Садржај

Глава 1

Реална и целобројна аритметика

Задатак: Рођендан

Аутор: Јелена Хаџи-Пурић

Такмичење: 2019/20 општинско, V разред, 1. задатак

Јелена се припрема за такмичење из историје. Ако је њен омиљени историјски херој прославио R-ти рођендан G-те године, напиши програм који одређује које године је тај херој рођен.

Опис улаза

Са улаза се учитавају природни бројеви R, G, сваки у посебном реду, $1 \leq R \leq 98, 100 \leq G \leq 2020$.

Опис излаза

Један природан број, година рођења хероја.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
`10`	`2010`	`46`	`1769`
`2020`		`1815`	

Решење

Ако је херој прославио G-те године R-ти рођендан, то значи да је рођен R година раније. Година рођења се израчунава тако што се од G-те године одброји R година уназад, тј. од G се одузме R.

```
#include <iostream>

using namespace std;

int main() {
  int R, G;
  cin >> R >> G;
  int rodjendan = G - R;
```

```
  cout << rodjendan << endl;
  return 0;
}
```

Задатак: Бананице

Такмичење: општинско 2018/2019, V разред, 1. задатак

Аутор: Филип Марић

Јована има m новчаница од 50 динара и двоструко више новчаница од 20 динара. Напиши програм који одређује колико јој новца остаје када купи n чоколадних бананица од којих свака кошта 15 динара (претпостави слободно да Јована има довољно новца да купи све бананице које жели).

Опис улаза

Са улаза се учитавају бројеви m и n (сваки у посебном реду).

Опис излаза

На излаз исписати број динара који јој остају када купи бананице.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
3	120	2	90
10		6	

Решење

Укупна количина новца који Јована има се може израчунати као $m \cdot 50 + (2m) \cdot 20$. Кусур се добија када се од тог броја одузме износ који је платила за бананице, а то је $n \cdot 15$.

```
#include <iostream>

using namespace std;

int main() {
  int m, n;
  cin >> m >> n;
  int novac = m*50 + (2*m)*20;
  int kusur = novac - n*15;
  cout << kusur << endl;
  return 0;
}
```

Задатак: Књига

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 1, V и VI разред, 1. задатак

Књига има N поглавља. Никола је првог дана прочитао A поглавља, а другог дана два поглавља више него првог. Написати програм који учитава целе позитивне бројеве N, A, и исписује колико поглавља је остало Николи да прочита.

Опис улаза

Са стандардног улаза се у првом реду уноси број N $(1 \leq N \leq 99)$, а у другом реду број A $(1 \leq A \leq 99)$. Улазни подаци су такви да преостали број поглавља никад није негативан.

Опис излаза

На стандардни излаз исписати један број, број поглавља која Никола још није прочитао.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
15	9	20	0
2		9	

Решење

Решење је сасвим једноставно. Ако је Никола првог дана прочитао A поглавља а другог дана $A + 2$ поглавља, то значи да му остаје да прочита још $N - A - (A + 2)$ поглавља.

```
#include <iostream>
using namespace std;

int main()
{
    int n, a;
    cin >> n >> a;
    cout << n - a - (a+2) << endl;
    return 0;
}
```

Задатак: Кусур

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 2, V разред, 1. задатак

Андрија и Бојан су у кафићу попили по један (исти) сок. Андрија је дао a, а Бојан b динара, а добили су кусур од k динара. Како треба да поделе кусур да би једнако платили?

Опис улаза

Са стандардног улаза се учитавају три цела позитивна броја, a, b и k редом, сваки у посебном реду. Сва три броја су мања или једнака 1000 и таква да постоји целобројно решење.

Опис излаза

На стандардни излаз исписати два броја, сваки у посебном реду. Први број је део кусура који треба да добије Андрија, а други број је Бојанов део кусура.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
80	0	500	390
100	20	150	40
20		430	

Решење

Два сока су коштала $a + b - k$, а један сок $s = \frac{a+b-k}{2}$ динара. Према томе, Андрија треба да добије $a - s$ а Бојан $b - s$ динара.

```
#include <iostream>

using namespace std;

int main() {
  int a, b, k;
  cin >> a >> b >> k;
  int sok = (a + b - k) / 2;
  cout << a - sok << endl;
  cout << b - sok << endl;
}
```

Задатак: Клацкалица

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 3, V и VI разред, 1. задатак

Ђаци прваци су после школе свратили у парк да се клацкају. Током игре покушавали су и да претегну једно друго. Испоставило се да је Милица са школском торбом тешка отприлике као Радоје без торбе, а Радоје са Милицином торбом као Бојан без торбе. Можете ли да одредите приближно Милицину тежину ако су познате тежине Радоја и Бојана?

Опис улаза

Са стандардног улаза се у првом реду уноси цео број R $(25 \leq R \leq 50)$, а у другом реду цео број B $(25 \leq B \leq 50)$, Радојева и Бојанова тежина редом.

Опис излаза

На стандардни излаз исписати један цео број, Милицину приближну тежину.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
35	28	39	37
42		41	

Решење

Тежину торбе можемо да одредимо као $T = B - R$. Након тога лако добијамо Милицину тежину као $B - T$.

```cpp
#include <iostream>
using namespace std;

int main() {
  int radoje, bojan;
  cin >> radoje >> bojan;
  int torba = bojan - radoje;
  int milica = radoje - torba;
  cout << milica << endl;
  return 0;
}
```

Задатак: Сестре

Аутор: Милан Вугделија

Такмичење: 2019/20 општинско, VI разред, 3. задатак

Ана, Бранка, Весна и Гоца су четири сестре. Ана је виша од Бранке исто колико Бранка од Весне, као и Весна од Гоце. Одредити висину Бранке и Весне ако је позната висина Ане и Гоце.

Опис улаза

У првом реду стандардног улаза је цео број A, Анина висина у центиметрима $(125 \leq A \leq 190)$. У другом реду стандардног улаза је цео број G, Гоцина висина у центиметрима $(85 \leq G \leq 160)$. Вредности A и G су тавке да су висине осталих сестара такође целобројне.

Опис излаза

У првом реду стандарног излаза исписати висину Бранке, а у другом висину Весне.

Пример

Улаз	*Излаз*
187	177
157	167

Решење

Означимо разлику у висини Ане и Бранке са R. Тада је $A = G + 3R$, односно $R = (A - G)/3$. Након што смо одредили вредност R, висина Бранке се једноставно одређује као $B = A - R$, а слично томе, Веснина висина је $V = B - R$.

```cpp
#include <iostream>

using namespace std;

int main() {
  int a, g;
  cin >> a >> g;
  int razlika = (a - g) / 3;
  int b = a - razlika;
  int v = b - razlika;
  cout << b << endl;
```

```
  cout << v << endl;
  return 0;
}
```

Задатак: Рам

Аутор: Нина Икодиновић

Такмичење: 2021/22. квалификације 1, V и VI разред, 1. задатак

Обим спољашњег дела рама правоугаоне слике је N cm. Колики је обим унутрашњег дела рама слике ако је рам широк M cm? Исписати вредност обима унутрашњег дела рама.

Опис улаза

У првом реду стандардног улаза налази се природан број N, а у другом природан број M. Бројеви N и M нису већи од 500.

Опис излаза

На стандардни излаз исписати само обим унутрашњег дела рама.

Пример 1

Улаз	*Излаз*
150 10	70

Пример 2

Улаз	*Излаз*
200 17	64

Решење

Потребно је израчунати обим унутрашњег дела рама слике када су нам дати обим спољашњег рама слике као и ширина рама. Може се приметити да се свака страница унутрашњег рама слике смањује за вредност двоструке ширине рама, тако да се обим смањује за осмоструку ширину рама.

```
#include <iostream>

using namespace std;

int main() {
   int n, m;
    cin >> n >> m;
    int obim = n - 8 * m;
    cout << obim << endl;
    return 0;
}
```

Задатак: Квиз

Аутор: Милан Вугделија

Такмичење: 2022/2023. квалификације 1, 6. разред, 1. задатак

Алиса и Бобан учествују у квизу, који се састоји од три игре. У првој игри Алиса је била боља од Бобана за A поена, а у другој је Бобан био бољи од Алисе за B поена. У трећој, одлучујућој игри, поени могу и да се губе и да се добијају. Алиса је ту игру управо завршила и освојила у њој T поена. Колико најмање поена Бобан треба да освоји у трећој игри, да би његов укупан резултат (збир поена из све три игре) био бољи од Алисиног?

Примети да Бобанов резултат у трећој игри може да буде и негативан, а да он ипак има бољи укупан резултат него Алиса.

Опис улаза

У првом реду стандардног улаза налази се број A, Алисина предност из прве игре. У другом реду се налази број B, Бобанова предност из друге игре. У трећем реду се налази број T, број Алисиних поена у трећој игри. Сва три броја су цели, већи од 0, а мањи од 50.

Опис излаза

На стандардни излаз исписати један цео број, најмањи број поена које Бобан треба да освоји, да би имао бољи укупан резултат.

Пример 1

Улаз	*Излаз*	*Објашњење*
3 5 12	11	После прве две игре Бобан је био у предности 2 поена. Након што Алиса одигра трећу игру, она прелази у вођство од 10 поена предности. Према томе, Бобану је довољно 11 поена да би имао бољи укупан резултат.

Пример 2

Улаз	*Излаз*
7 15 4	-3

Решење

Претпоставимо да је Алиса у предности након што одигра трећу игру. Тада је њена предност $A-B+T$ поена. Када би Бобан освојио управо толико поена у трећој игри, укупан резултат би био изједначен. Овај закључак важи и ако Алиса није остварила предност, само је тада вредност $A-B+T$ негативна или нула.

Да би победио, Бобану је довољно да освоји један поен више од $A-B+T$ у трећој игри, дакле $A-B+T+1$ поен.

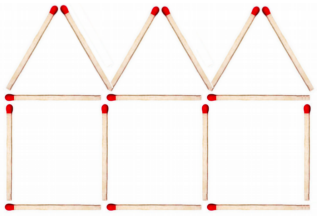
```cpp
#include <iostream>

using namespace std;

int main() {
    int a, b, t;
    cin >> a >> b >> t;
    cout << a-b+t+1 << endl;
    return 0;
}
```

Задатак: Палидрвца

Такмичење: општинско 2018/2019, V разред, 2. задатак

Аутор: Филип Марић

Напиши програм који одређује колико је палидрваца шибица потребно да би се направило n кућица (број n се учитава и вредност му је између 1 и 1000).



Опис улаза

Са улаза се учитава природан број n.

Опис излаза

На излаз исписати број палидрваца потребних да се изгради n кућица.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
2	11	3	16	83	416

Решење

Прва кућица је направљена од 6 палидрваца. Свака наредна додаје по 5 нових палидрваца. Укупан број палидрваца је зато $6 + 5(n - 1)$.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  cout << 6 + 5*(n-1) << endl;
  return 0;
}
```

Задатак: Слободна бацања

Аутори: Филип Марић, Владимир Кузмановић

Такмичење: окружно 2022/2023, VI разред, 1. задатак

У кошарци се сваки кош вреднује или са 3 поена (“тројка”, кош дат из далека током игре) или са 2 поена (“двојка”, кош дат са нормалне удаљености током игре) или са 1 поен (“слободно бацање”). Ако је познат резултат кошаркашке утакмице и број двојки и тројки који је постигао сваки од два тима, напиши програм који одређује број погођених слободних бацања за сваки од два тима, као и укупан број погођених слободних бацања.

Опис улаза

У првом реду стандардног улаза наведени су укупни поени прве и укупни поени друге екипе (цели бројеви између 0 и 200).

У другом реду стандардног улаза наведен је број двојки и број тројки које је дала прва екипа.

У трећем реду стандардног улаза наведен је број двојки и број тројки које је дала друга екипа.

Подаци су исправни.

Опис излаза

У првом реду стандардног излаза исписати број погођених слободних бацања прве и број погођених слободних бацања друге екипе, раздвојене размаком. У другом реду стандардног излаза исписати укупан број погодака из слободних бацања.

Пример

Улаз	*Излаз*
`100 85`	`5 3`
`40 5`	`8`
`35 4`	

Решење

Прво треба да учитамо све улазне параметре. Треба приметити да у решењу морамо да користимо целе бројеве, јер није могуће да било који од тимова као резултат има на пример вредност $95,5$. Први тим зваћемо `тим A`, а други тим зваћемо `тим B`.

Нека су редом укупни бројеви поена тимова обележени са A и B, бројеви постигнутих двојки тимова са A_2 и B_2 и бројеви постигнутих тројки са A_3 и B_3. Потребно је да одредимо бројеве слободних бацања оба тима, тј. вредности A_1 и B_1.

Да бисмо одредили број слободних бацања првог тима (A_1), потребно је да од укупног резултата првог тима (A) одузмемо бројеве поена који одговарају постигнутим двојкама и постигнутим тројкама. Приметимо да као улазне параметре добијамо број постигнутих двојки (A_2) и тројки (A_3), а не укупан број поена који тим кошевима одговора. Због тога, поене који одговарају двојкама добијамо тако што број двојки помножимо са 2, а број поена који одговарају тројкама добијамо тако што број тројки помножимо са 3. Одавде лако закључујемо да број слободних бацања првог тима одређујемо следећом формулом

$$A_1 = A - 2 \cdot A_2 - 3 \cdot A_3.$$

На сличан начин одређујемо и број слободних бацања другог тима B_1:

$$B_1 = B - 2 \cdot B_2 - 3 \cdot B_3.$$

Укупан број слободних бацања које су постигла оба тима одређујемо као збир вредности A_1 и B_1. На крају, само треба одштампати резултате у траженом формату.

Описани поступак је приказан у решењу.

```
#include <iostream>

using namespace std;

int main() {
  int A, B;
  cin >> A >> B;
  int A2, A3;
  cin >> A2 >> A3;
  int B2, B3;
  cin >> B2 >> B3;
  int A1 = A - 2*A2 - 3*A3;
  int B1 = B - 2*B2 - 3*B3;
  cout << A1 << " " << B1 << endl;
  cout << A1 + B1 << endl;
  return 0;
}
```

Задатак: Погрешна операција

Аутор: Небојша Варница

Такмичење: 2023/2024. квалификације 1, VII разред, 1. задатак

За дати природан број $n > 1$ одредити разломак који сабран са n даје исти резултат као и када се тај разломак помножи са n. Разломак треба да буде потпуно скраћен (бројилац и именилац треба да буду узајамно прости).

Опис улаза

У првом и једином реду стандардног улаза налази се природан број n, већи од 1.

Опис излаза

На стандардни излаз исписати бројилац и именилац траженог разломка у истом реду раздвојене само симболом `/`.

Пример

Улаз	*Излаз*	*Објашњење*
`3`	`3/2`	

$$3 + \frac{3}{2} = \frac{9}{2}$$

$$3 \cdot \frac{3}{2} = \frac{9}{2}$$

Решење

Претпоставимо да је разломак $\frac{a}{b}$. Тада треба да важи да је

$$n + \frac{a}{b} = n \cdot \frac{a}{b}$$

Одатле следи да је:

$$n = n \cdot \frac{a}{b} - \frac{a}{b} = (n-1)\frac{a}{b}$$

тј.

$$\frac{a}{b} = \frac{n}{n-1}$$

Пошто су n и $n-1$ узастопни природни бројеви, они су узајамно прости, па је решење разломак $\frac{n}{n-1}$.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  cout << n << "/" << n-1 << endl;
  return 0;
}
```

Задатак: Столњак

Аутор: Нина Икодиновић

Такмичење: 2021/22. квалификације 2, V разред, 1. задатак

Кројачица Мирјана је купила платно димензија N cm $\times$ M cm. Планирала је да од тог платна сашије што више квадратних столњака димензија K cm $\times$ K cm. Ако су познате димензије платна N и M, као и димензија столњака K, колико Мирјана може највише столњака да сашије? Остаци материјала не могу да се користе за састављање столњака.

Опис улаза

На стандадном улазу се налазе три целобројне вредности, свака у посебном реду. То су редом дужина платна N, ширина платна M и димензија столњака K. Ниједна од ових вредности није већа од 1000000.

Опис излаза

На стандардни излаз исписати једну целобројну вредност, број столњака које Мирјана може да сашије.

Пример 1

Улаз	*Излаз*
400 350 120	6

Пример 2

Улаз	*Излаз*
5000 3700 200	450

Решење

Потребно је израчунати укупан број столњака димензија $K \times K$ који се може сашити од платна димензија $N \times M$. Потребно је дужину и ширину платна целобројно поделити са димензијом столњака и те количнике помножити. Остатке при дељењу не треба узимати у обзир, јер према условима задатклја столњак не може да се састављa из делова.

```
#include <iostream>

using namespace std;

int main() {
int n, m, k, p, q;
    cin >> n >> m >> k;
    p = n / k;
    q = m / k;
    cout << p*q << endl;
   return 0;
}
```

Задатак: Звонце

ауīор: Душан Пōадић

Такмичење: 2021/22. квалификације 3, V разред, 1. задатак

На такмичењу у трчању такмичари трче на кружној стази дужине k метара. Поред стартне линије се налази звонце које зазвони сваки пут када такмичар протрчи поред њега (први пут звони када такмичар креће са стартне линије и онда након сваког пуног круга). Победник је онај такмичар који претрчи највећу дужину за задато време.

Дуња жели да учествује на овом такмичењу и пре тога жели да се припреми. Нажалост, она нема приступ стази потребне дужине, али се у парку поред њене куће налази кружна стаза дужине p метара коју може да користи. Дуња је отишла да вежба и истрчала је тачно t кругова, а сада је занима колику је дужину претрчала у метрима, као и колико пута би звоно зазвонило да је трчала на правој стази.

Опис улаза

У првој линији улаза се налази дужина такмичарске кружне стазе k ($0 < k \leq 600$, k је реалан број), у другој линији дужина стазе на којој Дуња вежба p ($0 < p \leq 600$), а у трећој линији је број кругова које је Дуња претрчала t ($1 \leq t \leq 10$, t је природан број).

Опис излаза

У првој линији излаза исписати колика је укупна дужина коју је Дуња претрчала, а у другој колико би се пута чуло звоно током њеног трчања да је трчала на правој стази.

Пример 1

Улаз	*Излаз*	*Објашњење*

`207.4`
`70.8`
`6`
	`424.8`
	`3`
		Дуња је претрчала тачно 6 кругова, а пошто је дужина стазе на којој трчи 70,8 метара укупно је претрчала 424,8 метара. Пошто је дужина праве тркачке стазе 207,4 метра, да је трчала на њој претрчала би нешто више од 2 цела круга, тако да звонце звони 3 пута (једном на старту и по једном за сваки претрчани круг).

Пример 2

Улаз	*Излаз*

`600`
`40.2`
`5`
	`201`
	`1`

Пример 3

Улаз	*Излаз*

`201.4`
`302.1`
`2`
	`604.2`
	`4`

Решење

Дуња је претрчала тачно t кругова, а како је сваки круг дужине p, укупна дужина коју је Дуња претрчала се добија множењем ова два броја. Сада када имамо укупну дужину потребно је одредити колико би се пута чуло звоно на стартној линији праве стазе. Овај проблем је практично обрнут од првог проблема - прво смо имали број кругова и дужину стазе, а тражили смо укупну претрчану дужину. Сада имамо укупну претрчану дужину и дужину стазе, а тражи се број претрчаних кругова. То се може добити дељењем укупне претрчане дужине дужином праве стазе. Нас занима колико је пута Дуња прошла поред стартне линије. Дуња пролази поред стартне линије сваки пут када заврши трчање круга, а на то треба додати и почетну позицију (тада се налазила на стартној линији, али није завршила ниједан круг). Уколико је, на пример, Дуња претрчала 4,6 кругова, она је заправо завршила 4 цела круга, па ће звонце зазвонити 5 пута (4 пута по завршетку кругова плус једном на почетку трке). Дакле, број звоњења се може добити када се број претрчаних кругова (количник укупне дужине и дужине стазе) заокружи на мањи цео број, а затим се дода још 1. Увек је добро проверити да ли наше решење ради за граничне случајеве, а у овом проблему је то када је број претрчаних кругова цео број. Уколико је нпр. Дуња претрчала тачно 500м, а дужина стазе је 500м, онда је она претрчала тачно 1 круг. По нашем алгоритму број звоњења је 1 + 1 = 2, што је тачно решење (једном кад је кренула са старта и једном када је завршила трку).

Треба поменути да решење у коме се број кругова само заокружује на већи број, без додавања јединице ради у свим случајевима, осим у граничним када је број кругова цео број. Овакво решење доноси 50% поена.

```cpp
#include <iostream>
#include <cmath>
#include <iomanip>

using namespace std;

int main() {
   double k, p;
   int t;
   cin >> k >> p >> t;
   double ukupnaDuz = p * t;
   int brojZ = floor(ukupnaDuz / k) + 1;
   cout << ukupnaDuz << endl << brojZ;
```

```
    return 0;
}
```

Задатак: Век

Аутор: Јелена Хаџи-Пурић

Такмичење: 2019/20 општинско, V разред, 3. задатак

Век је временско раздобље од 100 година. У овом задатку претпоставите да у нашој ери (н.е.) први век је трајао од 1. јануара 1. године до 31. децембра 100. године, као и да је други век н.е. почео 1. јануара 101. године. Завршетак другог века н.е. је 31.децембра 200. године. Ми смо тренутно у 21. веку који је почео 1. јануара 2001. године.

Напишите програм који за дату годину G исписује ком веку припада та година.

Опис улаза

Један природан број, број G, $1 \leq G \leq 2305$

Опис излаза

Један природан број, век коме припада текућа година.

Пример 1		**Пример 2**		**Пример 3**		**Пример 4**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
313	4	1499	15	1500	15	1601	17

Решење

Ознака века се поклапа са ознаком стотине у којој се налази број који означава текућу годину. Зато је довољно гледати целобројни количник при дељењу са 100. Када је година дељива са 100, онда је век једнак том количнику. Ако година није дељива са 100, онда се добијени количник увећа за 1, јер је наша ера почела првим, а не нултим веком.

```
#include <iostream>

using namespace std;

int main() {
  int godina;
  cin >> godina;

  int vek;
  if (godina % 100 == 0)
    vek = godina / 100;
  else
    vek = godina / 100 + 1;

  cout << vek << endl;
  return 0;
}
```

Ако би се векови и године бројали од нуле, систем би био много једноставнији, јер би се век увек добијао као целобројни количник при дељењу са 100. Рачунање можемо свести на овај систем ако од броја године одузмемо 1 (да бисмо са бројања година од 1 прешли на бројање од 0), а онда након израчунавања века на резултат додајемо један (да бисмо се са бројања векова од 0 вратили на бројање векова од 1).

```
#include <iostream>

using namespace std;

int main() {
  int godina;
  cin >> godina;
  int vek = (godina - 1) / 100 + 1;
  cout << vek << endl;
  return 0;
}
```

Задатак: Гориво

Аутор: Филип Марић

Такмичење: 2022/2023. квалификације 1, 6. разред, 2. задатак и 7. разред, 1. задатак

Три породице које живе у месту А желе да отпутују на излет у удаљени град Б. Свака породица путује својим аутомобилом и пре пута треба да наточи гориво, при чему је на пумпи неопходно купити увек цео број литара горива. Ако је познато растојање од места А до места Б и ако је позната потрошња горива сваког аутомобила на 100 километара, одредити најмању укупну количину коју све три породице заједно треба да купе да би сви успели да стигну до града Б.

Опис улаза

Са стандардног улаза се уноси прво цео број r ($10 \leq r \leq 1000$) који представља растојање од места А до места В (у километрима), а затим три цела броја p_1, p_2, p_3 ($2 \leq p_i \leq 20$) која представљају потрошњу сваког од три аутомобила на 100 километара.

Опис излаза

На стандардни излаз исписати тражену минималну количину горива.

Пример 1

Улаз	*Излаз*	*Објашњење*
850 7 8 9	205	Првом аутомобилу је потребно 59,5 литара да би прешао 850 километара, па ће купити 60 литара, другом је потребно тачно 68 литара, а трећем је потребно 76.5 литара, па ће купити 77 литара бензина, па ће сви заједно купити 60+68+77=205 литара бензина.

Пример 2

Улаз	*Излаз*
987 6 9 11	258

Решење

Претпоставимо да за аутомобил i купујемо g_i горива. Пошто тај аутомобил за један литар горива пређе $100/p_i$ километара, за g_i литара он пређе $\frac{100 \cdot g_i}{p_i}$ километара. Потребно је одредити најмањи број g_i такав да је $\frac{100 \cdot g_i}{p_i} \geq r$ тј. најмањи цео број g_i већи или једнак од $\frac{p_i \cdot r}{100}$, а то је $\left\lceil \frac{p_i \cdot r}{100} \right\rceil$. Резултат добијамо тако што израчунавамо ову вредност за све три вредности потрошње и саберемо их. Израчунавање горива за један аутомобил можемо извршити у посебној функцији.

```
#include <iostream>

using namespace std;

int gorivo(int potrosnja, int rastojanje) {
  // moze i
  // return (potrosnja * rastojanje + 99) / 100;
  if (potrosnja * rastojanje % 100 == 0)
    return (potrosnja * rastojanje) / 100;
  else
    return (potrosnja * rastojanje) / 100 + 1;
}

int main() {
  int rastojanje;
  cin >> rastojanje;
  int potrosnja1, potrosnja2, potrosnja3;
  cin >> potrosnja1 >> potrosnja2 >> potrosnja3;
  cout << gorivo(potrosnja1, rastojanje) +
          gorivo(potrosnja2, rastojanje) +
          gorivo(potrosnja3, rastojanje) << endl;
   return 0;
}
```

Задатак: Сат

Аутор: Милан Вугделија

Такмичење: 2019/20 општинско, VIII разред, 4. задатак

Кукавица из зидног сата се оглашава на пун сат онолико пута колико има сати (најмање 1 а највише 12), а на пун сат и 15, 30 или 45 минута по једном. Бака Ката је навила сат у S_1 сати и M_1 минута, а сада је S_2 сати и M_2 минута. Од навијања сата је прошло мање од 12 сати. Колико пута се кукавица огласила од навијања?

Опис улаза

- У првом реду стандардног улаза број S_1 ($1 \leq S_1 \leq 12$);
- У другом реду стандардног улаза број M_1 ($1 \leq M_1 \leq 59, M_1 \notin \{15, 30, 45\}$);
- У трећем реду стандардног улаза број S_2 ($S_1 < S_2 \leq 12$);
- У четвртом реду стандардног улаза број M_2 ($1 \leq M_2 \leq 59, M_2 \notin \{15, 30, 45\}$);

Опис излаза

На стандардни излаз исписати један цео број, број оглашавања кукавице.

Пример 1

Улаз	*Излаз*	*Објашњење*
3 16 4 1	6	Кукавица се огласила једном у 3:30, затим једном у 3:45 и четири пута у 4:00.

Пример 2

Улаз	*Излаз*	*Објашњење*
7 16 9 28	23	$1+1+8+1+1+1+9+1=23$

Решење

Задатак се може решити и помоћу циклуса, а овде ћемо показати аритметичко решење. Нека је $t_1 = 60 \cdot s_1 + m_1, t_2 = 60 \cdot s_2 + m_2$. Одредимо прво број различитих тренутака M, у којима се кукавица оглашавала. Тих тренутака има исто колико и времена (изражених у минутима) дељивих са 15, која су између t_1 и t_2, дакле $M = \lfloor \frac{t_2}{15} \rfloor - \lfloor \frac{t_1}{15} \rfloor$. Од тих M тренутака, њих $s_2 - s_1$ је било на пун сат. То значи да се кукавица на непун сат оглашавала $M - (s_2 - s_1)$ пута, сваки пут по једном. На овај број оглашавања треба још додати и оглашавања на пуне сате. На сваки пун сат кукавица се оглашава онолико пута колико има сати, па је број оглашавања на пуне сате једнак збиру (природних) бројева већих од s_1 а мањих од s_2. Знајући да је збир свих бројева од 1 до K једнак $\frac{K \cdot (K+1)}{2}$, тражени број оглашавања на пуне сате је $\frac{s_2 \cdot (s_2+1) - s_1 \cdot (s_1+1)}{2}$. Сабирањем броја оглашавања на пуне и непуне сате добијамо тражени укупан број оглашавања.

```cpp
#include <iostream>

using namespace std;

int main()  {
  int s1, m1, s2, m2;
  cin >> s1 >> m1 >> s2 >> m2;

  int brPunihSati = s2 - s1;
  int ukNaNepuneSate = (s2*60 + m2) / 15 - (s1*60 + m1) / 15 - brPunihSati;
  int ukNaPuneSate = (s2*(s2+1) - s1*(s1+1)) / 2;

  cout << ukNaPuneSate + ukNaNepuneSate << endl;
  return 0;
}
```

Задатак: Вода

Аутор: Милан Вујделија

Такмичење: 2021/22. квалификације 3, VI разред, 1. задатак

Из бурета могу да се напуне 4 канте, из канте може да се напуни 5 бокала, а из бокала може да се напуни 6 чаша. Написати програм који одређује колико пуних буради, канти, бокала и чаша је потребно да се напуни n чаша воде. Подразумева се да се вода држи у што мањем броју посуда.

Опис улаза

У првом и једином реду стандардног улаза налази се природан број n, не већи од 1000.

Опис излаза

На стандардни излаз исписати само четири неозначена цела броја, сваки у посебном реду. Први је број буради, други број канти (мањи од 4), трећи број бокала (мањи од 5), а четврти број чаша (мањи од 6).

Пример

Улаз	*Излаз*
202	1
	2
	3
	4

Решење

Задатак се једноставно решава применом операција целобројног дељења и остатка при дељењу (тзв. модула). Нека је C_u укупан број чаша. Када би се те чаше пресуле у бокале, добило би се укупно $B_u = \left\lfloor \frac{C_u}{6} \right\rfloor$ бокала и преостало би $C_p = C_u \bmod 6$ чаша.

Даљим пресипањем B_u бокала у канте добило би се укупно $K_u = \left\lfloor \frac{B_u}{5} \right\rfloor$ канти и преостало би $B_p = B_u \bmod 5$ бокала.

Коначно, пресипањем канти у бурад добило би се укупно $D_u = \left\lfloor \frac{K_u}{4} \right\rfloor$ буради и преостало би $K_p = K_u \bmod 4$ канти.

Резултати које треба исписати су редом укупан број буради D_u, преостали број канти K_p, преостали број бокала B_p и преостали број чаша C_p.

```
#include <iostream>

using namespace std;

int main() {
    int ukupnoCasa;
    cin >> ukupnoCasa;
    int preostaloCasa = ukupnoCasa % 6;
    int ukupnoBokala = ukupnoCasa / 6;

    int preostaloBokala = ukupnoBokala % 5;
    int ukupnoKanti = ukupnoBokala / 5;

    int preostaloKanti = ukupnoKanti % 4;
    int preostaloBuradi = ukupnoKanti/ 4;

    cout << preostaloBuradi << endl;
```

```
    cout << preostaloKanti << endl;
    cout << preostaloBokala << endl;
    cout << preostaloCasa << endl;
    return 0;
}
```

Задатак: Фестивал

Такмичење: окружно 2018/2019, V разред, 1. задатак

Аутор: Филип Марић

Током школског фестивала сви ученици школе ће се шетати улицама око школе. Током шетње сви ће се поређати у врсте од по 15 ученика и ходаће у пуно таквих врста које корачају једна иза друге. Ако ученике бројимо редом, почевши од првог ученика у првој врсти (како је приказано у наставку) напиши програм који за n-тог по реду ученика одређује у којој се врсти и којој колони налази.

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 врста 1
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 врста 2
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 врста 3
46 47 48 49 ...                              врста 4
```

Опис улаза

Са стандардног улаза се учитава број n ($1 \leq n \leq 1000$).

Опис излаза

На стандардни излаз исписати редни број врсте и колоне, раздвојене размаком.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
11	1 11	16	2 1	103	7 13

Решење

Када би се бројање вршило од нуле, довољно би било пронаћи целобројни количник и остатак броја n при дељењу са 15. Пошто се бројање врши од 1, можемо да одузмемо 1 од броја n, затим да одредимо врсту и колону које се броје од 0 и на те бројеве да додамо 1 (да бисмо прешли на бројање од 1).

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  int v = (n - 1) / 15 + 1;
  int k = (n - 1) % 15 + 1;
  cout << v << " " << k << endl;
```

```
  return 0;
}
```

Задатак: Дан у месецу

Аутор: Филип Марић

Такмичење: Ревијално такмичење 2019/2020., V и VI разред, 1. задатак

Ако се зна који је дан у недељи био први дан текућег месеца, напиши програм који одређује који дан је данас.

Опис улаза

Прва линија стандардног улаза садржи ознаку дана у недељи првог дана текућег месеца (1 - понедељак, 2 - уторак, 3 - среда, 4 - четвртак, 5 - петак, 6 - субота, 7 - недеља), а друга линија садржи редни број текућег дана у текућем месецу.

Опис излаза

На стандардни излаз исписати ознаку данашњег дана.

Пример 1

Улаз	*Излаз*	*Објашњење*
1 4	4	Први дан је био понедељак, па је четврти дан четвртак.

Пример 2

Улаз	*Излаз*	*Објашњење*
1 8	1	Први дан је био понедељак, па је и осми дан понедељак.

Пример 3

Улаз	*Излаз*	*Објашњење*
3 17	5	Први дан је била среда, па је седамнаести дан петак.

Решење

На ознаку првог дана у месецу додаћемо колико је дана прошло до данашњег. Тај број протеклих дана се добија тако што се од редног броја данашњег дана одузме 1. Пошто у обзир треба узети и периодично понављање дана, треба пронаћи остатак при дељењу са 7. Пошто се дани броје од 1 до 7, а остаци при дељењу са 7 су од 0 до 6, примењујемо “трик” да пре израчунавања остатка одузмемо 1, а након израчунавања додамо 1.

```
#include <iostream>

using namespace std;

int main() {
  int prvi;
  cin >> prvi;
  int danas;
  cin >> danas;
```

```
  cout << (prvi + (danas - 1) - 1) % 7 + 1 << endl;
  return 0;
}
```

Задатак: Одбијање

Аутор: Владан Ковачевић

Такмичење: државно 2022/2023, VIII разред, 1. задатак

Дат је билијарски сто облика правоугаоника висине n и ширине m. Бела кугла започиње кретање из доњег левог угла и удара у горњу ивицу, на растојању d од леве ивице. Она се затим одбија ка доњој ивици стола, па ка горњој и тако даље, све док не дотакне десну ивицу (слика). Одредити да ли ће она десну ивицу да дотакне у средишту, испод средишта или изнад средишта.

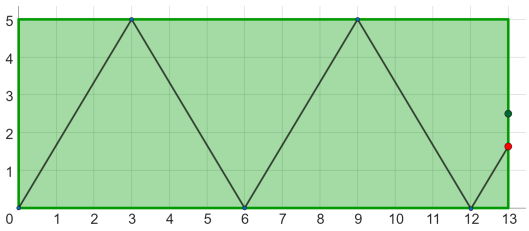
Опис улаза

Са стандардног улаза се у једном реду уносе три позитивна цела броја n, m и d ($1 \leq n, m \leq 100000$, $1 \leq d \leq 100$) раздвојена размацима.

Опис излаза

На стандардни излаз исписати 0, -1 или 1, у зависности од тога да ли ће кугла да дотакне десну ивицу у средишту, испод средишта или изнад средишта.

Пример 1

Улаз	*Излаз*	*Објашњење*
`5 13 3`	`-1`	

Пример 2

Улаз	*Излаз*
`4 7 2`	`0`

Решење

Приметимо да кугла прво удара горњу ивицу стола на растојању d од леве ивице, затим удара доњу ивицу на растојању $2d$,па опет горњу на растојању $3d$ и тако све док не удари десну ивицу стола.

Дакле, док не заврши кретање, кугла ће горњу ивицу да удара на растојањима d, $3d$, $5d$, … од леве ивице, односно доњу ивицу на растојањима $2d$, $4d$, $6d$, … од леве ивице.

Како кугла удара горњу (доњу) ивицу на растојањима дељивим са d, она ће десну ивицу ударити на растојању $r = m \bmod d$ од последњег ударца у горњу (доњу) ивицу, док ће се од горње и доње ивице одбити укупно $k = \lfloor \frac{m}{d} \rfloor$ пута пре него што удари десну ивицу.

Кугла ће ударити средиште десне ивице само ако се налази на пола путање између горње и доње ивице, без обзира на то да ли се креће од горње према доњој или обрнуто. Дакле, ако за $r = d/2$ одн. $2r = d$, резултат ће бити `0`.

Даље, приметимо да парно `k` значи да је кугла последњи пут ударила доњу ивицу, док непарно `k` значи да је последњи пут ударила горњу ивицу пре ударања десне ивице. Ако је кугла последњи пут ударила доњу ивицу, онда ће она десну ивицу ударити испод средишта ако пређе мање од пола путање, односно изнад средишта ако пређе више од пола путање између горње и доње ивице. Обрнуто важи ако је кугла последњи пут ударила горњу ивицу.

Дакле, за $k \bmod 2 = 0 \wedge 2r < d$ и $k \bmod 2 = 1 \wedge 2r > d$ решење ће бити -1, а за $k \bmod 2 = 0 \wedge 2r > d$ и $k \bmod 2 = 1 \wedge 2r < d$ решење ће бити 1.

```
#include <iostream>

using namespace std;

int main() {
  int n, m, d;
  cin >> n >> m >> d;

  int k = m / d;
  int r = m % d;

  if (2*r == d)
    cout << 0 << endl;
  else if (2*r < d)
    cout << (k % 2 ? 1 : -1) << endl;
  else
    cout << (k % 2 ? -1 : 1) << endl;

  return 0;
}
```

Задатак: Повећај сваку цифру

Аутор: Небојша Варница

Такмичење: 2021/22. квалификације 1, V и VI разред, 2. задатак

За дати двоцифрен природан број формирати број који настаје када се свака његова цифра повећа за 1

Опис улаза

У првом и једином реду стандардног улаза налази се двоцифрен број

Опис излаза

На стандардни излаз исписати број који се добија када се свака цифра датог броја повећа sa 1

Пример 1

Улаз	*Излаз*
47	58

Пример 2

Улаз	*Излаз*
39	410

Решење

Треба одвојити цифру десетица од цифре јединица (што можемо урадити одређивањем целобројног количника и остатка при дељењу са 10), повећати обе за 1 и добијене бројеве спојити.

```
#include <iostream>

using namespace std;

int main() {
    int n;
    cin >> n;
    int a = n / 10;
    int b = n % 10;
    cout << a + 1 << b + 1 << endl;
    return 0;
}
```

Задатак: Просек цифара

Аутор: Филип Марић

Такмичење: окружно 2021/22., V разред, 2. задатак

Напиши програм који израчунава просек цифара унетог петоцифреног броја.

Опис улаза

Са стандардног улаза се учитава петоцифрени број.

Опис излаза

На стандардни излаз исписати просек цифара унетог броја, заокружен на једну децималу.

Пример 1

Улаз	*Излаз*	*Објашњење*
`12345`	`3.0`	Просек се израчунава као $(1+2+3+4+5)/5$.

Пример 2

Улаз	*Излаз*	*Објашњење*
`10000`	`0.2`	Просек се израчунава као $(1+0+0+0+0)/5$.

Решење

Цифре можемо издвојити на уобичајени начин, одређивањем целобројног количника и остатка при дељењу тежинама цифара (на пример цифру на позицији хиљада можемо одредити одређивањем целобројног количника са 1000 и затим одређивањем остатка при дељењу са 10 — целобројни дељењем са 1000 уклањамо три цифре из декадног записа броја, а онда остатком при дељењу са 10 одређујемо последњу цифру тако добијеног броја).

Просек израчунавамо сабирањем добијених цифара и дељењем са 5.

```
#include <iostream>
#include <iomanip>
```

```cpp
using namespace std;

int main() {
  int n;
  cin >> n;
  int c0 = n % 10;
  int c1 = (n / 10) % 10;
  int c2 = (n / 100) % 10;
  int c3 = (n / 1000) % 10;
  int c4 = (n / 10000) % 10;
  cout << fixed << showpoint << setprecision(1) <<
    (c0 + c1 + c2 + c3 + c4) / 5.0 << endl;
  return 0;
}
```

Задатак: Цифре се преклапају

Аутор: Душан Попадић

Такмичење: 2023/2024. квалификације 1, VI разред, 1. задатак

Маре и Паја играју игрицу и дошли су до нивоа где треба решити шифру. Схватили су да је шифра заправо један петоцифрен број који је био записан на зиду на претходном нивоу игрице. Ниједан од њих не може да се сети тог броја, али се сећају делова. Маре је запамтио прве три цифре, а Паја последње три. Помозите им да пређу на следећи ниво тако што ћете за њих одредити тражени петоцифрен број. Претпоставити да је свако од њих исправно запамтио цифре, односно да су унети подаци исправни.

Опис улаза

У првом реду се налази троцифрен број - део који је запамтио Маре. У другом реду се налази троцифрен број - део који је запамтио Паја.

Опис излаза

Исписати тражени петоцифрен број.

Пример 1

Улаз	*Излаз*	*Објашњење*
145 563	14563	Маре је запамтио прве три цифре, тако да су прве три цифре броја 145. Паја је запамтио последње три цифре, тако да су последње три цифре броја 563. Једини петоцифрени број за који ово важи је 14563.

Пример 2

Улаз	*Излаз*
788 851	78851

Решење

Пошто се тражи петоцифрени број, а дате су прве три и последње три цифре тог броја, то знали да ће последња цифра Маретовог броја уједно бити и прва цифра Пајиног броја. Дакле потребно је изабрати један од следећа два приступа:

На цео Маретов број ћемо налепити последње две цифре Пајиног броја.

```
#include <iostream>

using namespace std;

int main() {
    int p, m;
    cin >> m >> p;
    int broj = m * 100 + p % 100;
    cout << broj << endl;
   return 0;
}
```

Одстранићемо последњу цифру Маретовог броја и на њега налепити цео Пајин број.

```
#include <iostream>

using namespace std;

int main() {
    int p, m;
    cin >> m >> p;
    int broj = (m / 10) * 1000 + p;
    cout << broj;
    return 0;
}
```

Задатак: Цврка

Аутор: Милан Вугделија

Такмичење: 2019/20 општинско, VI разред, 4. задатак

Кукавица Цврка из зидног сата се оглашава на пун сат онолико пута колико има сати, а на пун сат и 15, 30 или 45 минута по једном. Бака Ката је навила сат у S сати и M минута. Када ће се Цврка следећи пут огласити, ако знамо да се она никад не оглашава чим је навијена?

Опис улаза

У првом реду стандардног улаза број S ($1 \leq S \leq 10$); У другом реду стандардног улаза број M ($0 \leq M \leq 59$);

Опис излаза

У једном реду два цела броја, сат и минут следећег Цвркиног оглашавања, раздвојени једним размаком.

Пример 1

Улаз	*Излаз*
5 30	5 45

Пример 2

Улаз	*Излаз*
9 49	10 0

Решење

Означимо са t време у минутима протекло од последње поноћи. Број t можемо израчунати као $t = s \cdot 60 + m$. Време следећег Цвркиног оглашавања (у минутима протеклим од последње поноћи) је најмањи број који је већи од t и дељив са 15. Ово време можемо да израчунамо као $(\lfloor \frac{t}{15} \rfloor + 1) \cdot 15$. Приметимо да овде евентуални остатак желимо да изгубимо, што значи да је потребно целобројно дељење (оператор `//` у Пајтону). Остаје још да израчунато време претворимо у сате и минуте и испишемо.

```
#include <iostream>

using namespace std;

int main() {
  int s, m;
  cin >> s >> m;
  int t = s * 60 + m;
  t = (t / 15 + 1) * 15;
  int s2 = t / 60;
  int m2 = t % 60;
  cout << s2 << " " << m2 << endl;
  return 0;
}
```

Задатак: Царе господаре

Аутор: Милан Вугделија

Такмичење: 2019/20 општинско, VII разред, 2. задатак

Некада давно Ваши родитељи су играли игру Царе, царе, Господаре, колико је сати? У тој игри Цар свој одговор даје у облику: M минута до H сати. Напишите програм који за унете бројеве M и H исписује тачно време у уобичајеном облику час:минут. Претпоставити да је број M такав да је тражено време увек у истом дану. Дан траје од 00:00 до 23:59.

Опис улаза

Са улаза се учитавају природни бројеви M, H (сваки у посебном реду, $0 \leq M \leq 1440, 0 \leq H \leq 23$).

Опис излаза

Време у формату $HH : MM$, где је HH двоцифрена ознака за број часова у 24-часовном формату, а MM је двоцифрена ознака за број минута у часу.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
23 23	22:37	15 10	09:45	55 22	21:05

Решење

Цар свој одговор даје у облику: M минута до H сати. Најпре ћемо тај одговор претворити у укупан број минута који је протекао од поноћи ($ukupnominuta = H * 60 - M$). Укупан број минута ћемо претворити у текући час и минут користећи чињеницу да један час има 60 минута ($cas = ukupnominuta/60; minut = ukupnominuta\%60;$). На крају је потребно водити рачуна о форматираном испису два двоцифрена броја. Можемо користити посебне наредбе намењене испису целог броја у пољу минималне ширине 2 или уз помоћ наредби гранања дописивати водећу нулу у случају да су вредности часа или минута једноцифрени бројеви.

Треба повести рачуна и о специјалном случају када је $H = 0$ (он се најједноставније решава тако што се користи $H = 24$).

```
#include <iostream>

using namespace std;

int main() {
  int M, H;
  cin >> M >> H;
  if (H == 0) H = 24;
  int ukupnominuta = H * 60 - M;
  int cas, minut;
  cas = ukupnominuta / 60;
  minut = ukupnominuta % 60;
  if (cas < 10) cout << "0";
  cout << cas << ":";
  if (minut < 10) cout << "0";
  cout << minut << endl;
  return 0;
}
```

Глава 2

Елементарне операције са нискама

Задатак: Први понедељак у месецу

Аутор: Филип Марић

Такмичење: окружно 2021/22., VI разред, 2. задатак

Ако је познато ког дана у недељи је одређени датум, напиши програм за одређивање датума на који пада први понедељак у том месецу.

Опис улаза

У првом реду стандардног улаза је један датум у формату `dd-mm-gggg`, а у другом реду је цео број из опсега $[0, 6]$, редни број дана у недељи. Број 0 у другом реду је ознака за недељу, 1 за понедељак итд. до броја 6, који означава суботу.

Опис излаза

На стандардни излаз исписати датум првог понедељка у истом месецу, у формату `dd-mm-gggg`.

Пример

Улаз	*Излаз*	*Објашњење*
`19-02-2022` `6`	`07-02-2022`	Ако је деветнаести фебруар у суботу, онда је седмог фебруара понедељак, а то је и први понедељак датог месеца.

Решење

Када учитамо ниску која садржи датум, издвајамо њене делове који представљају дан, месец и годину. Дан преводимо у целобројну вредност. Датум првог понедељка у месецу можемо одредити на следећи начин. Одузимањем ознаке дана у недељи од текућег дана добијамо број који нам говори када је била недеља у текућој седмици. Тај број може бити и нула (на пример, ако је текући дан 2. у месецу и он је уторак), па и негативан (на пример, ако је текући дан 3. у месецу и он је петак, добијамо да је редни број недеље $3 - 5 = -2$), а може бити и већи од 7, па он још не представља редни број дана прве недеље у месецу. Међутим, проналажењем остатка тог броја при дељењу са 7, добија се број који је између 0 и 6, при чему, да бисмо

избегли негативне вредности пре проналажења остатка број увећавамо за 7 (израчунавање разлике по модулу k типично подразумева увећање за k пре одређивања остатка тј. важи да је $(a-b) \bmod k = (a \bmod k - b \bmod k + k) \bmod k$). Овим смо добили датум прве недеље у месецу, осим у случају када се добије 0 (прва недеља је тада 7. у месецу). Међутим, у свим случајевима се увећањем тог броја за 1 заиста добије датум првог понедељка у месецу (и када је остатак једнак 0, прва недеља је 7. у месецу, а први понедељак је 1. у месецу).

Ако је 19. у месецу субота (одређена бројем 6), разлика 13, даје остатак 6 при дељењу са 7 (то се не мења ако се разлика увећа на 20 пре проналажења остатка), што значи да је 6. у месецу била недеља, а 7. у месецу је понедељак.

Ако је 3. у месецу петак (одређен бројем 5), дан прве недеље у месецу се добија одређивањем остатка броја $3 - 5 + 7$ са 7, што је 5, па је први понедељак 6. у месецу.

```
#include <iostream>
#include <string>
#include <map>
#include <iomanip>

using namespace std;

int main() {
  string datum;
  cin >> datum;
  string dan, mesec, godina;
  dan = datum.substr(0, 2);
  mesec = datum.substr(3, 2);
  godina = datum.substr(6, 4);
  int dan_u_nedelji;
  cin >> dan_u_nedelji;
  int prvi_pon = (stoi(dan) - dan_u_nedelji + 7) % 7 + 1;
  cout << setfill('0') << setw(2) << prvi_pon << "-" << mesec << "-" << godina << endl;
  return 0;
}
```

Задатак: Украсна трака

Аутор: Небојша Варница

Такмичење: 2021/22. квалификације 3, V и VI разред, 2. задатак

Милена се бави израдом украсних трака тако што по њима исписује задати текст највећи могући број пута. На почетак и крај Милена увек ставља знак * а између два текста оставља једно празно место. Пошто има све више посла одлучила је да набави штампач који штампа по траци и жели да јој помогнеш.

Опис улаза

У првом реду је текст који Милена треба да испише (најмање дужине 1 а највише 30 знакова). У другом реду је дужина траке $N, 2 \geq N \geq 80$ - цео број који који представља укупан број знакова који се на траци могу записати.

Опис излаза

У једином реду приказује се изглед траке по завршетку исписа

Пример 1

Улаз *Излаз*

```
TAKMICENJE  *TAKMICENJE TAKMICENJE TAKMICENJE TA*
37
```

Пример 2

Улаз *Излаз*

```
SRECAN RODJENDAN  *SRECAN RODJENDAN SRECAN RODJENDAN SRECAN RODJENDAN SRECAN*
59
```

Пример 3

Улаз *Излаз*

```
SABAC     *SABAC SABAC SABAC SABAC *
26
```

Решење

На основу дужине n траке умањене за два (јер се на крајевима пишу две звездице) и дужине a датог текста проширеног једним размаком одређује се број понављања текста k и дужина m дела текста који може да стане на преостали део траке. Број понављања k је једнак целобројном количнику бројева n и a, а број m остатку при дељењу бројева n и a. Затим се текст креира и приказује.

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string t;
    getline(cin, t);
    t += " ";
    int n;
    cin >> n;
    n -= 2; // prostor za dve zvezdice
    int a = t.size();
    int k = n / a; // broj pojavljivanja teksta
    int m = n % a; // duzina preostalog dela teksta

    cout << '*';
    for (int i = 0; i < k; i++)
        cout << t;
    cout << t.substr(0, m) << '*';

    return 0;
}
```

Задатак: Запета

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 1, VII и VIII разред, 1. задатак

Дат је низ ASCII карактера, међу којима се појављује тачно један знак `,` (запета) и на самом крају тачно један знак `.` (тачка). Сви остали знаци, ако их има, су слова енглеске абецеде, цифре, заграде и размаци и они могу да се понављају. Написати програм који учитава дати низ карактера, а на стандардни излаз исписује текст у коме су део до запете и део од запете разменили места, док тачка остаје на крају текста.

Опис улаза

На стандардном улазу се у једном реду налази низ од највише 100 ASCII карактера.

Опис излаза

На стандардни излаз исписати измењени низ карактера.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`Da si hteo, mogao si.`	`mogao si,Da si hteo.`	`,a b c.`	`a b c,.`	`x,.`	`,x.`

Решење

У овом задатку је најпогодније да се прочита цео ред улаза као један стринг (без обзира на то да ли стринг садржи размаке). Након учитавања можемо да издвојимо део A од почетка стринга до запете (не укључујући запету) а затим и део B од запете до тачке (не укључујући запету и тачку).

У језику C++ позицију карактера `c` унутар стринга `s` можемо одредити позивом `s.find(c)`, док део стринга `s` који почиње на позицији `p` и има `d` карактера можемо одредити позивом `s.substr(p, d)`.

После оваквог издвајања остаје још само да испишемо делове у траженом редоследу (део B, запета, део A и на крају тачка).

```
#include <iostream>
using namespace std;

int main()
{
    string s;
    getline(cin, s);
    int pozZapete = s.find(',');
    int pozTacke = s.find('.');
    string a = s.substr(0, pozZapete);
    string b = s.substr(pozZapete + 1, pozTacke - pozZapete - 1);
    cout << b << "," << a << "." << endl;
    return 0;
}
```

Глава 3

Гранање

Задатак: Множење једноцифрених бројева

Аутор: Јелена Хаци-Пурић

Такмичење: 2019/20 општинско, V разред, 2. задатак

Мали Ика учи множење једноцифрених бројева. Али, Ика зна само бројеве прве десетице. Када измножи два броја, он тачно запише резултат само ако је производ у првој десетици. Иначе, напише 10.

Напиши програм који за два унета једноцифрена броја исписује производ који ће записати мали Ика. У случају да тачан резултат не припада првој десетици, у новом реду исписати и за колико је тачан резултат већи од броја 10.

Опис улаза

Са стандардног улаза се учитавају једноцифрени бројеви m и n (сваки у посебном реду, $1 \leq m, n \leq 9$).

Опис излаза

У првом реду један природан број, резултат који добиоја мали Ика. Ако то није тачан одговор, у другом реду још један природан број, а то је број за који је тачан резултат већи од броја 10.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`2`	`10`	`3`	`10`	`3`	`3`
`5`		`10`	`20`	`1`	

Решење

Помножимо два унета броја. Ако је резултат множења у првој десетици, онда је потребно исписати тај резултат. Ако је резултат множења већи од 10 онда је потребно (у засебним редовима) исписати два броја на стандардни излаз: број 10 у првом реду, разлику производа и броја 10 у другом реду.

```cpp
#include <iostream>

using namespace std;

int main() {
  int m, n;
  cin >> m >> n;
  int proizvod = m * n;

  if (proizvod > 10)
    cout << "10" << endl
         << proizvod - 10 << endl;
  else
    cout << proizvod << endl;

  return 0;
}
```

Задатак: До 100, па опет

Аутор: Јелена Хаџи-Пурић

Такмичење: 2019/20 општинско, VI разред, 1. задатак

Мали Ика је научио бројеве прве стотине. Сваког дана Ика броји од 1 до 100 и то по неколико пута узастопно. Напишите програм који ће на основу претпоследњег и последњег броја који је Ика изговорио исписати следећи број који ће Ика изговорити током бројања.

Опис улаза

Са стандардног улаза се учитавају два броја m и n, сваки у посебном реду, $1 \leq m, n \leq 100$.

Опис излаза

Један цео број, следећи број који ће Ика изговорити.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
3 4	5	98 99	100	99 100	1

Решење

Након уноса оба броја, довољно је увећати други број за 1 и исписати на стандардни излаз, осим када је тај други број једнак 100. Зато најпре проверимо да ли је други број једнак 100. У том случају, потребно је исписати 1, зато што ново одбројавање бројева прве стотине креће од броја 1 (јер је тако описан поступак Икиног бројања).

```cpp
#include <iostream>

using namespace std;

int main() {
```

```
  int m, n;
  cin >> m >> n;
  if (n == 100)
    cout << "1" << endl;
  else
    cout << n+1 << endl;
  return 0;
}
```

Задатак: Непливачи

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 1, V и VI разред, 3. задатак

Деда Раде жели да упише својих четворо унука непливача у школу пливања. Инструктор му је рекао да је остао само један слободан термин, за који могу да се пријаве само деца виша од 110 сантиметара. Деда Раде не жели да раздваја унуке, па ће их уписати у школу пливања само ако сви испуњавају услов. Написати програм који одређује да ли деда Раде већ сада може да упише свих четворо унука у школу пливања.

Опис улаза

Са стандардног улаза се уносе четири цела позитивна броја не већа од 180, сваки у посебном реду – висине деда Радетових унука.

Опис излаза

На стандардни излаз исписати реч SVI ако је свако од четворо деце више од 110cm, а реч NIKO ако бар једно дете није више од 110 cm.

Пример 1

Улаз	*Излаз*
131	SVI
111	
128	
117	

Пример 2

Улаз	*Излаз*
131	NIKO
110	
128	
117	

Решење

Након учитавања потребно је само проверити да ли су сва 4 учитана броја већа од 110. Најједноставнији начин је употреба сложеног логичког услова.

```
#include <iostream>
using namespace std;

int main()
{
    int a, b, c, d;
    cin >> a >> b >> c >> d;
    if (a > 110 && b > 110 && c > 110 && d > 110)
        cout << "SVI" << endl;
    else
```

```
        cout << "NIKO" << endl;
    return 0;
}
```

Задатак: Сијалица

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 1, V и VI разред, 2. задатак

Стубови уличне расвете су нумерисани редом по улицама. У свакој улици има по N стубова, тако да су стубови у првој улици нумерисани 1, 2, … N, у другој су бројеви стубова $N + 1$, $N + 2$, … $2N$ итд. Мирко је добио задатак да у свакој другој улици замени сијалицу на сваком трећем стубу. Када је дошао до стуба са бројем A, Мирко се забројао. Написати програм који учитава целе позитивне бројеве N и A и одговара на питање да ли на том стубу треба заменити сијалицу.

Опис улаза

Са стандардног улаза се у првом реду уноси број N $(1 \leq N \leq 1000)$, а у другом реду број A $(1 \leq A \leq 1000)$.

Опис излаза

На стандардни излаз исписати само реч da ili ne.

Пример 1

Улаз	*Излаз*
20	da
75	

Пример 2

Улаз	*Излаз*
100	ne
300	

Решење

Када би се стубови и улице бројали од 0, тада би редни број улице могао да се добије као A div N, а редни број стуба у улици као A mod N (где су са div и mod означени целобројни количник, тј. остатак при дељењу два цела броја). Међутим, пошто се у задатку броји од 1, учитану ознаку на стубу треба смањити за 1, а добијене резултате за редни број улице и стуба у улици повећати за 1. То значи да редни број улице треба рачунати као $(A - 1)$ div $N + 1$, а редни број стуба у улици као $(A - 1)$ mod $N + 1$.

Након овог израчунавања потребно је још само проверити да ли је редни борј улице дељив са 2 и редни број стуба у улици дељив са 3.

```
#include <iostream>
using namespace std;

int main()
{
    int n, a;
    cin >> n >> a;
    int ulica = (a-1) / n + 1;
    int brUUlici = (a-1) % n + 1;
    if ((ulica % 2 == 0) and (brUUlici % 3 == 0))
```

```
        cout << "da" << endl;
    else
        cout << "ne" << endl;
    return 0;
}
```

Задатак: Одељења

Такмичење: општинско 2018/2019, VI разред, 2. задатак

Аутор: Филип Марић

Сви ученици у школи иду на драмску секцију или тренирају кошарку. Одељење је супер ако у њему бар 10 ученика иде на драмску секцију, бар 10 ученика тренира кошарку и бар 5 ученика иде на обе активности. Напиши програм који на основу укупног броја ученика у одељењу, броја ученика који иду на драмску секцију и броја ученика који тренирају кошарку одређује број ученика који иду на обе активности, а затим одређује и да ли је одељење једно супер одељење.

Опис улаза

Са стандардног улаза се учитава број ученика у одељењу, број ученика који иде на драмску секцију и број ученика који тренира кошарку. У питању су три природна броја (или евентуално 0), мања или једнака 30, наведена у посебним редовима. Подаци су исправни (број ученика у одељењу је већи или једнак од броја ученика који иду на секцију тј. спорт).

Опис излаза

На стандардни излаз исписати број ученика који иду на обе активности, а ако је одељење супер, у наредном реду исписати текст `super`.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
27	4	25	7	30	6
14		15	super	8	
17		17		28	

Решење

Унија скупа D оних ђака који иду на секцију и скупа K оних ђака који тренирају кошарку једнака је скупу O свих ученика у одељењу. Стога је број елемената пресека та два скупа тј. број ђака који и тренирају и иду на секцију једнак $|D|+|K|-|O|$. Задатак се даље директно решава израчунавањем овог броја и провером да ли су услови за супер одељење задовољени.

```
#include <iostream>

using namespace std;

int main() {
  int broj_ucenika, dramska, kosarka;
  cin >> broj_ucenika >> dramska >> kosarka;
  int obe = (dramska + kosarka) - broj_ucenika;
  cout << obe << endl;
  if (dramska >= 10 && kosarka >= 10 && obe >= 5)
```

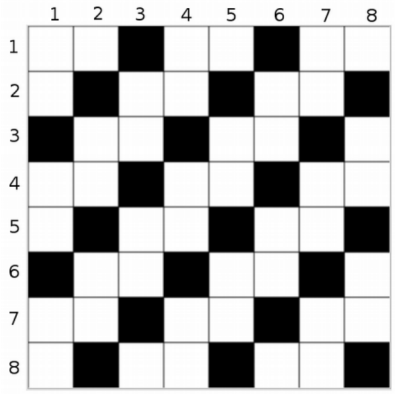
```
    cout << "super" << endl;
  return 0;
}
```

Задатак: Дијагонале на шаховској табли

Такмичење: окружно 2018/2019, V разред, 2. задатак, VI разред, 1. задатак

Аутор: Филип Марић

На једној необичној шаховској табли црна поља су распоређена по дијагоналама, као што је приказано на слици (на слици је приказано првих 8 врста и 8 колона, иако је табла већа). Напиши програм који одређује боју поља на основу унетог редног броја врсте и колоне.



Опис улаза

Са стандардног улаза се учитавају редни бројеви врсте и колоне три поља v ($1 \leq v \leq 100$) и k ($1 \leq k \leq 100$), раздвојени једним размаком, за свако поље у посебном реду.

Опис излаза

На стандардни излаз исписати боју поља (`crno` или `belo`), за свако поље у посебном реду

Пример

Улаз	*Излаз*
`1 5`	`belo`
`2 4`	`belo`
`3 4`	`crno`

Решење

Координате поља на првој дијагонали су (1, 3), (2, 2) и (3, 1) и лако се примети да им је збир увек 4 (када се прва координата повећа за 1, друга се смањи за 1, па збир све време остаје константан). Координате поља на другој дијагонали су (1, 6), (2, 5), (3, 4), (4, 3), (5, 2) и (6, 1) и збир им је увек 7. Слично важи и за наредне дијагонале. Дакле, може се приметити да је збир врсте и колоне на црним пољима 4, 7, 10, 13 итд., што су тачно бројеви који при дељењу са 3 дају остатак 1.

```cpp
#include <iostream>

using namespace std;

int main() {
  for (int i = 0; i < 3; i++) {
    int v, k;
    cin >> v >> k;
    if ((v + k) % 3 == 1)
      cout << "crno" << endl;
    else
      cout << "belo" << endl;
  }
}
```

Задатак: Исте цифре

Аутор: Филип Марић

Такмичење: државно 2021/22., V и VI разред, 1. задатак

Са стандардног улаза се уносе два двоцифрена броја чије су цифре различите. Напиши програм који одређује број парова заједничких цифара првог и другог броја.

Опис улаза

Опис улазних података.

Опис излаза

Опис излазних података.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
12 34	0	12 21	2	35 54	1

Решење

Ако са улаза учитамо двоцифрен број, тада његову цифру јединица можемо одредити као остатак при дељењу са 10, а цифру десетица као целобројни количник са 10. Бројач парова једнаких цифара се може иницијализовати на нулу и затим увећати за један ако је цифра јединица броја a једнака некој цифри броја b и зазим увећати за један ако је цифра десетица броја a једнака некој цифри броја b.

```cpp
#include <iostream>

using namespace std;

int main() {
  int a, b;
  cin >> a >> b;
  int a0 = a % 10, a1 = a / 10;
```

```
  int b0 = b % 10, b1 = b / 10;
  int broj = 0;
  if (a0 == b0 || a0 == b1)
    broj++;
  if (a1 == b0 || a1 == b1)
    broj++;
  cout << broj << endl;
  return 0;
}
```

Бројеви се могу учитати у облику ниски карактера што олакшава приступ цифрама (њима се тада приступа помоћу индекса 0 или 1).

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  string a, b;
  cin >> a >> b;
  int broj = 0;
  if (a[0] == b[0] || a[0] == b[1])
    broj++;
  if (a[1] == b[0] || a[1] == b[1])
    broj++;
  cout << broj << endl;
  return 0;
}
```

Уместо да постепено увећавамо бројач можемо посебно анализирати случајеве када постоји два пара једнаких цифара, када постоји један пар једнаких цифара и када нема парова једнаких цифара. Два пара постоје ако и само ако је $a_0 = b_0$ и $a_1 = b_1$ или је $a_0 = b_1$ и $a_1 = b_0$. Један пар постоји ако и само ако не важи претходно и важи да је $a_0 = b_0$ или $a_1 = b_1$ или $a_0 = b_1$ или $a_1 = b_0$. Ако ни то не важи, нема парова једнаких цифара.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  string a, b;
  cin >> a >> b;
  if ((a[0] == b[0] && a[1] == b[1]) ||
      (a[0] == b[1] && a[1] == b[0]))
    cout << 2 << endl;
  else if (a[0] == b[0] || a[0] == b[1] ||
           a[1] == b[0] || a[1] == b[1])
    cout << 1 << endl;
  else
```

```
    cout << 0 << endl;
  return 0;
}
```

Задатак: Недостајућа страница четвороугла

Аутор: Милан Вутделија

Такмичење: 2023/2024. квалификације 1, VIII разред, 1. задатак

Темена четвороугла су означена великим словима енглеске абецеде. Ознаке страница се добијају читањем темена у круг, увек у истом смеру. На пример, ако су темена редом (идући по контури) X, Y, Z, W, онда су ознаке страница XY, YZ, ZW и WX. Дате су ознаке неке три странице у произвољном редоследу страница, а треба одредити ознаку четврте.

Опис улаза

У сваком од три реда стандардног улаза налазе се по два велика слова енглеске абецеде, која представљају ознаку по једне од страница четвороугла. Редослед страница је произвољан.

Опис излаза

На стандардни излаз исписати два велика слова енглеске абецеде, која представљају ознаку четврте странице.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
AB	DA	QK	LP	HD	DF
BC		PQ		FG	
CD		KL		GH	

Решење

Нека су учитане странице редом a, b, c. Означимо непознату страницу са d. Обилазећи странице четвороугла редом у круг почевши од непознате странице d, на три дате странице можемо да наиђемо у једном од шест могућих редоследа: $abc, acb, bac, bca, cab, cba$. Сваки од ових редоследа можемо да "препознамо" поредећи слова у страницама за које претпостављамо да су узастопне. Конкретно:

- ако је $a_1 = b_0, b_1 = c_0$, тада је редослед наиласка на странице при обиласку редом у круг abc, па је $d_0 = c_1, d_1 = a_0$
- ако је $a_1 = c_0, c_1 = b_0$, тада је редослед наиласка на странице при обиласку редом у круг acb, па је $d_0 = b_1, d_1 = a_0$
- ако је $b_1 = a_0, a_1 = c_0$, тада је редослед наиласка на странице при обиласку редом у круг bac, па је $d_0 = c_1, d_1 = b_0$
- ако је $b_1 = c_0, c_1 = a_0$, тада је редослед наиласка на странице при обиласку редом у круг bca, па је $d_0 = a_1, d_1 = b_0$
- ако је $c_1 = a_0, a_1 = b_0$, тада је редослед наиласка на странице при обиласку редом у круг cab, па је $d_0 = b_1, d_1 = a_0$
- ако је $c_1 = b_0, b_1 = a_0$, тада је редослед наиласка на странице при обиласку редом у круг cba, па је $d_0 = a_1, d_1 = c_0$

Од ових шест услова увек је тачно један испуњен. Провером свих шест усова утврђујемо који је испуњен, а када то откријемо, лако израчунавамо непознату страницу.

```
#include <iostream>

using namespace std;

int main() {
    string a, b, c;
    cin >> a >> b >> c;
    if (a[1]==b[0] && b[1]==c[0]) cout << c[1] << a[0] << endl;
    else if (a[1]==c[0] && c[1]==b[0]) cout << b[1] << a[0] << endl;
    else if (b[1]==a[0] && a[1]==c[0]) cout << c[1] << b[0] << endl;
    else if (b[1]==c[0] && c[1]==a[0]) cout << a[1] << b[0] << endl;
    else if (c[1]==a[0] && a[1]==b[0]) cout << b[1] << c[0] << endl;
    else if (c[1]==b[0] && b[1]==a[0]) cout << a[1] << c[0] << endl;

    return 0;
}
```

Задатак: Папир камен маказе

Аутор: Небојша Варница

Такмичење: 2021/22. квалификације 1, V разред, 3. задатак

У игри "папир камен маказе" два играча бирају један од три објекта: папир, камен или маказе. Ако оба играча одаберу исти објекат сматрамо да нема победника односно да је нерешено. Ако један играч одабере папир а други камен победник је онај који одабере папир (јер папир обмотава камен). Ако један играч одабере папир а други маказе победник је онај који одабере маказе (јер маказе секу папир). Ако један играч одабере маказе а други камен победник је онај који одабере камен (јер камен ломи маказе). Написати програм који за унете објекте исписује који играч је победник.

Опис улаза

У сваком од прва два реда стандардног улаза налази се по једно од (латиничних) слова `P`, `K`, `M`. Слова представљају редом избор првог и другог играча: `P` - папир, `K` - камен, `M` - маказе.

Опис излаза

На стандардни излаз се исписује 0 (за нерешено), 1 (ако је победник први играч) или 2 (ако је победник други играч).

Пример 1

Улаз	*Излаз*
`P`	`2`
`M`	

Пример 2

Улаз	*Излаз*
`K`	`0`
`K`	

Решење

Треба проверити да ли су изабрани објекти исти. Ако нису онда их треба упоредити према условима датим у поставци.

```cpp
#include <iostream>
#include <string>

using namespace std;

int main() {
    string a, b, s;
    cin >> a >> b;
    s = a + b;
    if (a == b)
      cout << 0 << endl;
    else if (s == "PK" || s == "KM" || s == "MP")
      cout << 1 << endl;
    else
      cout << 2 << endl;
    return 0;
}
```

Задатак: Ветрење

Аутор: Милан Вугделија

Такмичење: Ревијално такмичење 2019/2020., V и VI разред, 2. задатак

Кућа има по један прозор на источној, јужној и западној страни. У кући је ваздух устајао ако није отворен ни један прозор, кућа се проветрава ако је отворен један или два прозора, а промаја је ако су отворена сва три прозора. Одредити шта се тренутно дешава у кући на основу тога који прозори су отворени.

Опис улаза

Улаз се састоји од 3 реда, а у сваком реду је реч `DA` или `NE`, према томе да ли је отворен прозор на једној страни куће. Дате речи се редом односе на прозоре на источној, јужној и западној страни.

Опис излаза

Један од текстова `promaja`, `vetrenje`, или `ustajao vazduh` (без наводника), који описује стање у кући.

Пример 1

Улаз	*Излаз*
`DA`	`vetrenje`
`NE`	
`DA`	

Пример 2

Улаз	*Излаз*
`NE`	`ustajao vazduh`
`NE`	
`NE`	

Пример 3

Улаз	*Излаз*
`DA`	`promaja`
`DA`	
`DA`	

Решење

Задатак се може решити гранањем, тј. конструкцијом `else-if`. Проверу да ли су сва три прозора отворена можемо извршити оператором “логичко и”, проверу да ли је бар један отворен можемо извршити оператором “логичко или”.

```cpp
#include <iostream>

using namespace std;

int main() {
    string s;
    cin >> s; bool istok = (s == "DA");
    cin >> s; bool jug = (s == "DA");
    cin >> s; bool zapad = (s == "DA");

    if (istok && jug && zapad)
        cout << "promaja" << endl;
    else if (istok || jug || zapad)
        cout << "vetrenje" << endl;
    else
        cout << "ustajao vazduh" << endl;
    return 0;
}
```

Задатак: Врста угла

Такмичење: општинско 2018/2019, V разред, 3. задатак

Аутор: Филип Марић

Напиши програм који одређује да ли је угао оштар, прав или туп.

Опис улаза

Са стандардног улаза се учитава величина угла у степенима (цео број између 0 и 179) и минутима (цео број између 0 и 59). Два броја су задата у посебним редовима.

Опис излаза

На стандардни излаз исписати текст `prav`, `tup` или `ostar`.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`130`	`tup`	`90`	`prav`	`89`	`ostar`
`38`		`0`		`14`	

Решење

Угао је прав ако има 90 степени и нула минута. Ако угао није прав, онда се број степени пореди са 90 и ако је већи од 90 онда је угао туп, а ако је мањи, онда је угао оштар.

```cpp
#include <iostream>

using namespace std;

int main() {
  int stepeni, minuti;
  cin >> stepeni >> minuti;
```

```
  if (stepeni == 90 && minuti == 0)
    cout << "prav" << endl;
  else if (stepeni >= 90)
    cout << "tup" << endl;
  else
    cout << "ostar" << endl;
  return 0;
}
```

Једно могуће решење је да се угао преведе у минуте (тако што се степени помноже са 60 и на то дода број минута), а затим упореде са правим углом израженим у минутама ($90 \cdot 60$).

```
#include <iostream>

using namespace std;

int uMinute(int stepeni, int minuti) {
  return stepeni * 60 + minuti;
}

int main() {
  int stepeni, minuti;
  cin >> stepeni >> minuti;
  if (uMinute(stepeni, minuti) < uMinute(90, 0))
    cout << "ostar" << endl;
  else if (uMinute(stepeni, minuti) > uMinute(90, 0))
    cout << "tup" << endl;
  else
    cout << "prav" << endl;
  return 0;
}
```

Задатак: Врста угла

Такмичење: општинско 2018/2019, VI разред, 1. задатак, VII разред, 1. задатак

Аутор: Филип Марић

Напиши програм који одређује да ли је угао оштар, прав или туп.

Опис улаза

Са стандардног улаза се учитава величина угла у степенима, минутима и секундама (угао је мањи од 180 степени, али је задат тако да број минута и број секунди може бити и већи од 59).

Опис излаза

На стандардни излаз исписати текст `prav`, `ostar` или `tup`.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`23`	`ostar`	`90`	`prav`	`89`	`tup`
`18`		`0`		`79`	
`370`		`0`		`3`	

Решење

Задатак се најједноставније решава ако се угао претвори само у секунде. У једном степену има $60 \cdot 60 = 3600$ секунди, у једном минуту 60 секунди па се претварање може извршити формулом $a \cdot 3600 + b \cdot 60 + c$, где је a број степени, b број минута, а c број секунди. Након што се угао претвори у секунде, врши се његово поређење са правим углом (такође изражемим у секундама као $90 \cdot 3600$).

```
#include <iostream>

using namespace std;

int uMinute(int stepeni, int minuti, int sekundi) {
  return (stepeni * 60 + minuti) * 60 + sekundi;
}

int main() {
  int stepeni, minuti, sekundi;
  cin >> stepeni >> minuti >> sekundi;
  int ugao = uMinute(stepeni, minuti, sekundi);
  int prav = uMinute(90, 0, 0);
  if (ugao < prav)
    cout << "ostar" << endl;
  else if (ugao > prav)
    cout << "tup" << endl;
  else
    cout << "prav" << endl;
  return 0;
}
```

Задатак: Врста троугла на основу страница

Такмичење: општинско 2018/2019, VIII разред, 1. задатак

Аутор: Филип Марић

Напиши програм који на основу дужина страница троугла проверава да ли такав троугао постоји и да ли је једнакостранични, једнакокраки или разностранични.

Опис улаза

Са стандардног улаза се уносе три позитивна цела броја a, b и c, сваки у посебном реду.

Опис излаза

На стандардни излаз исписати `ne postoji` ако не постоји троугао чије су дужине страница a,

b и c, `jednakostranicni` ако су ти бројеви дужине страница неког једнакостраничног троугла, `jednakokraki` ако су дужине страница неког једнакокраког троугла или `raznostranicni` ако су дужине страница неког разностраничног троугла.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
3	raznostranicni	3	jednakokraki	1	ne postoji
4		4		5	
5		3		1	

Решење

Провера да ли бројеви могу да представљају дужине страница неког троугла, се заснива на неједнакости троугла (збир дужина било које две странице мора бити већи од оне треће). Ако установимо да троугао не постоји, исписујемо одговарајућу поруку и завршавамо програм. Када установимо да троугао постоји (у грани у којој важи да троугао постоји), прелазимо на испитивање његовог типа. Најбоље је да прво испитамо да ли је у питању једнакостранични троугао. За то је довољно да проверимо једнакост два пара страница (ако је $a = b$ и $b = c$, на основу транзитивности једнакости ће следити и да је $a = c$). Ако установимо да троугао јесте једнакостраничан исписујемо одговарајућу поруку, а у другој грани овог гранања проверавамо да ли је једнакокраки тако што проверимо да ли је бар један пар страница једнак (као дисјункцију провере три једнакости). У зависности од резултата те провере исписујемо да је троугао једнакокраки или разностраничан.

Редослед ове две провере је битан. Ако би се прво проверавало да ли је троугао једнакокраки тако што се само провери да ли има један пар истих страница, могло би се десити да се једнакостранични троугао грешком прогласи једнакокраким. Слично томе, ако ове провере не би биле угнежђене, једнакостранични троуглови сигурно би били проглашени једнакокраким (јер сваки једнакостранични троугао задовољава услов једнакокраког).

```
#include <iostream>

using namespace std;

int main() {
  int a, b, c;
  cin >> a >> b >> c;
  if (a + b > c && a + c > b && b + c > a) {
    if (a == b && b == c /* && a == c */)
      cout << "jednakostranicni" << endl;
    else if (a == b || b == c || a == c)
      cout << "jednakokraki" << endl;
    else
      cout << "raznostranicni" << endl;
  } else
    cout << "ne postoji" << endl;

  return 0;
}
```

Још једна могућност одређивања врсте троугла је да се прво провери да ли је $a = b$, па ако јесте,

да се у зависности од услова $b = c$ одреди да ли је троугао једнакостранични или једнакокраки, а ако није да се у зависности од тога да ли важи $a = c$ или $b = c$ одреди да ли је троугао једнакокраки или је разностранични.

```
#include <iostream>

using namespace std;

int main() {
  int a, b, c;
  cin >> a >> b >> c;
  if (a + b > c && a + c > b && b + c > a) {
    if (a == b)
      if (b == c)
        cout << "jednakostranicni" << endl;
      else
        cout << "jednakokraki" << endl;
    else
      if (a == c || b == c)
        cout << "jednakokraki" << endl;
      else
        cout << "raznostranicni" << endl;
  } else {
    cout << "ne postoji" << endl;
  }
  return 0;
}
```

Задатак: Четвороугао

Аутор: Милан Вуīелија

Такмичење: 2021/22. квалификације 2, VII разред, 1. задатак

Написати програм који за дужине страница AB, BC, CD, DA четвороугла дате редом, утврђује да ли је тај четвороугао ромб, паралелограм (који није уједно и ромб), делтоид (који није уједно и ромб) или обичан четвороугао (који није ништа од претходног).

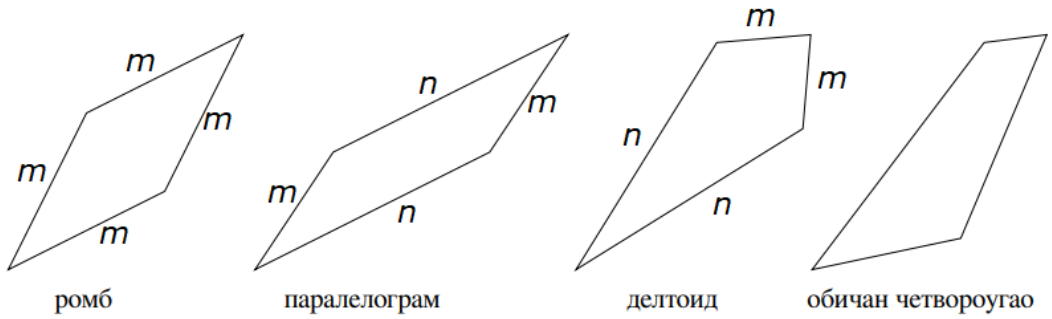
Ромб је четвороугао коме су све четири странице једнаке. Паралелограм је четвороугао коме су по две наспрамне странице једнаке. Делтоид је четвороугао коме су по две суседне странице једнаке. У овом здатку обичним називамо четвороугао који није ни ромб, ни паралелограм, ни делтоид.

Опис улаза

Четири цела позитивна броја, сваки у посебном реду. Бројеви нису већи од 100.

Опис излаза

Једна од речи `ROMB`, `PARALELOGRAM`, `DELTOID`, `OBICAN`



Пример 1

Улаз	*Излаз*
9	ROMB
9	
9	
9	

Пример 2

Улаз	*Излаз*
5	OBICAN
6	
6	
7	

Решење

Нека су a, b, c, d дужине страница четвороугла.

Услов да је тај четвороугао ромб је $a = b = c = d$.

Услов да је тај четвороугао паралелограм је $a = c \wedge b = d$.

Услов да је тај четвороугао делтоид је $(a = b \wedge c = d) \vee (a = d \wedge c = b)$.

Да не бисмо приликом провере да ли је четвороугао паралелограм (или делтоид), проверавали да при томе није ромб, можемо прво да проверимо да ли је ромб, јер се на тај начин остали услови поједностављују.

```
#include <iostream>

using namespace std;

int main() {
    int a1, a2, a3, a4;
    cin >> a1 >> a2 >> a3 >> a4;
    if (a1 == a2 && a2 == a3 && a3 == a4)
        cout << "ROMB" << endl;
    else if (a1 == a3 && a2 == a4)
        cout << "PARALELOGRAM" << endl;
    else if ((a1 == a2 && a3 == a4) || (a2 == a3 && a4 == a1))
        cout << "DELTOID" << endl;
    else
        cout << "OBICAN" << endl;
```

```
    return 0;
}
```

Задатак: Гол у гостима

аутор: Милан Вугделија

Такмичење: 2021/22. квалификације 3, VII разред, 1. задатак

Екипа A игра против екипе B, најпре код куће а затим у гостима. Прва утакмица је завршена резултатом $a_1 : b_1$, а друга резултатом $b_2 : a_2$ (у свакој утакмици прво се наводи број голова које је постигао домаћин, а затим гост). У следеће коло пролази екипа која је постигла укупно више голова, а у случају једнаког укупног броја голова, она екипа која је постигла више голова као гост. Ако су и по томе екипе изједначене, играју се продужеци. Написати програм који учитава бројеве a_1, b_1, b_2, a_2 и исписује ознаку екипе која наставља такмичење, или исписује да су потребни продужеци.

Опис улаза

На стандардном улазу су редом неозначени цели бројеви a_1, b_1, b_2, a_2, сваки у посебном реду. Бројеви на улазу нису већи од 100.

Опис излаза

На стандардни излаз исписати само једно слово, и то `a`, `p` или `b`.

Пример 1

Улаз	*Излаз*
1	a
0	
2	
1	

Пример 2

Улаз	*Излаз*
3	p
2	
3	
2	

Пример 3

Улаз	*Излаз*
4	b
2	
3	
0	

Решење

Најједноставније решење се добија надовезаним наредбама гранања.

Услове можемо да проверавамо истим редом као што су наведени у тексту задатка:

- да ли је екипа A постигла укупно више говова? (ако јесте, победник је A)
- ако није, да ли је екипа B постигла укупно више говова? (ако јесте, победник је B)
- ако није ни то, да ли је екипа A постигла више говова у гостима? (ако јесте, победник је A)
- ако није ни то, да ли је екипа B постигла више говова у гостима? (ако јесте, победник је B)
- ако ни један од претходних услова није испуњен, потребни су продужеци.

```
#include <iostream>

using namespace std;

int main() {
    int a1, b1, b2, a2;
    cin >> a1 >> b1 >> b2 >> a2;
```

```
    if (a1 + a2 > b1 + b2) cout << "a" << endl;
    else if (a1 + a2 < b1 + b2) cout << "b" << endl;
    else if (a2 > b1) cout << "a" << endl;
    else if (a2 < b1) cout << "b" << endl;
    else cout << "p" << endl;
    return 0;
}
```

Задатак: Класификација

Аутор: Филип Марић

Такмичење: окружно 2021/22., V разред, 1. задатак

Огњен на основу одговора на три питања треба да погоди замишљено живо биће. Питања су следећа:

1. Да ли је у питању животиња?
2. Да ли живи у води?
3. Да ли је велика?

Могући одговори су: пуноглавац (мала животиња која живи у води), кит (велика животиња која живи у води), мрав (мала животиња која живи на копну), слон (велика животиња која живи на копну), алга (мала "биљка" која живи у води), локвањ (велика биљка која живи у води), висибаба (мала биљка која живи на копну), храст (велика биљка која живи на копну).

Напиши програм који погађа непознато живо биће.

Опис улаза

Са стандардног улаза се учитавају одговори на три постављена питања, редом (`da` или `ne`), сваки одговор у посебном реду.

Опис излаза

На стандардни излаз исписати `punoglavac`, `kit`, `mrav`, `slon`, `alga`, `lokvanj`, `visibaba` или `hrast`.

Пример

Улаз	*Излаз*
`da`	`slon`
`ne`	
`da`	

Решење

Могуће је да користимо угнежђене наредбе `if-else`.

```
#include <iostream>

using namespace std;

int main() {
  string zivotinja, voda, velika;
  cin >> zivotinja >> voda >> velika;
```

```
  if (zivotinja == "da") {
    if (voda == "da") {
      if (velika == "da")
        cout << "kit" << endl;
      else
        cout << "punoglavac" << endl;
    } else {
      if (velika == "da")
        cout << "slon" << endl;
      else
        cout << "mrav" << endl;
    }
  } else {
    if (voda == "da") {
      if (velika == "da")
        cout << "lokvanj" << endl;
      else
        cout << "alga" << endl;
    } else {
      if (velika == "da")
        cout << "hrast" << endl;
      else
        cout << "visibaba" << endl;
    }
  }
  return 0;
}
```

Могуће је да се свако живо биће посебно испитају услови.

```
#include <iostream>
#include <string>

using namespace std;

int main(){
  string zivotinja, vodena, velika;
  cin >> zivotinja >> vodena >> velika;

  if (zivotinja == "da" && vodena == "da" && velika == "da")
    cout << "kit" << endl;
  if (zivotinja == "da" && vodena == "da" && velika == "ne")
    cout << "punoglavac" << endl;
  if (zivotinja == "da" && vodena == "ne" && velika == "da")
    cout << "slon" << endl;
  if (zivotinja == "da" && vodena == "ne" && velika == "ne")
    cout << "mrav" << endl;
  if (zivotinja == "ne" && vodena == "da" && velika == "da")
    cout << "lokvanj" << endl;
  if (zivotinja == "ne" && vodena == "da" && velika == "ne")
```

```
    cout << "alga" << endl;
  if (zivotinja == "ne" && vodena == "ne" && velika == "da")
    cout << "hrast" << endl;
  if (zivotinja == "ne" && vodena == "ne" && velika == "ne")
    cout << "visibaba" << endl;
}
```

Могуће је да се сви одговори убаце у низ, а да се затим индекс у низу израчуна на основу датих одговора, коришћењем бинарне репрезентације броја као $o_1 \cdot 2^2 + o_2 \cdot 2 + o_3$, где је o_1 једнако 1 ако је у питању животиња, а једнако 0 ако није, o_2 једнако 1 ако живи у води, а 0 ако не живи, а o_3 једнако 1 ако је велико, а 0 ако није.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  vector<string> odgovori
    {"kit", "punoglavac", "slon", "mrav", "lokvanj", "alga", "hrast", "visibaba"};
  string zivotinja, voda, velika;
  cin >> zivotinja >> voda >> velika;
  int k = (zivotinja != "da") * 4 + (voda != "da") * 2 + (velika != "da");
  cout << odgovori[k] << endl;
  return 0;
}
```

Могуће је направити мапу тј. речник која слика ниске које садрже надовезане одговоре у одговарајуће називе живих бића.

```
#include <iostream>

using namespace std;

int main() {
  string zivotinja, voda, velika;
  map<string, string> odgovor {{"dadada", "kit"},
                               {"dadane", "punoglavac"},
                               {"daneda", "slon"},
                               {"danene", "mrav"},
                               {"nedada", "lokvanj"},
                               {"nedane", "alga"},
                               {"neneda", "hrast"},
                               {"nenene", "visibaba"}};

  cout << odgovor[zivotinja + voda + velika] << endl;
  return 0;
}
```

Задатак: Сусрет

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 1, V и VI разред, 4. задатак, VII и VIII разред 2. задатак

Новак и Јагош су стари другари и они се често без договарања сачекују на свом омиљеном месту у неко уобичајено време, па ако први дочека другог онда проведу неко време дружећи се. Када Новак стигне први, он чека Јагоша 10 минута, а Јагош (ако он стигне први) чека Новака 15 минута.

Написати програм који за дата времена данашњег појављивања Новака и Јагоша одговара на питање да ли су се састали.

Опис улаза

Учитавају се четири цела броја, NS, NM, JS, JM редом, сваки у посебном реду стандардног улаза. Бројеви NS, NM представљају сат и минут Новаковог доласка, а JS, JM представљају сат и минут Јагошевог доласка. Подаци ће представљати исправно записана времена (то јест, важиће $0 \leq NS \leq 23$, $0 \leq NM \leq 59$, $0 \leq JS \leq 23$, $0 \leq JM \leq 59$).

Опис излаза

На стандардни излаз исписати само реч da ili ne.

Пример 1

Улаз	*Излаз*	*Објашњење*
10 15 10 25	da	Јагош је дошао 10 минута после Новака, што је довољно за сусрет.

Пример 2

Улаз	*Излаз*	*Објашњење*
21 15 21 26	ne	Јагош је дошао 11 минута после Новака, па до сусрета није дошло.

Пример 3

Улаз	*Излаз*	*Објашњење*
0 15 0 0	da	Јагош је дошао у поноћ, а Новак 15 минута касније, па су се срели.

Пример 4

Улаз	*Излаз*	*Објашњење*
0 0 23 59	ne	Разлика у временима доласка скоро читав дан и зато се нису срели.

Решење

Задатак се једноставније решава ако се после учитавања прво израчуна број минута од почетка дана до Николиног и Јагошевог доласка: $N = N_{sat} \cdot 60 + N_{min}$, и $J = J_{sat} \cdot 60 + J_{min}$.

Угнежђена гранања

Решавање задатка сада можемо да довршимо тако што најпре проверимо ко је први стигао на место састанка. Ако је то Новак, онда проверавамо да је Јагош дошао највише 10 минута након Новака, а ако је Јагош први стигао, онда проверавамо да ли је Новак дошао највише 15 минута након Јагоша.

```
#include <iostream>
using namespace std;

int main() {
  int nsat, nmin, jsat, jmin;
  cin >> nsat >> nmin >> jsat >> jmin;
  int n = nsat*60 + nmin;
  int j = jsat*60 + jmin;

  if (n < j) // ako je Novak dosao pre
    if (j <= n+10)
      cout << "da" << endl;
    else
      cout << "ne" << endl;
  else // ako je Jagos dosao pre (ili su dosli istovremeno)
    if (n <= j+15)
      cout << "da" << endl;
    else
      cout << "ne" << endl;
  return 0;
}
```

Сложени услов

Можемо и да формирамо сложени услов, који директно одговара на питање да ли је дошло до сусрета. Да би одговор био потврдан, треба да важи $N \leq J \leq N + 10$ или $J \leq N \leq j + 15$, у противном одговор је одречан.

```
#include <iostream>
using namespace std;

int main() {
    int nsat, nmin, jsat, jmin;
    cin >> nsat >> nmin >> jsat >> jmin;
    int n = nsat*60 + nmin;
    int j = jsat*60 + jmin;

    if ((n <= j && j <= n+10) || (j <= n && n <= j+15))
      cout << "da" << endl;
```

```
    else
      cout << "ne" << endl;

    return 0;
}
```

Задатак: Какуро провера

Аутор: Филип Марић

Такмичење: 2022/2023. квалификације 1, 6. разред, 3. задатак и 7. разред 2. задатак

У игри Какуро потребно је уписати цифре од 1 до 9 у празна поља табеле у складу са унапред задатим збировима врста и колона, тако да се сваки збир добија сабирањем различитих цифара. Ми ћемо размотрити једноставну варијанту ове игре у којој се бројеви уписују у 4 поља, распоређених у две хоризонталне (водоравне) врсте и две вертикалне (усправне) колоне, тако да су унапред дати збирови сваке врсте и сваке колоне. Напиши програм који проверава да ли су цифре уписане у складу са правилима игре Какуро.

Опис улаза

Прва линија стандардног улаза садржи два позитивна природна броја између 3 и 17 који представљају збирове у свакој од две колоне. Наредна линија садржи два позитивна природна броја између 3 и 17 који представљају збирове у свакој врсти. Након тога се уносе 4 броја (задата у две линије) који су уписани у поља.

Опис излаза

На стандардни излаз исписати `da` ако је табела попуњена исправно тј. `ne` ако није.

Пример 1

Улаз	*Излаз*

`14 8`
`16 6`
`9 7`
`5 1`
	`da`

Пример 2

Улаз	*Излаз*	*Објашњење*

`11 9`
`17 3`
`9 8`
`3 1`
	`ne`	Збир елемената прве колоне није 11 већ 12, а збир елемената друге врсте није 3 већ 4.

Пример 3

Улаз	*Излаз*	*Објашњење*

`11 10`
`17 4`
`9 8`
`2 2`
	`ne`	Иако се сви збирови поклапају, у другој врсти је два пута употребљена цифра 2.

Пример 4

Улаз	*Излаз*	*Објашњење*

`16 6`
`14 8`
`10 4`
`6 2`
	`ne`	Иако се сви збирови поклапају, употребљена је вредност 10 која није цифра.

Решење

Да бисмо избегли писање једног огромног логичког израза у коме бисмо проверили целокупну исправност решења какуро загонетке, можемо употребити логичку променљиву `OK` која ће имати вредност `true` ако и само ако је какуро исправно решен. Проверавамо редом сваки од три задата услова (да ли су сви уписани бројеви једноцифрени, да ли су збирови врста тј. колона исправни и да ли се неки бројеви у врстама или колонама понављају) и ако је било који од њих нарушен, закључујемо да какуро није исправно решен (и променљивој `OK` додељујемо вредност `false`).

```
#include <iostream>

using namespace std;

int main() {
  int k1, k2;
  cin >> k1 >> k2;
  int v1, v2;
  cin >> v1 >> v2;

  int a11, a12;
  cin >> a11 >> a12;
  int a21, a22;
  cin >> a21 >> a22;

  bool OK = true;
  if (a11 < 1 || a11 > 9 || a12 < 1 || a12 > 9 ||
      a21 < 1 || a21 > 9 || a22 < 1 || a22 > 9)
    OK = false;

  if (a11 + a12 != v1 || a11 + a21 != k1 ||
      a21 + a22 != v2 || a12 + a22 != k2)
    OK = false;

  if (a11 == a12 || a11 == a21 || a12 == a22 || a21 == a22)
    OK = false;

  if (OK)
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

Задатак: Мед

Аутор: Милан Вугделија

Такмичење: 2019/20 општинско, VI разред, 2. задатак

Пчелар Дуле је најпознатији произвођач меда. Да би мед био укусан, Дуле нам је открио тајну. Мед се чува у три посебне тегле. У прву теглу може се улити највише A килограма меда, у другу B килограма, а у трећу C килограма. Али, мед се обавезно улива у тегле редом почевши од прве тегле, не прелазећи на следећу теглу док се претходна не напуни. Ако Дуле има N килограма меда које улива у тегле, одреди колико је у којој тегли било меда након што је уливено свих N килограма.

Опис улаза

У првом реду стандардног улаза налази се природан број A, $1 \leq A \leq 10$, количина меда која стане у прву теглу. У другом реду налази се природан број B, $1 \leq B \leq 10$, количина меда која стане у другу теглу. У трећем реду налази се природан број C, $1 \leq C \leq 10$, количина меда која стане у трећу теглу. У четвртом реду налази се природан број N, $1 \leq N \leq A + B + C$, количина меда за уливање.

Опис излаза

На стандардном излазу написати три природна броја један испод другог. Први број је количина меда у првој тегли, други број је количина у другој, трећи број је количина меда у трећој тегли.

Пример 1		**Пример 2**		**Пример 3**		**Пример 4**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
1	1	3	3	3	3	2	2
1	0	7	7	7	7	8	4
1	0	8	0	5	1	6	0
1		10		11		6	

Решење

Замислимо да су испред нас три празне тегле ($teglaA = teglaB = teglaC = 0$). Знамо да у прву теглу можемо улити највише A килограма меда. Ако је почетна маса меда N таква да се може напунити до врха прва тегла, онда ћемо у прву теглу улити A килограма меда. Преостала маса меда за уливање постаје $N = N - A$. Горе описани поступак применимо над другом теглом, а потом над трећом теглом. Због услова $1 \leq N \leq A + B + C$ знамо да неће остати мед који није уливен у тегле.

```
#include <iostream>

using namespace std;

int main()  {
  int a, b, c, n;
  cin >> a >> b >> c >> n;
  int teglaA = 0, teglaB = 0, teglaC = 0;
  if (n >= a)
    teglaA = a;
  else
    teglaA = n;
```

```
  n = n - teglaA;

  if (n >= b)
    teglaB = b;
  else
    teglaB = n;
  n = n - teglaB;

  teglaC = n;

  cout << teglaA << endl << teglaB << endl << teglaC << endl;
  return 0;
}
```

Задатак: Ближе црти

Аутор: Милан Вуīелија

Такмичење: 2021/22. квалификације 1, V разред, 4. задатак

Пера и Боба играју стару игру крајцарице. Игра се тако што сваки играч баца новчић покушавајући да га набаци на претходно нацртану линију. Ко баци новчић преко црте, изгубио је. Ако ни Боба ни Пера не пребаце црту, победник је онај ко баци ближе црти.

Пошто је Боба старији, Пера има малу предност. Наиме, ако обојица баце преко црте, победник је Пера. Такође, ако не пребаце црту и баце једнако далеко од црте, и у том случају је победник Пера.

У данашњој игри, црта је удаљена 100 јединица од Бобе и Пере, па је циљ да се новчић баци на даљину 100, или што ближу мању даљину. Написати програм који за дате даљине B и P на које су редом бацили новчиће Боба и Пера, исписује име победника.

Опис улаза

У првом реду стандардног улаза природан број B, а у другом природан број P. Ниједан од ових бројева није већи од 150.

Опис излаза

На стандардни излаз исписати само реч `Boba` или реч `Pera`.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`96`	`Boba`	`100`	`Pera`	`91`	`Boba`
`85`		`100`		`102`	

Решење

Најједноставније решење се добија надовезаним наредбама гранања.

Једноставан случај у коме одмах знамо одговор је када је Боба бацио преко црте, а тада је победник Пера. У противном (ако Боба није бацио преко црте), следећи једноставан случај је када је Пера бацио преко црте, тада је Боба победио. Преостали су случајеви када ни један

играч није пребацио црту. Под тим условом, Пера је победио ако је бацио исто или даље него Боба, а Боба је победио ако је бацио даље него Пера.

```
#include <iostream>

using namespace std;

int main() {
    int boba, pera;
    cin >> boba >> pera;
    if (boba > 100)
      cout << "Pera" << endl;
    else if (pera > 100)
      cout << "Boba" << endl;
    else if (boba <= pera)
      cout << "Pera" << endl;
    else
      cout << "Boba" << endl;
    return 0;
}
```

Задатак: Куповина

Аутор: Небојша Варница

Такмичење: 2021/22. квалификације 1, VII и VIII разред 1. задатак

У радњи се продају кесице бомбона по цени од A дин за кесицу и кесице пеpеца по цени B дин за кесицу. Треба да купимо C кесица и потрошимо D динара. Написати програм који одређује колико кесица бомбона а колико пеpеца треба да купимо. Могу се куповати само целе кесице.

Опис улаза

Са стандардног улаза уносе се цели бројеви A, B, C, D - сваки у засебном реду. $1 \leq A, B, C \leq 1000, 1 \leq D \leq 10^6, A \neq B$.

Опис излаза

На стандардни излаз исписати два цела броја, сваки у посебном реду. Први број је број кесица бомбона, а други је број кесица пеpеца које треба да купимо. Ако задатак нема (целобројна) решења, приказати две нуле.

Пример 1

Улаз	*Излаз*
30	6
40	3
9	
300	

Пример 2

Улаз	*Излаз*
55	0
75	0
10	
1000	

Решење

Решење се добија решавањем система једначина: $x + y = C$, $Ax + By = D$, а то је $y = \frac{AC-D}{A-B}$ $x = C - y$ Према условима задатка решења морају бити цели позитивни бројеви.

```cpp
#include <iostream>

using namespace std;

int main() {
  int a,b,c,d;
  cin >> a >> b >> c >> d;
  if ((a*c-d)%(a-b) != 0)
    cout << 0 << endl << 0 << endl;
  else {
    int y = (a*c-d)/(a-b);
    int x = c-y;
    if (x<0 || y<0 || x>c || y>c)
      cout << 0 << endl << 0 << endl;
    else
      cout << x << endl << y << endl;
  }
  return 0;
}
```

Задатак: Распон крила

Аутор: Душан Пођадић

Такмичење: 2022/2023. квалификације 2, 7. разред, 3. задатак и 8. разред, 2. задатак

У друштвеној игри “Распон крила” (на енглеском “Wingspan”) играчима су на располагању 3 врсте потеза:

Постави карту птице на таблу

Играч може да из своје руке постави карту птице на своју таблу. Да би то урадио он мора да плати цену одигравања птице, тј. да из својих залиха у “банку” врати одређен број ресурса хране. Уколико нема довољно ресурса у својим залихама, играч не може да постави птицу док не набави још хране. Свака постављена птица играчу на крају партије доноси одређен број поена.

Положи јаја у гнезда

Играч може да положи k нових јаја на таблу, а свако положено јаје на крају игре доноси један поен.

Узми храну из хранилице

Играч може да из заједничке хранилице узме r нових ресурса хране и стави их у своје залихе.
Дуња игра ову игру и остала су јој још два потеза до краја партије. Она у руци има још две карте птица (звaћемо их прва и друга птица), а у својим залихама још n ресурса хране. Помозите Дуњи да победи у игри тако што ће у преостала два потеза да скупи што већи број поена.

Опис улаза

У првом реду стандардног улаза се уноси број k, који представља број јаја које Дуња може да положи у једном потезу. У другом реду се уноси број r, број ресурса које Дуња може да узме

из хранилице у једном потезу. У трећем реду се уноси број n, број ресурса хране које Дуња тренутно има у залихама. У четвртом реду се уносе два броја $c1$ и $p1$, цена у ресурсима коју Дуња мора да плати да би одиграла своју прву птицу и број поена које та птица доноси. У петом реду се уносе два цела броја $c2$ и $p2$, цена у ресурсима коју Дуња мора да плати да би одиграла своју другу птицу и број поена које та птица доноси.

Сви бројеви који се уносе су цели и између 1 и 500.

Опис излаза

У једином реду стандардног излаза исписати колико највише поена Дуња може да оствари у преостала два потеза.

Пример 1

Улаз	*Излаз*	*Објашњење*

`4`
`3`
`1`
`1 5`
`2 10`
	`10`	Пошто Дуњина друга птица доноси 10 поена што је више него да два пута положи јаја (што би било 8 поена) и више него да одигра прву птицу и једном положи јаја (што би било 9 поена), Дуњи се највише исплати да у свом првом потезу узме храну (онда ће имати 4 хране у руци) и да онда са 2 ресурса хране плати одигравање своје друге птице (две хране ће јој на крају остати у руци, али то није битно). Дуњи би се највише исплатило да одигра обе птице, али нажалост у руци на почетку има само једну храну што није довољно за одигравање обе птице, а када у првом потезу узме храну онда до краја има још само један потез, па не може да одигра обе птице (иако има довољно хране).

Пример 2

Улаз	*Излаз*	*Објашњење*

`3`
`1`
`6`
`2 1`
`3 2`
	`6`	Дуњи се највише исплати да два пута положи јаја, јер иако има довољно хране да одигра обе птице, свака од њених птица доноси мање поена од једне акције полагања јаја.

Решење

Прво је потребно уочити све могуће смислене потезе које Дуња може да направи:

- Дуња може да два пута положи јаја што јој доноси $2k$ поена.
- Дуња може да спусти две птице што јој доноси $p1 + p2$ поена. Наравно ово је могуће само уколико Дуња има довољно хране код себе, тј. ако важи $c1 + c2 \leq n$.
- Дуња може да једном узме храну из хранилице и да онда одигра прву (или другу) птицу. То јој доноси $p1$ ($p2$) поена. Услов да би ово могло да се деси је да Дуња након узимања ресурса из хранилице има довољно ресурса. Дуња је већ на почетку имала n ресурса и из хранилице узима r, дакле треба да важи $c1 \leq n + r$ ($c2 \leq n + r$).
- Дуња може да направи комбинацију одигравања птице и полагања јаја. Може да прво да положи јаја што јој доноси k поена, а онда да одигра једну од птица што јој доноси $p1$ или $p2$ поена у зависности од птице. Наравно услов је да Дуња има довољно хране тј. да важи $c1 \leq n$ или $c2 \leq n$, у зависности од тога коју птицу игра.

Остале комбинације потеза немају смисла. На пример, нема смисла да Дуња два пута узме храну јер јој то не доноси поене, а полагање јаја које је бесплатно јој сигурно доноси бар неки поен.

Такође, нема смисла да Дуња једном положи јаје и да онда узме храну јер јој та храна ништа не значи, а могла је уместо тога да два пута полаже јаја. Узимање хране се исплати само ако ће ту храну искористити да касније одигра птицу која доноси много поена.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  int k, r, n;
  int bodovi;

  cin >> k >> r >> n;
  int c1, c2, p1, p2;

  cin >> c1 >> p1 >> c2 >> p2;
  int m = 0;

  // dva puta jaja
  m = 2 * k;

  // dve ptice
  if (n >= c1 + c2) {
    bodovi = p1 + p2;
    if (bodovi > m) {
      m = bodovi;
    }
  }

  // hrana + ptica1
  if (n + r >= c1) {
    bodovi = p1;
    if (bodovi > m) {
      m = bodovi;
    }
  }

  // hrana + ptica2
  if (n + r >= c2) {
    bodovi = p2;
    if (bodovi > m) {
      m = bodovi;
    }
  }

  // jaja + ptica1
  if (n >= c1) {
    bodovi = k + p1;
```

```
    if (bodovi > m) {
      m = bodovi;
    }
  }

  // jaja + ptica2
  if (n >= c2) {
    bodovi = k + p2;
    if (bodovi > m) {
      m = bodovi;
    }
  }

  cout << m;
  return 0;
}
```

Задатак: AM/PM

Аутори: Филип Марић, Владимир Кузмановић

Такмичење: 2022/2023. квалификације 3, 6. разред, 1. задатак

Време на дигиталним сатовима може да се изражава било тако што се сати изразе вредностима од 0 до 23, било тако што се користе вредности од 1 до 12, уз напомену да ли је пре подне (`am`) или после подне (`pm`). Написати програм који на основу учитаног времена у првом формату исписује то време у другом формату.

Опис улаза

Прва линија стандардног улаза садржи сат (цео број од 0 до 23), а друга линија минут (цео број од 0 до 59).

Опис излаза

На стандардни излаз исписати одговарајуће време у `am` или `pm` формату.

Пример 1		**Пример 2**		**Пример 3**		**Пример 4**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`17` `23`	`5 23 pm`	`6` `17`	`6 17 am`	`0` `15`	`12 15 am`	`12` `5`	`12 5 pm`

Решење

У главном програму са стандардног улаза треба учитати редом вредности sat и $minut$. Треба приметити да се време на дигиталном часовнику који време у сатима изражава у формату 0 до 23 врло лако може поделити на преподневне (`am`) или поподневне (`pm`) термине у зависности од вредности промењиве sat. Такође, треба приметити да вредност $minut$ не игра никакву улогу у одређивању да ли се ради о преподневном или поподневном термину, па ћемо је непромењену преписати на стандардни излаз. На основу претходне дискусије, лако се закључује да у задатку треба само да прилагодимо вредност sat траженом формату.

Гранање са специјалним случајевима

Ако променљива sat има вредност у интервалу $[0, 11]$, тада се ради о преподневном термину. Овде треба бити обазрив и приметити да се поноћ у траженом формату изражава као `12 00 am`, па у случају да променљива sat има вредност 0, морамо да је променимо у 12. На крају, треба само да одштампамо резултат у облику `sat minut am`.

Уколико променљива sat има вредност у интервалу $[12, 23]$, тада се ради о поподневном термину. Прво треба да сведемо вредност променљиве на интервал $[0, 11]$ тако што ћемо одузети 12, тј. $sat = sat - 12$. У овом случају треба приметити да смо сада подне изразили са 0 сати, па у случају да променљива sat након одузимања има вредност 0, морамо да јој променимо вредност на 12. На крају, треба само да одштампамо резултат у облику `sat minut pm`.

```
#include <iostream>

using namespace std;

int main() {
  int sat;
  int min;
  cin >> sat >> min;
  if (sat < 12) {
    if (sat == 0)
      cout << 12 << " " << min << " " << "am" << endl;
    else
      cout << sat << " " << min << " " << "am" << endl;
  } else {
    if (sat == 12)
      cout << "12" << " " << min << " " << "pm" << endl;
    else
      cout << (sat - 12) << " " << min << " " << "pm" << endl;
  }
  return 0;
}
```

Модуларна аритметика

Задатак се лакше може решити ако се употреби модуларна аритметика. Одузимањем броја 1 од вредности sat, интервал сати $[0, 23]$, се своди на $[-1, 22]$ остаци ових бројева при дељењу са 12 су редом $[11, 0, 1, \ldots, 11, 0, \ldots, 10]$. Заиста, да не бисмо размишљали о дељењу негативних бројева, можемо прво увећати бројеве за 12, да бисмо добили интервал $[11, 24]$, чији су остаци редом заиста $[11, 0, 1, \ldots, 11, 0, \ldots, 10]$. Жељене вредности $[12, 1, 2, \ldots, 12, 1, \ldots 11]$ онда добијамо увећањем добијених остатака за 1.

```
#include <iostream>

using namespace std;

int main() {
  int sat;
  int min;
```

```
  cin >> sat >> min;
  cout << (sat - 1 + 12) % 12 + 1 << " " << min << " ";
  if (sat < 12)
    cout << "am" << endl;
  else
    cout << "pm" << endl;
  return 0;
}
```

Задатак: Тениски резултат гем

Аутори: Филип Марић, Владимир Кузмановић

Такмичење: 2022/2023. квалификације 3, 6. разред, 3. задатак и 7. разред 2. задатак

У току је један тениски гем и познато је колико поена је освојио сваки од играча. Напиши програм који на основу тога исписује резултат у уобичајеном формату (помоћу ознака `0`, `15`, `30`, `40`, `ad`).

Наиме, резултат у тениским гемовима се изражава на следећи начин. Када играч освоји свој први поен у гему каже да је освојио `15`, када освоји наредни поен каже се да је освојио `30`, када освоји наредни поен каже се да је освојио `40` и када освоји наредни поен осваја цео гем, осим ако у том тренутку противник већ има `40`. Ако има, онда се игра на разлику и играч не осваја цео гем, већ само има предност `ad`. Ако освоји поен док има предност осваја цео гем, а ако противник освоји поен резултат се враћа на `40:40`.

Опис улаза

У првој линији стандардног улаза налази се број освојених поена првог, а у другој линији број освојених поена другог играча. Претпоставити да је резултат исправан тј. да је гем и даље у току.

Опис излаза

На стандардни излаз исписати тренутни резултат у гему

Пример 1

Улаз	*Излаз*
`3` `1`	`40:15`

Пример 2

Улаз	*Излаз*
`0` `2`	`0:30`

Пример 3

Улаз	*Излаз*
`5` `6`	`40:ad`

Пример 4

Улаз	*Излаз*
`7` `7`	`40:40`

Решење

Да бисмо решили задатак, потребно је да прво успоставимо везу између броја освојених поена и уобичајеног приказа резултата у тенису. Ако је играч освојио 0 поена тада као његов резултат у тенису треба да прикажемо 0. Затим, ако је освојио један поен, тада треба да прикажемо 15, ако је освојио два поена тада треба да прикажемо 30 и на крају ако је освојио 3 поена, тада треба да прикажемо 40. Овде треба приметити да у ова 4 случаја, број који приказујемо не зависи од броја освојених поена противника.

Међутим, ако је било који играч стигао до 4 поена, тада се број поена другог играча може разликовати највише за 1, па у зависности од тога који играч има више поена (ако уопште има), њему треба да доделимо предност, тј. да одштампамо `ad`. Одавде закључујемо да приликом

одређивања резултата морамо да упоредимо првог играча са другим, али и другог играча са првим. Пошто се поени и за једног и за другог играча одређују истим поступком, тај поступак једноставности ради можемо дефинисати у засебној функцији.

```
#include <iostream>

using namespace std;

string rezultat(int x, int y) {
  if (x == 0)
    return "0";
  if (x == 1)
    return "15";
  if (x == 2)
    return "30";
  if (x == 3)
    return "40";
  if (x > y)
    return "ad";
  else
    return "40";
}

int main() {
  int a, b;
  cin >> a >> b;
  cout << rezultat(a, b) << ":" << rezultat(b, a) << endl;
  return 0;
}
```

Задатак: Додата цифра стотина

Аутори: Филип Марић, Владимир Кузмановић

Такмичење: 2022/2023. квалификације 3, 7. разред, 1. задатак

Двоцифреном броју x дописана је цифра c са леве стране и добијен је број који је k пута већи од броја x. На пример, када се броју 28 допише цифра 7 са леве стране добија се број 728 који 26 пута већи од полазног броја 28. Напиши програм који на основу вредности c и k одређује непознати број x.

Опис улаза

Са стандардног улаза се уносе цифра c и природни број k.

Опис излаза

На стандардни излаз исписати тражени број x или текст `ne` ако не постоји такав број за унете вредности c и k.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
7 26	28	6 11	60	3 27	ne

Решење

Да бисмо решили задатак, прво треба математички да дефинишемо проблем, тј. да из описа речима проблем пребацимо у једначину коју треба решити. Из текста задатка знамо да нови број добијамо тако што полазни двоцифрени број x проширимо са леве стране цифром c. Математички то можемо записати као $c \cdot 100 + x$. У наставку текста задатка, сазнајемо да је тако добијени број k пута већи од полазног броја x. Математички то можемо записати као $k \cdot x$, одакле лако формирамо једначину $c \cdot 100 + x = k \cdot x$. Након што смо проблем записали математички, можемо га лако решити.

Као улазни подаци дате су нам вредности c и k, па је очигледно да једначину треба да решимо по x. Пребацивањем непознате величине x на једну страну и познатих вредности на другу страну, добијамо следећи израз $c \cdot 100 = x \cdot (k - 1)$. Решавајући једначину по x, добијамо следећи израз $x = \frac{(c \cdot 100)}{(k-1)}$.

Иако делује да смо решили задатак, треба приметити да се у задатку ради о природним бројевима, па количник којим је описано решење не мора да постоји. Дакле, потребно је да испитатмо да ли је број $(c \cdot 100)$ дељив бројем $(k - 1)$. Ако број јесте дељив, тада треба да одредимо количник x и одштампамо га на стандардни излаз као решење задатка. Ако број није дељив, тада количник не постоји и на стандардни излаз треба да одштампамо `ne`.

```
#include <iostream>

using namespace std;

int main() {
  int c;
  cin >> c;
  int k;
  cin >> k;
  if ((c * 100) % (k - 1) == 0)
    cout << (c * 100) / (k - 1) << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

Задатак: Ко први игра

Аутор: Иван Дрецун

Такмичење: 2022/2023. квалификације 3, 8. разред, 1. задатак

Ана, Бобан и Влада желе да играју друштвену игру. Како би одредили којим редом ће играчи играти, свако од њих баца коцкицу. Први ће играти онај који је добио највећи број, други ће

играти онај који је добио наредни највећи број, итд. Напиши програм који исписује редослед којим играчи играју ако је познато ко је добио који број.

Опис улаза

Са стандардног улаза се уносе три броја a, b и v одвојена размаком, чије су вредности између 1 и 6. Они представљају бројеве на коцкици које су добили Ана, Бобан и Влада, тим редом. Познато је да ће сви бројеви бити различити.

Опис излаза

На стандардни излаз исписати слова `A`, `B` и `V`. Редослед исписа треба да одговара редоследу којим играчи играју. **Слова исписати без размака.**

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`2 6 3`	`BVA`	`5 4 3`	`ABV`	`2 1 4`	`VAB`

Решење

Испробавање свих редоследа

Задатак је могуће решити обрађивањем свих могућих редоследа игре, пошто их има јако мало (шест).

```
#include <iostream>

using namespace std;

int main() {
  int a, b, v;
  cin >> a >> b >> v;

  if (a > b && b > v) cout << "ABV\n";
  if (a > v && v > b) cout << "AVB\n";
  if (b > a && a > v) cout << "BAV\n";
  if (b > v && v > a) cout << "BVA\n";
  if (v > a && a > b) cout << "VAB\n";
  if (v > b && b > a) cout << "VBA\n";

  return 0;
}
```

Испробавање свих вредности на коцкицама

Задатак је могуће решити и провером свих бројева на коцкицама од 6 до 1.

```
#include <iostream>

using namespace std;

int main() {
  int a, b, v;
```

```
  cin >> a >> b >> v;

  for (int broj = 6; broj >= 1; broj--) {
    if (broj == a) cout << "A";
    if (broj == b) cout << "B";
    if (broj == v) cout << "V";
  }
  cout << '\n';

  return 0;
}
```

Задатак: Милионер

Аутор: Душан Пођадић

Такмичење: окружно 2022/2023, VII разред, 1. задатак

У квизу "Желите ли да постанете милионер" такмичари одговарају на питања која имају 4 понуђена одговора (a, b, c, d). Такмичарима су на располагању три помоћи:

- *Помоћ публике* - 100 гледалаца из публике гласа за један од 4 понуђена одговора и учесник може да види колико људи је гласало за који одговор.
- *Пола-пола* - рачунар насумично одбаци два нетачна одговора и остави играчу један тачан и један нетачан.
- *Позови пријатеља* - учесник телефоном зове свог пријатеља који му каже шта мисли да је тачан одговор.

Дарко је стигао до претпоследњег питања и остале су му све три помоћи. Пошто не зна одговор одлучио се за следећу тактику: прво ће питати публику, ако један одговор има више гласова од свих осталих заједно изабраће њега. Ако то није случај, узеће помоћ пола-пола. Када остану два понуђена одговора, Дарко ће изабрати онај који има бар 20 гласова публике више у односу на други. Ако ниједан одговор не испуњава услов, Дарко ће позвати пријатеља и одговориће онако како му пријатељ каже. Одредити Дарков коначан одговор на последње питање.

Опис улаза

У првом реду стандардног улаза се налазе 4 броја одвојена размацима који представљају број гледалаца који је гласао редом за одговоре a, b, c, и d. Сви ови бројеви су између 0 и 100 и збир им је 100. Уколико је потребно (тј. ако Дарко није на основу прве помоћи дао одговор), у другом реду се налазе два различита слова (из скупа слова a, b, c, d) одвојена размаком која представљају понуђене одговоре које је рачунар **оставио** Дарку. Уколико је потребно (тј. ако Дарко ни на основу друге помоћи није дао одговор), у последњем реду се налази једно слово - одговор који пријатељ предлаже. То слово је сигурно једно од оних које је рачунар оставио као могућ одговор.

Опис излаза

У једином реду стандардног излаза исписати једно слово (a, b, c или d) које представља коначан одговор који је Дарко дао.

Пример 1

Улаз	*Излаз*	*Објашњење*
`12 17 11 60`	`d`	Пошто је за одговор d гласало више гледалаца него за све остале одговоре заједно, Дарко одмах одлучује да то буде његов одговор и не користи остале помоћи.

Пример 2

Улаз	*Излаз*	*Објашњење*

`12 17 34 37`
`b c`
`c`
	`c`	Након помоћи публике ниједан одговор нема неопходну већину, па Дарко тражи помоћ пола-пола. Пошто остају одговори b и c чија је разлика у броју гласова публике мања од 20, Дарко тражи помоћ пријатеља и даје коначан одговор c јер је пријатељ тако предложио.

Решење

При решавању задатка је потребно спровести тачно онај поступак који се у задатку и тражи:

- Учитавамо 4 броја и проверавамо да ли је један већи од осталих заједно (пошто знамо да је збир тачно 100, довољно је проверити да ли је бар један строго већи од 50).
- Уколико постоји такав број исписујемо одговарајуће слово (a, b, c, или d), уколико не постоји учитавамо помоћ пола-пола.
- Уколико је разлика у гласовима публике између одговора који су учитани бар 20, исписујемо онај одговор који има више гласова публике, уколико није онда учитавамо помоћ пријатеља.
- Исписујемо одговор који учитамо као помоћ пријатеља.

Једно што у овом задатку може да нам направи проблем је повезивање одговора (a, b, c, и d) са гласовима публике. Једна могућа имплементација је да се бројеви који представљају гласове публике сместе у један низ који зовемо publika, а да се онда тај низ индексира коришћењем ASCII вредности слова из понуђених одговора, тј. одузимањем ASCII вредности слова ‘a’ од сваког понуђеног одговора. На нултом месту у низу се налази број гласова публике за одговор a (када одузмемо од ASCII вредности карактера ‘a’, ASCII вредност карактера ‘a’ добије се 0), на првом месту за одговор b (када одузимамо од ASCII вредности карактера ‘b’, ASCII вредност карактера ‘a’ добије се 1) и тако даље. Алтренативно могуће је користити и речник где би слова била кључеви, а гласови публике вредности.

```
#include <iostream>

using namespace std;

char odluka(){
  int publika[4];

  // ucitavamo pomoc publike
  cin >> publika[0] >> publika[1] >> publika[2] >> publika[3];

  for (int i = 0; i < 4; i++){
    if (publika[i] > 50)
      return 'a' + i;
  }
```

```
  char o1, o2;
  // ucitavamo pola-pola
  cin >> o1 >> o2;

  if (publika[o1-'a'] - publika[o2-'a'] >= 20)
    return o1;

  if (publika[o2-'a'] - publika[o1-'a'] >= 20)
    return o2;

  char o;
  // ucitavamo pomoc prijatelja
  cin >> o;
  return o;

}

int main() {
  cout << odluka();
  return 0;
}
```

Други начин имплементације решења би био да се гласови публике уместо у низ сместе у 4 променљиве (a, b, c и d), па да се, када је потребно користе наредбе гранања. За ову имплементацију је згодно направити засебну функцију која враћа коначан одговор.

```
#include <iostream>

using namespace std;

char odluka(){
  int a, b, c, d;
  // ucitavamo pomoc publike
  cin >> a >> b >> c >> d;

  if (a > 50) return 'a';
  if (b > 50) return 'b';
  if (c > 50) return 'c';
  if (d > 50) return 'd';

  // ucitavamo pola-pola
  char o1, o2;
  cin >> o1 >> o2;
  int p1 = 0, p2 = 0; // poeni za o1 i o2 redom

  if (o1 == 'a') p1 = a;
  else if (o1 == 'b') p1 = b;
  else if (o1 == 'c') p1 = c;
  else if (o1 == 'd') p1 = d;
```

```
  if (o2 == 'a') p2 = a;
  else if (o2 == 'b') p2 = b;
  else if (o2 == 'c') p2 = c;
  else if (o2 == 'd') p2 = d;

  if (p1 - p2 >= 20)
    return o1;
  if (p2 - p1 >= 20)
    return o2;

  // ucitavamo pomoc prijatelja
  char o;
  cin >> o;
  return o;

}

int main() {
  cout << odluka();
  return 0;
}
```

Задатак: Две слике на папиру

Аутори: Филип Марић, Владимир Кузмановић

Такмичење: окружно 2022/2023, VII разред, 2. задатак

Написати програм који испитује колико постоји исправних распореда две слике познатих димензија на папиру познатих димензија, тако да су ивице слика паралелне ивицама папира, да су слике приказане целе и да се не преклапају. Слике није допуштено ротирати, али папир јесте (могуће је постављати га било у хоризонталној "пејзаж" оријентацији, било у вертикалној "портрет" оријентацији). Слике је допуштено постављати било једну испод друге било једну поред друге, на папир који је окренут било хоризонтално, било вертикално (постоји, дакле, највише 4 могућа распореда, при чему неки од њих могу бити неисправни, јер су слике превелике).

Опис улаза

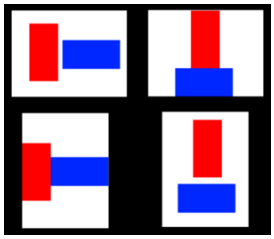
Са стандардног улаза се уносе димензије (ширина и висина) папира, димензије прве слике и димензије друге слике.

Опис излаза

На стандардни излаз исписати број од 0 до 4 који представља број могућих начина да се слике распореде на папир.

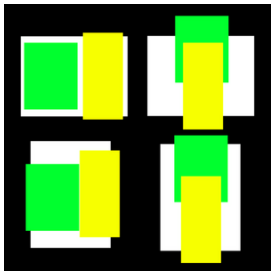
Пример 1

Улаз	*Излаз*	*Објашњење*

`800 600`
`200 400`
`400 200`
	`4`	

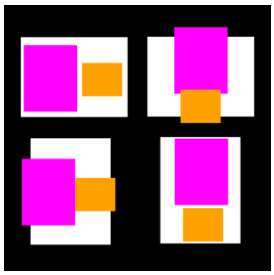
Пример 2

Улаз	*Излаз*	*Објашњење*

`800 600`
`400 500`
`300 650`
	`0`	

Пример 3

Улаз	*Излаз*	*Објашњење*

`800 600`
`400 500`
`300 250`
	`2`	

Решење

Опис главног решења

Да бисмо урадили задатак потребно је да прво извршимо анализу могућих случајева. Према условима задатка слике можемо да поставимо или једну поред друге или једну испод друге. Поред распореда слика, додатни услов задатка је и оријентација папира, портрет или пејзаж. Одавде закључујемо да су у задатку могућа 4 случаја:

- **Случај 1**: Папир је у Портрет оријентацији и слике су поређане једна до друге.
- **Случај 2**: Папир је у Портрет оријентацији и слике су поређане једна испод друге.
- **Случај 3**: Папир је у Пејзаж оријентацији и слике су поређане једна до друге.
- **Случај 4**: Папир је у Пејзаж оријентацији и слике су поређане једна испод друге.

Папир је описан следећим вредностима *papir_sirina* (ili kraće sp) i *papir_visina* (ili kraće vp). Слике су редом описане вредностима *slika1_sirina*, *slika1_visina*, *slika2_sirina* и *slika2_visina* (ili kraće $s1, v1, s2, v2$). На почетку рада програма учитаћемо све ове вредности. Поред тога, број исправних распоред слика на папиру обележићемо са $broj$ и на почетку рада програма иницијализоваћемо га на 0.

У првом случају, слике ређамо једну до друге и папир је у портрет положају. Укупна ширина коју заузимају тако поређане слике једнака је збиру ширина слика, а укупна висина тако поређаних слика једанака је максимуму висина те две слике. Дакле, да би слике стале на папир у портрет оријентацији укупна ширина слика мора да буде мања или једнака од ширине папира. Такође укупна висина слика мора бити мања или једнака од висине папира. Ако је тај услов испуњен, инрементираћемо вредност бројача $broj$.

У другом случају, слике ређамо једну испод друге и папир је у портрет положају. Укупна висина коју заузимају тако поређане слике једнака је збиру висина слика, а укупна ширина тако поређаних слика једанака је максимуму ширина те две слике. Дакле, да би слике стале на папир у портрет оријентацији укупна ширина слика мора да буде мања или једнака од ширине папира. Такође укупна висина слика мора бити мања или једнака од висине папира. Ако је тај услов испуњен, инрементираћемо вредност бројача $broj$.

У случајевима три и четири, услови се испитују аналогно, само је потребно променити оријентацију папира у пејзаж. То се лако постиже тако што у условима заменимо ширину и висину папира.

Описани поступак је приказан у решењу.

```
#include <iostream>
using namespace std;

int main() {
  int papir_sirina, papir_visina;
  cin >> papir_sirina >> papir_visina;
  int slika1_sirina, slika1_visina;
  cin >> slika1_sirina >> slika1_visina;
  int slika2_sirina, slika2_visina;
  cin >> slika2_sirina >> slika2_visina;
  int broj = 0;
  if (slika1_sirina + slika2_sirina <= papir_sirina &&
      max(slika1_visina, slika2_visina) <= papir_visina)
    broj++;
  if (slika1_visina + slika2_visina <= papir_visina &&
      max(slika1_sirina, slika2_sirina) <= papir_sirina)
    broj++;
  if (slika1_sirina + slika2_sirina <= papir_visina &&
      max(slika1_visina, slika2_visina) <= papir_sirina)
    broj++;
  if (slika1_visina + slika2_visina <= papir_sirina &&
      max(slika1_sirina, slika2_sirina) <= papir_visina)
    broj++;
  cout << broj << endl;
  return 0;
```

```
}
```

Задатак: Однос суседних

Аутор: Милан Вуīделија

Такмичење: државно 2022/2023, VI разред, 2. задатак

За пет непознатих различитих бројева, a, b, c, d, e, познат је однос величина свака два узастопна. Тачније, познато је:

- да ли је $a < b$ или $a > b$
- да ли је $b < c$ или $b > c$
- да ли је $c < d$ или $c > d$
- да ли је $d < e$ или $d > e$

Написати програм који на основу ових информација одређује који од ових бројева може да буде најмањи, а који највећи.

Опис улаза

На стандардном улазу се у једном реду налазе четири карактера, раздвојена по једном размаком. Сваки од тих карактера је $<$ или $>$.

- Ако је први карактер једнак $<$, то значи да је $a < b$, у противном је $a > b$.
- Ако је други карактер једнак $<$, то значи да је $b < c$, у противном је $b > c$.
- Ако је трећи карактер једнак $<$, то значи да је $c < d$, у противном је $c > d$.
- Ако је четврти карактер једнак $<$, то значи да је $d < e$, у противном је $d > e$.

Опис излаза

На стандардни излаз исписати два реда. У првом реду исписати редом (од a до e) ознаке свих бројева који могу да буду најмањи, а у другом реду редом (такође од a до e) ознаке свих бројева који могу да буду највећи.

Пример 1

Улаз	*Излаз*	*Објашњење*
`< < > >`	`ae` `c`	Улазни подаци значе да је $a < b, b < c, c > d, d > e$. Из ових односа може да се закључи да је c највећи од ових бројева, а најмањи може да буде само a или e.

Пример 2

Улаз	*Излаз*
`< > > <`	`ad` `be`

Решење

Број може да буде најмањи ако и само ако ниједан од његових непосредних суседа (у редоследу набрајања) није мањи од њега. У решењу можемо да користимо ниску (стринг) `najmanji`, у коју ћемо да убацујемо ознаке оних бројева који би могли да буду најмањи. конкретно, у ниску `najmanji` убацујемо

- слово `a` ако је први знак на улазу једнак `<`

- слово `b` ако је први знак на улазу једнак `>`, а други `<`
- слово `c` ако је други знак на улазу једнак `>`, а трећи `<`
- слово `d` ако је трећи знак на улазу једнак `>`, а четврти `<`
- слово `e` ако је четврти знак на улазу једнак `>`

Слично томе, у ниску `najveci` убацујемо

- слово `a` ако је први знак на улазу једнак `>`
- слово `b` ако је први знак на улазу једнак `<`, а други `>`
- слово `c` ако је други знак на улазу једнак `<`, а трећи `>`
- слово `d` ако је трећи знак на улазу једнак `<`, а четврти `>`
- слово `e` ако је четврти знак на улазу једнак `<`

```
#include <iostream>

using namespace std;

int main() {
    string ab, bc, cd, de;
    cin >> ab >> bc >> cd >> de;
    string najmanji = "";
    string najveci = "";

    if (ab == "<")
        najmanji += "a";
    else
        najveci += "a";

    if (ab == ">" && bc == "<")
        najmanji += "b";
    else if (ab == "<" && bc == ">")
        najveci += "b";

    if (bc == ">" && cd == "<")
        najmanji += "c";
    else if (bc == "<" && cd == ">")
        najveci += "c";

    if (cd == ">" && de == "<")
        najmanji += "d";
    else if (cd == "<" && de == ">")
        najveci += "d";

    if (de == ">")
        najmanji += "e";
    else
        najveci += "e";

    cout << najmanji << endl;
    cout << najveci << endl;
```

```
    return 0;
}
```

Алтернативно, можемо да (у склопу ниске) чувамо све кандидате за најмањи тј. највећи број, и да затим избацујемо оне за које је установљено да не могу бити највећи тј. најмањи. Ако је први карактер `<` тада први број не може бити највећи, а други најмањи, а ако је први карактер `>` тада први број не може бити најмањи, а други највећи. Сличну анализу вршимо и за остале карактере које учитавамо (најједноставније у петљи).

```
#include <iostream>
#include <string>
#include <set>

using namespace std;

int main() {
  string znak;
  getline(cin, znak);
  set<char> najmanji = {'a', 'b', 'c', 'd', 'e'};
  set<char> najveci = {'a', 'b', 'c', 'd', 'e'};

  for (int i = 0; i < 4; i++) {
    if (znak[2*i] == '<') {
      najveci.erase('a' + i);
      najmanji.erase('a' + i+1);
    } else {
      najveci.erase('a' + i+1);
      najmanji.erase('a' + i);
    }
  }

  cout << string(najmanji.begin(), najmanji.end()) << endl;
  cout << string(najveci.begin(), najveci.end()) << endl;
  return 0;
}
```

Задатак: Последњи меч

Аутор: Филип Марић

Такмичење: 2022/2023. квалификације 2, 7. и 8. разред, 1. задатак

У току је светско првенство у фудбалу. У групи од 4 тима (`A`, `B`, `C` и `D`) одигране су све утакмице, осим последње у којој се састају тимови `C` и `D`. Ако је познат број освојених поена сва 4 тима, напиши програм који одређује да ли тим `D` има шансе да се квалификује даље и ако има, колико је поена потребно да освоји у последњем мечу (за победу се осваја 3 поена, а за нерешен резултат оба тима добијају по 1 поен). Даље иду две екипе са највећим освојеним бројем поена, а ако екипа `D` има исти број поена као нека друга, сматрати да она има предност, јер је домаћин.

Опис улаза

Са стандардног улаза се учитава тренутно освојени број поена екипа `A`, `B`, `C` и `D` (тим редом).

Опис излаза

На стандардни излаз исписати најмањи довољан број поена за екипу `D` у последњем мечу или `ne` ако екипа нема шансе да се квалификује.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`6`	`ne`	`6`	`0`	`9`	`1`
`4`		`4`		`0`	
`4`		`0`		`1`	
`0`		`4`		`3`	

Решење

Основна идеја нам је да анализирамо редом ситуације у којима је екипа D изгубила у последњем мечу, било је нерешено и екипа D је победила. Екипа D се пласира даље ако није лошија бар од две друге екипе, што можемо урадити поређењем D са екипама A, B, затим A, C и на крају B, C. Ако D изгуби она задржава свој број поена а екипа C има три поена више и ако се на тај начин екипа D пласира даље исписујемо 0. Ако не, онда проверавамо нерешен исход што значи да D и C имају по један додатни поен и ако се на тај начин екипа D пласира, исписујемо 1. Ако не, онда проверавамо случај да је D победила и тада она има 3 поена више. Ако се на тај начин екипа D пласира исписујемо 3, а у супротном исписујемо `ne`.

```
#include <iostream>

using namespace std;

int main() {
  int A, B, C, D;
  cin >> A >> B >> C >> D;

  if ((D >= A && D >= B) || (D >= B && D >= C+3) || (D >= B && D >= C+3))
    cout << 0 << endl;
  else if ((D+1 >= A && D+1 >= B) || (D+1 >= B && D+1 >= C+1) || (D+1 >= B && D+1 >= C+1))
    cout << 1 << endl;
  else if ((D+3 >= A && D+3 >= B) || (D+3 >= B && D+3 >= C) || (D+3 >= B && D+3 >= C))
    cout << 3 << endl;
  else
    cout << "ne" << endl;

  return 0;
}
```

Много лепше решење добијамо ако проверу да ли се екипа D пласирала издвојимо у посебну функцију.

```
#include <iostream>
using namespace std;

bool proveri(int A, int B, int C, int D) {
```

```
  return (D >= A && D >= B) || (D >= A && D >= C) || (D >= B && D >= C);
}

int main() {
  int bodoviA, bodoviB, bodoviC, bodoviD;
  cin >> bodoviA >> bodoviB >> bodoviC >> bodoviD;

  if (proveri(bodoviA, bodoviB, bodoviC + 3, bodoviD))
    cout << 0 << endl;
  else if (proveri(bodoviA, bodoviB, bodoviC + 1, bodoviD + 1))
    cout << 1 << endl;
  else if (proveri(bodoviA, bodoviB, bodoviC, bodoviD + 3))
    cout << 3 << endl;
  else
    cout << "ne" << endl;
}
```

Проверу можемо извршити и тако што сортирамо низ поена опадајуће (при чему ако исте екипе имају исти број поена, сортирамо их абецедно, опадајуће). Екипа D пролази ако је након тог сортирања на првом или другом месту у низу.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

bool proveri(int bodoviA, int bodoviB, int bodoviC, int bodoviD) {
  vector<pair<int, char>> plasman{make_pair(bodoviA, 'A'),
                                  make_pair(bodoviB, 'B'),
                                  make_pair(bodoviC, 'C'),
                                  make_pair(bodoviD, 'D')};
  sort(begin(plasman), end(plasman));
  return plasman[3].second == 'D' || plasman[2].second == 'D';
}

int main() {
  int bodoviA, bodoviB, bodoviC, bodoviD;
  cin >> bodoviA >> bodoviB >> bodoviC >> bodoviD;

  if (proveri(bodoviA, bodoviB, bodoviC + 3, bodoviD))
    cout << 0 << endl;
  else if (proveri(bodoviA, bodoviB, bodoviC + 1, bodoviD + 1))
    cout << 1 << endl;
  else if (proveri(bodoviA, bodoviB, bodoviC, bodoviD + 3))
    cout << 3 << endl;
  else
    cout << "ne" << endl;

  return 0;
```

```
}
```

Задатак: Такси растојање око правоугаоника

Аутор: Милан Вуīделија

Такмичење: 2021/22. квалификације 1, VII и VIII разред 3. задатак

Дате су тачке A, B и правоугаоник $MPNQ$, страница паралелних осама, задат теменима P, Q једне своје дијагонале. Одредити дужину најкраће полигоналне линије са почетком A и крајем B, страница паралелних осама, која не садржи унутрашње тачке правоугаоника $MPNQ$.

Опис улаза

Са стандардног улаза се уноси 8 реалних бројева из интервала $[-1000, 1000]$, заокружених на највише четири децимале. У првом реду се уносе x и y координата тачке A, у другом тачке B, у трећем тачке P, а у четвртом тачке Q.

Подаци су такви да тачке A и B нису унутар правоугаоника.

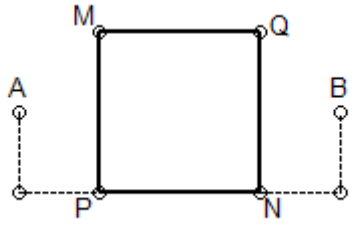
Опис излаза

На стандардни излаз исписати тражену дужину на четири децимале.

Пример

Улаз	*Излаз*	*Објашњење*

`0.0 -2.00`
`0.0 2.00`
`-2.0 1.0`
`1.0 -1.0`
	`6.0000`	Испод је шематски приказан распоред датих тачака $A(0,-2)$, $B(0,2)$, $P(-2,1)$, $Q(1,-1)$. Тражена најкраћа полигонална линија је $AXYB$, где је $X(1,-2)$, $Y(1,2)$. Дужине дужи AX, XY, YB су редом 1, 4, 1, што у збиру даје 6.

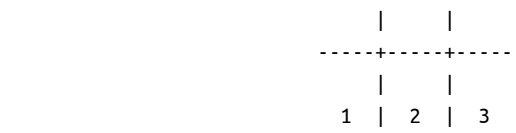


Решење

Упутство:

Поделимо раван на 9 области у односу на правоугаоник и нумеришимо их бројевима од 1 до 9 у следећем редоследу: лево-испод, испод, десно-испод, лево, унутар, десно, лево-изнад, изнад, десно-изнад.

```
  7  |  8  |  9
     |     |
-----+-----+-----
     |     |
  4  |  5  |  6
```



Задатак може да се реши анализирањем различитих случајева према томе којим областима припадају тачке A и B. Посебно се издвајају случајеви када крајње тачке припадају областима 2 и 8, или областима 4 и 6, док се сви остали случајеви не морају анализирати посебно. Конкретније:

- У случају да тачке A и B припадају областима 4 и 6, тражена дужина је једнака мањем од збирова $|a_y - p_y| + |a_x - b_x| + |p_y - b_y|$ и $|a_y - q_y| + |a_x - b_x| + |q_y - b_y|$.
- У случају да тачке A и B припадају областима 2 и 8, тражена дужина је једнака мањем од збирова $|a_x - p_x| + |a_y - b_y| + |p_x - b_x|$ и $|a_x - q_x| + |a_y - b_y| + |q_x - b_x|$.
- У свим осталим случајевима тражена дужина је једнака $|ax - bx| + |ay - by|$

```
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

bool Poredak(double t1, double t2, double t3, double t4) {
  double m = min(t1, t4), M = max(t1, t4);
  return (m <= t2 && m <= t3 && M >= t2 && M >= t3);
}

int main() {
  double ax, ay, bx, by, px, py, qx, qy;
  cin >> ax >> ay >> bx >> by >> px >> py >> qx >> qy;

  double d;
  if (Poredak(ax, px, qx, bx) && Poredak(py, ay, by, qy))
      d = min(abs(ay-py) + abs(by-py), abs(qy-ay) + abs(qy-by))
        + abs(ax-bx);
  else if (Poredak(ay, py, qy, by) && Poredak(px, ax, bx, qx))
      d = min(abs(ax-px) + abs(bx-px), abs(qx-ax) + abs(qx-bx))
        + abs(ay-by);
  else
      d = abs(ax-bx) + abs(ay-by);

  cout << fixed << showpoint << setprecision(4) << d << endl;
  return 0;
}
```

Задатак: Jaja

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 1, VII и VIII разред 3. задатак

У сваку кутију за јаја може да стане K јаја. Када Јоца пакује јаја, он користи најмањи број кутија у које јаја могу да стану, али пошто је сујеверан, он никада не оставља у кутији (позитиван) број јаја дељив са 3.

Написати програм који одређује колико кутија треба Јоци да би спаковао J јаја.

Опис улаза

у првом реду стандардног улаза је број K, који је увек 10 или 15. У другом реду је цео број J, такав да је $0 \leq J \leq 100$.

Опис излаза

На стандардни излаз исписати један цео број, број кутија које ће Јоца употребити.

Пример 1

Улаз	*Излаз*
10	2
18	

Пример 2

Улаз	*Излаз*
15	1
10	

Решење

Прво што можемо да приметимо је да у кутије од 15 јаја можемо да ставимо највише 14 јаја. Према томе, оваквим кутијама можемо одмах након учитавања да смањимо величину за 1 и поједноставимо даље разматрање.

Погледајмо шта би се догодило када би се кутије пуниле редом до краја. Тада би број потребних кутија био $\lceil \frac{J}{K} \rceil$, где је J број јаја која треба спаковати, а K број јаја која стају у кутију.

Пошто након почетне исправке величина кутије K више није дељива са 3, то услов дељивости може да утиче само на последњу употребљену кутију. Зато још треба посебно испитати ситуацију када је број јаја у последњој употребљеној кутији дељив са 3. Таква сиутација може бити разрешена или узимањем нове кутије (само ако је неопходно), или пребацивањем једног или више јаја из претходно попуњених кутија у последњу. Додавање јаја у последњу кутију није могуће само у следећа два случаја:

(1) када нема ниједне претходно попуњене кутије из које бисмо "позајмили" јаја (то јест када је $J \leq K$).

(2) када у последњој кутији има места само за још једно јаје (то јест када је $J \bmod K = K - 1$). У овом случају додавање није могуће зато што би пребацивањем једног јајета из било које попуњене кутије у тој претходно попуњеној кутији остао број јаја дељив са 3.

У свим осталим случајевима се из претпоследње у последњу кутију могу пребацити бар једно или два јаја. На тај начин, број јаја у последњој кутији у сваком случају више неће бити дељив са 3, а од два могућа пребацивања (једно или два јаја) једно од тих пребацивања мора одговарати и претпоследњој кутији, тако да ни у њој број јаја не буде дељив са 3. Према томе, у свим случајевима осим два издвојена раније, број кутија се не мора повећавати.

```
#include <iostream>
using namespace std;

int main()
{
```

```
    int velKutije, brJaja;
    cin >> velKutije >> brJaja;
    if (velKutije % 3 == 0) velKutije--;

    int brKutija = (brJaja + velKutije - 1) / velKutije;
    int uPoslednjoj = brJaja % velKutije;

    // ako je u poslednjoj kutiji nezgodan broj
    if (uPoslednjoj > 0 && uPoslednjoj % 3 == 0)
      // ako nemamo odakle u nju da dodamo (poslednja kutija je jedina)
      // ili ako bi se dodavanjem pokvarila prethodna kutija
      if (brKutija == 1 || uPoslednjoj + 1 == velKutije)
        brKutija++;  // onda ce trebati kutija vise

    cout << brKutija << endl;
    return 0;
}
```

Задатак: Одбојка

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 2, VII и VIII разред, 1. задатак

Данас се игра важан квалификациони меч између две локалне одбојкашке екипе, A и B. Бранко, који је велики љубитељ одбојке, због обавеза није могао да прати меч, па је видео само како се играчи поздрављају по завршетку утакмице. Тада се на екрану појавио последњи од оних досадних статистичких података који каже да је екипа A освојила укупно a поена, а екипа B укупно b поена током целе утакмице.

На основу ове информације Бранко сада покушава да закључи која екипа је победила. Наравно, Бранко зна да се меч играо на три добијена сета, а СВАКИ сет на 25 освојених поена (ни пети сет није скраћен), с тим да сет не може да се заврши са $25 : 24$. У случају резултата $24 : 24$ сет се наставља док једна од екипа не стекне предност од 2 поена.

Напишите програм који Бранку даје одговор на питање које га мучи.

Опис улаза

Са стандардног улаза се у првом реду уноси цео број a ($0 \leq a \leq 100$), а у другом реду цео број b ($0 \leq b \leq 100$), бројеви поена које су освојиле екипе током целе утакмице.

Опис излаза

На стандардни излаз исписати само једно слово.

- Ако је на основу укупног броја освојених поена извесно да је меч добила екипа A, исписати слово A.
- Ако је на основу укупног броја освојених поена извесно да је меч добила екипа B, исписати слово B.
- Ако није извесно која екипа је добила меч (то јест, ако је меч могла добити било која екипа), исписати слово N.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
72 90	B	89 82	N

Решење

Размотримо најпре колико најмање поена треба да освоји екипа да би добила меч. Да би екипа добила меч, треба да добије три сета, а у сваком добијеном сету треба да освоји бар 25 поена. То значи да екипа која на крају меча има мање од 75 поена никако није могла да добије меч. Са друге стране, могуће је да екипа која освоји 75 поена добије меч, али то зависи и од броја поена противничке екипе.

Размотримо зато и колико екипа која изгуби три сета (а тиме и меч) може да надмаши победничку екипу у простом збиру поена. Екипа која је изгубила може да добије два сета са највише по 25 поена предности (резултатом 25:0), а да изгуби три сета са по најмање два поена заостатка. То значи да поражена екипа може да сакупи чак $2 \cdot 25 - 3 \cdot 2 = 50 - 6 = 44$ поена више од противника а да ипак изгуби меч. Екипа која има 45 или више поена предности у збиру, није могла да изгуби меч.

Из ове анализе произилази следећи закључак:

- ако је $a \geq 75$ и $b - a \leq 44$, екипа A је могла да добије меч.
- ако је $b \geq 75$ и $a - b \leq 44$, екипа B је могла да добије меч.

Пошто је меч завршен победом једне екипе, бар један од ових услова ће бити испуњен. Јасно, ако је испуњен само један од ових услова онда знамо победника, а ако су испуњена оба услова, онда је победник могла бити било која екипа.

```
#include <iostream>

using namespace std;

int main() {
  int a, b;
  cin >> a >> b;
  bool mogla_a = (a > 74) && (a + 44 >= b);
  bool mogla_b = (b > 74) && (b + 44 >= a);
  if (mogla_a && mogla_b)
    cout << 'N' << endl;
  else if (mogla_a)
    cout << 'A' << endl;
  else // if (mogla_b)
    cout << 'B' << endl;
}
```

Глава 4

Примене минимума и максимума неколико бројева

Задатак: Паковање

Аутор: Филип Марић

Такмичење: 2021/22. квалификације 2, V разред, 2. задатак и VI разред, 2. задатак

Ближи се Нова година и деца припремају поклоне за своје другаре. Поклони су у кутијама облика квадра и приликом паковања се облепљују украсним папиром. Ако је квадар димензија $a \times b \times c$, његове стране су два правоугаоника површине $a \times b$, два правоугаоника површине $b \times c$ и два правоугаоника површине $a \times c$. Укупна површина је збир свих 6 површина правоугаоника. Ипак, да би се поклон могао лепо запаковати, потребно је да се одвоји мало вишка папира – најекономичније је да тај вишак буде површине која је једнака најмањој од три површине правоугаоника. Напиши програм који одређује колико је папира потребно да се запакује поклон.

Опис улаза

Са стандардног улаза се учитавају три цела броја a, b и c, између 1 и 100, који представљају димензије поклона.

Опис излаза

На стандардни излаз исписати тражену најмању површину папира за увијање.

Пример 1

Улаз	*Излаз*
2 4 3	58

Пример 2

Улаз	*Излаз*
12 10 8	672

Решење

Површина квадра се лако може израчунати формулом $P_k = 2(ab + ac + bc)$. Након тога потребно је наћи додатну површину на основу формуле $P_v = \min(ab, ac, bc)$ и исписати збир те

две површине. Минимум три броја је најједноставније наћи коришћењем библиотечке функције.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int a, b, c;
  cin >> a >> b >> c;
  cout << 2*a*b + 2*b*c + 2*a*c + min({a*b, a*c, b*c}) << endl;
  return 0;
}
```

Задатак: Кувар

Аутор: Милан Вуīелија

Такмичење: 2022/2023. квалификације 1, 8. разред, 1. задатак

Кувару је за једну порцију потребно за главно јело (поред осталих састојака) m грама меса, а за салату или k грама купуса или c грама цвекле. На располагању је M грама меса, K грама купуса и C грама цвекле (а свих осталих састојака има много више). Написати програм који учитава потребне количине за једну порцију m, k, c, а затим и расположиве количине MKC, а исписује за колико целих порција (главно јело и салата) кувар има довољно састојака.

Опис улаза

На стандардном улазу су редом бројеви m, k, c, M, K, C, сваки у посебном реду. Сви бројеви су цели, позитивни и мањи од 50000.

Опис излаза

На стандарни излаз исписати само један цео неозначен број, а то је број целих порција које могу да се припреме.

Пример 1

Улаз	*Излаз*	*Објашњење*
200 250 150 2200 1400 890	10	Меса има довољно за 11 порција, а салате за 10 (5 салата од купуса и 5 од цвекле), па зато може да се припреми 10 комплетних оброка.

Пример 2

Улаз	*Излаз*	*Објашњење*
150 200 100 6100 5100 1000	35	Меса има довољно за 40 порција, а салате за 35 (25 салата од купуса и 10 од цвекле), па зато може да се припреми 35 комплетних оброка.

Решење

Број порција меса P_m може да се израчуна целобројним дељењем расположиве количине меса количином потребном за једну порцију.

На исти начин се израчунава број порција купус салате P_k, као и број порција салате од цвекле P_c. Укупан број порција салате је $P_s = P_k + P_c$.

Коначно, број оброка који могу да се припреме једнак је мањем од бројева P_m и P_s.

```
#include <iostream>

using namespace std;

int main() {
    int m, k, c, M, K, C;
    cin >> m >> k >> c >> M >> K >> C;
    int brPorcijaM = M / m;
    int brPorcijaK = K / k;
    int brPorcijaC = C / c;
    cout << min(brPorcijaM, brPorcijaK + brPorcijaC) << endl;
    return 0;
}
```

Задатак: Највећи двоцифрени број

Аутор: Милан Вуīелија

Такмичење: 2021/22. квалификације 1, VII разред 2. задатак

Написати програм који за унети троцифрени број исписује највећи од три двоцифрена броја, који се добијају изостављањем по једне цифре из полазног броја.

Опис улаза

У првом и једином реду стандардног улаза се налази један троцифрен број.

Опис излаза

На стандардни излаз исписати само један двоцифрен број.

Пример 1

Улаз	*Излаз*	*Објашњење*
`235`	`35`	Највећи од бројева 23, 25, 35.

Пример 2

Улаз	*Излаз*	*Објашњење*
`412`	`42`	Највећи од бројева 41, 42, 12.

Решење

Мада то није неопходно, можемо ради јасноће решења прво да издвојимо цифре датог троцифреног броја, а затим да формирамо могуће двоцифрене бројеве.

Највећи од три двоцифрена броја можемо да одредимо помоћу библиотечке функције за одређивање максимума.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
    int abc;
    cin >> abc;
    int a = abc / 100;
    int b = abc / 10 % 10;
    int c = abc % 10;
    int ab = a * 10 + b;
    int ac = a * 10 + c;
    int bc = b * 10 + c;
    int najveci = max({ab, ac, bc});
    cout << najveci << endl;
    return 0;
}
```

Задатак: Бака

Такмичење: општинско 2018/2019, V разред, 4. задатак, VI разред, 3. задатак

Аутор: Филип Марић

Напиши програм који учитава број година баке, тате (њеног сина) и унуке (његове ћерке) и који одређује за колико година је бака старија од своје унуке.

Опис улаза

Са стандардног улаза се уносе три природна броја мања од 100 који представљају године баке, тате и унуке. Бројеви се уносе у произвољном редоследу.

Опис излаза

На стандардни излаз исписати разлику година баке и унуке.

Пример 1

Улаз	*Излаз*
73 42 17	56

Пример 2

Улаз	*Излаз*
26 3 49	46

Решење

Бака има највише година, а унука најмање година, па се проблем своди на одређивање разлике између највећег и најмањег учитаног броја. Највећи и најмањи од три броја можемо наћи применом алгоритма за одређивање минимума и максимума.

```
#include <iostream>

using namespace std;

int main() {
  int a, b, c;
  cin >> a >> b >> c;
  int min = a, max = a;
  if (b < min)
    min = b;
  else if (b > max)
    max = b;
  if (c < min)
    min = c;
  else if (c > max)
    max = c;
  cout << max - min << endl;
  return 0;
}
```

Још једноставније је да се примене библиотечке функције за одређивање минимума и максимума.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int a, b, c;
  cin >> a >> b >> c;
  cout << max({a, b, c}) - min({a, b, c}) << endl;
  return 0;
}
```

Алтернативно, бројеве је могуће сместити у низ и тај низ можемо сортирати.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int a[3];
  cin >> a[0] >> a[1] >> a[2];
  sort(a, a + 3);
  cout << a[2] - a[0] << endl;
  return 0;
}
```

Задатак: Две домине

Аутор: Јелена Хаџи-Пурић, *Аутор решења*: Филип Марић

Такмичење: општинско 2018/2019, VI разред, 4. задатак

Зоран има две домине. На свакој домини су тачкицама представљене две цифре од 1 до 9. Напиши програм који одређује највећи број који Зоран може написати помоћу своје две домине (сваку од њих може окренути како жели и домине може поређати како жели).

Опис улаза

Први ред стандардног улаза садржи бројеве на првој домини (раздвојене размаком), а други ред садржи бројеве на другој домини.

Опис излаза

На стандардни излаз исписати тражени највећи број.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`2 3` `1 6`	`6 1 3 2`	`1 9` `3 4`	`9 1 4 3`	`6 2` `3 6`	`6 3 6 2`

Решење

Сваку од две домине је потребно окренути тако да већа цифра на тој домини буде испред мање (ако су цифре једнаке, свеједно је како се домина окреће). Редослед две домине се одређује тако да број записан на првој домини буде већи од броја записаног на другој.

Прву цифру сваке од две домине одређујемо као максимум две цифре на тој домини, а другу као њихов минимум. Два двоцифрена броја можемо упоредити и лексикографски: прва домина иде испред друге ако је њена прва цифра већа од прве цифре друге домине или су те цифре једнаке, а друга цифра прве домине је већа од друге цифре друге домине.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int d11, d12, d21, d22;
  cin >> d11 >> d12 >> d21 >> d22;
  // odredjujemo vecu i manju cifru na prvoj domini
  int m11 = max(d11, d12), m12 = min(d11, d12);
  // odredjujemo vecu i manju cifru na drugoj domini
  int m21 = max(d21, d22), m22 = min(d21, d22);
  // odredjujemo bolji redosled domina leksikografskim poredjenjem parova
  // (m11, m12) i (m21, m22)
  if (m11 > m21 || (m11 == m21 && m12 > m22))
    cout << m11 << " " << m12 << " " << m21 << " " << m22 << endl;
  else
    cout << m21 << " " << m22 << " " << m11 << " " << m12 << endl;
```

```
  return 0;
}
```

Једно могуће решење је и да експлицитно формирамо двоцифрене бројеве множењем прве цифре са 10 и додавањем друге, да их затим упоредимо и да тражени четвороцифрени број добијемо множењем већег двоцифреног са 100 и додавањем мањег (ово решење је исправно јер смо сигурни да се не јавља цифра 0). Када је број формиран, цифре можемо да издвојимо коришћењем целобројног дељења са 10.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  // ucitavamo cifre na dve domine
  int d11, d12, d21, d22;
  cin >> d11 >> d12 >> d21 >> d22;
  // veci dvocifreni broj koji se moze napraviti pomocu prve domine
  int d1 = d11 > d12 ? 10*d11 + d12 : 10*d12 + d11;
  // veci dvocifreni broj koji se moze napraviti pomocu druge domine
  int d2 = d21 > d22 ? 10*d21 + d22 : 10*d22 + d21;
  // najveci cetvorocifreni broj koji se moze napraviti od domina
  int d = d1 > d2 ? 100*d1 + d2 : 100*d2 + d1;
  // ispisujemo njegovu jednu po jednu cifru
  cout << (d / 1000) % 10 << " "
       << (d / 100) % 10 << " "
       << (d / 10) % 10 << " "
       << (d / 1) % 10 << endl;
  return 0;
}
```

Задатак: Деца

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 2, V разред, 2. задатак и VI разред 1. задатак

Сања има тачно онолико година колико и њено троје деце заједно. Дате су године Сање и њено двоје деце у произвољном редоследу. Одредити године трећег Сањиног детета.

Опис улаза

Са стандардног улаза се учитавају три цела позитивна броја који нису већи од 100, сваки у посебном реду. Један од бројева представља Сањине године, а остала два године двоје од њене деце.

Опис излаза

На стандардни излаз исписати један број, број година трећег Сањиног детета.

Пример 1

Улаз	*Излаз*
17 43 12	14

Пример 2

Улаз	*Излаз*
15 20 50	15

Решење

Од три учитана броја прво треба одредити највећи - то је број година маме Сање. Збир година два детета можемо да одредимо тако што од збира сва три броја одузмемо мамин број година. Коначно, године трећег детета добијамо када од маминог броја година одузмемо збир година два детета.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int a, b, c;
  cin >> a >> b >> c;
  int mama = max({a, b, c});
  int prva_dva = a + b + c - mama;
  int trece = mama - prva_dva;
  cout << trece << endl;
}
```

Задатак: Правоугаоник по квадрантима

Такмичење: окружно 2018/2019, VII разред, 2. задатак, VIII разред, 1. задатак

Аутор: Филип Марић

Дат је правоугаоник чије су ивице паралелне координатним осама. Напиши програм који одређује површину дела правоугаоника у сваком од четири квадранта и исписује највећу од њих.

Опис улаза

Са стандардног улаза уносе се координате доњег левог (x_1, y_1) и горњег десног (x_2, y_2) темена правоугаоника – четири цела броја између -100 и 100, при чему важи $x_1 < x_2$ и $y_1 < y_2$ (уносе се по два броја у сваком реду, раздвојени размаком).

Опис излаза

На стандардни излаз исписати тражену највећу површину.

Пример 1

Улаз	*Излаз*
-3 -5 2 4	15

Пример 2

Улаз	*Излаз*
-3 3 -1 5	4

Решење

Да бисмо одредили површине делова правоугаоника у разним квадрантима, потребно је да израчунамо дужину дела хоризонталне странице правоугаоника који лежи лево од y-осе (у негативном делу) и дужину дела који лежи десно од y-осе (у позитивном делу).

Ако је x_1 позитиван, онда је дужина "позитивног" дела хоризонталне странице $x_1 - x_2$, а ако је негативан, онда је дужина $x_2 - 0$ (ако је x_2 позитиван) или 0 ако је x_2 негативан. Сви се ови случајеви могу обухватити изразом $\max(x_2 - \max(x_1, 0), 0)$. Заиста, од десног краја x_2 се одузима леви крај који једнак $\max(x_1, 0)$ тј. или је x_1 или је 0 (ако је x_1 негативан). Ако је разлика $x_2 - \max(x_1, 0)$ позитивна, то значи да је x_2 позитивно и та разлика је жељени резултат, а у супротном је резултат 0 тј. резултат је једнак максимум ове разлике и броја 0.

Слично, дужину "негативног" дела странице се може израчунати као $\max(\min(x_2, 0) - x_1, 0)$.

Аналогно рачунамо и "позитиван" и "негативан" део вертикалне странице.

На основу тога лако израчунавамо површине делова правоугаоника у сва 4 квадранта (на пример, површина дела у првом квадранту је једнака производу дужина "позитивних" делова страница) и одређујемо њихов максимум (најједноставније библиотечком функцијом).

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int x1, y1, x2, y2;
  cin >> x1 >> y1 >> x2 >> y2;

  int posX = max(x2 - max(0, x1), 0);
  int negX = max(min(0, x2) - x1, 0);
  int posY = max(y2 - max(0, y1), 0);
  int negY = max(min(0, y2) - y1, 0);

  int P1 = posX * posY;
  int P2 = negX * posY;
  int P3 = negX * negY;
  int P4 = posX * negY;

  cout << max({P1, P2, P3, P4}) << endl;
  return 0;
}
```

Задатак: Мајице

Аутор: Милан Вуīелија

Такмичење: 2021/22. квалификације 1, VI разред, 3. задатак

Сваки учесник кампа треба да добије по једну мајицу. *A* учесника је рекло да им одговара мала или средња величина, а *B* учесника да им одговара средња или велика. На располагању већ

имамо S малих, M средњих и L великих мајица. Одредити колико мајица је потребно додатно поручити, да би их било довољно за све учеснике кампа.

Опис улаза

Цели неозначени бројеви A, B, S, M, L редом, сваки у посебном реду стандардног улаза. Ниједан од ових бројева није већи од 15.

Опис излаза

Један цео број, број мајица које треба поручити.

Пример 1		**Пример 2**		**Пример 3**		**Пример 4**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
5	1	5	3	5	5	5	0
6		6		6		6	
7		1		2		2	
1		1		2		9	
4		8		2		2	

Решење

Нека од A људи којима одговара мала или средња величина за њих A' има малих мајица (A' ће бити једнак мањем од бројева A, S). Број преосталих људи из ове групе означимо са $A'' = A - A'$ (овај број може да буде и 0). Слично, нека од B људи којима одговара средња или велика мајица за њих B' има великих мајица и нека је $B'' = B - B'$.

Ако је број преосталих људи у обе групе заједно мањи или једнак броју мајица средње величине ($A'' + B'' \leq M$), додатне мајице нису потребне и одговор је 0. У противном, одговор је $A'' + B'' - M$.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
    int a, b, s, m, l;
    cin >> a >> b >> s >> m >> l;
    a -= min(a, s);
    b -= min(b, l);
    int rez = max(a + b - m, 0);
    cout << rez << endl;
    return 0;
}
```

Задатак: Стубићи од коцкица

Аутор: Милан Вујделија

Такмичење: 2022/2023. квалификације 1, 8. разред, 2. задатак

Два друга се играју коцкицама. Од претходне игре су им остала два стуба чије су висине A и B. Сада, за нову грађевину они имају две идеје. По једној идеји потребни су им стубови висина A_1 и B_1, а по другој стубови висина A_2 и B_2. Да би променио висину неког стуба за 1, једноме од другара треба T_x времена, а другом T_y времена. За колико најмање времена другари могу да преправе (продуже или скрате) постојеће стубове и направе од њих стубове који се уклапају у једну од идеја, ако сваки од њих преправља по један стуб? Време се мери од тренутка када другари истовремено почну са преправкама, до тренутка када су обојица завршила.

Опис улаза

У четири реда стандардног улаза су по два приридна броја раздвојена размаком. У првом реду су A и B, у другом A_1 и B_1, у трећем A_2 и B_2, сви из интервала $[1, 1000000]$, а у четвртом T_x и T_y, из интервала $[1, 100]$.

Опис излаза

На стандардни излаз исписати један природан број, тражено најмање време.

Пример 1

Улаз	*Излаз*	*Објашњење*
`10 20`	`20`	Бржи од њих двојице може да промени стуб висине 10 у 28 за $18 \cdot 1 = 18$
`18 28`		јединица времена, а спорији може да промени стуб висине 20 у 18 за $2 \cdot 10 =$
`21 31`		20 јединица времена, тако да им је укупно довољно 20 јединица времена.
`1 10`		Брже уклапање у једну од две идеје није могуће.

Пример 2

Улаз	*Излаз*
`7 4`	`15`
`9 13`	
`14 12`	
`3 2`	

Решење

Решење набрајањем свих 8 могућности

Другари треба да донесу три одлуке:

- да ли праве стгубове висина висина A_1 и B_1 или A_2 и B_2
- који стуб се преправља у који
- ко од њих преправља стуб A, а ко стуб B

То даје укупно осам могућих начина рада. При сваком начину рада, време се добија као дуже од појединачних времена потребних другарима за преправку. Оптимално време је најкраће од тих осам времена.

```
#include <iostream>

using namespace std;

int main() {
    int a, b, a1, b1, a2, b2, tx, ty;
    cin >> a >> b >> a1 >> b1 >> a2 >> b2 >> tx >> ty;
    int t    = max(abs(a - a1)*tx, abs(b - b1)*ty);
```

```
    t = min(t, max(abs(a - a1)*ty, abs(b - b1)*tx));
    t = min(t, max(abs(a - b1)*tx, abs(b - a1)*ty));
    t = min(t, max(abs(a - b1)*ty, abs(b - a1)*tx));
    t = min(t, max(abs(a - a2)*tx, abs(b - b2)*ty));
    t = min(t, max(abs(a - a2)*ty, abs(b - b2)*tx));
    t = min(t, max(abs(a - b2)*tx, abs(b - a2)*ty));
    t = min(t, max(abs(a - b2)*ty, abs(b - a2)*tx));
    cout << t << endl;
    return 0;
}
```

Решење разматрањем подслучајева

Нека се један од стубова преправља за дужину P_1, а други за дужину P_2. Време потребно за преправљање једнако је већем од времена за сваку преправку појединачно. Ово време ће бити минимално ако бржи другар ради већу преправку. Без умањења општости, можемо да претпоставимо да је $T_x \leq T_y$ (ако није, можемо да разменимо вредности T_x и T_y). Сада потребно време можемо да изразимо као

$$Vreme(P_1, P_2) = \max((\min(P_1, P_2) \cdot T_y, \max(P_1, P_2) \cdot T_x).$$

У случају да се другари одлуче за преправку на A_1, B_1, они имају две могућности: или стуб висине A преправљају на висину A_1 а стуб висине B на висину B_1, или обрнуто. Од ове две могућности изабраће ону за коју је потребно мање времена и добиће да је оптимално време потребно за преправку по првој идеји

$$T_1 = \min(Vreme(|A - A_1|, |B - B_1|), Vreme(|A - B_1|, |B - A_1|))$$

На исти начин добијамо да је оптимално време потребно за преправку по другој идеји

$$T_2 = \min(Vreme(|A - A_2|, |B - B_2|), Vreme(|A - B_2|, |B - A_2|))$$

Коначно, одговор на питање из задатка добијамо као $\min(T_1, T_2)$.

```
#include <iostream>

using namespace std;

void Uredi(int& m, int& n) {
  if (m > n) { int pom = m; m = n; n = pom; }
}
int Vreme(int da, int db, int tx, int ty) {
    Uredi(da, db);
    return max(da * ty, db * tx);
}
int OptVreme(int a0, int b0, int ac, int bc, int tx, int ty) {
    int tc1 = Vreme(abs(a0 - ac), abs(b0 - bc), tx, ty);
    int tc2 = Vreme(abs(a0 - bc), abs(b0 - ac), tx, ty);
    return min(tc1, tc2);
}
```

```
int main() {
    int a, b, a1, b1, a2, b2, tx, ty;
    cin >> a >> b >> a1 >> b1 >> a2 >> b2 >> tx >> ty;
    Uredi(tx, ty);
    int rez1 = OptVreme(a, b, a1, b1, tx, ty);
    int rez2 = OptVreme(a, b, a2, b2, tx, ty);
    cout << min(rez1, rez2) << endl;
    return 0;
}
```

Задатак: Финалиста

Аутор: Милан Вујделија

Такмичење: окружно 2021/22., VI разред, 1. задатак

Једно полуфинале играју екипе A и B, а друго екипе C и D. Победници полуфиналних утакмица се састају у финалу. Рејтинзи екипа су различити, а сваку утакмицу добија екипа са већим рејтингом. Написати програм који учитава рејтинге ових екипа редом, а исписује ознаку екипе која је изгубила у финалу.

Опис улаза

На стандардном улазу се налазе 4 различита неозначена цела броја, сваки у посебном реду. Ови бројеви су редом рејтинзи екипа A, B, C и D и нису већи од 2000.

Опис излаза

На стандардни излаз исписати само једно од слова `A`, `B`, `C` или `D`.

Пример

Улаз	*Излаз*	*Објашњење*

`4`
`1`
`5`
`8`
	`A`	Финале су играле екипе A и D, а изгубила је екипа A.

Решење

Задатак можемо да решимо у две фазе: најпре одредимо рејтинг екипе која је изгубила у финалу, а затим пронађемо ознаку те екипе.

Ако већи од бројева A и B означимо са E, већи од бројева C и D означимо са F, а мањи од бројева E и F означимо са G, тада је G рејтинг тражене екипе.

У другој фази сваки од бројева A, B, C и D поредимо са G, и исписујемо ознаку оног који је једнак са G.

```
#include <iostream>

using namespace std;

int main() {
    int a, b, c, d;
```

```
    cin >> a >> b >> c >> d;
    int finalista = min(max(a, b), max(c, d));
    if (finalista == a) cout << "A" << endl;
    else if (finalista == b) cout << "B" << endl;
    else if (finalista == c) cout << "C" << endl;
    else if (finalista == d) cout << "D" << endl;
    return 0;
}
```

Задатак: Прозор и ограда

Аутор: Милан Вуīделија

Такмичење: 2021/22. квалификације 2, VIII разред, 2. задатак

У Хакатонској улици постоји једна дугачка ограда. Места на огради ћемо изражавати у метрима од почетка ограде. Кроз Петров прозор се види део ограде од a до b. Неко је у току ноћи обојио део ограде од c до d у зелено. Следећег јутра је у црвено обојен део ограде од e до f. Колико метара зелене ограде се након тога види са Петровог прозора?

Опис улаза

У првом реду стандардног улаза цели бројеви a и b раздвојени размаком. У другом реду стандардног улаза цели бројеви c и d раздвојени размаком. У трећем реду стандардног улаза цели бројеви e и f раздвојени размаком. Сви бројеви су цели, позитивни и нису већи од 10^7. При томе важи $a < b, c < d, e < f$.

Опис излаза

Један цео број, део ограде обојен у зелено који може да се види кроз Петров прозор, изражен у метрима.

Пример

Улаз	*Излаз*
10 20 5 15 12 17	2

Решење

Део ограде који је видљив кроз прозор може да се опише као неки интервал P. Делови обојени зеленом и црвеном бојом су такође интервали, означимо их са Z и C.

Део ограде који је током ноћи обојен у зелено и видљив кроз прозор је пресек интервала P и Z, $PZ = P \cap Z$, па је и сам интервал. Онај део овог интервала који је накнадно обојен црвено је његов пресек са интервалом C, $PZC = PZ \cap C$. Приметимо да један или оба од интервала PZ и PZC могу да буду и празни.

Коначно, дужина дела који је после оба бојења зелен и видљив кроз прозор једнака је разлици дужина интервала PZ и PZC.

```
#include <iostream>

using namespace std;
```

```
pair<int, int> Presek(pair<int, int>& a, pair<int, int>& b) {
    pair<int, int> rez;
    rez.first = max(a.first, b.first);
    rez.second = min(a.second, b.second);
    return rez;
}

int duzina(pair<int, int>& interval) {
    return max(0, interval.second - interval.first);
}

int main() {
    pair<int, int> prozor, zeleno, crveno, pz, pzc;
    cin >> prozor.first >> prozor.second;
    cin >> zeleno.first >> zeleno.second;
    cin >> crveno.first >> crveno.second;
    pz = Presek(prozor, zeleno);
    pzc = Presek(pz, crveno);
    cout << duzina(pz) - duzina(pzc) << endl;
   return 0;
}
```

Задатак: Шарко

Аутор: Милан Вутделија

Такмичење: 2021/22. квалификације 3, VIII разред, 2. задатак

Малени Коста заједно са својим псом Шарком чува три јагњета. Ливада је представљена бројевном правом, а јагњад су увек на међусобно различитим целобројним позицијама. Сваки пут када Шарко лане, једно од два крајња јагњета се бар мало примакне осталим. Формално, ако су позиције јагњади $a < b < c$, тада се или a повећа тако да остане различито од b и мање од c, или се c смањи тако да остане различито од b и веће од a. Ово се понавља док се јагањци не нађу на три узастопне позиције. После тога Шарко престаје да лаје и само маше репом.

Косту занима колико најмање, а колико највише пута ће Шарко да лане, пре него што се јагањци збију једно до другог.

Опис улаза

На стандардном улазу су три различита цела позитивна броја, сваки у посебном реду. Ови бројеви представљају почетне позиције јагњади и нису већи од 1000.

Опис излаза

У првом реду стандардног излаза исписати колико најмање пута Шарко може да лане. У другом реду стандардног излаза исписати колико највише пута Шарко може да лане.

Пример

Улаз	*Излаз*
7	1
2	3
6	

Решење

Задатак можемо лакше да решимо ако израчунамо најмању и највећу од три дате позиције, $L = \min(a, b, c), D = \max(a, b, c)$. Највећи број лајања Шарка добијамо ако се на свако лајање једно од крајњих јагњади помери ка средњем за само једно место. Ово је могуће увек, осим када су сва три јагњета једно до другог. Према томе, највећи број лајања је једнак броју празних места између јагњади, а то је $D - L - 2$.

Да бисмо одредили најмањи број лајања, потребно је да разликујемо неколико случајева. Прво, ако су на почетку сва три јагњета једно до другог, најмањи број лајања је нула. Друго, ако нису већ, јагњад могу после једног лајања да се нађу на три узастопне позиције ако је растојање између било која два јагњета једнако један или два. У свим осталим случајевима довољна су им два лајања (на прво лајање се два јагњета групишу, а на друго лајање им прође и преостало јагње).

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int a, b, c;
  cin >> a >> b >> c;
  int levo = min(min(a, b), c);
  int desno = max(max(a, b), c);
  int najvise = desno - levo - 2;
  int najmanje = -1;
  if (levo + 2 == desno)
    najmanje = 0;
  else if (abs(a - b) < 3 || abs(b - c) < 3 || abs(c - a) < 3)
    najmanje = 1;
  else
    najmanje = 2;
  cout << najmanje << endl;
  cout << najvise << endl;
  return 0;
}
```

Задатак: Из бајке у басну

Аутор: Милан Вуīделија

Такмичење: 2022/2023. квалификације 3, 6. разред, 2. задатак

Ислужени магарац, пас, мачка и петао су решили да побегну из познате бајке у неку басну у

којој имају пријатеље спремне да им помогну. Ауто којим одлазе из бајке је такође ислужен, па је важно да се оптерећење леве и десне стране не разликују превише, а исто важи за предњу и задњу страну. Подразумева се да на сваком од 4 седишта седи по једна животиња. Ако су познате масе све 4 животиње, која је најмања разлика оптерећења коју оне могу да гарантују и између леве и десне стране, и између предње и задње стране?

Опис улаза

У првом реду стандардног улаза је тежина магарца, у другом тежина пса, у трећем тежина мачке а у четвртом тежина петла. Сви бројеви су цели, позитивни и мањи од 500. Гарантује се да је најтежа животиња увек магарац или пас, а најлакша мачка или петао.

Опис излаза

На стандардни излаз исписати један цео неозначен број, разлику оптерећења коју животиње могу да гарантују и између предњег и задњег реда, и између леве и десне стране.

Пример 1

Улаз	*Излаз*	*Објашњење*
170 60 20 6	124	Ако, на пример, магарац седи на месту возача (предње лево седиште), петао до њега, мачка иза магарца, а пас до мачке и иза петла, онда је разлика између оптерећења предњег и задњег реда 176-80=96, а између леве и десне стране 190-66=124. Одавде се види да животиње могу да гарантују да разлика оптерећења неће прећи 124 ни између леве и десне, ни између предње и задње стране. При томе не постоји распоред седења којим би се постигло да су обе ове разлике мање од 124.

Пример 2

Улаз	*Излаз*
85 90 14 15	6

Решење

Нека су тежине животиња a, b, c, d редом од најлакше до најтеже. Пошто је пар a, b најлакши, а пар c, d најтежи, највећа разлика између тежина парова је $(d + c) - (a + b)$. Ову разлику треба избећи, па животиња тежине c не треба да седи ни поред ни испред/иза животиње тежине d, већ дијагонално од ње. Преостале две разлике су $|(d + a) - (c + b)|$ и $(d + b) - (c + a)$. Разлика која може да се гарантује једнака је већој од ове две, а може се доказати да је то увек разлика $(d + b) - (c + a)$ (размислите зашто).

Поменимо још да се вредности a, b, c, d из ове анализе једноставно одређују, као што се види из програма – решења.

```
#include <iostream>

using namespace std;

int main() {
    int magarac, pas, macka, petao;
    cin >> magarac >> pas >> macka >> petao;
    int a = min(macka, petao);
```

```
    int b = max(macka, petao);
    int c = min(pas, magarac);
    int d = max(pas, magarac);
    cout << (d+b)-(c+a) << endl;
    return 0;
}
```

Задатак: Усељавање

Аутори: Иван Дрецун, Владимир Кузмановић

Такмичење: окружно 2022/2023, VIII разред, 1. задатак

Душан се усељава у нови стан и жели да донесе своје ствари у кутијама. Познат је план стана који је правоугаоног облика. У стану постоји један стуб који је у плану такође правоугаоног облика. Потребно је одредити максималне димензије правоугаоне кутије тако да се кутија може кретати са свих страна стуба. Странице свих правоугаоника су паралелне осама и кутија неће бити ротирана. Напиши програм који за дати план стана помаже Душану да одреди димензије кутије.

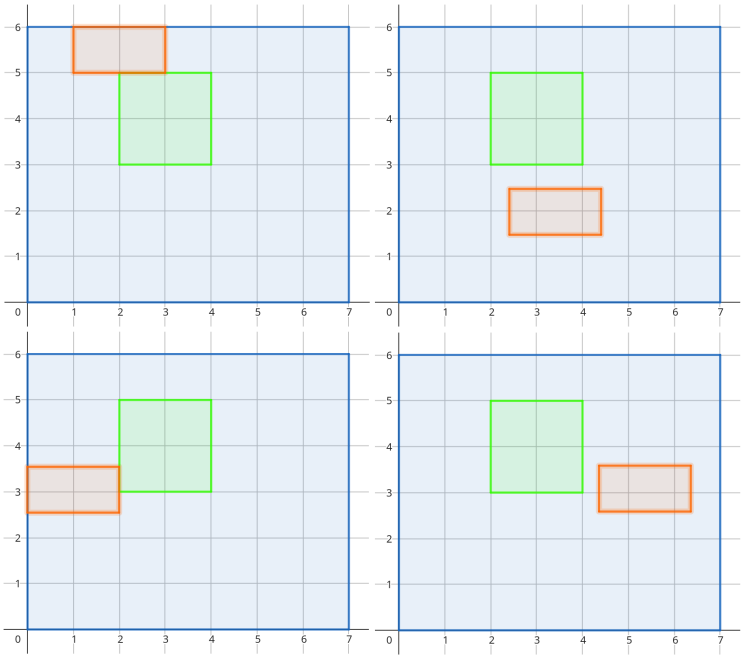
Опис улаза

Са стандардног улаза се уноси шест целих бројева. Прва два представљају дужину и ширину стана. Наредна два представљају дужину и ширину стуба. Последња два представљају удаљености стуба од левог и доњег зида стана, тим редом (растојања су увек већа од нуле). Сваки број се уноси у засебном реду.

Опис излаза

На стандардни излаз исписати два цела броја који редом представљају дужину и ширину кутије. Бројеве исписати у истом реду, одвојене размаком.

Пример 1

Улаз	*Излаз*	*Објашњење*
7 6 2 2 2 3	2 1	

Пример 2

Улаз	*Излаз*
6 6 3 3 2 1	1 1

Пример 3

Улаз	*Излаз*
6 5 3 1 2 2	1 2

Решење

Нека су са $standD$, $standS$, $stubD$, $stubS$, $stubX$ и $stubY$ редом обележени дужина и ширина стана, дужина и ширина стуба и удаљеност стуба од левог и удаљеност стуба од доњег зида.

Максималну дужину кутије d ћемо наћи као минимум две вредности. Прва вредност је растојање стуба од левог зида ($stubX$), а друга вредност је растојање од десне ивице стуба до десног зида ($standD - stubX - stubD$). Дакле, максималну дужину кутије добијамо на следећи начин: $d = min(stubX, standD - stubX - stubD)$.

На сличан начин одређујемо и максималну ширину кутије s. Потребно је да нађемо минимум растојања од доњег зида до стуба и од горње ивице стуба до горњег зида. Максималну ширину кутије добијамо на следећи начин: $s = min(stubY, standS - stubY - stubS)$.

На крају, само треба да прикажемо вредности d и s.

Описани поступак је приказан у решењу.

```
#include <iostream>
```

```
using namespace std;

int main() {
  int stanD, stanS, stubD, stubS, stubX, stubY;
  cin >> stanD >> stanS >> stubD >> stubS >> stubX >> stubY;

  int d = min(stubX, stanD - stubX - stubD);
  int s = min(stubY, stanS - stubY - stubS);

  cout << d << ' ' << s << endl;
  return 0;
}
```

Задатак: Гусари

Аутор: Милан Вујделија

Такмичење: окружно 2022/2023, VIII разред, 2. задатак

Од G љутих гусара, током разних жестоких борби X њих је изгубило једно око, U једно уво, R једну руку, а N једну ногу. Колико је најмање, а колико највише гусара који су изгубили и око и уво и руку и ногу?

Опис улаза

На стандардном улазу су редом цели бројеви G, X, U, R, N, сваки у посебном реду. За ове бројеве важи $1 \leq G \leq 50\,000, 0 \leq X \leq G, 0 \leq U \leq G, 0 \leq R \leq G, 0 \leq N \leq G$.

Опис излаза

На стандарни излаз исписати два цела неозначена броја, сваки у посебном реду. У првом реду исписати најмањи могући број тражених гусара, а у другом њихов највећи могући број.

Пример 1

Улаз	*Излаз*
15	2
12	10
14	
11	
10	

Пример 2

Улаз	*Излаз*
15	15
15	15
15	
15	
15	

Решење

Нека је m најмањи од бројева x, u, r, n. Могуће је да је истих m гусара изгубило и око и уво и руку и ногу, али није могуће да таквих има више од m. Зато је навећи број гусара са све 4 повреде управо $m = min(x, u, r, n)$.

Оба ока има $x_1 = g - x$ гусара, оба ува $u_1 = g - u$, обе руке $r_1 = g - r$, а обе ноге $n_1 = g - n$ гусара. Да би гусара са све 4 повреде било што мање, треба да међу оних x_1, u_1, r_1 и n_1 који немају бар по једну повреду да буде што више различитих. Ако је $x_1 + u_1 + r_1 + n_1 < g$ онда могу сви они да буду различити, а преосталих $g - (x_1 + u_1 + r_1 + n_1)$ гусара би имали све 4 повреде, и то би био најмањи могућ број таквих гусара. У противном, тј. ако је $x_1 + u_1 + r_1 +$

$n_1 \geq g$, могуће је да је сваки од гусара избегао бар једну побреду, па је најмањи могућ број оних са све 4 повреде једнак 0.

На пример, ако су g, x, u, r, n редом једнаки $15, 12, 14, 11, 10$, распоред повреда може да буде

```
               1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
12 без ока     +  +  +  +  +  +  +  +  +  +  +  +  .  .  .
14 без ува     +  +  +  +  +  +  +  +  +  +  +  .  +  +  +
11 без руке    +  +  +  +  +  +  +  .  .  .  .  +  +  +  +
10 без ноге    +  +  .  .  .  .  .  +  +  +  +  +  +  +  +
```

где тачкице означавају гусаре без повреде, а плусеви оне са повредом. Видимо да чак и када су све тачкице у различитим колонама (тј. сви гусари без неке повреде различити) морају да постоје бар две колоне са свим плусевима, тј. два гусара са свим повредама.

Истакнимо још једном закључак ове анализе: ако је $x_1 + u_1 + r_1 + n_1 < g$, најмањи могућ број гусара са све 4 повреде је $g - (x_1 + u_1 + r_1 + n_1)$, у противном најмањи могућ број гусара са све 4 повреде је 0.

```cpp
#include <iostream>

using namespace std;

int main() {
    int g, x, u, r, n;
    cin >> g >> x >> u >> r >> n;
    int najvise = min(min(x,u),min(r,n));
    int najmanje = max(0, g - ((g-x) + (g-u) + (g-r) + (g-n)));
    cout << najmanje << endl;
    cout << najvise << endl;
    return 0;
}
```

Задатак: Највећа предност домаћих

Аутор: Милан Вујделија

Такмичење: 2023/2024. квалификације 1, VI разред, 2. задатак

Дат је резулатат кошаркашке утакмице након сваке четвртине. Колика је највећа могућа предност домаћих?

Опис улаза

У сваком од четири реда стандардног улаза налазе се по два цела ненегативна броја раздвојена размаком. Сваки ред улаза одговара резултату након једне четвртине редом. Први број у реду је број поена које је постигла домаћа екипа, а други број је број поена које је постигла гостујућа екипа. Ниједан од бројева на улазу није већи од 200.

Опис излаза

На стандардни излаз исписати само један цео број, највећу могућу предност домаће екипе. Ако домаћа екипа ни у једном тренутку није могла да има предност, исписати 0.

Пример 1

Улаз	*Излаз*	*Објашњење*

`21 18`
`39 40`
`63 54`
`78 77`
	`24`	Резултат је у једном тренутку могао да буде $78 : 54$, што је 24 поена предности.

Пример 2

Улаз	*Излаз*	*Објашњење*

`0 1`
`1 3`
`2 5`
`4 9`
	`0`	Домаћа екипа никада није имала предност. Најповољнији резулатат за њу је био на почетку утакцмице $(0 : 0)$.

Решење

Нека су резултати по четвртинама редом $a_1 : b_1$, $a_2 : b_2$, $a_3 : b_3$ и $a_4 : b_4$. Најповољнији резултат за домаћу екипу током прве четвртине је могао да буде $a_1 : 0$, док су најповољнији могући резултати током друге, треће и четврте четвртине редом $a_2 : b_1$, $a_3 : b_2$, $a_4 : b_3$. Ови резултати настају ако током сваке четвртине прво домаћа екипа постигне све своје поене, а затим гостујућа све своје.

Према томе, највећа вођства или најмањи заостаци домаћих у поенима по четвртинама су a_1, $a_2 - b_1$, $a_3 - b_2$, $a_4 - b_3$. Тражени резултат је највећи од ова четири броја. Приметимо да највећи од ових бројева не може да буде негативан, јер је $a_1 \geq 0$.

```
#include <iostream>

using namespace std;

int main() {
    int a1, b1, a2, b2, a3, b3, a4, b4;
    cin >> a1 >> b1 >> a2 >> b2 >> a3 >> b3 >> a4 >> b4;
    int prednost = max(max(a1, a2-b1), max(a3-b2, a4-b3));
    cout << prednost << endl;
    return 0;
}
```

Глава 5

Сортирање малих серија

Задатак: Кутије

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 2, V разред, 3. задатак, и VI, VII и VIII разред, 2. задатак

Дате су две кутије за ципеле облика квадра. Свака кутија је дата својом дужином, ширином и висином. Испитати да ли се нека од ових кутија може убацити у другу. Једна кутија се може убацити у другу ако се може окренути тако да јој одговарајуће димензије буду строго мање од одговарајућих димензија друге кутије.

Опис улаза

У првом реду стандардног улаза налазе се три цела броја d_1, s_1 и v_1 раздвојени по једним размаком, дужина, ширина и висина прве кутије. У другом реду стандардног улаза налазе се три цела броја d_2, s_2 и v_2 такође раздвојени по једним размаком, дужина, ширина и висина друге кутије. Сви бројеви су позитивни и мањи од 1000.

Опис излаза

На стандардни излаз исписати `DA` ако се било која од кутија може убацити у другу, а `NE` иначе.

Пример 1

Улаз	*Излаз*	*Објашњење*
`5 9 2` `3 6 10`	`DA`	Ако другу кутију окренемо да јој висина постане ширина, ширина постане дужина, а дужина висина, моћи ћемо да убацимо прву кутију у другу.

Пример 2

Улаз	*Излаз*	*Објашњење*
`3 4 5` `3 4 5`	`NE`	Странице морају бити строго мање.

Пример 3

Улаз	*Излаз*	*Објашњење*
`3 4 5` `2 3 4`	`DA`	Кутије су већ окренуте тако да се друга може убацити у прву.

Решење

Нека су $a_1 \leq b_1 \leq c_1$ димензије прве, а $a_2 \leq b_2 \leq c_2$ димензије друге кутије. Да не бисмо проверавали све могуће оријентације кутија, доказаћемо да прва кутија може да стане у другу ако и само ако је $a_1 \leq a_2, b_1 \leq b_2, c_1 \leq c_2$. Смер $\Leftarrow$ овог тврђења је очигледан, па је довољно доказати смер $\Rightarrow$. Претпоставимо да нису испуњена сва три услова $a_1 \leq a_2, b_1 \leq b_2, c_1 \leq c_2$.

Ако није $a_1 \leq a_2$, онда $c_1 \geq b_1 \geq a_1 \geq a_2$, па ни једна димензија прве кутије није мања од a_2 и прва кутија не може да стане у другу.

Ако није $b_1 \leq b_2$, онда $c_1 \geq b_1 \geq b_2 \geq a_2$, па пошто не могу обе ивице c_1 и b_1 да се поставе паралелно са c_2, то прва кутија ни у овом случају не може да стане у другу.

Ако није $c_1 \leq c_2$, онда $c_1 \geq c_2 \geq b_2 \geq a_2$, па c_1 није мање ни од једне стране друге кутије и поново прва кутија не може да стане у другу. Овим је доказ завршен.

Према томе, да бисмо проверили да ли прва кутија може да стане у другу, довољно је да након сортирања ивица сваке кутије проверимо да важе сва три услова $a_1 \leq a_2, b_1 \leq b_2, c_1 \leq c_2$ (ако прва кутија не може да стане у другу када су овако окренуте, онда не може да стане ни на који начин).

Наравно, из истог разлога важи да друга кутија може да стане у прву само ако је $a_2 \leq a_1, b_2 \leq b_1, c_2 \leq c_1$.

На питање из задатка дајемо потврдан одговор било да је $a_1 \leq a_2, b_1 \leq b_2, c_1 \leq c_2$ или $a_2 \leq a_1, b_2 \leq b_1, c_2 \leq c_1$, а ако не једна од од ових тројки услова није испуњена, одговор ће бити одречан.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int a[3];
  int b[3];
  cin >> a[0] >> a[1] >> a[2];
  cin >> b[0] >> b[1] >> b[2];
  sort(a, a+3);
  sort(b, b+3);
  if ((a[0] < b[0] && a[1] < b[1] && a[2] < b[2]) ||
      (b[0] < a[0] && b[1] < a[1] && b[2] < a[2]))
    cout << "DA" << endl;
  else
    cout << "NE" << endl;
  return 0;
}
```

Задатак: Ризико

Аутор: Филип Марић

Такмичење: 2019/20 квалификације 3, V и VI разред, 3. задатак и VII и VIII разред, 1. задатак

Током игре ризико нападачки и одбрамени играч улазе у битке бацајући коцкице. Сваки од њих има 3 коцкице а на основу тренутног стања игре одређује се колико од њих сме да баци. Када се баце коцкице, упоређују се најјача коцкица нападача (она са највећим бројем) са најачом коцкицом одбрамбеног играча, затим друга по јачини са другом по јачини и најмања са најмањом. Ако је неки играч бацио мање коцкица, тада се најслабије коцкице оног другог играча занемарују (онај ко је бацао више коцкица је у предности). Када се две коцкице упоређују, ако нападач има строго јачу коцкицу (коцикцу на којој је број строго већи) одбрана губи један поен, док у супротном напад губи један поен (тиме је одбрамбени играч у благој предности). Напиши програм који на основу резултата бацања коцкица одређује број поена које губе један и други играч.

Опис улаза

Први ред стандардног улаза садржи број бачених коцкица нападача (1, 2 или 3), а наредни ред садржи бројеве (од 1 до 6) на коцкицама које је нападач бацио. Наредни ред стандардног улаза садржи број бачених коцкица одбрамбеног играча (1, 2 или 3), а наредни ред садржи бројеве (од 1 до 6) на коцкицама које је одбрамбени играч бацио.

Опис излаза

На стандардни излаз исписати број поена које губи нападач и број поена које губи одбрамбени играч, раздвојене једним размаком.

Пример 1

Улаз	*Излаз*	*Објашњење*

`3`
`5 6 3`
`3`
`5 3 4`
	`1 2`	Нападач 6 се пореди са одбраном 5, нападач 5 са одбраном 4 и нападач 3 са одбраном 3. У прве две борбе побеђује нападач, па одбрана губи два поена, док у трећој побеђује одбрана (јер нападач није добио строго већи број), па нападач губи 1 поен.

Пример 2

Улаз	*Излаз*	*Објашњење*

`3`
`1 4 6`
`2`
`5 5`
	`1 1`	Нападач 6 се пореди са одбраном 5, нападач 4 са одбраном 5, док се нападач 1 занемарује. У првој борби побеђује нападач, па одбрана губи поен, док у другој побеђује одбрана, па нападач губи поен.

Решење

Задатак решавамо тако што сортирамо низ коцкица нападача и низ коцкица одбрамбеног играча опадајуће и онда редом поредимо једну по једну коцкицу од почетка низа све док се неки од низова не исцрпи и на одговарајући начин ажурирамо бројаче изгубљених фигура за играча који напада и играча који се брани.

```
#include <iostream>
#include <algorithm>
#include <functional>

using namespace std;

int main() {
  int n_napad;
  cin >> n_napad;
```

```
  int napad[3];
  for (int i = 0; i < n_napad; i++)
    cin >> napad[i];

  int n_odbrana;
  cin >> n_odbrana;
  int odbrana[3];
  for (int i = 0; i < n_odbrana; i++)
    cin >> odbrana[i];

  sort(napad, napad + n_napad, greater<int>());
  sort(odbrana, odbrana + n_odbrana, greater<int>());

  int gubi_napad = 0, gubi_odbrana = 0;
  for (int i = 0; i < min(n_napad, n_odbrana); i++)
    if (napad[i] > odbrana[i])
      gubi_odbrana += 1;
    else
      gubi_napad += 1;

  cout << gubi_napad << " " << gubi_odbrana << endl;

  return 0;
}
```

Задатак: Прекидач

Аутор: Милан Вугделија

Такмичење: 2019/20 општинско, V разред, 4. задатак

Прекидач укључује сијалицу код степеништа када се на њега притисне, а сијалица се аутоматски искључује 60 секунди након последњег притиска на прекидач. Колико секунди је светлела сијалица ако је прекидач притиснут у тренуцима T_1, T_2 и T_3?

Опис улаза

Са стандардног улаза се учитавају цели позитивни бројеви T_1, T_2 и T_3, сваки у посебном реду, $1 \leq T_1, T_2, T_3 \leq 1\,000$, времена притисака на прекидач изражена у секундама протеклим од неког тренутка у прошлости.

Опис излаза

На стандардни излаз испишите један природан број, укупно време светљења сијалице изражено у секундама.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
158	157	11	97	3	180
300		25		100	
121		48		182	

Решење

Задатак је лакше решити ако преуредимо временске тренутке T_1, T_2 и T_3 тако да важи $T_1 \leq T_2 \leq T_3$. Укупно време светљења можемо да разложимо на три дела. У сваком делу светљење почиње притиском на прекидач, а завршава се или следећим притиском на прекидач или истеком 60 секунди, у зависности од тога шта се пре догоди. Тако је трајање првог дела једнако мањем од бројева $(T_2 - T_1)$ и 60, трајање другог дела мањем од бројева $(T_3 - T_2)$ и 60, а трајање трећег дела је увек 60. Сортирање времена је најједноставније извршити коришћењем библиотечке функције.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main()  {
  int t[3];
  cin >> t[0] >> t[1] >> t[2];
  sort(t, t+3);
  cout << (min(60, t[1] - t[0]) + min(60, t[2] - t[1]) + 60) << endl;
  return 0;
}
```

Сортирање је могуће извршити и ручно, поређењем и разменом елемената.

```
#include <iostream>

using namespace std;

int main()  {
  int t1, t2, t3;
  cin >> t1 >> t2 >> t3;

  if (t1 > t2) swap(t1, t2);
  if (t2 > t3) swap(t2, t3);
  if (t1 > t2) swap(t1, t2);

  cout << (min(60, t2 - t1) + min(60, t3 - t2) + 60) << endl;
  return 0;
}
```

Задатак: Одсутна

Аутор: Милан Вугделија

Такмичење: 2019/20 општинско, VII и VIII разред, 1. задатак

Ана, Бранка, Весна и Гоца су четири сестре. Ана је виша од Бранке исто колико Бранка од Весне, као и Весна од Гоце. Ана и још две од сестара су биле на систематском прегледу, а четврта није. Одредити висину одсутне сестре ако су познате висине остале три.

Опис улаза

У сваком од три реда стандардног улаза по један цео број x $(85 \leq x \leq 200)$, висине трију сестара у центиметрима, међу којима је и Анина. Подаци су дати у произвољном редоследу, али тако да увек постоји јединствено целобројно решење.

Опис излаза

Један цео број, висина четврте сестре у центиметрима.

Пример

Улаз	*Излаз*
176 146 156	166

Решење

Задатак се знатно лакше решава ако прво уредимо три дате висине a, b, c, од највеће до најмање. Након уређивања, разлику првог и другог броја означимо са r_1, а разлику другог и трећег са r_2. Могућа су три случаја. У случају да недостаје Гоцина висина, вредности r_1 и r_2 ће бити једнаке, а Гоцина висина је $c - r_1$; ако недостаје Веснина висина, r_2 ће бити двоструко веће од r_1, а Веснину висину можемо израчунати као $b - r_1$ или $c + r_1$; у преосталом случају недостаје Бранкина висина и тада је r_1 двоструко веће од r_2, а Бранкина висина се израчунава као $a - r_2$. Који је случај наступио, лако можемо да установимо поредећи вредности r_1 и r_2.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int visina[3];
  cin >> visina[0] >> visina[1] >> visina[2];
  sort(visina, visina+3);

  int r1 = visina[2] - visina[1];
  int r2 = visina[1] - visina[0];

  if (r1 == r2)
    cout << visina[0] - r1 << endl;
  else if (r1 == 2 * r2)
    cout << visina[2] - r2 << endl;
  else
    cout << visina[1] - r1 << endl;

  return 0;
}
```

Висине можемо уредити и ручним поређењем и разменом.

```
#include <iostream>

using namespace std;
```

```
int main() {
  int a, b, c;
  cin >> a >> b >> c;

  if (a < b) swap(a, b);
  if (b < c) swap(b, c);
  if (a < b) swap(a, b);

  int r1 = a - b;
  int r2 = b - c;

  if (r1 == r2)
    cout << c-r1 << endl;
  else if (r1 == 2 * r2)
    cout << a - r2 << endl;
  else
    cout << b - r1 << endl;

  return 0;
}
```

Задатак: По три

Аутор: Филип Марић

Такмичење: 2021/22. квалификације 3, V разред, 2. задатак

Златокоса је три пута долазила у кућу у којој су живели тата медвед, мама медвед и њихова три медведића. Сваки пут када би ушла у кућу, нашла би пет пуних тањира каше (мама медвед је сваки дан постављала ручак на потпуно исти начин), а она је јела кашу из нека три од њих. Први дан је појела кашу из три тањира у којима је било најмање каше, други дан је појела кашу из три тањира у којима је било највише каше, а трећи дан је појела кашу из три средња тањира (није појела кашу из тањира у којем је било најмање и у којем је било највише каше).

Опис улаза

Са стандардног улаза се учитава 5 целих бројева између 10 и 100, који представљају количину каше у пет тањира (сваки број је наведен у посебном реду).

Опис излаза

На стандардни излаз исписати укупну количину каше коју је Златокоса појела првог, другог и трећег дана.

Пример

Улаз	*Излаз*
20 30 50 10 40	60 120 90

Решење

Кључни корак у решењу је да се чиније са кашом сортирају по количини каше коју садрже. То је најлакше урадити смештањем бројева у петочлани низ коришћењем библиотечке функције. У језику могуће је користити функцију `sort`. Након тога исписујемо збир прва три елемента низа, затим последња три елемента низа и збир средња три елемента низа.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int a[5];
  cin >> a[0] >> a[1] >> a[2] >> a[3] >> a[4];
  sort(a, a+5);
  cout << a[0] + a[1] + a[2] << endl;
  cout << a[2] + a[3] + a[4] << endl;
  cout << a[1] + a[2] + a[3] << endl;
  return 0;
}
```

Задатак: Збир средњих

Аутор: Филип Марић

Такмичење: државно 2021/22., V и VI разред, 2. задатак

Златокоса је улазила у кућице медведића и увек је од три каше бирала средњу. Напиши програм који одређује колико је каше Златокоса укупно појела.

Опис улаза

Са стандардног улаза се уноси број кућица n ($1 \leq n \leq 100$), а затим у наредних n редова по три различита природна броја (између 1 и 100) која представљају редом количину каше у три чиније које је Златокоса затекла (нажалост, она не зна у којој чинији се налази најмања, у којој највећа, а у којој средња количина каше).

Опис излаза

На стандардни излаз исписати укупну количину каше коју је Златокоса појела.

Пример

Улаз	*Излаз*	*Објашњење*

`3`
`5 1 7`
`3 4 8`
`3 1 2`
	`11`	У првој кући Златокоса је појела 5, у другој 4, а у трећој 2.

Решење

Библиотечко сортирање

Кључни део задатка је одређивање средње од три вредности. Најједноставнији начин је да се сва три учитана броја сместе у трочлани низ који се затим сортира библиотечком функцијом. Током извршавања петље сабирамо средње вредности тако сортираних низова.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int n;
  cin >> n;
  int zbir = 0;
  for (int i = 0; i < n; i++) {
    int a[3];
    cin >> a[0] >> a[1] >> a[2];
    sort(a, a+3);
    zbir += a[1];
  }
  cout << zbir << endl;
  return 0;
}
```

Ручно сортирање

Три променљиве се могу сортирати и ручно. За почетак у прву променљиву доводимо најмању од три вредности (тако што упоредимо прву и другу, а затим и прву и трећу променљиву, размењујући им вредности ако су у лошем поретку). Након тога у другу променљиву доводимо средишњу вредност тако што упоредимо другу и трећу променљиву и разменимо им вредност ако је потребно.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int n;
  cin >> n;
  int zbir = 0;
  for (int i = 0; i < n; i++) {
    int a, b, c;
    cin >> a >> b >> c;
    if (a > b)
      swap(a, b);
    if (a > c)
      swap(a, c);
```

```
    if (b > c)
      swap(b, c);
    zbir += b;
  }
  cout << zbir << endl;
  return 0;
}
```

Задатак: Најдаљи најближи

Аутор: Милан Вујделија

Такмичење: 2023/2024. квалификације 1, VII разред, 4. задатак

Написати програм који од 4 учитана цела броја исписује онај, коме је растојање до најближег од осталих бројева максимално. Ако има више таквих бројева, исписати их све, редом по величини од најмањег од највећег.

Опис улаза

На стандардном улазу се налазе четири цела броја из интервала $[-1000000, 1000000]$, сваки посебном реду.

Опис излаза

На стандардни излаз исписати оне од учитаних бројева који испуњавају описани услов, сваки у посебном реду.

Пример 1

Улаз	*Излаз*
7	7
2	
4	
1	

Пример 2

Улаз	*Излаз*
4	2
2	4
7	7
9	9

Пример 3

Улаз	*Излаз*
4	1
1	9
6	
9	

Решење

Задатак се лакше решава ако најпре уредимо 4 учитана броја a, b, c, d по величини. Након што постигнемо да важи $a \leq b \leq c \leq d$, можемо лако за сваки од њих да одредимо даљину до најближег од преостала три:

- $a_m = b - a$
- $b_m = \min(b - a, c - b)$
- $c_m = \min(c - b, d - c)$
- $d_m = d - c$

Нека је $m = max(a_m, b_m, c_m, d_m)$. Потребно је још исписати сваки од бројева за који је растојање од њега до најближег од осталих једнако m.

```
#include <iostream>
#include <algorithm>

using namespace std;
```

```
int main() {
    int a, b, c, d;
    cin >> a >> b >> c >> d;
    if (a>b) swap(a, b);
    if (b>c) swap(b, c);
    if (c>d) swap(c, d);

    if (a>b) swap(a, b);
    if (b>c) swap(b, c);
    if (a>b) swap(a, b);
    int doA = b - a;
    int doB = min(b - a, c - b);
    int doC = min(c - b, d - c);
    int doD = d - c;
    int dMax = max(max(doA, doB), max(doC, doD));
    if (dMax == doA) cout << a << endl;
    if (dMax == doB) cout << b << endl;
    if (dMax == doC) cout << c << endl;
    if (dMax == doD) cout << d << endl;
    return 0;
}
```

Сортирање се још једноставније спроводи ако су бројеви учитани у низ.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
    int a[4];
    cin >> a[0] >> a[1] >> a[2] >> a[3];
    sort(a, a+4);
    int doA = a[1] - a[0];
    int doB = min(a[1] - a[0], a[2] - a[1]);
    int doC = min(a[2] - a[1], a[3] - a[2]);
    int doD = a[3] - a[2];
    int dMax = max({doA, doB, doC, doD});
    if (dMax == doA) cout << a[0] << endl;
    if (dMax == doB) cout << a[1] << endl;
    if (dMax == doC) cout << a[2] << endl;
    if (dMax == doD) cout << a[3] << endl;
    return 0;
}
```

Глава 6

Основни итеративни алгоритми за обраду серија елемената

Задатак: Мерење са три тега

Аутор: Милан Вуīделија

Такмичење: окружно 2022/2023, VI разред, 2. задатак

На двостраној ваги (теразијама), предмет се мери тако што се стави на једну страну ваге, а затим се тегови познатих маса стављају на обе стране док се не постигне равнотежа. Тада је збир маса на једној страни једнак збиру маса на другој страни, па маса предмета може да се израчуна.

На пример, ако је вага у равнотежи када се на страни са предметом налази тег од 2Kg, а на супротној страни тег од 5Kg, закључујемо је маса предмета $X = 5\text{Kg} - 2\text{Kg} = 3\text{Kg}$.

Написати програм који одређује најмању масу која може да се измери једним мерењем на двостраној ваги, ако су нам на располагању три тега познатих маса. Приликом мерења могу да се користе један, два или сва три тега, а сваки тег може да се стави на једну или другу страну.

Опис улаза

На стандардном улазу се налазе три цела неозначена броја, сваки у посебном реду, масе тегова. Бројеви припадају интервалу $[1, 10000]$.

Опис излаза

На стандардни излаз исписати један цео позитиван број, тражену најмању масу која може да се измери.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
2	2	5	1	3	2
5		12		12	
7		4		7	

Решење

Претпоставимо да је $a > b$. Стављањем тега масе a на једну, а мерене масе и тега масе b на другу страну, можемо да измеримо масу $a - b$. Слично, ако је $a < b$, можемо да измеримо масу $b - a$. У сваком случају, ако је $a \neq b$, стављањем тегова маса a и b на различите стране теразија, можемо да измеримо масу $|a - b|$.

Јасно је да нема сврхе да се два или сва три тега ставе на једну страну ваге, а ниједан тег на супротну страну, јер то сигурно није најмања маса која може да се измери (мања маса може да се измери остављањем само једног од тих тегова на првој страни).

Набрајањем могућих распореда тегова долазимо до тога да су кандидати за решење величине $a, b, c, |a - b|, |b - c|, |c - a|, |a + b - c|, |b + c - a|, |c + a - b|$.

Решење ће бити најмања од ових величина, која није једнака нули.

Један начин да нађемо намању позитивну од наведених величина је да употребимо неколико наредби гранања.

```
#include <iostream>

using namespace std;

int main() {
    int a, b, c;
    cin >> a >> b >> c;
    int m = min(a, min(b, c));

    int r1 = abs(a-b);
    int r2 = abs(b-c);
    int r3 = abs(c-a);
    int r4 = abs(a+b-c);
    int r5 = abs(b+c-a);
    int r6 = abs(c+a-b);

    if (r1 > 0 && r1 < m)
        m = r1;
    if (r2 > 0 && r2 < m)
        m = r2;
    if (r3 > 0 && r3 < m)
        m = r3;
    if (r4 > 0 && r4 < m)
```

```
        m = r4;
    if (r5 > 0 && r5 < m)
        m = r5;
    if (r6 > 0 && r6 < m)
        m = r6;

    cout << m << endl;
    return 0;
}
```

Набројане изразе који су кандидати за решење можемо и да сместимо у торку или листу, а затим да помоћу петље за сваку од тих вредности проверимо да ли је позитивна и мања од најмање пронађене до тад.

```
#include <iostream>

using namespace std;

int main() {
    int a, b, c;
    cin >> a >> b >> c;
    int m = min(a, min(b, c));
    int razlike[] = {
        abs(a+b-c), abs(b+c-a), abs(c+a-b),
        abs(a-b), abs(b-c), abs(c-a)
    };
    for (int r : razlike)
        if (r > 0 && r < m)
            m = r;

    cout << m << endl;
    return 0;
}
```

Задатак: Шљиве

Аутор: Милан Вујделија

Такмичење: 2022/2023. квалификације 2, 6. разред, 1. задатак

Јанко и Наташа су јели шљиве. Из чиније су узимали истовремено, Јанко по две, а Наташа по три шљиве, све док је то било могуће, тј. док није остало мање од пет шљива. Преостале шљиве (ако их је било) појео је касније Марко. Написати програм који учитава број шљива на почетку, а исписује колико је ко појео, као и број оних који су појели бар једну шљиву.

Опис улаза

На стандардном улазу се налази један цео неозначен број мањи од 51, број шљива на почетку.

Опис излаза

На стандардни излаз у четири реда исписати по један цео број. Прва три броја треба да буду

бројеви шљива које су појели Јанко, Наташа и Марко, тим редом. Четврти број треба да буде број оних који су појели бар једну шљиву.

Пример 1

Улаз	*Излаз*
14	4
	6
	4
	3

Пример 2

Улаз	*Излаз*
3	0
	0
	3
	1

Пример 3

Улаз	*Излаз*
15	6
	9
	0
	2

Решење

Јанко и Наташа узимају заједно по пет шљива док је то могуће, а то је док преостали број шљива не постане мањи од 5. Према томе, број њихових истовремених, заједничких узимања је $u = s \operatorname{div} 5 = \lfloor \frac{s}{5} \rfloor$, где је s број шљива на почетку. Током тих u узимања, Јанко је узео $j = 2u$, а Наташа $n = 3u$ шљива. Преостале шљиве, којих има $m = n \bmod 5$, добија Марко.

Преостаје још да се одреди колико од бројева j, n, m је позитивно. То можемо да урадимо тако што бројач позитивних бројева поставимо на 0, за сваки од та три броја проверимо да ли је позитиван и за оне који јесу повећамо бројач позитивних.

```
#include <iostream>

using namespace std;

int main() {
    int n;
    cin >> n;
    int janko = (n / 5) * 2;
    int natasa = (n / 5) * 3;
    int marko = n % 5;

    int brPozitivnih = 0;
    if (janko > 0) brPozitivnih++;
    if (natasa > 0) brPozitivnih++;
    if (marko > 0) brPozitivnih++;

    cout << janko << endl;
    cout << natasa << endl;
    cout << marko << endl;
    cout << brPozitivnih << endl;
    return 0;
}
```

Пребројавање позитивних може да буде нешто краће ако приметимо да Јанка и Наташу не морамо одвојено да проверавамо (ако је неко од њих појео бар једну шљиву, онда то важи за обоје). Према томе, ако је испуњен било који од услова $j > 0 \iff n > 0 \iff s \geq 5$, бројач позитивних можемо одмах да повећамо за 2 (и накнадно за 1 ако је $m > 0$).

```
#include <iostream>

using namespace std;
```

```
int main() {
    int n;
    cin >> n;
    int janko = (n / 5) * 2;
    int natasa = (n / 5) * 3;
    int marko = n % 5;

    int brPozitivnih = (n>4) ? 2 : 0;
    if (marko > 0) brPozitivnih++;

    cout << janko << endl;
    cout << natasa << endl;
    cout << marko << endl;
    cout << brPozitivnih << endl;
    return 0;
}
```

Задатак: Самогласници

Аутор: Филип Марић

Такмичење: 2021/22. квалификације 2, V разред, 3. задатак и VI разред, 2. задатак

Матеја воли речи које почињу и завршавају се самогласницима (на пример, **ауто**, **игла**, **окно**) - њима би дала оцену 5.

Мало су јој мање речи које само почињу или се само завршавају самогласником (на пример, **песма**, **авион**) - њима би дала оцену 3.

Најмање су јој драге речи које почињу и завршавају се сугласницима (на пример, **телевизор**, **рам**, **курсор**) - њима би дала оцену 1.

Опис улаза

Са стандардног улаза се учитавају три речи записане малим словима енглеске абецеде, свака у посебном реду. Речи нису дуже од 20 слова.

Опис излаза

На стандардни излаз исписати укупну оцену коју би Матеја дала за све три речи.

Пример 1

Улаз	*Излаз*
`auto`	`9`
`avion`	
`televizor`	

Пример 2

Улаз	*Излаз*
`oboa`	`7`
`klarinet`	
`klavir`	

Решење

Проверу да ли је дати карактер самогласник можемо извршити поређењем са свих пет самогласника. Пошто ову проверу треба вршити више пута, згодно је издвојити је у посебну функцију. Када се учита реч, потребно је проверити јој прво и последње слово. Првом слову ни-

ске `s` можемо приступити са `s[0]` или `s.front()`, док последњем слову можемо приступити са `s[s.size()-1]` или `s.back()`. Гранањем прво проверавамо да ли су оба слова самогласници (треба да буде самогласник и једно и друго слово). Ако јесу, оцена речи је 5. Ако нису, проверамо ли је бар једно слово самогласник (треба да буде самогласник или једно или друго слово). Ако јесте, оцена речи је 3. Ако није, онда су оба слова сугласници и оцена речи је 1.

Пошто је потребно оценити три речи, згодно је оцену дефинисати у склопу посебне функције.

```
#include <iostream>
#include <string>

using namespace std;

bool samoglasnik(char c) {
  return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';
}

int ocena(const string& rec) {
  if (samoglasnik(rec.front()) && samoglasnik(rec.back()))
    return 5;
  if (samoglasnik(rec.front()) || samoglasnik(rec.back()))
    return 3;
  return 1;
}

int main() {
  int ukupno = 0;
  string rec;
  getline(cin, rec);
  ukupno += ocena(rec);
  getline(cin, rec);
  ukupno += ocena(rec);
  getline(cin, rec);
  ukupno += ocena(rec);
  cout << ukupno << endl;
  return 0;
}
```

Задатак: Приземље

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 3, V и VI разред, 2. задатак

У једној згради лифт се веома ретко користи. Кад год уђете у приземље те зграде, лифт је слободан и стоји на неком спрату, а често управо у приземљу.

Пензионер Божур се из хобија бави електроником, па га је интересовало да ли би се исплатило да се лифт подеси тако да чим је слободан, да се врати у приземље. Зато је током прошлог месеца аутоматски прикупио податке, а сада хоће да упосли унука Уроша, програмера, да одреди колико се у приземљу просечно чека на лифт. Урошу би помоћ добро дошла.

Опис улаза

Са стандардног улаза се у првом реду уноси цео број N ($1 \leq N \leq 1000$), број позива лифта из приземља. У наредних N редова је по један цео број S ($0 \leq S \leq 15$), број спратова које је празан лифт прешао по позиву из приземља (да би стигао у приземље).

Опис излаза

На стандардни излаз исписати један цео број, просечан број спратова које лифт пређе да би празан дошао у приземље на позив, заокружен навише.

Пример 1

Улаз	*Излаз*
5 4 0 0 2 0	2

Пример 2

Улаз	*Излаз*
2 0 0	0

Решење

Да се добије средња вредност, потребно је сабрати свих N вредности S, добијени збир поделити са N. На крају треба још заокружити добијену вредност навише, што се може постићи стандардном функцијом из библиотеке или коришћењем везе да је $\lceil \frac{a}{b} \rceil = \frac{a+b-1}{b}$.

```
#include <iostream>
using namespace std;

int main() {
  int n;
  cin >> n;
  int zbir = 0;
  for (int i = 0; i < n; i++) {
    int s; cin >> s;
    zbir += s;
  }
  cout << (zbir + n - 1) / n << endl;
  return 0;
}
```

Задатак: Велико снижење цена

Аутор: Филип Марић

Такмичење: 2021/22. квалификације 3, V разред, 4. задатак

У току је велика распродаја која подразумева да се сваки производ у радњи продаје по сниженој цени. Ако је позната оригинална цена сваког производа који купац купује и проценат снижења те цене, исписати укупан износ новца који купац треба да плати.

Напомена: ако се цена c снижава за p процената, тада се она умањује за $c \cdot \frac{p}{100}$. На пример, ако се цена од 500 динара снижава за 20%, тада се цена умањује за $500 \cdot \frac{20}{100} = 100$ динара, па је

цена након снижења једнака 400 динара.

Опис улаза

Са стандардног улаза се уноси број производа n ($1 \leq n \leq 100$), а затим из наредних n редова по два цела броја: цена производа (број између 1 и 100000) и проценат снижења те цене (број између 1 и 90).

Опис излаза

На стандардни излаз исписати укупан износ заокружен на две децимале.

Пример

Улаз	*Излаз*
3 1000 20 850 30 450 15	1777.50

Решење

У петљи која се извршава n пута учитавамо два броја цену c и попуст p и затим број $c - \frac{c \cdot p}{100} = c \cdot (1 - \frac{p}{100})$ додајемо на укупну цену (коју иницијално постављамо на нулу и на крају исписујемо).

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
  double ukupno = 0.0;
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    int cena, popust;
    cin >> cena >> popust;
    ukupno += cena * (1 - popust / 100.0);
  }
  cout << fixed << showpoint << setprecision(2) << ukupno << endl;
  return 0;
}
```

Задатак: Огрлица

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 1, V разред, 5. задатак

Сара чува перле у N кутијица нумерисаних бројевима од 1 до N. Она увек прави огрлицу тако што из сваке од N кутијица редом по бројевима узме по једну перлу и у истом редоследу наниже перле на конац.

Написати програм који одређује на колико начина Сара може да изабере перле за следећу огрлицу.

Опис улаза

у првом реду стандардног улаза је цео позитиван број N, број кутијица, не већи од 10. У сваком од следећих N редова је по један цео једноцифрен број, број перли у одговарајућој кутијици по реду.

Опис излаза

На стандардни излаз исписати један цео број, број начина на које Сара може да направи следећу огрлицу.

Пример 1

Улаз	*Излаз*
5 3 3 2 5 1	90

Пример 2

Улаз	*Излаз*
3 7 0 2	0

Решење

Пошто се перле могу комбиновати свака са сваком, број начина да се наниже огрлица је једнак производу броја перли из појединих кутија. Дакле, потребно је само помножити дате бројеве перли и исписати производ тих бројева.

```
#include <iostream>
using namespace std;

int main() {
  int n, p;
  cin >> n;
  p = 1;
  for (int i = 0; i < n; i++)  {
    int a;
    cin >> a;
    p *= a;
  }
  cout << p << endl;
  return 0;
}
```

Задатак: Укупно време

Аутор: Филип Марић

Такмичење: 2021/22. квалификације 1, V разред, 5. задатак и VI разред 4. задатак

Петар жели да измери укупно време које проводи играјући своју омиљену видео-игру. Сваки пут када је покренуо игру он је записао време почетка и када је искључио игру он је записао

време завршетка играња. Напиши програм који на основу тих времена израчунава укупно време играња.

Опис улаза

Са стандардног улаза се учитава број n ($1 \leq n \leq 100$) пута колико је Петар играо игру током недељу дана. Након тога се учитава n линија које садрже времена почетка и завршетка игре. Свако време се задаје у формату `h m` (сат и минут раздвојени размаком), при чему су време почетка и време завршетка игре увек у истом дану (Петар никад није остао будан до поноћи).

Опис излаза

На стандардни излаз исписати укупно време (број сати и минута раздвојене размаком).

Пример 1

Улаз	*Излаз*

`3`
`14 35 14 55`
`9 20 10 13`
`7 13 9 7`
	`3 7`

Пример 2

Улаз	*Излаз*
`1` `14 35 14 55`	`0 20`

Решење

Потребно је израчунати трајање сваке појединачне игре и затим сабрати тако добијена времена. Трајање појединачне игре можемо најлакше израчунати ако и време почетка и време завршетка претворимо у минуте (тако што сате помножимо са 60 и саберемо са минутима). Пошто на тај начин добијамо укупно време у минутима, по завршетку сабирања је потребно да израчунамо број сати (као целобројни количник при дељењу са 60) и број минута (као остатак при дељењу са 60).

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n >> ws;
  int mukupno = 0;
  for (int i = 0; i < n; i++) {
    int spoc, mpoc, skraj, mkraj;
    cin >> spoc >> mpoc >> skraj >> mkraj;
    mukupno += (60*skraj + mkraj) - (60*spoc + mpoc);
  }
  cout << mukupno / 60 << " " << mukupno % 60 << endl;
  return 0;
}
```

Задатак: Промена школе

Аутор: Душан Пајадић, Јелена Петровић

Такмичење: 2023/2024. квалификације 1, VI и VII разред, 3. задатак

На крају првог разреда средње школе Ивана је одлучила да се пребаци из медицинске школе у гимназију. У гимназији су одлучили да ће јој као просек у првом разреду рачунати само одређене предмете - оне који постоје у обе школе. Закључено је да се k Иваниних оцена неће рачунати за просек у новој школи. Одредити нови Иванин просек оцена ако је познат просек из медицинске школе, укупан број предмета које је слушала у медицинској (n) и k оцена које јој се неће рачунати. Претпоставити да је Ивана завршила разред, тј. да није имала ниједну закључену јединицу.

Просек оцена $a_1, a_2, ..., a_n$ рачуна се на следећи начин:

$$\frac{a_1 + a_2 + ... + a_n}{n}.$$

Опис улаза

У првом реду стандардног улаза налази се природан број n, а у другом реду број k ($1 \leq k < n \leq 50$). У наредних k редова је дато k Иваниних оцена (природни бројеви од 2 до 5) које се не рачунају. У последњем реду се налази број p који представља Иванин просек из медицинске школе који је заокружен на две децимале.

Опис излаза

Исписати нови Иванин просек, заокружен на 2 децимале.

Пример 1

Улаз	*Излаз*
3 1 4 4.33	4.50

Пример 2

Улаз	*Излаз*
12 3 4 5 4 4.58	4.67

Решење

Нека су $a_1, a_2, ..., a_n$ Иванине оцене на крају првог разреда средње медицинске школе, где су са $a_1, ..., a_k$ означене оцене које се неће рачунати. Просек свих оцена, означен са p, рачуна се на следећи начин:

$$p = \frac{a_1 + a_2 + ... + a_n}{n}.$$

Са $\hat{p}$ ћемо означити број p заокружен на две децимале. Како су $\hat{p}$ и $a_1, ..., a_k$ познати, збир преосталих оцена можемо добити користећи претходну формулу, водећи притом рачуна о заокруживању јер су оцене природни бројеви:

$$a_{k+1} + ... + a_n = p \cdot n - (a_1 + ... + a_k) = \text{round}(\hat{p} \cdot n) - (a_1 + ... + a_k).$$

Пошто преосталих оцена има $n - k$, њихов просек се може израчунати дељењем израчунатог збира преосталих оцена са $n - k$ (из услова задатка важи $n - k > 0$):

$$\frac{a_{k+1} + ... + a_n}{n-k} = \frac{\text{round}(\hat{p} \cdot n) - (a_1 + ... + a_k)}{n-k}.$$

```
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

int main()
{
    int n, k;
    cin >> n >> k;

    int zbirOcena = 0;
    for (int i = 0; i < k; ++i)
    {
        int ocena;
        cin >> ocena;
        zbirOcena += ocena;
    }

    double prosek;
    cin >> prosek;

    double noviProsek = ((int)round(prosek*n) - zbirOcena) / (n-k);
    cout << fixed << showpoint << setprecision(2) <<  noviProsek << endl;

    return 0;
}
```

Задатак: Трансформације

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 2, V разред, 4. задатак и VI разред 3. задатак

Трансформацијом броја x зовемо замену броја x бројем $x \cdot x + 1$. Одредити троцифрени завршетак броја који се добија после n трансформација броја a.

Опис улаза

Са стандардног улаза се у првом реду уноси број a $(2 \leq a \leq 9)$, а у другом реду број n $(10 \leq n \leq 1000)$.

Опис излаза

На стандардни излаз исписати три цифре без размака, цифре којима се завршава трансформисани број.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
5	802	402	026
11		2	

Решење

Број a је трансформисањем веома брзо увећава, тако да већ после неколико трансформација даље рачунање или није тачно (због прекорачења) или је врло споро (због величине броја).

Пошто су нам потребне само последње три цифре трансформисаног броја, користићемо модуларну аритметику. Уместо са $a \cdot a + 1$, замењиваћемо a са $(a \cdot a + 1) \bmod 1000$. Нови резултат ће бити конгруентан старом по модулу 1000, па се зато при овој измени последње три цифре резулатата неће променити.

Након израчунавања трансформисаног броја по модулу 1000 треба још повести рачуна да се испишу и водеће нуле ако их има.

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    int a, n;
    cin >> a >> n;
    for (int i = 0; i < n; i++)
        a = (a * a + 1) % 1000;
    cout << setfill('0') << setw(3) << a << endl;
    return 0;
}
```

Задатак: Висок притисак

Аутор: Филип Марић

Такмичење: окружно 2021/22., V разред, 3. задатак

Током једног месеца вршено је мерење ваздушног притиска. Напиши програм који одређује број мерења при којима је притисак био строго већи од дате вредности.

Опис улаза

Са стандардног улаза се учитава број мерења n ($1 \leq n \leq 31$), а затим у сваком од n наредних редова подаци о једном мерењу: редни број дана у месецу када је извршено мерење и вредност притиска заокружена на две децимале. На крају се учитава гранична вредност притиска, такође заокружена на две децимале.

Опис излаза

На стандардни излаз исписати један неозначен цео број, тражени број дана.

Пример

Улаз	*Излаз*
4 1 1001.23 5 999.71 8 1003.42 13 998.73 1000.00	2

Решење

Податке о измереном притиску уписујемо у низ, занемарујући податке о дану када је мерење извршено (тај податак је небитан). Након учитавања граничне вредности у петљи пролазимо кроз низ и бројимо колико је елемената низа веће од учитане границе.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int n;
  cin >> n;
  vector<double> pritisak(n);
  for (int i = 0; i < n; i++) {
    int dan;
    cin >> dan >> pritisak[i];
  }
  double granica;
  cin >> granica;
  int brojDana = 0;
  for (double p : pritisak)
    if (p > granica)
      brojDana++;
  cout << brojDana << endl;
  return 0;
}
```

Задатак: Секција

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 1, V разред, 6. задатак и VI разред 5. задатак

У школи је велико интересовање за програмерску секцију, на којој ће се правити рачунарске игре. Наставник је поставио услов да на секцију могу да иду само они ученици који имају 5 из информатике и бар 4 из математике.

Написати програм који одређује колико ученика може да се пријави за секцију.

Опис улаза

у првом реду број N, број ученика заинтересованих за секцију, природан број не већи од 200. У сваком од следећих N редова низ оцена једног од заинтересованих ученика из свих 11 предмета редом, без размака. Оцена из математике је шеста, а из информатике девета у низу од 11 цифара.

Опис излаза

На стандардни излаз исписати један цео број, број ученика који испуњавају услов.

Пример 1

Улаз	*Излаз*
4 55555355455 55555455455 55555455555 22222522522	2

Пример 2

Улаз	*Излаз*
2 55555555455 55555355555	0

Решење

Најједноставније је да учитавамо сваки ред као стринг од 11 карактера (цифара). Потребно је да пребројимо оне стрингове код којих је шеста цифра слева (бројећи од 1) већа од 3, а девета слева једнака 5.

Након пребројавања исписујемо вредност бројача.

```
#include <iostream>
using namespace std;

int main()
{
    int n, br;
    cin >> n;
    string s;
    br = 0;
    for (int i = 0; i < n; i++)
    {
        cin >> s;
        if (s[5] - '0' > 3 && s[8] - '0' == 5)
            br++;
    }
    cout << br << endl;
    return 0;
}
```

Задатак: Топ

Аутор: Иван Дрецун

Такмичење: 2023/2024. квалификације 1, VI разред, 4. задатак

Мали Душан је тек почео да учи како се игра шах. Учитељ му је на шаховску таблу ставио једног топа. Топ је фигура која може да нападне било које поље у врсти и колони у којој се налази. Врсте на шаховској табли означене су бројевима од 1 до 8, а колоне словима од `A` до `H` енглеске

абецеде. Малог Душана занима да ли ће топ моћи да нападне његову фигуру ако је стави на неко поље. Напиши програм који даје Малом Душану одговор на то питање за сва поља за која га то занима.

Опис улаза

Са стандардног улаза се уноси позиција топа у стандардном шаховском запису. У наредном реду се уноси број n који представља број питања која Мали Душан поставља. Након тога се у наредних n редова уносе позиције поља за које Мали Душан жели да добије одговор.

Опис излаза

На стандардни излаз за свако питање исписати `DA` уколико топ напада то поље, односно `NE` у супротном. Одговоре исписивати у засебним редовима.

Пример

Улаз	*Излаз*
A4	NE
3	DA
B7	NE
D4	
H2	

Решење

Поље на ком се налази топ ћемо унети као ниску карактера (која садржи 2 карактера). У петљи ћемо затим уносити n поља (поново као ниске од 2 карактера). Топ напада унето поље ако је оно у истој врсти или истој колони, што се своди на то да је или први карактер две ниске једнак или други карактер две ниске једнак.

```
#include <iostream>

using namespace std;

int main() {
  string top;
  cin >> top;

  int n;
  cin >> n;

  for (int i = 0; i < n; i++) {
    string polje;
    cin >> polje;

    if (top[0] == polje[0] || top[1] == polje[1])
      cout << "DA" << endl;
    else
      cout << "NE" << endl;
  }

  return 0;
```

```
}
```

Задатак: Мејлови

Аутор: Филип Марић

Такмичење: општинско 2018/2019, VII разред, 2. задатак, VIII разред, 2. задатак

Пера ради у софтверској компанији у Нишу чија је централа у САД и послао је свом шефу n мејлова. За сваки мејл је познато време (сат и минут) слања, изражено у локалном, нишком времену. Ако његов шеф ради у Њујорку од 9 до 17h, по локалном, њујоршком времену које за нишким касни тачно 6 сати, напиши програм који одређује колико је мејлова шефу стигло ван његовог радног времена (мејлови у 9:00 се рачунају да су унутар, а у 17:00 да су ван радног времена).

Опис улаза

Са стандардног улаза се учитава број мејлова n ($1 \leq n \leq 100$), а затим се учитава време сваког мејла (сат између 0 и 23 и минут између 0 и 59). Свако време је задато у посебном реду.

Опис излаза

На стандардни излаз исписати број мејлова који су послати ван њујоршког радног времена.

Пример

Улаз	*Излаз*
7 14 0 22 59 15 0 9 0 19 24 17 0 23 23	3

Решење

У петљи се учитавају времена свих мејлова и одржава се број оних који су ван радног времена. За свако учитано време проверавамо да ли се након умањења сата за 6 добија број између 9 и 17 и ако се не добија, увећава се бројач мејлова ван радног времена.

```
#include <iostream>

using namespace std;

int main() {
  int broj_mejlova;
  cin >> broj_mejlova;
  int broj_van_radnog_vremena = 0;
  for (int i = 0; i < broj_mejlova; i++) {
    int sat, minut;
    cin >> sat >> minut;
    if (sat - 6 < 9 || sat - 6 >= 17)
```

```
      broj_van_radnog_vremena++;
  }
  cout << broj_van_radnog_vremena << endl;
  return 0;
}
```

Задатак: Медаље

Такмичење: окружно 2018/2019, V разред, 3. задатак

Аутор: Филип Марић

На окружном такмичењу из информатике медаљом се награђују најуспешнији такмичари. Праг за бронзану медаљу је 70 освојених поена, за сребрну 80, а за златну 90 освојених поена. Са стандардног улаза се уноси број ученика (њих највише 100) и затим поени сваког од њих. Напиши програм који одређује број бронзаних, сребрних и златних медаља.

Опис улаза

Са стандардног улаза се учитава број такмичара n ($1 \leq n \leq 100$), а затим за сваког такмичара број поена у посебном реду (природан број између 0 и 100).

Опис излаза

На стандардни излаз исписати број бронзаних, сребрних и златних медаља (сваки број у посебном реду).

Пример

Улаз	*Излаз*
5	1
83	2
95	1
55	
70	
82	

Решење

Број бронзаних, сребрних и златних медаља ћемо чувати у три посебне променљиве, које иницијализујемо на нулу. У петљи учитавамо један по један број поена. Ако је број поена већи или једнак од 90 поена, увећавамо број златних медаља. У супротном, ако је већи или једнак од 80 поена, увећавамо број сребрних медаља. У супротном, ако је већи или једнак од 70 поена, увећавамо број бронзаних медаља. Након петље исписујемо вредности три променљиве.

```
#include <iostream>

using namespace std;

int main() {
  int bronzanih = 0;
  int srebrnih = 0;
  int zlatnih = 0;
  int n; cin >> n;
```

```
  for (int i = 0; i < n; i++) {
    int poeni;
    cin >> poeni;
    if (poeni >= 90)
      zlatnih++;
    else if (poeni >= 80)
      srebrnih++;
    else if (poeni >= 70)
      bronzanih++;
  }
  cout << bronzanih << endl;
  cout << srebrnih << endl;
  cout << zlatnih << endl;
  return 0;
}
```

Задатак: Купци

Аутор: Милан Вугделија

Такмичење: 2019/20 општинско, VII и VIII разред, 3. задатак

Менаџер једног великог ланца продавница жели да зна структуру прихода. Он је зато све куповине разврстао у мале, средње и велике на следећи начин: ако је износ рачуна за неку куповину X, та куповина се сматра за малу ако је $X \leq M$, за средњу ако је $M < X \leq V$, а за велику ако је $V < X$. Менаџер жели да за посматрани период зна колики је укупан приход продавница од малих, средњих и велилких куповина редом.

Опис улаза

Са стандардног улаза се у првом реду учитавају бројеви M и V ($1 \leq M < V \leq 100000$). У другом реду је цео број n, укупан број куповина у свим продавницама у посматраном периоду ($1 \leq n \leq 50$). У трећем и последњем реду су износи рачуна за n куповина, цели позитивни бројеви до 200000 раздвојени по једним размаком.

Опис излаза

Три цела броја, сваки у посебном реду. Ови бројеви су укупни приходи од малих, средњих и великих куповина.

Пример

Улаз	*Излаз*
`100 1000`	`188`
`5`	`512`
`88 1010 512 100 2500`	`3510`

Решење

Формирамо три суме (sm, ss, sv), које ће редом садржати приходе од малих, средњих и великих куповина. Након иницијализације све три суме, разматрамо једну по једну куповину, поредимо је са границама за мале, средње и велике куповине и разврставамо, а затим је додајемо на одговарајућу суму. На крају исписујемо све три суме.

```
#include <iostream>

using namespace std;

int main()  {
  int m, v, n, racun;
  cin >> m >> v >> n;
  int sm = 0, ss = 0, sv = 0;
  for (int i = 0; i < n; i++) {
    cin >> racun;
    if (racun <= m)
      sm += racun;
    else if (racun <= v)
      ss += racun;
    else
      sv += racun;
  }
  cout << sm << endl;
  cout << ss << endl;
  cout << sv << endl;
  return 0;
}
```

Задатак: Цифре и слова

Аутор: Филип Марић

Такмичење: окружно 2021/22., VII разред, 2. задатак

Напиши програм који за унету ниску одређује апсолутну вредност разлике између броја цифара и броја слова у тој ниски. Као слова се броје мала и велика слова енглеске абецеде.

Опис улаза

Једина линија стандардног улаза садржи реч дужине највише 20 карактера. У ниски, поред малих и великих слова енглеске абецеде и цифара, могу да се појаве и празнине и други ASCII карактери (неће се појављивати слова других писама).

Опис излаза

На стандардни излаз исписати тражену вредност.

Пример

Улаз	*Излаз*
`Programiranje 1`	`12`

Решење

Учитавамо једну линију текста, пролазимо кроз све њене карактере, одржавајући два бројача: један бројач цифара и један бројач слова. Бројаче увећавамо сваки пут када се наиђе на карактер одговарајуће врсте. Проверу врсте карактера можемо вршити било библиотечким функцијама, било поређењем ASCII тј. UNICODE кодова.

```cpp
#include <iostream>
#include <string>
#include <cmath>

using namespace std;

int main() {
  string linija;
  getline(cin, linija);
  int brojCifara = 0, brojSlova = 0;
  for (char c : linija)
    if (isdigit(c))
      brojCifara++;
    else if (isalpha(c))
      brojSlova++;
  cout << abs(brojCifara - brojSlova) << endl;
  return 0;
}
```

Задатак: Крађа дијаманта

Аутор: Павле Секешан

Такмичење: 2023/2024. квалификације 1, VIII разред, 3. задатак

Озлоглашени пљачкаши Тоша и Моша испланирали су да украду драгоцени дијамант. Оно што овај подухват чини тежим него раније јесте нови ласерски безбедносни систем постављен у просторијама где се дијамант налази.

Моша је успео да сазна да су ласери постављени веома специфично; постоји тачно n вертикалних ласера, где је i-ти ласер постављен на позицију v_i и m хоризонталних ласера, постављени на позицију h_i (ласере можемо посматрати као праве представљене једначином $x = v_i$ за вертикалне, односно $y = h_i$ за хоризонталне ласере).

У међувремену, Тоша је сазнао да један прозор на врху зграде може да се отвори, па ако се спусте низ конопац кроз њега, наћи ће се на координатама (x_1, y_1) у соби, а да је тачна локација дијаманта на координатама (x_2, y_2).

Све што је преостало је да се ласери онеспособе. Зато, Тошу и Мошу интересује колико минимално ласера морају да искључе да би могли несметано да дођу до дијаманта, не прелазећи ни преко једног ласера. Да ли можете да им помогнете?

Опис улаза

- У првом реду стандардног улаза налазе се целобројне вредности x_1 и y_1 раздвојене размаком, координате места у соби на коме се Тоша и Моша налазе на почетку.
- У другом реду налазе се целобројне вредности x_2 и y_2 раздвојене размаком, координате дијаманта у просторији. У трећем реду налази се природан број n, број вертикалних ласера.
- У четвртом реду налази се n целобројних вредности раздвојених размаком, где i-та вредност v_i представља x координату вертикалног ласера.
- У петом реду налази се природан број m, број хоризонталних ласера.

- У шестом реду налази се m целобројних вредности раздвојених размаком, где i-та вредност h_i представља y координату хоризонталног ласера.

Опис излаза

Природан број који представља минималан број ласера које Тоша и Моша морају да искључе.

Ограничења

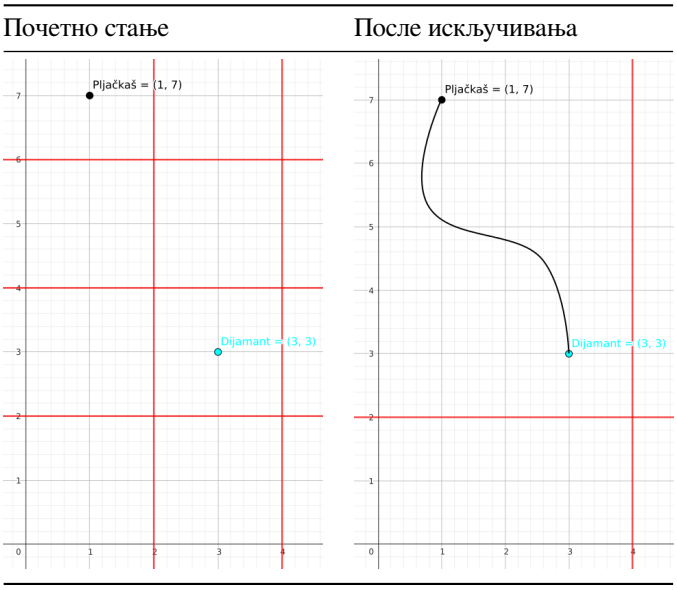
$$-10^9 \leq x_1, y_1, x_2, y_2 \leq 10^9$$

$$1 \leq n, m \leq 10^5$$

$$-10^9 \leq v_i, h_i \leq 10^9$$

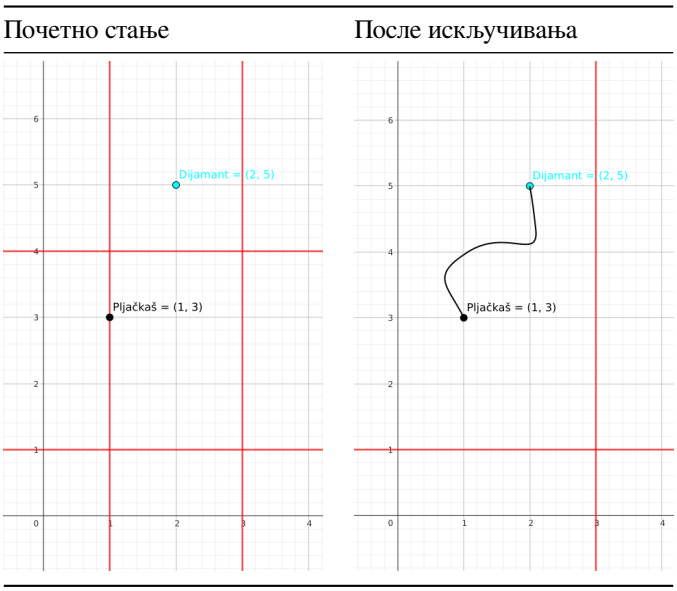
Пример 1

Улаз	*Излаз*	*Објашњење*
1 7 3 3 2 2 4 3 2 4 6	3	Минималан број ласера које морамо да искључимо је 3, на пример хоризонталне ласере на позицијама $y = 6$ и $y = 4$ и вертикалан ласер на позицији $x = 2$.



Пример 2

Улаз	*Излаз*	*Објашњење*
1 3 2 5 2 1 3 2 1 4	2	Минималан број ласера које морамо да искључимо је 2, на пример хоризонтални ласер на позицији $y = 4$ и вертикалан ласер на позицији $x = 1$.



Решење

Потребно је искључити све оне хоризонталне ласере који се налазе између y-координата пљачкаша и дијаманта и све оне вертикалне ласере који се налазе између x-координата пљачкаша и дијаманта. Један начин да се утврди да се тачка a налази између тачака a_1 и a_2 је да се провери услов $|a_1 - a| + |a_2 - a| = |a_1 - a_2|$ (овај услов је коректан без обзира на међусобни однос вредности a_1 и a_2).

```
#include <iostream>

using namespace std;

bool je_izmedju(int a1, int a2, int a) {
  return abs(a1 - a) + abs(a2 - a) == abs(a1 - a2);
}

int izbroj_izmedju(int c1, int c2, int k) {
  int br = 0;
  for (int i = 0; i < k; i++) {
    int c;
    cin >> c;
    if (je_izmedju(c1, c2, c))
      br++;
  }
  return br;
}
```

```
int main() {
  int x1, y1, x2, y2;
  cin >> x1 >> y1 >> x2 >> y2;
  int br = 0;
  int n;
  cin >> n;
  br += izbroj_izmedju(x1, x2, n);
  int m;
  cin >> m;
  br += izbroj_izmedju(y1, y2, m);
  cout << br << endl;
  return 0;
}
```

Задатак: Број посета кућици

Аутор: Филип Марић

Такмичење: 2021/22. квалификације 2, VI разред, 3. задатак и 7. разред, 2. задатак

Лик се у игрици креће по неограниченој мрежи квадратних поља. Свако поље је одређено својим координатама (координата у расте на горе) а играч креће са поља $(0, 0)$, које се сматра његовом "кућицом". Лик се премешта на суседно поље када год играч притисне неку од четири стрелице (означимо их са `<` за лево, `>` за десно, `^` за горе и `v` за доле). Написати програм који одређује колико се пута током игре играч налази у својој кућици.

Опис улаза

Са стандардног улаза се учитава ниска састављена од карактера `<`, `>`, `^` и `v` (њих највише 10^6).

Опис излаза

На стандардни излаз исписати тражени број посета пољу $(0, 0)$.

Пример 1

Улаз	*Излаз*	*Објашњење*
`<>^v><v^>^`	`5`	Играч редом обилази поља $(0, 0)$, $(-1, 0)$, $(0, 0)$, $(0, 1)$, $(0, 0)$, $(1, 0)$, $(0, 0)$, $(0, -1)$, $(0, 0)$, $(1, 0)$, $(1, 1)$, што значи да се на пољу $(0, 0)$ налази 5 пута.

Пример 2

Улаз	*Излаз*
`>>>^^^<<<vvv>>>^^^<<<vvv`	`3`

Решење

Симулираћемо кретање лика тако што ћемо у сваком тренутку пратити његове координате на табли. Обрађиваћемо један по један карактер и у складу са тим који смер кретања карактер описује, мењаћемо тренутну позицију (за карактер `<` смањујемо координату x за 1, за карактер `>` повећавамо координату x за 1, за карактер `^` повећавамо координату y за 1, а за карактер `v` смањујемо координату y за 1). Када одредимо нове координате проверавамо да ли су обе

једнаке нули и ако јесу, тада увећавамо бројач посета кућици (пошто се на почетку лик налази на пољу $(0, 0)$ тј. у кућици, тај бројач треба иницијлизовати на 1).

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  int x = 0, y = 0;
  int brojPoseta = 1;
  char c;
  while ((cin >> c) && c != '\n') {
    switch(c) {
    case '<':
      x--;
      break;
    case '>':
      x++;
      break;
    case '^':
      y++;
      break;
    case 'v':
      y--;
      break;
    }
    if (x == 0 && y == 0)
      brojPoseta++;
  }
  cout << brojPoseta << endl;
  return 0;
}
```

Задатак: Triple-Double од 3 категорије

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 1, VI разред, 6. задатак

Чувени кошаркаш Дабло Трипловић има толико добре статистике, да се већ помало сумња да су његови успеси добро пребројани.

Написати програм који одређује колико пута је Трипловић постигао такозвани трипл–дабл.

Опис улаза

На стандардноим улазу се у првом реду налази цео број N, $(1 \leq N \leq 100)$, број утакмица које је Трипловић одиграо у сезони. У сваком од следећих N редова по 3 цела броја раздвојена по једним размаком: број поена, скокова и асистенција редом (ови бројеви нису већи од 1000).

Опис излаза

На стандардни излаз исписати један број, број утакмица на којима је Трипловић постигао трипл–дабл.

Напомена

За потребе овог задатка трипл–дабл дефинишемо као најмање двоцифрен учинак (10 или више) у све три категорије које се прате. Другим речима: сва три броја већа од 9 у једном реду улаза.

Пример

Улаз	*Излаз*
3 20 9 7 127 12 11 11 14 12	2

Решење

Потребно је пребројати редове улаза у којима су сва три броја већа од 9. Увешћемо бројач који пре петље постављамо на 0, а затим у петљи учитавамо по три броја, прверавамо да ли су сва три већа од 9, и ако јесу - увећавамо бројач.

По завршетку петље исписујемо вредност бројача.

```
#include <iostream>
using namespace std;

int main()
{
    int n, brTriplDabl = 0, a, b, c;
    cin >> n;
    for (int utak = 0; utak < n; utak++)
    {
        cin >> a >> b >> c;
        if (a > 9 && b > 9 && c > 9)
            brTriplDabl ++;
    }
    cout << brTriplDabl << endl;
    return 0;
}
```

Задатак: Triple-Double од 5 категорија

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 1, VII и VIII разред, 5. задатак

Чувени кошаркаш Дабло Трипловић има толико добре статистике, да се већ помало сумња да су његови успеси добро пребројани.

Написати програм који одређује колико пута је Трипловић постигао такозвани трипл–дабл.

Опис улаза

Са стандардног улаза се у првом реду уноси цео број N $(1 \leq N \leq 100)$, број утакмица које је Трипловић одиграо у сезони. У сваком од следећих N редова је по 5 целих бројева раздвојених по једним размаком: број поена, скокова, асистенција, блокада и украдених лопти редом (ови бројеви нису већи од 1000).

Опис излаза

На стандардни излаз исписати један цео број, број утакмица на којима је Трипловић постигао трипл–дабл.

Напомена

За потребе овог задатка трипл–дабл дефинишемо као најмање двоцифрен учинак (10 или више) у бар три од пет категорија које се прате. Другим речима: бар три броја већа од 9 у једном реду улаза.

Пример

Улаз	*Излаз*
3 20 9 7 2 1 127 12 11 3 0 0 10 11 14 12	2

Решење

У овом задатку треба пребројати редове улаза у којима су бар три од пет бројева већи од 9. Да бисмо установили да ли неки ред улаза треба бројати, треба у сваком реду пребројати елементе који су већи од 9. То значи да ћемо у овом задатку применити технику пребројавања "на два нивоа": глобално бројимо редове који испуњавају услов, а у сваком реду бројимо елементе веће од 9.

Формираћемо двоструку петљу: спољна петља ће ићи по утакмицама (тј. редовима улаза), а унутрашња по категоријама које се прате на свакој утакмици. Бројач трипл-даблова иницијализујемо пре спољне петље, а бројач двоцифрених учинака пре унутрашње.

```
#include <iostream>
using namespace std;

int main()
{
    int n, brTriplDabl = 0;
    int a[5];
    cin >> n;
    for (int utak = 0; utak < n; utak++)
    {
        cin >> a[0] >> a[1] >> a[2] >> a[3] >> a[4];
        int brDvoc = 0;
        for (int kateg = 0; kateg < 5; kateg++)
            if (a[kateg] >= 10)
                brDvoc++;

        if (brDvoc >= 3)
            brTriplDabl++;
```

```
    }
    cout << brTriplDabl << endl;
    return 0;
}
```

Задатак: Клуб

Аутор: Милан Вуīделија

Такмичење: окружно 2022/2023, VI разред, 3. задатак

У билијар клуб долазе и одлазе чланови клуба од ујутру до увече према распореду. Сви чланови имају кључ од клуба и договор је да први ујутру који дође откључа, цео дан клуб буде откључан и увече последњи који оде закључа. За дата времена долазака и одлазака људи из клуба одредити колико је сати и минута клуб укупно био откључан, као и колико људи је дошло у клуб пре 12 сати.

Опис улаза

У првом реду стандардног улаза се налази природан број n ($1 \leq n \leq 100$), број људи који је током посматраног дана дошло у клуб. У наредних n редова се налазе по 4 природна броја раздвојена размацима: сати и минути када је члан дошао у клуб и сати и минути када је члан отишао из клуба (сати су у опсегу од 0 до 23, а минути од 0 до 59). Сви доласци и одласци су у току једног дана.

Опис излаза

У првом реду стандардног излаза исписати време у сатима и минутима (два броја раздвојена размаком) током ког је клуб био откључан. Сати треба да буду у опсегу од 0 до 23, а минути од 0 до 59. У другом реду исписати један природан број - колико људи је дошло у клуб пре 12 сати.

Пример 1

Улаз	*Излаз*	*Објашњење*

`5`
`10 20 16 40`
`11 0 15 17`
`7 15 9 30`
`17 45 21 20`
`11 59 14 20`
	`14 5`
	`4`
		Клуб је откључао 3. члан у 7:15, а закључао 4. у 21:20, па је клуб укупно био откључан 14 сати и 5 минута. Сви чланови осим 4. су у клуб дошли пре 12, дакле 4 члана.

Пример 2

Улаз	*Излаз*	*Објашњење*

`3`
`9 20 13 10`
`12 0 12 15`
`11 15 13 10`
	`3 50`
	`2`
		Клуб је откључао 1. члан у 9:20, а закључали су га заједно 1. и 3. у 13:10, па је клуб укупно био откључан 3 сата и 50 минута. Чланови 1 и 3 су у клуб ушли пре 12, па је одговор на друго питање 2.

Решење

Пажљивим читањем задатка лако се уочава да се задатак своди на истовремено одређивање минимума и максимума улазне серије вредности. Прецизније, да бисмо одредили време када се отворио билијар клуб треба да одредимо време када је дошао први играч, а то време је заправо

минимум од свих унетих времена долазака. На исти начин, може се закључити да је време затварања билијар клуба време када је отишао последњи играч, што је максимум од свих унетих времена одлазака.

Ради једноставнијег решавања задатка, сва времена која се уносе ћемо из формата сат-минут превести у минуте. Ако је ds време доласка у сатима и dm време доласка у минутима једног играча, тада његово време доласка изражено само у минутима добијамо следећим изразом $ds \cdot 60 + dm$. Ако су са os и om редом обележени његово време одласка у сатима и минутима, на исти начин можемо изразити време одласка тог играча само у минутима, тј. са $os \cdot 60 + om$.

Нека је са dol обележено време доласка првог играча у минутима и са odl време одласка последњег играча у минутима. На почетку, dol иницијализујемо са вредности која је већа од свих вредности дозвољених задатком, на пример $dol = 24 \cdot 60$. Такође, odl иницијализујемо са вредности која је мања од свих дозвољених вредности, на пример $odl = 0$. Нека је са br обележен број играча који су дошли пре 12 сати. На почетку програма, вредност br иницијализујемо са 0.

Затим, потребно је да учитамо колико играча је дошло у билијар клуб, тј. вредност n. Након тога, у свакој од n итерација петље учитавамо времена доласка и одласка једног играча. Времена преводимо у минуте према раније описаном поступку и упоређујемо их са тренутним минимумом dol и тренутним максимум odl. Ако је време доласка мање од тренутног минимума, ажурираћемо вредност dol. Ако је време одласка веће од тренутног максимума, ажурираћемо вредност odl. Поред тога, уколико је време доласка играча у сатима мање од 12, инкрементираћемо вредност br.

Након што се петља изврши, потребно је само да одредимо колико дуго је билијар клуб био откључан. То ћемо лако постићи тако што нађемо разлику вредности odl и dol, тј. одузмемо минималну вредност од максималне. Време које та разлика представља је изражено у минутима. Време колико је клуб био отворен у сатима добијамо као количник при дељењу те разлике са 60, а време у минутима добијамо као остатак при дељењу те разлике са 60. На крају остаје само да прикажемо резултат у траженом формату.

Описани поступак је приказан у решењу.

```
#include <iostream>

using namespace std;

int main() {

   int n;
   cin >> n;

   int dol = 24 * 60;
   int odl = 0;
   int br = 0;

   for (int i = 0; i < n; i++){
      int ds, dm, os, om;
      cin >> ds >> dm >> os >> om;
      if (ds * 60 + dm < dol)
        dol = ds * 60 + dm;
```

```
      if (os * 60 + om > odl)
        odl = os * 60 + om;
      if (ds < 12)
        br++;
   }

   int r = odl - dol;
   cout << r / 60 << " " << r % 60 << endl << br;

   return 0;
}
```

Задатак: Списак производа

Аутор: Филип Марић

Такмичење: 2022/2023. квалификације 1, 6. разред, 4. задатак

Љиља је сваки дан школе куповала за ужину јабуку, кифлу или лизалицу. Ако је познато колика је цена сваког производа и ако је познат списак производа које је она купила током неколико дана, одредити колико је укупно новца потрошила.

Опис улаза

Са стандрадног улаза се учитавају цена јабуке, кифле и лизалице (три природна броја између 10 и 100, сваки у посебном реду). Након тога се учитава ниска од највише 30 карактера `j`, `k` и `l` која одређује редом производе које је Љиља купила.

Опис излаза

На стандардни излаз исписати укупну количину утрошеног новца.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
`32`	`259`	`10`	`60`
`30`		`20`	
`45`		`30`	
`jjkllkl`		`jkl`	

Решење

Цене сва три производа чувамо у засебним променљивама. Након тога учитавамо ниску која садржи списак купљених производа и у петљи `for` анализирамо један по један њен карактер. У зависности од прочитаног карактера укупан збир увећавамо за цену одговарајућег производа.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  int j, k, l;
  cin >> j >> k >> l;
```

```
  string proizvodi;
  cin >> proizvodi;
  int novac = 0;
  for (char c : proizvodi)
    if (c == 'j')
      novac += j;
    else if (c == 'k')
      novac += k;
    else if (c == 'l')
      novac += l;
  cout << novac << endl;
  return 0;
}
```

Задатак: Дежурство

Аутор: Милан Вуђелија

Такмичење: 2021/22. квалификације 2, V разред 4. задатак

У једном великом и сложеном систему се повремено дешавају инциденти, а тада је потребно да дежурни службеник брзо интервенише и отклони или премости проблем. Имамо податке о времену дешавања инцидената током N дана, као и податке о томе када је Илија био дежуран током тог периода. Потребно је одредити укупан број инцидената, као и број инцидената који су се догодили за време Илијиног дежурства.

Опис улаза

У првом реду стандардног улаза је цео број N ($1 \leq N \leq 20$), број дана током којих су праћени инциденти. У свакој од наредних N група по три реда улаза налазе се подаци за један дан, и то:

У првом реду у оквиру групе је низ од 24 слова. Свако од слова је `S` или `D`, а означава да ли је Илија био слободан или дежуран у одговарајућем сату тог дана. У другом реду у оквиру групе је један цео број, број инцидената B ($1 \leq B \leq 9$), који су се догодили тог дана. У трећем реду у оквиру групе је B целих бројева, сваки од 0 до 23, укључујући границе. Ти бројеви представљају сате у којима су се тог дана догодили инциденти.

Опис излаза

На стандардни излаз исписати два цела броја, раздвојена размаком. Први број је укупан број инцидената, а други је број инцидената који су се догодили за време Илијиног дежурства.

Пример

Улаз	*Излаз*	*Објашњење*

`3`
`DSSSDDSSDSSDSSSDSSSSDSSS`
`2`
`7 9`
`DSSDDSSDSSDSSSDSSSSDSSSS`
`5`
`3 3 4 8 8`
`DSSSDDSSDSSDSSDSSSSDSSSS`
`4`
`0 14 14 14`
	`11 7`	Бројећи за сваки дан слова у распореду Илијиног дежурства од нуле, видимо да је Илија био дежуран за време 7 од укупно 11 инцидената (у трећем и четвртом сату другог дана и нултом и четрнаестом сату трећег дана).

Решење

Укупан број инцидената можемо да одредимо сабирајући бројеве инцидената по данима. Да бисмо пребројали инциденте током којих је Илија био дежуран, потребно је да за сваки дан прочитамо времена инцидената из тог дана и да бројимо оне инциденте којима одговара слово `D` у Илијином распореду дежурства.

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
    int nDana, nIncidenata, vreme;
    int brSvihInc = 0;
    int brDezInc = 0;
    string raspored;
    cin >> nDana;
    for (int d = 0; d < nDana; d++) {
        cin >> raspored >> nIncidenata;
        brSvihInc += nIncidenata;
        for (int i = 0; i < nIncidenata; i++) {
            cin >> vreme;
            if (raspored[vreme] == 'D')
                brDezInc++;
        }
    }
    cout << brSvihInc << " " << brDezInc << endl;
    return 0;
}
```

Задатак: Најмања дубина

Аутор: Филип Марић

Такмичење: 2021/22. квалификације 3, VI разред, 3. задатак и VII разред, 2. задатак

Подморница је кренула своје путовање са нивоа мора (дубине 0) а затим се спуштала и подизала. Ако су познате промене дубине након сваког минута, напиши програм који одређује најнижу тачку (тј. највећу дубину, изражену најмањим негативним бројем) на којој се подморница током пута налазила. Позитивна промена означава да се подморница токог тог минута подигла, а негативна да се спустила. Могуће је да су подаци неисправни, па програм треба да провери и да ли је подморница све време била на дубинама које су испод нивоа мора (или на нивоу мора).

Опис улаза

Са стандардног улаза се уноси број минута n ($3 \leq n \leq 100$), а затим низ од n целих бројева (између -100 и 100) који означавају промену дубине (негативни бројеви означавају да се подморница спуштала, а позитивни да се подизала).

Опис излаза

На стандардни излаз исписати тражену најнижу тачку, или - ако подаци нису исправни.

Пример 1

Улаз	*Излаз*
5 -2 -3 1 -4 3	-8

Пример 2

Улаз	*Излаз*
4 -2 4 -7 3	-

Решење

У једном променљивој ћемо чувати тренутну дубину подморнице. Њу ћемо иницијализовати на нулу и у сваком кораку ћемо је увећавати за учитану промену дубине подморнице. Класичним алгоритмом одређивања минимума пронаћи ћемо најмању вредност те променљиве (тако што у сваком тренутку упоредимо текућу дубину са дотада најмањом и ажурирамо најмању дубину ако је то потребно). Ако је у неком тренутку дубина строго позитивна, у логичкој променљивој ћемо регистровати да је дошло до грешке. На крају ћемо проверити да је дошло до грешке и ако није, исписаћемо најмању дубину.

```
#include <iostream>

using namespace std;

int main() {
  int dubina = 0;
  int minDubina = 0;
  int n;
  cin >> n;
  bool OK = true;
  for (int i = 0; i < n; i++) {
    int d;
    cin >> d;
    dubina += d;
    if (dubina > 0) {
      OK = false;
```

```
      break;
    }
    if (dubina < minDubina)
      minDubina = dubina;
  }
  if (OK)
    cout << minDubina << endl;
  else
    cout << "-" << endl;
  return 0;
}
```

Задатак: Највиши дечак и девојчица

Такмичење: државно 2018/2019, V и VI разред, 1. задатак

Аутор: Филип Марић

Познате су висине свих ученика једне школе. Напиши програм који одређује највишег дечака и највишу девојчицу у тој школи, одређује ко је од њих виши и исписује разлику њихових висина.

Опис улаза

Са стандардног улаза се уноси број ученика ($5 \leq n \leq 100$), а затим у сваком од n наредних редова подаци за по једног ученика. За сваког ученика се уноси висина (цео број између 120 и 200) и ознака пола (`m` за мушки пол и `z` за женски), раздвојени једним размаком. Претпоставити да постоји бар један дечак и бар једна девојчица.

Опис излаза

Ако је највиша девојчица виша од највишег дечака, на стандардни излаз исписати `z`, размак и затим за колико је виша. Ако је највиши дечак виши од највише девојчице, на стандардни излаз исписати `m`, размак и затим за колико је виши. Ако су исте висине, исписати само `=`.

Пример

Улаз	*Излаз*
5 152 z 174 m 165 z 172 m 170 z	m 4

Решење

Циљ је да одредимо висину највишег дечака и висину највише девочице. То можемо урадити или тако што формирамо низ висина дечака и низ висина девојчица и позовемо библиотечку функцију или тако што применимо класичан алгоритам одређивања максимума.

Након што су одређене највиша висина дечака и највиша висина девојчица, гранањем испитујемо која од њих је већа и исписујемо тражени резултат.

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int maks_m = 0, maks_z = 0;
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    int visina; string pol;
    cin >> visina >> pol;
    if (pol == "m")
      maks_m = max(maks_m, visina);
    else if (pol == "z")
      maks_z = max(maks_z, visina);
  }

  if (maks_m > maks_z)
    cout << "m" << " " << maks_m - maks_z << endl;
  else if (maks_z > maks_m)
    cout << "z" << " " << maks_z - maks_m << endl;
  else
    cout << "=" << endl;

  return 0;
}
```

Задатак: Грип

Аутор: Милан Вугделија

Такмичење: 2019/20 општинско, VII разред, 3. задатак и 8. разред, 1. задатак

Због сезонског грипа, мали Пера данас мери температуру тела сваких пола сата. Написати програм који ће за дато време првог мерења вредности температуре и N мерења, одредити и исписати највишу измерену температуру и тачно време када је та температура измерена. Можете претпоставити да су сва мерења извршена у истом дану, као и да ће постојати само једна највиша измерена температура.

Опис улаза

У првом реду стандардног улаза налази се један цео број H, $0 \leq H \leq 23$, час у коме је извршено прво мерење температуре. У другом реду налази се један цео број M, $0 \leq M \leq 59$, минут у ком је забележено прво мерење температуре. У трећем реду налази се један природан број N, $1 \leq N \leq 10$, број извршених мерења. У следећих N редова налази се по један природан број T, $1 \leq T \leq 42$, вредност измерене температуре у сваком следећем мерењу.

Опис излаза

У првом реду стандардног излаза исписати највишу измерену температуру. У другом реду ис-

пишите час, а у трећем реду минут у ком је измерена та највиша температура.

Пример 1		**Пример 2**		**Пример 3**		**Пример 4**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
14	39	14	39	0	39	10	37
0	15	0	14	0	0	21	11
3	0	3	30	2	30	3	21
37		36		38		35	
36		39		39		36	
39		37				37	

Решење

Након учитавања почетног времена и броја мерења n, тражимо највећи од наредних n бројева и његов редни број i_{max}. Након проласка кроз сва мерења, треба да одредимо време највише температуре, а оно се добија додавањем i_{max} пута по 30 минута на почетно време. У случају да је температура била највиша при првом мерењу, не треба додавати ништа на почетно време. Према томе, при одређивању редног броја највише температуре згодно је да се броји од 0 (редни број прве температуре је 0).

```
#include <iostream>

using namespace std;

int main()  {
  int h, m, n, t;
  cin >> h >> m >> n;
  int tMax = 0, iMax = 0;
  for (int i = 0; i < n; i++) {
    cin >> t;
    if (t > tMax) {
      tMax = t;
      iMax = i;
    }
  }
  int ukMinuta = h*60 + m + 30*iMax;
  cout << tMax << endl << ukMinuta / 60 << endl << ukMinuta % 60 << endl;
  return 0;
}
```

Задатак: Преферанс

Аутор: Небојша Варница

Такмичење: 2021/22. квалификације 1, VIII разред 2. задатак

Карте за игру имају ознаке A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2 - сложено од најјаче до најслабије карте. Осим ознаке свака карта има и "боју": пик, каро, херц и треф. Са картама се могу играти различите игре: "пасијанс", "таблић", "реми", "преферанс", "бриџ", "покер", "ајнц", "црна дама", …

У преферансу учествују три играча. На почетку игре се бира адутска боја. У сваком кругу игре играчи бацају по једну карту. Победник је играч који има најјачу карту у адутској боји. Ако нико не баци такву карту онда је победник играч који баци најјачу карту у боји коју је бацио први играч.

Примери: ако је адутска боја херц а играчи су бацили:

- K херц, 9 херц, A треф - победник је први играч јер је бацио најјачу карту у адутској боји
- A пик, 10 херц, K каро - победник је други играч јер је једини бацио адутску боју
- J треф, K каро, Q треф - победник је трећи играч јер нико није бацио адутску боју а трећи играч је бацио најјачу карту у боји коју је бацио први играч

Напишите програм који за унету адутску боју и три унете различите карте израчунава који је играч победник круга.

Опис улаза

Са стандардног улаза у првом реду се задаје адутска боја (почетно латинично слово назива боје - P, K, H или T) а затим у три следећа реда три карте: прво боја (почетно латинично слово назива боје - P, K, H или T) а затим ознака карте (A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3 или 2) - без размака

Опис излаза

На стандардном излазу приказати редни број играча који је победио у том кругу

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
H	1	H	2	H	3
HK		PA		TJ	
H9		H10		KK	
TA		KK		TQ	

Решење

Опис главног решења

Упоређујемо прве две карте па се јача од њих упоређује са трећом.

Приликом упоређивања прво обрадимо случај када су обе адутске, па затим када је једна адутска а друга није и на крају када ниједна није адутска.

```
#include <iostream>

using namespace std;

int vrednost(char ozn) {
  if (ozn >= '2' && ozn <= '9')
    return ozn-'0';
  switch(ozn) {
  case '1' : return 10;
  case 'J' : return 12;
  case 'Q' : return 13;
  case 'K' : return 14;
  case 'A' : return 15;
```

```
  }
  return 0;
}

int main() {
  char adut, boja[3], oznaka[3];
  cin >> adut;
  for (int i=0; i<3; i++) {
    string x;
    cin >> x;
    boja[i] = x[0];
    oznaka[i] = x[1];
  }
  int res = 0;
  for (int i=1; i<3; i++) {
    if (boja[res] == adut && boja[i] == adut) {
      if (vrednost(oznaka[res]) < vrednost(oznaka[i]))
        res = i;
    } else if (boja[res] != adut && boja[i] == adut) {
      res = i;
    } else if (boja[res] == boja[i]) {
      if (vrednost(oznaka[res]) < vrednost(oznaka[i]))
        res = i;
    }
  }
  cout << (res+1) << endl;
  return 0;
}
```

Задатак: Састанак

Такмичење: окружно 2018/2019, VI разред, 2. задатак, VII разред, 1. задатак

Аутор: Филип Марић

Пет програмера током једног дана борави у фирми. Ако се за сваког од њих зна сат и минут доласка и сат и минут одласка напиши програм који одређује да ли је могуће да организују заједнички састанак (на коме морају сви бити присутни) и ако је могуће, колико је најдуже трајање таквог састанка (у сатима и минутима).

Опис улаза

Сваки од пет редова стандардног улаза садржи четири цела броја раздвојена са по једним размаком (сат и минут доласка и сат и минут одласка сваког програмера).

Опис излаза

На стандардни излаз исписати време састанака тако да су број сати и минута раздвојени једним размаком или реч ne ако састанак није могуће одржати.

Пример 1

Улаз	*Излаз*
8 15 16 30 9 40 17 15 9 20 14 30 8 45 16 45 9 15 15 20	4 50

Пример 2

Улаз	*Излаз*
7 30 12 30 7 48 13 29 12 28 19 12 12 33 17 37 11 36 18 51	ne

Решење

Да бисмо лакше радили са временом, претворићемо сате и минуте у минуте (док ћемо при приказу коначног трајања састанка минуте претворити у сате и минуте).

Састанак може да почне када је последњи од 5 људи дошао на посао и мора да се заврши када први од њих напусти посао. Зато је довољно да нађемо највеће време доласка (алгоритмом тражења максимума) и најмање време одласка (алгоритмом тражења минимума). Ако је највеће време доласка веће од најмањег времена одласка, састанак се не може одржати.

```
#include <iostream>

using namespace std;

int main() {
  int poslednji_dosao = 0;
  int prvi_otisao = 24 * 60;
  for (int i = 0; i < 5; i++) {
    int sat_dosao, minut_dosao, sat_otisao, minut_otisao;
    cin >> sat_dosao >> minut_dosao >> sat_otisao >> minut_otisao;

    int dosao = sat_dosao * 60 + minut_dosao;
    int otisao = sat_otisao * 60 + minut_otisao;
    if (dosao > poslednji_dosao)
      poslednji_dosao = dosao;
    if (otisao < prvi_otisao)
      prvi_otisao = otisao;
  }

  if (prvi_otisao > poslednji_dosao) {
    int sati = (prvi_otisao - poslednji_dosao) / 60;
    int minuta = (prvi_otisao - poslednji_dosao) % 60;
    cout << sati << " " << minuta << endl;
  } else
    cout << "ne" << endl;

  return 0;
}
```

Задатак: Мајстор

Аутор: Иван Дрецун

Такмичење: 2022/2023. квалификације 1, 6. разред, 5. задатак

Након извођења радова мајстор Пери је враћено n кутија са шрафовима. Кутије су коришћене и могу садржати различите бројеве шрафова. Мајстор Пера хоће да користи две кутије са најмањим бројем шрафова. Помозите мајстор Пери да израчуна колико укупно шрафова има у тако изабране две кутије.

Опис улаза

У првој линији стандардног улаза уноси се укупан број кутија n ($2 \leq n \leq 30$). Затим се у наредних n линија уносе бројеви шрафова у кутијама k_i ($1 \leq k_i \leq 100$), редом.

Опис излаза

На стандардни излаз исписати цео број који представља збир шрафова у две кутије са најмање шрафова.

Пример 1

Улаз	*Излаз*
5 45 32 9 15 67	24

Пример 2

Улаз	*Излаз*
7 764 455 721 231 138 97 333	235

Пример 3

Улаз	*Излаз*
5 28 14 30 15 14	28

Решење

Одржавање најмања два броја приликом проласка кроз низ

У задатку је потребно одредити две најмање вредности које се јављају у низу. До решења можемо доћи читајући један по један елемент низа, притом чувајући две најмање вредности међу онима које смо до сада прочитали, на пример у променљивама $\min_1$ (најмањи) и $\min_2$ (други најмањи). Када читамо наредну вредност низа, разликујемо три случаја:

- Уколико је прочитана вредност мања од $\min_1$, тада је то нова најмања вредност. Не треба да заборавимо да је у овом случају друга најмања вредност она која је претходно била најмања.
- Уколико је прочитана вредност мања од $\min_2$, али не и од $\min_1$, тада је то нова друга најмања вредност.
- Уколико је прочитана вредност већа или једнака од $\min_2$, нема потребе ажурирати променљиве.

Нека је на пример дат низ вредности `8 6 4 5 7`. На почетку издвајамо $\min_1 = 6$ и $\min_2 = 8$ као почетне две најмање вредности. Након тога читамо број 4. Како је 4 мање и од 6 и од 8, сада треба да буде $\min_1 = 4$, а $\min_2 = 6$. Након читања броја 5 треба да буде $\min_1 = 4$, а $\min_2 = 5$. Након читања броја 7 треба да буде $\min_1 = 4$, а $\min_2 = 5$ (најмање две вредности се не мењају).

```
#include <iostream>
#include <vector>
```

```
using namespace std;

int main() {
  int n;
  cin >> n;

  vector<int> kutije(n);
  for (int i = 0; i < n; i++)
    cin >> kutije[i];

  int min1 = kutije[0];
  int min2 = kutije[1];
  if (min2 < min1)
    swap(min1, min2);

  for (int i = 2; i < n; i++) {
    if (kutije[i] < min1) {
      min2 = min1;
      min1 = kutije[i];
    }
    else if (kutije[i] < min2)
      min2 = kutije[i];
  }

  cout << min1 + min2 << endl;
  return 0;
}
```

Примена сортирања

Задатак је могуће решити применом сортирања. Након сортирања улазног низа, најмања два броја ће се наћи на прве две позиције у низу. Једноставно се може исписати њихов збир. Нагласимо да иако је ово решење најједноставније имплементирати, оно није најефикасније (јер се непотребно одређује прецизан редослед елемената након прва два).

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int n;
  cin >> n;

  vector<int> kutije(n);
  for (int i = 0; i < n; i++)
    cin >> kutije[i];

  sort(begin(kutije), end(kutije));
```

```
  cout << kutije[0] + kutije[1] << endl;
  return 0;
}
```

Задатак: Две полице

Аутор: Филип Марић

Такмичење: 2019/20 квалификације 2, VI разред, 6. задатак и VII и VIII разред, 5. задатак

У продавници се продају две врсте полица чије су цене и дужине познате. Желимо да поставимо полице дуж једног зида тако да попунимо што већи део дужине зида. При том, ако се иста дужина зида може попунити на више начина, бирамо јефтинију варијанту. Напиши програм који одређује највећу могућу дужину дела зида која се може попунити полицама и најмању цену по којој се то може урадити.

Опис улаза

У првом реду стандардног улаза се налази број z ($1 \leq z \leq 10^6$) који представља дужину зида. У два наредна реда се налазе описи полица: дужина (природан број између 1 и 10^6) и цена (природан број између 1 и 1000).

Опис излаза

На стандардни излаз исписати највећу могућу дужину дела зида који се може попунити полицама и најмању могућу цену, раздвојене једним размаком.

Пример

Улаз	*Излаз*	*Објашњење*

`24`
`3 10`
`5 8`
	`24 54`	Цео зид се може попунити са 3 полице дужине 3 и 3 полице дужине 5 или са 8 полица дужине 3. Цена у првом случају је 54, а у другом случају је 80.

Решење

Задатак решавамо грубом силом тј. исцрпном претрагом свих могућности. Крећемо од варијанте у којој немамо полица типа 1 и затим додајемо једну по једну полицу типа 1, све док је то могуће (док је дужина дела зида попуњеног полицама типа 1 мања или једнака од укупне дужине зида). За сваки број полица типа 1 који тренутно разматрамо, израчунавамо највећи могући број полица типа 2 који се може поставити (њега одређујемо целобројним дељењем дужине дела зида иза постављених првих полица дужином друге полице). Након тога израчунавамо преосталу дужину зида иза свих полица и цену и ако је потребно ажурирамо минимум.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int zid;
  cin >> zid;
  int polica1, cena1;
```

```
  cin >> polica1 >> cena1;
  int polica2, cena2;
  cin >> polica2 >> cena2;

  if (polica1 < polica2) {
     swap(polica1, polica2);
     swap(cena1, cena2);
  }

  int min_ostalo = zid;
  int min_cena = 0;

  // isprobavamo sve mogucnosti
  // dodajemo jednu po jednu policu tipa 1 dok god je to moguce
  for (int broj1 = 0; broj1*polica1 <= zid; broj1++) {
    // racunamo najveci moguci broj preostalih polica tipa 2
    int broj2 = (zid - broj1*polica1) / polica2;
    int ostalo = zid - broj1*polica1 - broj2*polica2;
    // cena svih polica
    int cena = broj1*cena1 + broj2*cena2;
    // azuriramo minimum ako je potrebno
    if (ostalo < min_ostalo ||
       (ostalo == min_ostalo && cena < min_cena)) {
      min_ostalo = ostalo;
      min_cena = cena;
    }
  }

  cout << zid - min_ostalo << " " << min_cena << endl;
  return 0;
}
```

Задатак: Флаше

Аутор: Душан Пошадић

Такмичење: 2022/2023. квалификације 2, 6. разред, 2. задатак

Ранко претаче разне врсте течности из n пуних канистера различитих запремина у флаше запремине v литара. За сваки канистер одредити колико флаша ће бити потпуно напуњено течношћу из тог канистера, као и колико течности ће остати у канистеру када се те флаше попуне. Течности се не смеју међусобно мешати.

Опис улаза

У првом реду стандардног улаза налазе се 2 цела броја n (број канистера) и v (запремина флаша у литрима). У наредних n редова се налази по један цео број. Сваки тај број представља запремину једног канистера у литрима.

Сви бројеви су цели и између 1 и 10000.

Опис излаза

У n редова исписати по 2 цела броја раздвојена размаком. Први број представља број флаша које ће бити потпуно напуњене од стране канистера, а други број колико литара течности ће остати у канистеру након попуњавања флаша.

Пример

Улаз	*Излаз*	*Објашњење*
7 6 24 13 6 10 17 23 40	4 0 2 1 1 0 1 4 2 5 3 5 6 4	Први канистер има запремину 24 литра Пошто је запремина флаше 6 литара, могу се напунити 4 флаше и 0 литара остаје у канистеру. Други канистер има запремину 13 литара, дакле могу се напунити 2 флаше, а 1 литар остаје у канистеру. Слично и за остале канистере.

Решење

За сваки канистер, број флаша које се могу потпуно напунити течношћу из тог канистера можемо добити целобројним количником запремина канистера и запремине флаша (то је практично количник заокружен наниже), док се вишак течности може добити као остатак при дељењу запремина канистера и флаша.

```
#include <iostream>
using namespace std;

int main() {
  int n, v;
  cin >> n >> v;
  for (int i = 0; i < n; i++) {
    int t;
    cin >> t;
    cout << t / v << " " << t % v << endl;
  }
  return 0;
}
```

Задатак: По три цифре

Аутори: Филип Марић, Владимир Кузмановић

Такмичење: 2022/2023. квалификације 3, 6. разред, 5. задатак

Читљивости ради, велики бројеви се записују тако што се раздвајају класе од по три цифре. Напиши програм који унети природан број штампа у том облику.

Опис улаза

Са стандардног улаза се учитава један природни број мањи од 10^{18}.

Опис излаза

На стандардни излаз исписати тај број са издвојеним класама цифара.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`123456`	`123 456`	`12345`	`12 345`	`123456789007`	`123 456 789 007`

Решење

Опис главног решења

У задатку прво треба приметити опсег броја који се учитава (10^{18}) и одатле закључити да не могу да се користе 32-битни целобројни типови података, јер потенцијално може доћи до прекорачења. Поред тога, треба приметити да нам нигде у задатку не треба вредност броја, већ само укупан број његових цифара које треба да раздвојимо по класама. Због тога, број можемо да учитамо као ниску, тј. стринг или низ карактера.

Да бисмо лакше решили задатак, такође треба приметити да саме класе не зависе од вредности цифара, већ само од тежине цифре у запису броја. Нумерисање тежина цифара започећемо од позиције најмање тежине, тј. од позиције јединица (са десна на лево). Позицију јединица нумерисаћемо бројем 0, позицију десетица бројем 1, позицију стотина бројем 2 и тако даље све док не дођемо до позиције највеће тежине. Ако број у свом запису има n цифара, тада ће цифра највеће тежине имати индекс $n - 1$.

Ако је дат број 123456, чији је запис дужине 6, тада ће цифра јединица (6) бити на позицији 0, цифра десетица (5) на позицији 1, цифра стотина (4) на позицији 2, цифра хиљада (3) на позицији 3 и тако даље све до цифре највеће тежине (1) чија је позиција 5. Одавде треба приметити да иза сваке цифра чија је позиција дељива са 3 треба да одштампамо један размак. Додавање размака не треба урадити само у случају цифре на позицији најмање тежине.

С обзиром да број штампамо са лева на десно, потребно је да у петљи пролазимо редом кроз цифре броја. Број смо учитали као текст, па ће цифра највеће тежине у низу карактера имати индекс 0, следећа цифра индекс 1 и тако даље све до цифре најмање тежине чији ће индекс бити $n - 1$. На основу индекса у низу карактера треба да израчунамо редни број позиције цифре у броју. Да бисмо то урадили потребно је само да приметимо да цифри на индексу i одговара редни број позиције $n - 1 - i$ у запису броја. Специјално, изостављамо испис размака након последње цифре.

Описани поступак је приказан у решењу.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  string broj;
  cin >> broj;
  int n = broj.size();
  for (int i = 0; i < n; i++) {
    cout << broj[i];
    if ((n - 1 - i) % 3 == 0 && i != n-1)
      cout << " ";
  }
```

```
  cout << endl;
  return 0;
}
```

Задатак: Парно-парни-непарно-непарни низови

Аутор: Филип Марић

Такмичење: државно 2022/2023, VII разред, 1. задатак

Низ је ппнн (парно-парно-непарно-непаран) ако се на парним позицијама у низу (0, 2, 4, …) налазе искључиво парни, а на непарним позицијама у низу искључиво непарни бројеви. Напиши програм који за неколико учитаних низова проверава да ли су ппнн низови.

Опис улаза

Са стандардног улаза се учитавају бројеви $n(5 \leq n \leq 100)$ и $k(5 \leq k \leq 100)$ а затим из n наредних редова n низова од по k елемената чија је вредност између 0 и 1000.

Опис излаза

За сваки од учитаних низова на стандардни излаз исписати `da` ако задовољава ппнн услов, а у супротном `ne`.

Пример

Улаз	*Излаз*	*Објашњење*
`3 5`	`ne`	У првом низу се на позицији 0 која је парна налази број 1, који је непаран.
`1 2 3 4 5`	`da`	У другом низу се на парним позицијама налазе бројеви 2, 4 и 6, који су парни, а на непарним позицијама бројеви 3 и 5, који су непарни.
`2 3 4 5 6`	`da`	
`4 7 0 9 2`		У трећем низу се на парним позицијама налазе бројеви 4, 0 и 2, који су парни, а на непарним позицијама бројеви 7 и 9, који су непарни.

Решење

Задатак се решава тако што се линеарном претрагом (у петљи) испитају сви чланови низа. Ако се пронађе макар један паран елемент на непарној позицији или непаран елемент на парној позицији низ није *ппнн* (што региструјемо посебном логичком променљивом). Линеарна претрага се извршава независно за сваки учитани низ (зато у програму користимо двоструку петљу).

```
#include <iostream>

using namespace std;

int main() {
  int n, k;
  cin >> n >> k;
  for (int i = 0; i < n; i++) {
    bool ppnn = true;
    for (int j = 0; j < k; j++) {
      int x;
      cin >> x;
      if (j % 2 != x % 2)
        ppnn = false;
```

```
    }
    cout << (ppnn ? "da" : "ne") << endl;
  }
  return 0;
}
```

Задатак: Парно непарни

Аутор: Филип Марић

Такмичење: државно 2022/2023, VI разред, 1. задатак

Низ је *ппнн* (парно-парно-непарно-непаран) ако се на парним позицијама у низу (0, 2, 4, …) налазе искључиво парни, а на непарним позицијама у низу (1, 3, 5, …) искључиво непарни бројеви. Напиши програм који за учитани низ проверава да ли је *ппнн*.

Опис улаза

Са стандардног улаза се учитава број $k (5 \leq k \leq 100)$ а затим k елемената низа чија је вредност између 0 и 1000.

Опис излаза

За учитани низ на стандардни излаз исписати `da` ако задовољава ппнн услов, а у супротном `ne`.

Пример 1

Улаз	*Излаз*	*Објашњење*
`5` `1 2 3 4 5`	`ne`	На позицији 0 која је парна налази се број 1 који је непаран.

Пример 2

Улаз	*Излаз*	*Објашњење*
`7` `2 3 4 5 6 7 8`	`da`	На парним позицијама се налазе бројеви 2, 4, 6 и 8 који су парни, а на непарним позицијама бројеви 3, 5 и 7 који су непарни.

Пример 3

Улаз	*Излаз*	*Објашњење*
`5` `4 7 0 9 2`	`da`	На парним позицијама се налазе бројеви 4, 0 и 2 који су парни, а на непарним позицијама бројеви 7 и 9 који су непарни.

Решење

Задатак се решава тако што се линеарном претрагом (у петљи) испитају сви чланови низа. Ако се пронађе макар један паран елемент на непарној позицији или непаран елемент на парној позицији низ није *ппнн* (што региструјемо посебном логичком променљивом).

```
#include <iostream>

using namespace std;

int main() {
  int k;
  cin >> k;
  bool ppnn = true;
```

```
  for (int j = 0; j < k; j++) {
    int x;
    cin >> x;
    if (j % 2 != x % 2)
      ppnn = false;
  }
  cout << (ppnn ? "da" : "ne") << endl;

  return 0;
}
```

Задатак: Сличне речи

Аутор: Милан Вуīделија

Такмичење: 2021/22. квалификације 1, VI разред, 5. задатак

Написати програм, који за низ од N датих речи једнаке дужине проверава да ли се сваке две узастопне речи у низу разликују на тачно једној позицији.

Опис улаза

У првом реду стандардног улаза је природан број N ($2 \leq N \leq 20$). У сваком од N наредних редова је по једна реч, која се састоји од великих слова енглеске абецеде. Дужина свих N речи је иста, и то најмање једно, а највише 30 слова.

Опис излаза

На стандардни излаз исписати само реч `da` или реч `ne`.

Пример 1

Улаз	*Излаз*
4	da
KOBILA	
POBILA	
POBIJA	
DOBIJA	

Пример 2

Улаз	*Излаз*
3	ne
KAKAV	
KAKVA	
KAKVO	

Пример 3

Улаз	*Излаз*
2	ne
PERCE	
PERCE	

Решење

Користимо логичку променљиву `reciSuSlicne`, у којој ће се налазити одговор на питање да ли су сви прегледани парови узастопних речи слични, тачније: да ли се разликују на тачно једној позицији. На крају рада програма ћемо у овој променљивој имати коначан одговор. На почетку "сви испитани" парови испуњавају услов (јер сви елементи празног скупа испуњавају било који услов), па је почетна вредност променљиве `true`.

Пошто сваку нову реч поредимо са претходном, згодно је да прву реч учитамо пре петље. У петљи учитавамо нову реч и пребројимо позиције на којима се она разликује од претходне речи. Ако је број тих позиција различит од један, коначан одговор је познат (одречан је) и може се прекинути даље испитивање. Ако је број тих позиција једнак један, испитивање се наставља. У том случају потребно је још да ажурирамо претходну реч (у следећем проласку кроз петљу претходна реч ће бити ова која је у текућем проласку била нова).

```cpp
#include <iostream>

using namespace std;

int main() {
    bool reciSuSlicne = true;
    string prethRec, sledRec;
    int n;
    cin >> n >> prethRec;
    for (int rec = 1; rec < n && reciSuSlicne; rec++) {
        cin >> sledRec;
        int brRazlSlova = 0;
        for (int slovo = 0; slovo < prethRec.size(); slovo++)
            if (prethRec[slovo] != sledRec[slovo])
                brRazlSlova++;

        if (brRazlSlova != 1)
            reciSuSlicne = false;

        prethRec = sledRec;
    }
    if (reciSuSlicne)
        cout << "da";
    else
        cout << "ne";
   return 0;
}
```

Задатак: Најмањи број са највећим збиром цифара

Такмичење: окружно 2018/2019, VI разред, 3. задатак

Аутор: Филип Марић

Напиши програм који за n унетих природних бројева мањих од милијарду одређује онај који има највећи збир цифара. Ако постоји више таквих бројева, исписати најмањи од њих.

Опис улаза

Са стандардног улаза се уноси број n ($1 \leq n \leq 100$), а затим бројеви, сваки у посебном реду.

Опис излаза

На стандардни излаз исписати тражени број.

Пример

Улаз	*Излаз*
5 345 555 456 711 123	456

Решење

Задатак се своди на алгоритам одређивања максимума, при чему је критеријум поређења два броја специфичан. Пошто се критеријум заснива на упоређивању збирова цифара бројева, дефинисаћемо помоћну функцију за израчунавање збира цифара. Чуваћемо максимални број и његов збир цифара и иницијализоваћемо те променљиве на нулу (јер ни један број не може имати збир цифара мањи од нуле). Учитавамо један по један број, израчунавамо му збир цифара. Резултат ћемо ажурирати ако је збир цифара учитаног броја већи од досадашњег највећег збира или ако је збир цифара једнак досадашњем највећем збиру, али је сам број мањи од досадашњег најбољег.

```
#include <iostream>

using namespace std;

int zbirCifara(int x) {
  int zbir = 0;
  while (x > 0) {
    zbir += x % 10;
    x = x / 10;
  }
  return zbir;
}

int main() {
  int n; cin >> n;
  int maxBroj = 0, maxZbirCifara = 0;
  for (int i = 0; i < n; i++) {
    int broj; cin >> broj;
    int zbir = zbirCifara(broj);
    if (zbir > maxZbirCifara || (zbir == maxZbirCifara && broj < maxBroj)) {
      maxBroj = broj;
      maxZbirCifara = zbir;
    }
  }
  cout << maxBroj << endl;
  return 0;
}
```

Задатак: Чудесна цифра

Аутори: Јелена Хаџи-Пурић, Милан Вугделија

Такмичење: 2019/20 општинско, VIII разред, 2. задатак

Кажемо да је цифра неког броја чудесна ако је њена вредност једнака позицији на којој се налази у броју (позиције се броје од један, слева надесно). На пример, у броју 92 цифра два је чудесна, јер се налази на другој позицији. У броју 1838 постоје две чудесне цифре: 1 и 3 (цифра један је на првој позицији и цифра три је на трећој позицији).

Написати програм који за задати природан број N исписује колико је у њему чудесних цифара, а потом у следећем реду исписује и вредности тих цифара гледано слева надесно.

Опис улаза

На стандардном улазу је један природан број, број N ($1 \leq N \leq 999\,999\,999$)

Опис излаза

- У првом реду стандардног излаза број чудесних цифара броја N.
- У другом реду стандардног излаза редом чудесне цифре броја N.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
94	0	423	2 2 3	1838	2 1 3

Решење

У овом задатку је згодније да се број N учита као стринг (ниска). Задатак решавамо у два пролаза кроз цифре броја N. У првом пролазу само бројимо чудесне цифре, па ако их има онда их у другом пролазу исписујемо. Услов да је i-та цифра слева чудесна је да бројчана вредност те цифре једнака i (односно једнака $i + 1$ пошто цифре бројимо од 0).

```
#include <iostream>

using namespace std;

int main()  {
  string a;
  cin >> a;

  // odredjujemo broj cudesnih cifara
  int n = 0;
  for (int i = 0; i < a.size(); i++)
    if (a[i] - '0' == i+1)
      n++;

  // ispisujemo cudesne cifre
  cout << n << endl;
  if (n > 0) {
    for (int i = 0; i < a.size(); i++)
      if (a[i] - '0' == i+1)
```

```
        cout << a[i] << " ";
    cout << endl;
  }
  return 0;
}
```

Alternativno, cifre možemo određivati deljenjem sa 10 i možemo ih smestiti u niz, pa zatim analizirati sadržaj tog niza. Pošto se na taj način cifre dobijaju zdesna nalevo, a dodaju se uvek na kraj niza, u nizu će biti smeštene naopako.

Задатак: Телеграф

Аутори: Душан Пајадић, Милица Мићић

Такмичење: 2021/22. квалификације 2, VI разред, 4. задатак

У доба пре интернета и телефона, за брзи пренос порука често се користио телеграф. Код телеграфа је специфично што се сваки знак (слово, цифра…) шаље посебно. Маре и Паја су старомодни и за своју комуникацију још увек користе телеграф, али да би били сигурни да их нико не прислушкује развили су шифру за међусобно препознавање. Препознавање се одвија у 3 корака:

1. Паја замисли један природан број (A) и пошаље га Марету цифру по цифру.
2. Маре одреди просек цифара броја A и направи нови број који се добија тако што се из Пајиног броја избаце цифре које су строго мање од израчунатог просека. Након тога Маре помножи добијени број унапред договореним природним бројем k и резултат (број B) шаље Паји.
3. Паја формира нови број (C) тако што пролази кроз цифре броја B и наизменично их сабира и одузима, почевши од последње цифре (од последње цифре одузме претпоследњу, затим на њу дода ону испред претпоследње, па ону испред те опет одузима и тако наизменично сабира и одузима док не потроши све цифре). Тако добијен број поново шаље Марету и након тога настављају разговор нормално.

Опис улаза

У првом реду налазе се два броја n ($1 \leq n \leq 8$), број цифара броја A, и k ($1 \leq k \leq 20$), број којим Маре множи свој број. У наредних n редова се налазе цифре броја A.

Опис излаза

Исписати број B који је Маре послао Паји, а након тога у следећем реду број C који је Паја на крају послао Марету.

Пример 1

Улаз	*Излаз*	*Објашњење*
6 3 2 4 3 7 8 1	234 3	Пајин број има 6 цифара и то је 243781. Просек цифара тог броја је (2+4+3+7+8+1)/6 = 4,17. Након избацивања свих цифара које су мање од просека добија се 78, који Маре множи са k и добија $78 \cdot 3 = 234$. Паја на основу броја 234 добија свој нови број 4-3+2=3. Дакле, исписује се прво 234, а затим 3.

Пример 2

Улаз	*Излаз*
3 6 1 6 5	390 -6

Решење

Учитавамо цифре полазног броја A и смештамо их у низ цифара (почевши од цифре највеће тежине). Просек цифара је једнак количнику збира свих цифара са бројем цифара, па зато израчунавамо збир свих цифара. Након израчунавања просека филтрирамо цифре задржавајући само оне које су веће или једнаке од просека и од њих Хорнеровом схемом формирамо број који множимо са k и тако добијамо вредност B коју исписујемо. Формирање броја Хорнеровом схемом се врши тако што при сваком уносу цифре досадашњи број помножимо са 10 и додамо нову цифру као последњу, при чему се цифре обрађују слева надесно (од цифре највеће тежине).

Да бисмо добили број C који Паја враћа Марету потребно је да уклањамо једну по једну цифру са десног краја броја B, док нам не остане 0. Издвајамо једну по једну цифру добијеног броја, кренувши од цифре најмање тежине и од тих цифара формирамо тражен збир, водећи рачуна о томе да цифре на непарним позицијама одузимамо, а на парним додајемо на збир.

```
#include <iostream>

using namespace std;

int main() {
  // broj cifara i koeficijent kojim se množi
  int n, k;
  cin >> n >> k;

  // ucitavamo cifre broja A
  int cifreA[8];
  for (int i = 0; i < n; i++)
    cin >> cifreA[i];

  // izracunavamo prosek cifara broja A
  int zbir = 0;
  for (int i = 0; i < n; i++)
    zbir += cifreA[i];
  double prosek = (double)zbir / (double)n;

  // formiramo broj B od cifara koje su veće ili jednake od proseka
  long long B = 0;
  for (int i = 0; i < n; i++)
    if (cifreA[i] >= prosek)
      B = B * 10 + cifreA[i];
  // množimo ga sa k i ispisujemo dobijeni rezultat
  B *= k;
```

```
  cout << B << endl;

  // izdvajamo cifre broja B zdesna na levo i dodajemo ih ili
  // oduzimamo ih od rezultata u zavisnosti od parnosti tekuće
  // pozicije cifre
  bool parnaPozicija = true;
  int C = 0;
  while (B > 0) {
    int cifra = B % 10;
    if (parnaPozicija)
      C += cifra;
    else
      C -= cifra;
    B /= 10;
    parnaPozicija = !parnaPozicija;
  }
  cout << C << endl;

  return 0;
}
```

Задатак: Ексел

Аутор: Душан Пођадић

Такмичење: 2023/2024. квалификације 1, VIII разред, 2. задатак

Мајкрософт Ексел програм садржи табеле чији су редови означени бројевима (1, 2, 3 …), а колоне словима (A, B, C,… Y, Z). Када се у табелу дода превише колона (више од 26 колико слова има енглеска абецеда), колоне почињу да се означавају са два слова (AA, AB, AC …, AZ, BA, BB, BC … ZZ), затим са три (AAA, AAB, AAC … AAZ, ABA, ABB, … ZZZ) и тако даље. Ако знамо да је у табели последња колона означена карактером (или низом караткера) s, одредити колико колона има у табели.

Опис улаза

У првој линији стандардног улаза се уноси ниска карактера s. Сви карактери су велика слова енглеске абецеде и има их највише 6.

Опис излаза

У једини ред стандардног излаза исписати цео број који представља укупан број колона у табели.

Пример 1

Улаз	*Излаз*	*Објашњење*
`F`	`6`	Колоне редом иду: A, B, C, D, E i F - дакле 6 колона.

Пример 2

Улаз	*Излаз*
`AC`	`29`

Пример 3

Улаз	*Излаз*
`EXCEL`	`2708874`

Решење

Овај задатак се практично своди на пребацивање из бројног система са основом 26 у бројни систем са основом 10. То се ради тако што се тражени број иницијализује на 0 и онда се иде редом кроз цифре. Досада одређени број се множи са 26 (основом система из кога се пребацује) и додаје се вредност цифре. Треба обратити пажњу да колоне у екселу нису класичан систем у основи 26 јер не постоји 0 већ се креће од 1 ('A'). Једина модификација која је потребна у алгоритму да би се ово испоштовало је то да се при додавању нове цифре на број још додатно дода 1.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
   string kol;
   cin >> kol;

   int brk = 0;
   for (int i = 0; i < kol.length(); i++) {
      brk *= 26;
      brk += kol[i] - 'A' + 1;
   }

   cout << brk;
}
```

Задатак: Број дана одмереног тренинга

Аутор: Филип Марић

Такмичење: 2022/2023. квалификације 1, 7. разред, 3. задатак

Приликом припреме за предстојећа такмичења, спортиста сваког дана мери резултат који је постигао. Пошто жели да равномерно подиже форму, жели да сваки дан резултат буде све бољи (већи број представља бољи резултат). За неки дан тренинга ћемо рећи да је добар, ако је резултат постигнут тог дана строго бољи него претходног, а строго лошији него наредног (да би био добар, за први дан је довољно да је строго лошији од наредног, а за последњи дан да је строго бољи од претходног).

Напиши програм који израчунава број добрих дана.

Опис улаза

Са стандардног улаза се учитава број дана $n (3 \leq n \leq 100)$ током којих је спортиста тренирао, а у другом реду резултати које је спортиста постизао током сваког од тих n дана.

Опис излаза

На стандардни излаз исписати број добрих дана.

Пример 1

Улаз	*Излаз*
`6` `100 120 130 110 140 150`	`4`

Пример 2

Улаз	*Излаз*
`6` `10 20 30 40 50 60`	`6`

Решење

Елементе можемо учитати у низ. Након тога можемо анализирати све елементе низа од оног на другој позицији (редни број 1), до оног на претпоследњој позицији (редни број $n-2$) и за сваки од њих проверавати да ли је већи од претходног, а мањи од следећег елемента, увећавајући бројач ако је тај услов испуњен. Посебно треба проверити први елемент (редни број 0) и испитати да ли је он мањи од њему наредног и последњи елемент (редни број $n-1$) и испитати да ли је он већи од њему претходног.

Пошто се у сваком тренутку анализирају само три узастопна елемента низа коришћење низа се може избацити тако што се ти елементи памте у три променљиве (`prethodni`, `tekuci`, `sledeci`).

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  int broj = 0;
  int prethodni, tekuci;
  cin >> prethodni >> tekuci;
  if (prethodni < tekuci)
    broj++;
  for (int i = 2; i < n; i++) {
    int sledeci;
    cin >> sledeci;
    if (prethodni < tekuci && tekuci < sledeci)
      broj++;
    prethodni = tekuci;
    tekuci = sledeci;
  }
  if (prethodni < tekuci)
    broj++;
  cout << broj << endl;
  return 0;
}
```

Задатак: Растуће цифре

Такмичење: државно 2018/2019, V и VI разред, 3. задатак

Аутор: Филип Марић

Напиши програм који међу унетим бројевима проналази оне којима су цифре строго растуће, гледајући од цифре највеће тежине (прве цифре слева).

Опис улаза

Са стандардног улаза се учитава број n ($1 \leq n \leq 5000$), а затим и n природних бројева већих или једнаких од 10 и мањих или једнаких од 10^9, сваки у посебном реду

Опис излаза

На стандардни излаз исписати тражене бројеве са растућим цифрама, сваки у посебном реду.

Пример

Улаз	*Излаз*
3 123 222 321	123

Решење

Проверу да ли су све цифре броја растуће је најбоље реализовати у засебној функцији. Пошто је цифре једноставније цифре одређивати здесна налево (од цифре најмање тежине), алгоритам ће заправо проверавати да ли је текућа цифра строго мања од претходне (да ли су цифре строго опадајуће гледајући здесна налево).

Претходну цифру иницијализујемо на цифру јединица (која је једнака остатку при дељењу броја са 10), коју уклањамо из броја (целобројним дељењем са 10). Након тога улазимо у петљу која се извршава све док је број већи од 0. Одређујемо и уклањамо последњу цифру броја (поново коришћењем остатка и целобројног количника при дељењу са 10). Ако је текућа цифра већа или једнака од претходне, број нема строго растуће цифре (слева надесно). У супротном се припремамо за наредну итерацију тако што текућа цифра у овој постаје претходна цифра у наредној итерацији.

У главном програму само учитавамо серију од n бројева и филтрирамо је на основу функције која проверава да ли су цифре строго растуће.

```
#include <iostream>

using namespace std;

int rastuceCifre(int x) {
  int prethodna = x % 10; x /= 10;
  while (x > 0) {
    int tekuca = x % 10; x /= 10;
    if (tekuca >= prethodna)
      return false;
    prethodna = tekuca;
  }
  return true;
}

int main() {
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
```

```
    int x; cin >> x;
    if (rastuceCifre(x))
      cout << x << endl;
  }
  return 0;
}
```

Задатак: Долине

Аутор: Иван Дрецун

Такмичење: 2022/2023. квалификације 2, 6. разред, 6. задатак

Дат је низ надморских висина неког терена. Долина је подниз једнаких узастопних висина такав да су висина пре и висина после њега строго веће од њега. Напиши програм који одређује колико на датом терену постоји долина.

Опис улаза

Са стандардног улаза се уноси број делова терена n ($1 \leq n \leq 150$). У наредној линији се уносе n бројева одвојених размаком који представљају надморске висине терена. Све висине су цели бројеви из интервала $[0, 100]$.

Опис излаза

На стандардни излаз исписати један цео број који представља број долина.

Пример 1

Улаз	*Излаз*	*Објашњење*
`8` `5 3 3 7 4 4 4 8`	`2`	Долине су `5 3 3 7` и `7 4 4 4 8`.

Пример 2

Улаз	*Излаз*	*Објашњење*
`5` `4 3 1 2 4`	`1`	Једина долина је `3 1 2`.

Пример 3

Улаз	*Излаз*	*Објашњење*
`11` `2 2 3 4 4 2 2 2 5 4 5`	`2`	Долине су `4 2 2 2 5` и `5 4 5`.

Решење

Приликом учитавања низа висина можемо памтити два помоћна податка - висину тренутног дела и висину претходног дела терена. На пример, ако смо учитали податке `3 4 4 2 2 2`, висина тренутног дела је 2, док је висина претходног дела 4.

Када учитавамо следећу висину, потребно је да проверимо да ли је тренутни део долина. Нека смо у наставку поменутог примера учитали висину 6. Тада је тренутни део (део висине 2) долина - претходни део је висине 4, а следећи део је висине 6. Учитали смо податке `3 4 4 2 2 2 6` што значи да је сада претходни део висине 2, а тренутни висине 6.

Да смо уместо висине 6 на улазу имали висину 2 учитани низ висина би био `3 4 4 2 2 2 2`, што значи да се висине претходног и тренутног дела се не би мењале.

```cpp
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;

  int prethodni = -1;
  int trenutni = -1;
  int brojDolina = 0;
  for (int i = 0; i < n; i++) {
    int sledeci;
    cin >> sledeci;

    if (sledeci > trenutni && prethodni > trenutni)
      brojDolina++;

    if (sledeci != trenutni) {
      prethodni = trenutni;
      trenutni = sledeci;
    }
  }

  cout << brojDolina << endl;
  return 0;
}
```

Задатак: Врхови

Аутор: Иван Дрецун

Такмичење: 2022/2023. квалификације 3, 6. разред, 4. задатак

Напиши програм који за унети низ природних бројева одређује колико има врхова. Број је врх ако је строго већи од свог левог и десног суседа.

Опис улаза

Са стандардног улаза се уноси природан број н који представља величину низа. У наредном реду се уноси н целих бројева одвојених размаком.

Опис излаза

На стандардни излаз исписати колико у датом низу постоји врхова. Сматрати да је први елемент низа врх уколико је строго већи од десног суседа и да је последњи елемент низа врх уколико је строго већи од левог суседа.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
7 1 3 4 3 3 1 6	2	4 8 3 7 1	2	2 4 3	1

Решење

Опис решења

Потребно је једном петљом проћи кроз све елементе низа. Први и последњи елемент низа су специјални случајеви зато што немају и левог и десног суседа, па њих можемо обрадити засебно, а у оквиру петље обрађујемо све остале елементе.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int n;
  cin >> n;

  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  int brojVrhova = 0;

  // racunamo vrhove u sredini
  for (int i = 1; i < n - 1; i++)
    if (a[i] > a[i - 1] && a[i] > a[i + 1])
      brojVrhova++;

  // specijalno obradjujemo pocetak
  if (a[0] > a[1])
    brojVrhova++;

  // specijalno obradjujemo kraj
  if (a[n - 1] > a[n - 2])
    brojVrhova++;

  cout << brojVrhova << endl;

  return 0;
}
```

Задатак: Чудне степенице

Аутор: Иван Дрецун

Такмичење: окружно 2021/22., VI разред, 3. задатак

Познато је да постоје две врсте степеница. *Обичне степенице* су такве да је разлика у висини између свака два суседна степеника иста. *Чудне степенице* су такве да степеници могу бити разних висина, без обзира на висине суседних степеника. Низом целих бројева задат је облик чудних степеница. Ширина свих степеника је једнака, а висина i-тог степеника задата је i-тим

бројем у низу. Напиши програм који из тог низа издваја најдужи подниз (узастопних елемената) који представља обичне степенице.

Опис улаза

Са стандардног улаза се уноси дужина низа чудних степеница n ($2 \leq n \leq 60$). У наредном реду се уноси n целих бројева који представљају висине степеника (које могу бити и негативне), раздвојене размаком.

Опис излаза

На стандардни излаз исписати елементе најдужег подниза који представља обичне степенице, раздвојене размаком. Уколико постоји више оваквих поднизова, исписати најлевљи.

Пример

Улаз	*Излаз*	*Објашњење*
`9`	`1 3 5`	Постоје два подниза обичних степеница дужине 3, и то су `[1 3 5]`
`1 2 1 3 5 8 9 8 7`		и `[9 8 7]`, па се међу њима бира најлевљи.

Решење

У овом задатку је потребно одредити најдужи подниз бројева такав да су свака два суседна броја на истој раздаљини. На пример, ако је дат низ `4 6 4 2`, у поднизу `6 4 2` се свака два суседна елемента разликују за -2 ($4 - 6 = -2$, $2 - 4 = -2$).

Задатак је могуће решити једним проласком кроз низ. Потребно је памтити дужину и почетак тренутног подниза обичних степеница, као и исте те податке о најдужем таквом поднизу. Какав год да је низ чудних степеница, сигурно ће постојати обичне степенице дужине бар 2. На пример, ако су дате чудне степенице низом `1 4 2 5 6`, поднизови `1 4`, `4 2`, `2 5` и `5 6` сви чине низове обичних степеница. Због овога на почетку постављамо дужину подниза обичних степеница на 2, а почетак тог подниза на позицију 0. Након тога, почевши од позиције 2 у низу, крећемо се кроз низ и разматрамо два случаја.

Уколико је тренутно посматрани број удаљен од претходног исто колико је и претходни од њему претходног, онда тренутни подниз обичних степеница можемо продужити за један. На пример, ако је дат низ `1 3 5 7` и тренутно разматрамо број 5, њега можемо надовезати на подниз `1 3` да добијемо подниз `1 3 5`.

Са друге стране, ако је та разлика другачија, то значи да смо прешли на нови подниз обичних степеница. На пример, ако је дат низ `1 3 5 4 3`, у тренутку када разматрамо број 4, он се не може надовезати на претходни подниз обичних степеница `1 3 5`, па од тог тренутка посматрамо нови подниз `5 4`.

Приликом проласка кроз низ, у сваком тренутку памтимо податке о најдужем пронађеном поднизу обичних степеница, који по завршетку исписујемо.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int n;
  cin >> n;
```

```
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  int duzina = 2, pocetak = 0;
  int duzinaMax = 2, pocetakMax = 0;
  for (int i = 2; i < n; i++) {
    if (a[i] - a[i - 1] == a[i - 1] - a[i - 2])
      duzina++;
    else {
      duzina = 2;
      pocetak = i - 1;
    }

    if (duzina > duzinaMax) {
      duzinaMax  = duzina;
      pocetakMax = pocetak;
    }
  }

  for (int i = 0; i < duzinaMax; i++)
    cout << a[pocetakMax + i] << ' ';
  cout << '\n';

  return 0;
}
```

Задатак: Продужавање титлова

Аутор: Филип Марић

Такмичење: Ревијално такмичење 2019/2020., V и VI разред, 3. задатак

Станислав је ђак првог разреда и тек учи да чита. Покушава да прочита титлове на цртаном филму који гледа током карантина, али му често недостаје неколико секунди да би их прочитао до краја. Његова сестра жели да му помогне тако што ће продужити трајање сваког титла за пет секунди. Међутим, некада је пауза до наредног титла краћа од 5 секунди, па да се титлови не би преклапали, она титл продужава тачно до почетка наредног. Последњи титл се сигурно може продужити за 5 секунди. Пошто је филм дугачак, ово је напоран посао. Напиши програм који би Станислављевој сестри помогао да овај посао брже заврши.

Опис улаза

Са стандардног улаза се уноси укупан број титлова n ($1 \leq n \leq 1000$), и након тога у наредних n линија подаци о титловима. За сваки титл се уноси време почетка и време краја (у односу на почетак филма), раздвојени размаком. Свако време задато је у облику `hh:mm:ss`. Филм не траје дуже од 3 сата.

Опис излаза

На стандардни излаз исписати податке о продуженим титловима (у истом формату у ком су унети).

Пример

Улаз	*Излаз*
5	5
00:01:43 00:01:48	00:01:43 00:01:53
00:01:56 00:01:59	00:01:56 00:02:04
00:02:17 00:02:23	00:02:17 00:02:24
00:02:24 00:02:26	00:02:24 00:02:30
00:02:30 00:02:38	00:02:30 00:02:43

Решење

Подацима о времену најједноставније можемо баратати ако их претворимо у секунде (протекле од претходне поноћи). Једноставности ради, издвојићемо функције за учитавање и испис времена и података о титлу (време почетка и краја). Након учитавања података претворићемо их у секунде, а пре исписивања ћемо их претворити у сате, минуте и секунде.

Титлове ћемо обрађивати у петљи и у сваком тренутку ћемо одржавати податке о текућем и наредном титлу. Текући титл ћемо учитати пре петље, а затим ћемо у петљи $n - 1$ пута учитавати наредни титл, поправљати текући (знајући време почетка наредног) и након тога наредни проглашавати текућим за следећу итерацију. Сваки текући титл у петљи обрађујемо тако што препишемо време почетка и мање од увећаног времена завршетка и времена почетка наредног титла. Последњи титл ћемо обрадити након петље, тако што ћемо преписати време почетка и увећано време завршетка.

```
#include <iostream>
#include <iomanip>
#include <algorithm>

using namespace std;

int uSekunde(int h, int m, int s) {
  return h * 60 * 60 + m * 60 + s;
}

void odSekundi(int S, int& h, int& m, int& s) {
  s = S % 60; S /= 60;
  m = S % 60; S /= 60;
  h = S;
}

// ucitavamo vreme sa standardnog ulaza u formatu hh:mm:ss
int ucitajVreme() {
  int h, m, s;
  char c;
  cin >> h >> c >> m >> c >> s;
  return uSekunde(h, m, s);
}
```

```
// ucitavamo podatke o titlu (pocetak i kraj u formatu hh:mm:ss)
void ucitajTitl(int& start, int& end) {
  start = ucitajVreme();
  cin >> ws;
  end = ucitajVreme();
  cin >> ws;
}

// ispisujemo podatke o vremenu u formatu hh:mm:ss
void ispisiVreme(int S) {
  int h, m, s;
  odSekundi(S, h, m, s);
  cout << setfill('0') << setw(2) << h << ":"
       << setfill('0') << setw(2) << m << ":"
       << setfill('0') << setw(2) << s;
}

// ispisiujemo podatke o titlu (pocetak i kraj u formatu hh:mm:ss)
void ispisiTitl(int start, int end) {
  ispisiVreme(start);
  cout << " ";
  ispisiVreme(end);
  cout << endl;
}

int main() {
  // ukupan broj titlova
  int n;
  cin >> n;
  cout << n << endl;
  // ucitavamo podatke o tekucem titlu
  int start_t, end_t;
  ucitajTitl(start_t, end_t);
  for (int i = 0; i < n - 1; i++) {
    // ucitavamo podatke o narednom titlu
    int start_n, end_n;
    ucitajTitl(start_n, end_n);
    // obradjujemo podatke o tekucem titlu, uzevsi u obzir pocetak narednog
    ispisiTitl(start_t, min(end_t + 5, start_n));
    // naredni postaje tekuci za sledecu iteraciju
    start_t = start_n; end_t = end_n;
  }
  // poslednji titl se sigurno produzava
  ispisiTitl(start_t, end_t+5);
}
```

Уместо претварања у секунде, податке о времену можемо чувати и помоћу три независне промењиве, помоћу посебно дефинисане структуре, или, најбоље, помоћу уређене торке у коју се прво смешта сат, затим минут и на крају секунд. На тај начин се постиже једноставно поређење

времена. Увећање времена за одређени број секунди можемо извршити у засебној функцији у којој се примењује алгоритам сабирања позиционих записа.

```
#include <iostream>
#include <iomanip>
#include <tuple>
#include <algorithm>

using namespace std;

// dati vremenski trenutak vreme uvecava za dati broj ds sekundi
tuple<int, int, int> uvecajZaNekolikoSekundi(tuple<int, int, int> vreme, int ds) {
  tuple<int, int, int> rez = vreme;
  int& h = get<0>(rez);
  int& m = get<1>(rez);
  int& s = get<2>(rez);
  // uvecavamo sekunde i vrsimo popravke zbog prekoracenja
  s += ds;
  if (s >= 60) {
    m += s / 60;
    s = s % 60;
    if (m >= 60) {
      h += m / 60;
      m = m % 60;
    }
  }
  return rez;
}

// ucitavamo vreme sa standardnog ulaza u formatu hh:mm:ss
void ucitajVreme(tuple<int, int, int>& vreme) {
  char c;
  cin >> get<0>(vreme) >> c >> get<1>(vreme) >> c >> get<2>(vreme);
}

// ucitavamo podatke o titlu (pocetak i kraj u formatu hh:mm:ss)
void ucitajTitl(tuple<int, int, int>& start, tuple<int, int, int>& end) {
  ucitajVreme(start);
  cin >> ws;
  ucitajVreme(end);
  cin >> ws;
}

// ispisujemo podatke o vremenu u formatu hh:mm:ss
void ispisiVreme(const tuple<int, int, int>& vreme) {
  cout << setfill('0') << setw(2) << get<0>(vreme) << ":"
       << setfill('0') << setw(2) << get<1>(vreme) << ":"
       << setfill('0') << setw(2) << get<2>(vreme);
}
```

```
// ispisiujemo podatke o titlu (pocetak i kraj u formatu hh:mm:ss)
void ispisiTitl(const tuple<int, int, int>& start,
                const tuple<int, int, int>& end) {
  ispisiVreme(start);
  cout << " ";
  ispisiVreme(end);
  cout << endl;
}

int main() {
  // ukupan broj titlova
  int n;
  cin >> n;
  cout << n << endl;
  // ucitavamo podatke o tekucem titlu
  tuple<int, int, int> start_t, end_t;
  ucitajTitl(start_t, end_t);
  for (int i = 0; i < n - 1; i++) {
    // ucitavamo podatke o narednom titlu
    tuple<int, int, int> start_n, end_n;
    ucitajTitl(start_n, end_n);
    // obradjujemo podatke o tekucem titlu, uzevsi u obzir pocetak narednog
    ispisiTitl(start_t, min(uvecajZaNekolikoSekundi(end_t, 5), start_n));
    // naredni postaje tekuci za sledecu iteraciju
    start_t = start_n; end_t = end_n;
  }
  // poslednji titl se sigurno produzava
  ispisiTitl(start_t, uvecajZaNekolikoSekundi(end_t, 5));
}
```

Глава 7

Угнежђене петље

Задатак: К пута број к

Такмичење: окружно 2018/2019, V разред, 4. задатак

Аутор: Филип Марић

Анђа пише на папир бројеве од m до n тако да сваки број k (за који важи $m \leq k \leq n$) понови k пута. Напиши програм који за унете m и n исписује све бројеве које је Анђа записала.

Опис улаза

Са стандардног улаза се учитавају бројеви m и n ($1 \leq m \leq n \leq 100$, сваки у посебном реду.

Опис излаза

На стандрадни излаз исписати тражене бројеве (иза сваког броја исписати по један размак).

Пример

Улаз	*Излаз*
3 5	3 3 3 4 4 4 4 5 5 5 5 5

Решење

У спољашњој петљи чија је бројачка променљива k обрађујемо све бројеве до m до n. У унутрашњој петљи чије се тело понавља k пута исписујемо број k.

```
#include <iostream>

using namespace std;

int main() {
  int m, n;
  cin >> m >> n;
  for (int k = m; k <= n; k++)
    for (int i = 0; i < k; i++)
      cout << k << " ";
```

```
  return 0;
}
```

Задатак: К пола пута паран број к

Такмичење: окружно 2018/2019, VI разред, 4. задатак

Аутор: Филип Марић

Анђа пише на папир редом парне бројеве који су већи или једнаки m и мањи или једнаки n тако да сваки од тих бројева k понови $k/2$ пута. Напиши програм који за унете m и n раздвојене једним размаком исписује бројеве које је Анђа записала.

Опис улаза

Са стандардног улаза се учитавају бројеви m и n ($1 \leq m \leq n \leq 100$), раздвојени размаком.

Опис излаза

На стандардни излаз исписати тражене бројеве (иза сваког броја исписати по један размак).

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`3 6`	`4 4 6 6 6`	`1 3`	`2`	`5 8`	`6 6 6 8 8 8`

Решење

У петљи обрађујемо све бројеве до m до n. Ако је текући број паран, онда у унутрашњој петљи чије се тело понавља $k/2$ пута исписујемо број k.

```
#include <iostream>

using namespace std;

int main() {
  int m, n;
  cin >> m >> n;
  for (int k = m; k <= n; k++)
    if (k % 2 == 0)
      for (int i = 0; i < k / 2; i++)
        cout << k << " ";
  return 0;
}
```

Задатак: Серије вежби

Аутори: Филип Марић, Владимир Кузмановић

Такмичење: окружно 2022/2023, VI разред, 4. задатак

Пера вежба тако што n пута изводи вежбу која садржи k покрета. Да би лакше знао колико треба да вежба, он броји тако што на почетку сваког новог извођења вежбе каже редни број тог извођења, а затим наставља тако што каже редни број покрета у том извођењу вежбе. На

пример, ако 3 пута изводи вежбу од 4 покрета, он говори “јен два три чет, два два три чет, три два три чет”. Напиши програм који броји на овај начин.

Опис улаза

Са стандардног улаза се уноси број n ($2 \leq n \leq 100$) и k ($2 \leq k \leq 10$), сваки у посебном реду.

Опис излаза

На стандардни излаз исписати бројеве који се изговарају, за свако извођење вежбе у посебном реду.

Пример

Улаз	*Излаз*
3	1 2 3 4
4	2 2 3 4
	3 2 3 4

Решење

Опис главног решења

На почетку програма, прво треба да учитамо број понављања n и број покрета k којима је вежба дефинисана. Да бисмо урадили, задатак потребно је да пажљиво анализирамо текст задатка и описани поступак бројања.

На основу текста задатка лако се закључује да је потребно да бројимо понављања вежби, али и да за свако понављање треба да бројимо и покрете којима је та вежба дефинисана. Из овог описа директно се види да је решење заправо двострука или угнежђена петља. Спољашња петља ће бројати понављања, а унутрашња петља ће бројати кораке.

Бројач i у спољашњој петљи треба да крене од 1 и да се заустави када достигне вредност n. У телу петље треба да одштампамо вредност бројача i праћену размаком и затим да покренемо унутрашњу петљу. Бројач j у унутрашњој петљи треба да крене од 2 и да се заустави када достигне вредност k. У телу унутрашње петље треба да одштампамо вредност j праћену размаком. Након што се унутрашња петља заврши, у телу спољашње треба само да одштампамо нови ред.

Описани поступак је приказан у решењу.

```
#include <iostream>

using namespace std;

int main() {
  int n, k;
  cin >> n >> k;
  for (int i = 1; i <= n; i++) {
    cout << i << " ";
    for (int j = 2; j <= k; j++) {
      cout << j;
      if (j < k)
        cout << " ";
    }
    cout << endl;
```

```
  }
  return 0;
}
```

Задатак: Непарно-парни троугао

Аутор: Филип Марић

Такмичење: општинско 2018/2019, VII разред, 3. задатак

Врсте троугла садрже наизменично непарне, па затим парне бројеве. Прва врста садржи само број 1, а свака наредна један број више од претходне. Напиши програм који за дати број врста n исписује описани троугао.

Опис улаза

Са стандардног улаза се учитава број врста n ($1 \leq n \leq 100$).

Опис излаза

На стандардни излаз исписати троугао, врсту по врсту.

Пример

Улаз

```
4
```

Излаз

```
1
2 4
1 3 5
2 4 6 8
```

Решење

Задатак се решава угнежђеним петљама. У спољашњој петљи променљивом i пролазимо кроз редне бројеве троугла од 1 до n. У унутрашњој петљи набрајам елементе врсте чији је редни број i ако је i непаран, онда бројање у унутрашњој петљи почиње од 1, а ако је паран, почиње од 2. Ако је i непаран, тада је последњи елемент врсте број $2i-1$, а ако је паран, тада је последњи елемент врсте број $2i$. У оба случаја бројачка променљива у унутрашњој петљи се у сваком кораку увећава за 2.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  for (int i = 1; i <= n; i++) {
    for (int j = i % 2 == 0 ? 2 : 1; j <= 2*i; j += 2)
```

```
      cout << j << " ";
    cout << endl;
  }
  return 0;
}
```

Задатак: Цртеж

Аутор: Душан Пођадић

Такмичење: окружно 2021/22., V разред, 4. задатак

За унети непаран број n нацртати лептир машну. Лептир машна се црта од знакова - и * као на сликама у примерима.

У првом реду постоје две звездице на крајевима, а у сваком наредном реду се број звездица повећава за 2. Средишњи ред има све звездице, а затим број звездица опада и последњи ред поново има само две. Сваки ред има n карактера.

Опис улаза

У једином реду стандардног улаза се учитава број n ($5 \leq n \leq 1000$).

Опис излаза

У n редова стандардног излаза исписати по n карактера тако да се добије тражена фигура.

Пример 1

Улаз	*Излаз*
5	*---* **-** ***** **-** *---*

Пример 2

Улаз	*Излаз*
7	*-----* **---** ***-*** ******* ***-*** **---** *-----*

Пример 3

Улаз	*Излаз*
9	*-------* **-----** ***---*** ****-**** ********* ****-**** ***---*** **-----** *-------*

Решење

Да би се решио овај задатак потребно је проучити фигуру коју желимо да нацртамо. Приметимо да први ред фигуре има по једну звездицу на оба краја, други има по две, трећи по три и то се тако наставља до средњег $\left(\frac{n}{2}\right)$ реда. Средњи ред је попуњен звездицама, а након тога се број звездица на свакој страни поново смањује за 1 у сваком реду.

Уводимо променљиву brz која ће означавати колико се у неком реду са сваке стране налази звездица (на почетку је то 1) и затим ћемо повећавати вредност те променљиве док не стигнемо до половине свих редова, а затим ћемо смањивати за 1.

Када у унутрашњем циклусу исписујемо карактере, првих и последњих brz карактера (то су они за које важи $j < brz$ или $j \geq n - brz$) ће бити звездице, а између тога цртице.

```cpp
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;

  int brz = 1;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (j < brz || j >= n - brz)
        cout << "*";
      else
        cout << "-";
    }
    cout << endl;
    if (i < n / 2)
      brz++;
    else
      brz--;
  }

  return 0;
}
```

Задатак: Цртеж

Аутор: Душан Пођадић

Такмичење: окружно 2021/22., VI разред, 4. задатак

За унети непаран број n нацртати фигуру ромба са дијагоналама, тако да су му дијагонале паралелне координатним осама. Цртеж нацртати користећи знакове - и * као на сликама приказаним у примерима.

Опис улаза

У једином реду стандардног улаза се учитава број n ($5 \leq n \leq 1000$).

Опис излаза

У n редова стандардног излаза исписати по n карактера тако да се добије тражена фигура.

Пример 1

Улаз	*Излаз*
5	--*-- -***- ***** -***- --*--

Пример 2

Улаз	*Излаз*
7	---*--- --***-- -*-*-*- ******* -*-*-*- --***-- ---*---

Пример 3

Улаз	*Излаз*
9	----*---- ---***--- --*-*-*-- -*--*--*- ********* -*--*--*- --*-*-*-- ---***--- ----*----

Решење

Да би се решио овај задатак потребно је проучити фигуру коју желимо да нацртамо. Приметимо да у првом и последњем реду имамо звездице на средини, у средњем реду имамо само звездице, док у осталим редовима имамо по 3 звездице од којих је једна увек у средини, а друге две су симетричне у односу на средину. У другом реду су те звездице одмах до средине и у сваком наредном реду се удаљавају за по једно место. Када се прође средњи ред, тада се звездице више не удаљавају од средине, већ се приближавају.

Уводимо променљиву r која означава удаљеност додатних звездица од средине реда. Та удаљеност је на почетку 0 (у првом реду нема додатних звездица, па можемо да кажемо да се оне налазе на удаљености 0 од средине), а онда се повећава за 1 како пролазимо кроз редове. Када прођемо средњи ред, удаљеност се више не повећава, већ се смањује за 1 у сваком реду.

Када у унутрашњем циклусу исписујемо карактере, проверавамо да ли се налазимо у средњем реду ($i == s$), на средини реда ($j == s$) или на удаљености r од средине реда ($abs(j-s) == r$). У наведеним случајевима исписујемо звездицу, а у свим осталим исписујемо цртицу.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;

  int r = 0;
  int s = n / 2;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (abs(j - s) == r || j == s || i == s)
        cout << "*";
      else
        cout << "-";
    }
    cout << endl;
    if (i < s)
      r++;
    else
```

```
      r--;
  }
  return 0;
}
```

Задатак: Рам од слова

Аутор: Филип Марић

Такмичење: 2021/22. квалификације 3, VII разред, 3. задатак и VIII разред, 1. задатак

Напиши програм који исцртава рам на основу слова дате речи.

Опис улаза

Са стандардног улаза се учитава реч састављена од малих слова енглеске абецеде (њих између 3 и 20).

Опис излаза

На стандардни излаз исписати рам формиран од слова учитане речи и тачкица. У првом реду рама исписана је учитана реч, у последњем та реч је исписана у обратном редоследу у првој колони су слова учитане речи исписана једно испод другог у учитаном редоследу, а у последњој колони у обратном редоследу.

Пример 1

Улаз	*Излаз*

`abc`	`abc`
	`b.b`
	`cba`

Пример 2

Улаз	*Излаз*

`abcd`	`abcd`
	`b..c`
	`c..b`
	`dcba`

Решење

Размотримо позиције слова која се исписују.

```
0   1   2   ...  n-3 n-2  n-1
1   .   .   ...   .   .   n-2
2   .   .   ...   .   .   n-3
... ... ... ...  ... ...  ...
n-3 .   .   ...   .   .    2
n-2 .   .   ...   .   .    1
n-1 n-2 n-3 ...   2   1    0
```

Примећујемо да се први и последњи ред разликују од унутрашњих редова, тако да ћемо испис организовати у три фазе (први ред, сви средњи редови и последњи ред).

Прво учитавамо реч. Затим је исписујемо. Након тога у петљи са бројачем i исписујемо свако слово речи од позиције 1, до позиције $n-2$, где је n дужина учитане речи. Након тога исписујемо $n-2$ тачке (коришћењем петље или изградњом ниске), а затим и слово на позицији $n-i-1$ (када је $i=1$, исписујемо слово на позицији $n-2$, када је $i=2$, исписујемо слово на позицији $n-3$ итд., а када је $i=n-2$ исписујемо слово на позицији 1. На крају, у последњем реду, исписујемо слова речи у обратном редоследу.

```cpp
#include <iostream>
#include <string>

using namespace std;

int main() {
  string rec;
  cin >> rec;
  int n = rec.size();
  // prvi red
  cout << rec << endl;
  // unutrašnji redovi
  for (int i = 1; i < n - 1; i++) {
    cout << rec[i];
    for (int j = 0; j < n - 2; j++)
      cout << ".";
    cout << rec[n - 1 - i];
    cout << endl;
  }
  // poslednji red
  for (int j = n - 1; j >= 0; j--)
    cout << rec[j];
  cout << endl;

  return 0;
}
```

Глава 8

Основне структуре података

Задатак: Јамб

Аутор: Филип Марић

Такмичење: државно 2022/2023, VIII разред, 2. задатак

У игри јамб се баца 5 коцкица са бројевима од 1 до 6. Потребно је написати програм који ће проверити да ли је на коцкицама постигнут неки од следећих исхода:

- Пет истих тј. јамб (`5`): ово значи да су на свих 5 коцкица исти бројеви. На пример: 6, 6, 6, 6, 6
- Четири исте тј. каро (`4`): ово значи да су на 4 коцкице исти бројеви (не и на свих 5). На пример: 5, 1, 5, 5, 5
- Три исте и две исте тј. фул (`3+2`): ово значи да постоје три коцкице на којима су исти бројеви и да су на остале две коцкице опет исти бројеви (али није свих 5 бројева исто). На пример: 3, 2, 3, 2, 3
- Три исте тј. трилинг (`3`): ово значи да постоје три коцкице на којима су исти бројеви, док се на преостале две коцке налазе неки други, различити бројеви. На пример: 4, 2, 4, 4, 3
- Две исте тј. пар (`2`): ово значи да постоје две коцкице на којима су исти бројеви, док се на преостале три коцке налазе неки други, различити бројеви. На пример: 4, 5, 6, 4, 2
- Две исте и две исте тј. два пара (`2+2`): ово значи да постоје две коцкице на којима су исти бројеви и још две коцкице на којима су исти бројеви (али различити од бројева на прве две исте коцкице), а да се на преосталој, петој коцкици налази неки трећи број.
- Све различите (`-`): ово значи да су на свим коцкицама различити бројеви.

Опис улаза

Са стандардног улаза се учитава 5 бројева између 1 и 6, раздвојених размаком.

Опис излаза

На стандардни излаз исписати исход бацања тј. једну од следећих порука `5`, `4`, `3+2`, `3`, `2+2`, `2`, `-`.

Пример 1		**Пример 2**		**Пример 3**		**Пример 4**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`2 2 3 2 2`	`4`	`5 6 5 6 5`	`3+2`	`1 2 3 4 3`	`2`	`2 2 3 4 3`	`2+2`

Пример 5

Улаз	*Излаз*
3 3 2 5 3	3

Решење

Задатак можемо решити тако што пребројимо колико се пута који број јавио на коцкицама. Након тога се програм једноставно довршава анализом разних случајева.

Алтернативно, могуће је и сортирати низ коцкица.

```
#include <iostream>

using namespace std;

int main() {
  int kockice[5];
  for (int i = 0; i < 5; i++)
    cin >> kockice[i];

  int broj[7] = {0};
  for (int i = 0; i < 5; i++)
    broj[kockice[i]]++;

  bool pet = false, cetiri = false, tri = false;
  int parova = 0;
  for (int i = 1; i <= 6; i++)
    if (broj[i] == 5)
      pet = true;
    else if (broj[i] == 4)
      cetiri = true;
    else if (broj[i] == 3)
      tri = true;
    else if (broj[i] == 2)
      parova++;

  if (pet)
    cout << "5" << endl;
  else if (cetiri)
    cout << "4" << endl;
  else if (tri && parova > 0)
    cout << "3+2" << endl;
  else if (tri)
    cout << "3" << endl;
  else if (parova == 2)
    cout << "2+2" << endl;
  else if (parova == 1)
    cout << "2" << endl;
  else
    cout << "-" << endl;
```

```
  return 0;
}
```

Задатак: Највише одличних

Аутор: Милан Вуīелија

Такмичење: 2022/2023. квалификације 2, 6. разред, 5. задатак

Написати програм који за дати разред и успех сваког од n ученика исписује разред у коме има највише одличних ученика.

Опис улаза

У првом реду стандардног улаза је цео број n $(1 \leq n \leq 100)$. У сваком од наредних n редова налази се један природан и један реалан број раздвојени размаком. Природан број је из интервала $[1, 8]$ и представља разред, а реалан број је из интервала $[1.0, 5.0]$ и представља успех. Број једнак или већи од 4.5 представља одличан успех.

Опис излаза

На стандардни излаз исписати један цео број, број разреда са највише одличних ученика. Уколико одговр није једнозначан, исписати број највећег разреда са највише одличних ученика.

Пример

Улаз	*Излаз*
6 4 3.58 4 4.67 5 4.83 6 4.5 4 4.93 6 5.0	6

Решење

Задатак се најлакше решава помоћу низа бројача, где сваки елемент низа броји одличне ученике у једном разреду. Док читамо податке, можемо успут да бројимо одличне ученике у сваком разреду. Након тога, у једном проласку кроз низ бројача налазимо максимум низа (највећи број одличних у једном разреду), а уједно и највећи разред који је достигао тај број одличних ученика.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int n, razred;
  vector <int> brOdlicnih(9);
  double uspeh;
  cin >> n;
```

```
  for (int i = 0; i < n; i++) {
    cin >> razred >> uspeh;
    if (uspeh >= 4.5)
      brOdlicnih[razred]++;
  }

  int najRazred = 0;
  for (int razred = 1; razred <= 8; razred++)
    if (brOdlicnih[razred] >= brOdlicnih[najRazred])
      najRazred = razred;
  cout << najRazred << endl;
  return 0;
}
```

Задатак: Словарица

Аутор: Милан Вуīделија

Такмичење: 2022/2023. квалификације 1, 7. разред, 4. задатак

Ана учи да саставља речи од слова. Данас жели да од слова која има на располагању састави што више примерака речи коју је смислила. Написати програм који одређује колико пута дата реч може да се састави од слова дате колекције.

Опис улаза

У првом реду стандардног улаза налази се ниска карактера која представља колекцију, а у другом ниска која представља реч. Обе ниске се састоје искључиво од малих слова енглеске абецеде и дужина им је до 50000 карактера.

Опис излаза

На стандардни излаз исписати један цео број, број могућих састављања дате речи.

Пример 1

Улаз	*Излаз*
`matematikaiprogramiranje` `meta`	`2`

Пример 2

Улаз	*Излаз*
`nekavelikakolekcijaslova` `kea`	`3`

Пример 3

Улаз	*Излаз*
`babanedanijevisesedapajenekogleda` `deda`	`1`

Решење

Нека се одређено слово појављује k пута у колекцији и $r > 0$ пута у речи. Тада овог слова има довољно за $\left\lfloor \frac{k}{r} \right\rfloor$ понављања речи, тј. није могуће саставити више понављања речи. Према томе, број речи који може да се састави је једнак најмањем од ових количника по свим словима која се појављују у речи.

Да бисмо добили ефикасно решење, најбоље је да се прво преброје појављивања сваког слова у колекцији и у речи, а затим од највише 26 количника да се нађе најмањи.

```cpp
#include <iostream>

using namespace std;

void Prebroj(string s, int br[]) {
    for (char c : s)
        br[c-'a']++;
}
int main() {
    string a, b;
    cin >> a >> b;
    int brPoj = a.size() / b.size();
    int kolekcija[26] = {0}, rec[26] = {0};
    Prebroj(a, kolekcija);
    Prebroj(b, rec);
    for (int i = 0; i < 26; i++)
        if (rec[i] > 0) {
            if (kolekcija[i] == 0) { brPoj = 0; break; }
            else brPoj = min(brPoj, kolekcija[i] / rec[i]);
        }

    cout << brPoj << endl;
    return 0;
}
```

Задатак: Скочко

Аутор: Филип Марић

Такмичење: 2021/22. квалификације 2, VI разред, 5. задатак

У игри "Скочко" играч погађа непознату комбинацију S, која се састоји од n симбола (једноставности ради, претпоставимо да су то цифре од 1 до 9). Након што играч напише свој низ P од n симбола, рачунар треба да му каже колико постоји симбола у непознатој комбинацији S, који се поклапају са симболима у низу P. При том се посебно броје симболи који су на истој позицији у оба низа, а посебно симболи из непознате комбинације који се налазе у низу P, али нису на истој позицији.

Опис улаза

Са стандардног улаза се уносе два низа симбола дужине n ($1 \leq n \leq 15$). Први је непозната комбинација S, а други је низ P, који је унео играч покушавајући да погоди непознату комбинацију.

Опис излаза

На стандардни излаз исписати број погођених симбола који су на одговарајућим позицијама и број погођених симбола који нису на одговарајућим позицијама (два броја раздвојена размаком).

Пример 1

Улаз	*Излаз*
12345	1 4
54321	

Пример 2

Улаз	*Излаз*
112233	6 0
112233	

Пример 3

Улаз	*Излаз*
112233	3 3
132123	

Пример 4

Улаз	*Излаз*
111111	1 0
122222	

Пример 5

Улаз	*Излаз*
111122	0 2
222233	

Пример 6

Улаз	*Излаз*
1222	0 1
2333	

Решење

Решење заснивамо на бројању појављивања сваке цифре. Учитавамо две ниске карактера, и затим бројимо колико пута се свака од 10 цифара јавила у првој и у другој ниски. Након тога одређујемо колико се цифара јавља на одговарајућим позицијама, тако што обрађујемо редом једну по једну позицију у нискама и поредимо карактере на тој позицији. Пошто те цифре желимо да изузмемо приликом одређивања цифара који се поклапају, али нису на одговарајућим позицијама, умањиваћемо број појављивања пронађених поклапајућих карактера. Број цифара које се поклапају, али нису на правом месту ћемо одредити тако што ћемо за сваку цифру пронаћи минимум броја појављивања у обе ниске (након уклањања појављивања на одговарајућим позицијама) и сабраћемо тако добијене минимуме за све цифре.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  string resenje, pogodak;
  cin >> resenje >> pogodak;

  // koliko puta se svaka cifra javlja u resenju i u pogotku
  int brojResenje[10] = {0};
  for (char c : resenje)
    brojResenje[c - '0']++;

  int brojPogodak[10] = {0};
  for (char c : pogodak)
    brojPogodak[c - '0']++;

  // koliko je cifara na pravom mestu (njih izuzimamo iz brojanja
  // pogodjenih cifara koje su van pravog mesta)
  int naPravomMestu = 0;
  for (int i = 0; i < resenje.size(); i++)
    if (resenje[i] == pogodak[i]) {
      naPravomMestu++;
      brojPogodak[resenje[i] - '0']--;
      brojResenje[resenje[i] - '0']--;
    }

  // racunamo broj cifara van pravog mesta (cifre koje su na pravom
```

```
  // mestu su vec izuzete)
  int vanPravogMesta = 0;
  for (int i = 0; i < 10; i++)
    vanPravogMesta += min(brojResenje[i], brojPogodak[i]);

  cout << naPravomMestu << " " << vanPravogMesta << endl;
  return 0;
}
```

Задатак: Најсрећнији дан

Аутор: Филип Марић

Такмичење: општинско 2018/2019, VII разред, 4. задатак, VIII разред, 3. задатак

Лидија је за сваку петицу коју је добила током једног полугодишта записала име радног дана у коме је ту петицу добила. Напиши програм који одређује који јој је био најсрећнији дан током тог полугодишта (претпостави слободно да је добијала различит број оцена за сваки од 5 радних дана).

Опис улаза

Учитава се број петица n ($1 \leq n \leq 100$) које је добила и затим ознаке дана `pon`, `uto`, `sre`, `cet`, `pet`, у којима је те петице добијала свака у посебном реду.

Опис излаза

На стандардни излаз исписати ознаку најсрећнијег дана.

Пример

Улаз	*Излаз*
8 pon sre cet pon pon pet cet sre	pon

Решење

Потребно је да пребројимо колико се пута који дан појавио на улазу. То је најједноставније урадити коришћењем мапе (речника). За сваки учитани дан увећавамо одговарајући елемент у мапи.

У другој фази одређујемо најсрећнији дан алгоритмом одређивања максимума. Пролазимо редом кроз све елементе мапе и бележимо дан који се најчешће појављује. Ако текући дан има више појављивања од до тада најчешћег, ажурирамо најчешћи дан.

```
#include <iostream>
#include <string>
```

```cpp
#include <map>

using namespace std;

int main() {
  int n;
  cin >> n;

  map<string, int> brojPetica;
  for (int i = 0; i < n; i++) {
    string dan;
    cin >> dan;
    brojPetica[dan]++;
  }

  int max = 0;
  string max_dan;
  for (auto it : brojPetica)
    if (it.second > max) {
      max = it.second;
      max_dan = it.first;
    }

  cout << max_dan << endl;

  return 0;
}
```

Задатак: Речник

Аутор: Филип Марић

Такмичење: државно 2022/2023, VII разред, 2. задатак

Напиши програм који имплементира српско-енглески речник. Прво се учитава списак речи и њихових превода, а затим корисник од система тражи да му се преведу неке речи (било са српског на енглески, било са енглеског на српски).

Опис улаза

Са стандардног улаза се уноси број $n (1 \leq n \leq 50\,000)$ парова речи на српском и енглеском језику. Све речи на српском су међусобно различите и све речи на енглеском су међусобно различите. Након тога се уноси $m (1 \leq m \leq 50\,000)$ упита. Сваки упит садржи жељени језик (или `sr` или `en`) и затим реч на супротном језику коју треба превести.

Опис излаза

На стандардни излаз исписати све преводе. Ако за неку реч није унет превод, исписати `?`.

Пример

Улаз	*Излаз*
3 cat macka dog pas mouse mis 4 en pas sr mouse sr lion en mis	dog mis ? mouse

Решење

Као што име задатка каже, задатак се најлакше решава ако се употреби структура података речник (тј. мапа, асоцијативни низ). У једном речнику пресликавамо речи на српском у речи на енглеском, а у другом речи на енглеском у речи на српском језику.

```
#include <iostream>
#include <map>

using namespace std;

int main() {
  map<string, string> sr_en;
  map<string, string> en_sr;
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    string en, sr;
    cin >> en >> sr;
    sr_en[sr] = en;
    en_sr[en] = sr;
  }

  int m;
  cin >> m;
  for (int i = 0; i < m; i++) {
    string jezik, rec;
    cin >> jezik >> rec;
    if (jezik == "en") {
      if (sr_en.count(rec))
        cout << sr_en[rec] << endl;
      else
        cout << "?" << endl;;
    } else if (jezik == "sr") {
      if (en_sr.count(rec))
        cout << en_sr[rec] << endl;
      else
        cout << "?" << endl;
```

```
    }
  }

  return 0;
}
```

Задатак: Аритмогриф

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 2, VII и VIII разред, 4. задатак

При решавању једне познате врсте аритметичких ребуса потребно је иста слова заменити истим цифрама, а различита слова различитим цифрама, тако да се добије тачна једнакост. На пример, за ребус $SNEG + KRUG = SPORT$ једно од решења је $1937 + 8627 == 10564$. Можемо се уверити да је наведена једнакост заиста тачна, али то није тема овог задатка. Овде нас интересује други услов, а то је да ли је замена слова цифрама правилно изведена. На пример, понуђено "решење" $4832 + 5962 == 10794$ није исправно (иако је, узгред речено, једнакост тачна) зато што је слово S на једном месту замењено цифтом 4, а на другом цифром 1. Такође, цифра 4 одговара двама различитим словима (S и T).

Написати програм који проверава да ли је у овом типу ребуса замена слова цифрама правилно изведена.

Опис улаза

Са стандардног улаза се у првом реду уноси стринг, који представља описани аритметички ребус. Стринг може да садржи велика слова енглеске абецеде, размаке, знаке математичких операција и и знак једнакости. Укупна дужина стринга није већа од 100 а број различитих слова која се појављују у стрингу није већи од 10.

У другом реду се уноси стринг који представља понуђено решење ребуса из првог реда. Други стринг се од првог разликује искључиво по томе што је свако слово замењено неком цифром.

Опис излаза

На стандардни излаз исписати "da" ако је замена правилна (без обзира на то да ли је једнакост тачна), а "ne" ако замена није правилна.

Пример 1

Улаз	*Излаз*	*Објашњење*
`SNEG + KRUG = SPORT` `2341 + 8751 = 20679`	`da`	Иста слова су замењена истим цифрама а различита слова различитим цифрама, што значи да је сама замена правилна (мада једнакост није тачна).

Пример 2

Улаз	*Излаз*
`SNEG + KRUG = SPORT` `4832 + 5962 = 10794`	`ne`

Решење

Задатак ћемо свакако решавати пролазећи истовремено кроз оба дата стринга. Пошто пролазећи кроз дате стрингове словима ребуса придружујемо цифре понуђеног решења, природно је да

формирамо речник у коме памтимо та придруживања.

Када у петљи дохватимо текуће слово и текућу цифру, интересује нас да ли је ово прво појављивање тог слова.

Ако текуће слово није у речнику, ово је његово прво појављивање. Сада је важно да ли се и текућа цифра први пут појављује. Да бисмо то установили, користићемо скуп искоришћених цифара. Ако цифра није у скупу, то је њено прво појављивање и можемо у речнику да текућем слову придружимо текућу цифру, а у скуп искоришћених цифара да додамо текућу цифру. Ако је цифра већ била у скупу, то значи да понуђено решење не ваља.

Ако је текуће слово већ у речнику, онда само проверавамо да је њему придружена вредност из речника једнака текућој цифри. Ако није једнака, онда то поново значи да понуђено решење не ваља.

На крају, ако нисмо констатовали да понуђено решење не ваља, онда је замена слова цифрама исправна.

```
#include <iostream>
#include <map>
#include <set>

using namespace std;

int main() {
  string rebus, resenje;
  getline(cin, rebus);
  getline(cin, resenje);

  map<char, int>  zamene;
  set<char> iskoriscene_cifre;
  bool dobraZamena = true;

  for (size_t i = 0; i < rebus.size(); i++) {
    auto cifra_it = zamene.find(rebus[i]);
    if (cifra_it == zamene.end()) {
      if (iskoriscene_cifre.find(resenje[i]) != iskoriscene_cifre.end()) {
        dobraZamena = false;
        break;
      }
      zamene[rebus[i]] = resenje[i];
      iskoriscene_cifre.insert(resenje[i]);
    } else if (cifra_it->second != resenje[i]) {
      dobraZamena = false;
      break;
    }
  }
  if (dobraZamena)
    cout << "da" << endl;
  else
    cout << "ne" << endl;
```

```
}
```

Друга могућност је да у речнику за свако слово памтимо све цифре које су том слову придружене. Дакле кључеви тог речника су слова, а вредности су скупови цифара. Након формирања речника проверавамо да су сви скупови цифара једночлани (ако нису, понуђено решење не ваља). Овом провером смо потврдили да је сваком слову придружена тачно једна цифра, али не снамо да ли су различитим словима придружене различите цифре. Да бисмо то проверили, можемо да претходну проверу ставимо у функцију, а затим да функцију позовемо још једном, али са датим стринговима (ребус и понуђено решење) у замењеним улогама. Тако за сваку цифру добијамо скуп слова којима је та цифра придружена. Ако су и сви ови скупови једночлани, тиме потврђујемо и да је свака цифра придружена тачно једном слову, што значи да је замена слова цифрама исправна.

```
#include <iostream>
#include <map>
#include <set>

using namespace std;

bool dobra_zamena(const string& a, const string& b) {
  map<char, set<char>> zamene;
  for (int i = 0; i < a.length(); i++)
    zamene[a[i]].insert(b[i]);

  for (int i = 0; i < a.length(); i++)
    if (zamene[a[i]].size() != 1)
      return false;
  return true;
}

int main() {
  string rebus;
  getline(cin, rebus);
  string resenje;
  getline(cin, resenje);
  if (dobra_zamena(rebus, resenje) && dobra_zamena(resenje, rebus))
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

Задатак: ATP победник

Аутор: Филип Марић

Такмичење: 2019/20 квалификације 3, V и VI разред, 6. задатак

Током године играју се многи тениски турнири на којима тенисери освајају поене. Поени се сабирају и на крају године објављује се завршна листа на којој су тенисери рангирани на основу

укупног броја поена током те године. Напиши програм који на основу резултата свих турнира одређује најбољег тенисера и његове поене (претпоставити да ће победник имати строго већи број поена од свих осталих).

Опис улаза

Са стандардног улаза се уноси број n, а затим у наредних n редова подаци о освојеним поенима тенисера: презиме тенисера и затим број освојених поена.

Опис излаза

На стандардни излаз исписати презиме и укупан број поена победника.

Пример 1

Улаз	*Излаз*

`9`
`Djokovic 1000`
`Nadal 800`
`Federer 600`
`Nadal 1000`
`Federer 800`
`Djokovic 600`
`Djokovic 1000`
`Cicipas 800`
`Nadal 600`
	`Djokovic 2600`

Пример 2

Улаз	*Излаз*

`6`
`Zverev 1000`
`Cicipas 800`
`Medvedev 700`
`Thiem 1000`
`Cicipas 800`
`Medvedev 600`
	`Cicipas 1600`

Решење

У првој фази програма је потребно да израчунамо укупан број поена за сваког играча. Податке можемо чувати у мапи која пресликава име играча у његов број поена. Након тога одређујемо највећу вредност у мапи (што можемо урадити било применом библиотечке функције, било проласком кроз све елементе мапе и ажурирањем максимума).

```
#include <iostream>
#include <algorithm>
#include <map>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  int n;
  cin >> n;
  map<string, int> bodovi;
  for (int i = 0; i < n; i++) {
    string teniser; int bodovi_na_turniru;
    cin >> teniser >> bodovi_na_turniru >> ws;
    bodovi[teniser] += bodovi_na_turniru;
  }

  auto it = max_element(bodovi.begin(), bodovi.end(),
                    [](const pair<string, int>& p1,
```

```
                        const pair<string, int>& p2) {
                      return p1.second < p2.second;
                    });
  cout << it->first << " " << it->second << endl;
  return 0;
}
```

Задатак: Хороскоп

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 2, VII и VIII разред, 3. задатак

Јединствени матични број грађана (скраћено ЈМБГ) је низ од 13 цифара, који у нашој земљи свако добија још као сасвим мали. Прве две цифре овог низа представљају дан, а следеће две месец рођења. На пример, ако нечији ЈМБГ почиње са 0904, он је рођен деветог априла, а ако почиње са 3112 он је рођен тридесет првог децембра.

Перица воли да се забавља читајући разне хороскопе, а у једном тренутку се запитао колико има људи рођених у сваком од хороскопских знакова, за које би требало да важи оно што пише у хороскопу. Перица је некако успео да дође до повеће листе матичних бројева, а зна и почетне и завршне датуме за сваки знак:

- ОВАН – од 21. марта до 20. априла
- БИК – од 21. априла до 21. маја
- БЛИЗАНЦИ – од 22. маја до 21. јуна
- РАК – од 22. јуна до 22. јула
- ЛАВ – од 23. јула до 22. августа
- ДЕВИЦА – од 23. августа до 22. септембра
- ВАГА – од 23. септембра до 23.октобра
- ШКОРПИЈА – од 24. октобра до 22. новембра
- СТРЕЛАЦ – од 23. новембра до 21. децембра
- ЈАРАЦ – од 22. децембра до 20. јануара
- ВОДОЛИЈА – од 21. јануара до 19. фебруара
- РИБЕ – од 20. фебруара до 20. марта

Перица сада жели да преброји рођења у сваком од хороскопских знакова. Напишите програм који помаже Перици у бројању.

Опис улаза

На стандардном улазу се у првом реду налази број N $(1 \leq N \leq 200)$, број матичних бројева које је Перица набавио. У сваком од N наредних редова је по један матични број (13 цифара без размака).

Опис излаза

На стандардни излаз исписати 12 целих бројева, сваки у посебном реду. Први број је број особа рођени у знаку овна, други - у знаку бика, итд. Последњи, дванаести број је број особа рођених у знаку рибе.

Пример

Улаз	*Излаз*
5	2
1504279718463	0
0412622712918	0
1903326718649	0
0710262713307	0
0104646719372	0
	1
	0
	1
	0
	0
	1

Решење

Потребно је за сваки прочитани матични број одредити датум рођења особе, а затим за тај датум рођења одредити одговарајући хороскопски знак. Ово можемо да урадимо на различите начине. Један начин је да прво направимо списак уређених тројки, које се састоје од индекса хороскопског знака, и дана и месеца када се тај знак завршава. Тада је довољно да у петљи за текући ЈМБГ издвојимо месец и дан рођења особе и поредимо га редом са завршним датумима хороскопских знакова, док датум рођења не буде мањи од краја неког знака (пошто је број знакова веома мали, можемо слободно употребљавати линеарну претрагу). Када се то догоди, знамо којем знаку припада текући датум рођења. Датуме можемо представити било уређеним паровима (месец, дан), било их кодирати целим бројевима по принципу 100 пута месец + дан.

Да бисмо довршили задатак, можемо да уведемо листу од 12 бројача, тако да сви почињу од 0, а када за текућу особу установимо хороскопски знак, увећавамо за 1 бројач који одговара том знаку.

```
#include <iostream>
#include <string>
#include <vector>
#include <tuple>
using namespace std;

int main() {
  enum Znak {OVAN = 0, BIK, BLIZANCI, RAK, LAV, DEVICA,
             VAGA, SKORPIJA, STRELAC, JARAC, VODOLIJA, RIBE};

  // datume kodiramo kao 100 * mesec + dan
  vector<tuple<Znak, int>> poslednjiDanZnaka {
    make_tuple(JARAC, 120), // JARAC do 20. januara
    make_tuple(VODOLIJA, 219), // VODOLIJA do 21. februara
    make_tuple(RIBE, 320), // RIBE do 20 marta
    make_tuple(OVAN, 420), // OVAN do 20. aprila
    make_tuple(BIK, 521), // BIK do 21. maja
    make_tuple(BLIZANCI, 621), // BLIZANCI do 21. juna
    make_tuple(RAK, 722), // RAK do 22. jula
```

```
    make_tuple(LAV, 822), // LAV do 22. avgusta
    make_tuple(DEVICA, 922), // DEVICA do 22. septembra
    make_tuple(VAGA, 1023), // VAGA do 23. oktobra
    make_tuple(SKORPIJA, 1122), // SKORPIJA do 22. novembra
    make_tuple(STRELAC, 1221),  // STRELAC do 21. decembra
    make_tuple(JARAC, 1231)  // JARAC do 31. decembra
  };
  vector<int> broj(12, 0);

  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    // ucitavamo jmbg i izdvajamo datum rodjenja osobe
    string jmbg;
    cin >> jmbg;
    int dan = stoi(jmbg.substr(0, 2));
    int mesec = stoi(jmbg.substr(2, 2));
    int datum = 100 * mesec + dan;
    // linearnom pretragom pronalazimo znak osobe
    for (int znak = 0; znak <= 12; znak++)
      if (datum <= get<1>(poslednjiDanZnaka[znak])) {
        broj[get<0>(poslednjiDanZnaka[znak])]++;
        break;
      }
  }

  // ispisujemo broj osoba svakog znaka
  for (int i = 0; i < 12; i++)
    cout << broj[i] << endl;

  return 0;
}
```

Задатак: Датум са највећом зарадом

Аутор: Филип Марић

Такмичење: 2022/2023. квалификације 1, 8. разред, 4. задатак

Познати су сви фискални рачуни које је издала једна књижара. Са сваког рачуна се може прочитати датум када је тај рачун издат и износ који је наплаћен. Напиши програм који одређује највећи укупан износ који је књижара наплатила у једном дану, тј. максималан дневни пазар, као и датум када је тај износ био највећи (ако има више таквих датума, исписати их све, уређене хронолошки од првог до последњег датума).

Опис улаза

Са стандардног улаза се учитава број издатих рачуна n ($1 \leq n \leq 10^5$), а затим подаци о сваком рачуну у облику `dd-mm-gggg iznos`, где је износ реалан број заокружен на две децимале.

Опис излаза

На стандардни излаз исписати максимални износ (реалан број заокружен на две децимале), а затим у наредним редовима датуме када је тај максимални износ наплаћен. Ако има више датума, они треба да буду исписани редом од најстаријег до најновијег.

Пример

Улаз	*Излаз*
5	440.50
03-07-2021 340.00	05-04-2021
05-04-2021 285.50	03-07-2021
03-07-2021 100.50	
04-07-2021 270.00	
05-04-2021 155.00	

Решење

Зараду за сваки дан (дневни пазар) можемо израчунати тако што датум сваког рачуна пресликамо у укупан износ рачуна за тај дан. Да би смо постигли да у крајњем испису датуми буду хронолошки сортирани, датуме који су кључеви мапе тј. речника ћемо представити као ниске облика `gggg-mm-dd`. Након израчунавања износа за сваки датум, одредићемо максимални износ (уобичајеним алгоритмом одређивања максимума), а затим поново проћи кроз све датуме редом и исписати оне код којих се јавља тај максимални износ.

```
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int n;
  cin >> n >> ws;
  vector<pair<string, double>> racuni;
  for (int i = 0; i < n; i++)  {
    string linija;
    getline(cin, linija);
    string dan = linija.substr(0, 2);
    string mesec = linija.substr(3, 2);
    string godina = linija.substr(6, 4);
    double iznos = stod(linija.substr(11));
    racuni.emplace_back(godina + "-" + mesec + "-" + dan, iznos);
  }

  sort(begin(racuni), end(racuni));

  vector<pair<string, double>> po_datumima;
  for (int i = 0; i < racuni.size(); i++) {
    if (i == 0 || racuni[i].first != racuni[i-1].first)
      po_datumima.emplace_back(racuni[i].first, 0.0);
```

```
    po_datumima.back().second += racuni[i].second;
  }

  double maks_iznos = 0.0;
  for (int i = 0; i < po_datumima.size(); i++)
    maks_iznos = max(maks_iznos, po_datumima[i].second);

  cout << fixed << setprecision(2) << maks_iznos << endl;
  for (int i = 0; i < po_datumima.size(); i++)
    if (po_datumima[i].second == maks_iznos) {
      string datum = po_datumima[i].first;
      string dan = datum.substr(8, 2);
      string mesec = datum.substr(5, 2);
      string godina = datum.substr(0, 4);
      cout << dan << "-" << mesec << "-" << godina << endl;
    }
  return 0;
}
```

Да бисмо ефикасно израчунали зараду за сваки дан (дневни пазар), можемо и да сортирамо податке по датумима. Датуми су представљени нискама карактера, и да би се сортирање ниски лексикографски поклопило са хронолошким редоследом датума, датуме можемо представити у облику `gggg-mm-dd`. Након сортирања, сви рачуни за један дан се налазе на узастопним позицијама у низу. Након одређивања зараду за сваки дан, можемо одредити и износ највеће дневне зараде. Након тога поново можемо проћи кроз низ дневних зарада (који је сортиран хронолошки) исписујемо датуме код којих се износ поклапа са тим максималним.

```
#include <iostream>
#include <iomanip>
#include <string>
#include <map>
#include <algorithm>

using namespace std;

int main() {
  int n;
  cin >> n >> ws;
  map<string, double> poDatumima;
  for (int i = 0; i < n; i++)  {
    string linija;
    getline(cin, linija);
    string dan = linija.substr(0, 2);
    string mesec = linija.substr(3, 2);
    string godina = linija.substr(6, 4);
    string datum = godina + "-" + mesec + "-" + dan;
    double iznos = stod(linija.substr(11));
    poDatumima[datum] += iznos;
  }
```

```
  double maks_iznos = 0.0;
  for (const auto& [datum, iznos] : poDatumima)
    maks_iznos = max(maks_iznos, iznos);

  cout << fixed << setprecision(2) << maks_iznos << endl;
  for (const auto& [datum, iznos] : poDatumima)
    if (iznos == maks_iznos) {
      string dan = datum.substr(8, 2);
      string mesec = datum.substr(5, 2);
      string godina = datum.substr(0, 4);
      cout << dan << "-" << mesec << "-" << godina << endl;
    }
  return 0;
}
```

Задатак: Фудбалска група табела

Аутор: Филип Марић

Такмичење: 2022/2023. квалификације 2, 6. разред, 4. задатак

На светском првенству у фудбалу свака група се састоји од 4 екипе које играју свака са сваком. У наредну фазу такмичења пласирају се две најбоље пласиране екипе. За сваку победу добијају се 3 поена, а за нерешен резултат 1 поен. Напиши програм који на основу резултата 6 утакмица одређује број поена сваке екипе и гол разлику (разлику између броја датих и броја примљених голова).

Опис улаза

Са стандардног улаза се уносе резултати свих мечева и то редом AB, CD, AC, BD, AD, BC. Сваки резултат се уноси у облику два броја раздвојена размаком.

Опис излаза

На стандардни излаз исписати број поена и гол разлику редом за екипе A, B, C и D.

Пример 1

Улаз	*Излаз*
3 2	9 6
2 1	3 -2
4 2	3 -2
1 3	3 -2
3 0	
1 0	

Пример 2

Улаз	*Излаз*
0 0	7 7
2 0	5 1
6 2	3 -3
1 1	1 -5
3 0	
1 0	

Решење

За сваку од 4 екипе треба да памтимо број поена и гол разлику. То можемо да урадимо било помоћу четвороелементног низа, било помоћу мапе тј. речника.

Пролазимо кроз низ мечева, учитавамо резултат (број датих голова прве и број датих голова друге екипе) и ажурирамо поене и гол разлику. Број поена израчунавамо поређењем броја датих голова обе екипе (ако је прва екипа дала више голова, њен број поена увећавамо за 3,

ако су обе екипе дале исти број голова, број поена обе екипе увећавамо за по 1, а ако је друга екипа дала већи број голова, њен број поена увећавамо за 3). Гол разлика прве екипе се увећава за разлику између броја голова који је она дала и броја голова која је дала друга екипа, а гол разлика друге екипе се увећава за разлику између броја голова који је она дала и броја голова која је дала прва екипа.

На крају, пролазимо кроз екипе и за сваку исписујемо број поена и гол-разлику.

```
#include <iostream>
#include <vector>
#include <map>

using namespace std;

int main() {
  map<char, int> poeni;
  map<char, int> gol_razlika;

  for (const string& mec: {"AB", "CD", "AC", "BD", "AD", "BC"}) {
    int x, y;
    cin >> x >> y;
    if (x > y)
      poeni[mec[0]] += 3;
    else if (x < y)
      poeni[mec[1]] += 3;
    else {
      poeni[mec[0]] += 1;
      poeni[mec[1]] += 1;
    }

    gol_razlika[mec[0]] += x - y;
    gol_razlika[mec[1]] += y - x;
  }

  for (char ekipa: {'A', 'B', 'C', 'D'})
    cout << poeni[ekipa] <<  " " << gol_razlika[ekipa] << endl;
   return 0;
}
```

Задатак: Рачуни

Аутор: Иван Дрецун

Такмичење: окружно 2021/22., VII разред, 3. задатак

Потребно је симулирати банковни систем за k различитих корисника. Сваки корисник има рачун са почетним стањем 0. Потребно је подржати две врсте операција:

- `upit x` одређује колико постоји корисника чији рачун садржи тачно x динара
- `ime x` додаје x динара на рачун особе са именом `ime` (x може бити и негативно)

Написати програм који пордржава извршавање n оваквих операција.

Опис улаза

Са стандардног улаза се уносе бројеви n и k. Након тога се у n редова уноси по једна операција.

Опис излаза

За сваки упит (операцију првог типа) исписати одговор, сваки у засебном реду.

Пример

Улаз	*Излаз*

`6 4`
`marko 2`
`milan 5`
`dragana 4`
`upit 0`
`milan -1`
`upit 4`
	`1`
	`2`

Решење

Задатак се једноставно и лако може решити употребом две мапе тј. речника. Једна се користи да преслика име корисника у износ на његовом рачуну, а друга да преслика износ на рачуну у број рачуна који садрже тај износ.

```
#include <iostream>
#include <map>

using namespace std;

int main() {
  int n, m, x;
  cin >> n >> m;

  map<string, int> racun;
  map<int, int> brPojavljivanja;
  brPojavljivanja[0] = m;

  for (int i = 0; i < n; i++) {
    string s;
    cin >> s >> x;

    if (s == "upit")
      cout << brPojavljivanja[x] << '\n';
    else {
      brPojavljivanja[racun[s]]--;
      racun[s] += x;
      brPojavljivanja[racun[s]]++;
    }
  }
```

```
  return 0;
}
```

Задатак: Зрак

Аутор: Филип Марић

Такмичење: 2022/2023. квалификације 1, 6. разред, 6. задатак

Познато је да радио-таласима сметају препреке које им се нађу на путу. Потребно је истражити да ли радио-таласи могу да прођу кроз правоугаону област у којој је засађено дрвеће. Радио-талас се испушта из горњег левог угла те области и то у правцу који је одређен вертикалним и хоризонталним померајем (то су бројеви који одређују разлику између координата наредног и текућег поља које је зрак достигао). На пример, ако су помераји редом 2 и 1, то значи да се зрак у сваком кораку помери две врсте наниже и једну колону надесно, па редом прелази поља чије су координате (v, k) једнаке $(0, 0)$, $(2, 1)$, $(4, 2)$ итд. Напиши програм који одређује колико стабала ће се наћи на путу радио-таласа.

Опис улаза

Прва линија стандардног улаза садржи димензије матрице bv и bk ($3 \leq bv, bk \leq 100$), раздвојене размаком. Након тога се налази опис матрице, при чему су поља која садрже стабла означена са 1, а поља која не садрже стабла са 0. На крају се налази линија у којој су дати помераји (природни бројеви раздвојени размаком).

Опис излаза

На стандардни излаз исписати број стабала које радио сигнал посети.

Пример 1

Улаз	*Излаз*	*Објашњење*

`8 8`
`0 1 1 0 0 1 0 1`
`0 0 1 0 1 0 1 0`
`1 0 1 0 1 0 1 0`
`1 1 0 1 0 1 0 1`
`0 1 1 0 1 1 1 0`
`0 0 0 0 0 1 0 1`
`1 0 1 1 0 1 1 0`
`0 1 1 0 1 1 1 0`
`2 1`
	`2`	На слици су приказана поља преко којих ће прелазити зрак. Симболом `X` су означена поља на којима постоје стабла, а симболом `-` поља на којима не постоје стабла.
		`- 1 1 0 0 1 0 1`
		`0 0 1 0 1 0 1 0`
		`1 - 1 0 1 0 1 0`
		`1 1 0 1 0 1 0 1`
		`0 1 X 0 1 1 1 0`
		`0 0 0 0 0 1 0 1`
		`1 0 1 X 0 1 1 0`
		`0 1 1 0 1 1 1 0`

Пример 2

Улаз	*Излаз*
8 8 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 1 1 2	2

Решење

Основни део овог задатка представља учитавање матрице логичких вредности. Она може бити представљена у облику низа низова. Када се матрица учита, анализирамо њене елементе, кренувши од елемента на позицији $(0, 0)$ и увећавајући у сваком кораку координате за (d_v, d_k), све док су обе координате унутар граница матрице. Када наиђемо на елемент чија је вредност 1, увећавамо бројач препрека који на крају програма исписујемо.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int bv, bk;
  cin >> bv >> bk;
  vector<vector<int>> M(bv);
  for (int v = 0; v < bv; v++) {
    M[v].resize(bk);
    for (int k = 0; k < bk; k++)
      cin >> M[v][k];
  }
  int dv, dk;
  cin >> dv >> dk;
  int brojPrepreka = 0;
  for (int v = 0, k = 0; v < bv && k < bk; v += dv, k += dk) {
    if (M[v][k] == 1)
      brojPrepreka++;
  }
  cout << brojPrepreka << endl;
  return 0;
}
```

Задатак: Лопте

Аутор: Филип Марић

Такмичење: државно 2021/22., VII и VIII разред, 2. задатак

Непомична камера снима из висине неколико лопти које се котрљају. Свака слика са те камере се састоји од m врста и n колона. За сваку лопту је познат почетни положај на слици и брзина кретања. Написати програм који одређује положај лопти на слици након одређеног броја секунди (неке лопте могу и да изађу ван слике и њих не треба приказивати). Могуће судare лопти занемарити, тј. рачунати као да лопте пролазе једна кроз другу.

Опис улаза

Са стандардног улаза се учитавају димензије слике m и n ($1 \leq m, n \leq 100$), након тога број секунди k ($0 \leq k \leq 100$)и након тога, до краја стандардног улаза линије које садрже по четири броја: v и k ($0 \leq v < m$, $0 \leq k < n$) означавају редни број врсте и колоне на којој се лопта налази у почетном тренутку, а v_v и v_k ($-3 \leq v_v, v_k \leq 3$) означавају брзину кретања лотпе по врстама и по колонама (у свакој секунди врста се промени за v_v, а колона за v_k).

Опис излаза

На стандардни излаз исписати положај сваке лопте након K секунди (празна поља означити карактерима `.` а поља на којима је лопта карактером `#`).

Пример

Улаз	*Излаз*	*Објашњење*
`6 7`	`...#...`	Положај лопти на почетку је
`1`	`..#.#..`	`.......`
`1 1 0 1`	`..#.#..`	`.#####.`
`1 2 1 0`	`.#####.`	`.......`
`1 3 -1 0`	`.#...#.`	`#.###.#`
`1 4 1 0`	`.#...#.`	`.......`
`1 5 0 -1`		`.##.##.`
`3 0 0 1`		Када се лопте помере на начин како то њихове брзине описују добије се
`3 2 0 0`		приказ великог слова `A`.
`3 3 0 0`		
`3 4 0 0`		
`3 6 0 -1`		
`5 2 -1 -1`		
`5 4 -1 1`		
`5 1 0 0`		
`5 5 0 0`		

Решење

Текући положај сваке лопте ћемо чувати у матрици (можемо је реализовати као низ ниски тј. низ низова карактера). Након учитавања почетног положаја и брзине сваке лопте израчунавамо њен нов положај и ако је он унутар матрице, у матрицу на одговарајуће место уписујемо карактер `#`. Ако су почетне координате лопте (p_x, p_y), а вектора брзине (v_x, v_y), координате положаја после s секунди једнаке су $(p_x + s \cdot v_x, p_y + s \cdot v_y)$.

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;
```

```
int main() {
  int m, n;
  cin >> m >> n;
  vector<string> A(m);
  for (string& s : A)
    s.resize(n, '.');
  int s;
  cin >> s;
  int pv, pk, vv, vk;
  while (cin >> pv >> pk >> vv >> vk) {
    int v = pv + s * vv;
    int k = pk + s * vk;
    if (0 <= v && v < m && 0 <= k && k < n)
      A[v][k] = '#';
  }
  for (const string& s : A)
    cout << s << endl;
  return 0;
}
```

Задатак: Икс-окс

Аутор: Душан Пођадић

Такмичење: окружно 2021/22., VII и VII разред, 1. задатак

Два играча играју игру икс-окс. Ова игра се игра на табли величине 3×3 на коју играчи наизменично уписују икс или окс. Победник је онај играч који успе да у један ред, једну колону или на једну дијагоналу упише 3 своја знака. За задату крајњу позицију игре, одреди да ли је победио играч икс, играч окс, или је игра завршена нерешено.

Опис улаза

У три реда стандардног улаза се налазе по три знака `x`, `o` или `*`. Први ред улаза описује како је попуњен први ред табле за играње итд. Ознака `x` означава икс, `o` окс, а `*` да је то поље на крају партије празно.

Опис излаза

Уколико је победио икс исписати знак `x`, уколико је победио окс исписати `o`, а уколико је нерешено исписати `nereseno`.

Пример 1

Улаз	*Излаз*

`x*x`
`ox*`
`xoo`
	`x`

Пример 2

Улаз	*Излаз*

`xxo`
`oox`
`xox`
	`nereseno`

Решење

Табла може бити представљена низом од три ниске дужине 3 (или матрицом карактера димензије 3×3).

За сваку врсту, за сваку колону и за обе дијагонале треба проверити да ли су сви елементи међусобно једнаки и да ли су једнаки `x` или `o`. Пошто се ова провера често врши, згодно је издвојити је у засебну функцију.

У главној функцији провере, покрећемо проверу елемената сваке врсте (једноставности ради, то можемо урадити уз помоћ петље), сваке колоне и обе дијагонале.

```
#include <bits/stdc++.h>
using namespace std;

char proveri(char a, char b, char c) {
  if (a == b && b == c)
    if (a == 'x' || a == 'o')
      return a;
  return '*';
}

char proveriTablu(string s[]) {
  char p;

  for (int i = 0; i < 3; i++) {
    p = proveri(s[i][0], s[i][1], s[i][2]);
    if (p == 'x' || p == 'o')
      return p;
  }

  for (int j = 0; j < 3; j++) {
    p = proveri(s[0][j], s[1][j], s[2][j]);
    if (p == 'x' || p == 'o')
      return p;
  }

  p = proveri(s[0][0], s[1][1], s[2][2]);
  if (p == 'x' || p == 'o')
    return p;

  p = proveri(s[0][2], s[1][1], s[2][0]);
  if (p == 'x' || p == 'o')
    return p;

  return '*';
}

int main() {
  string s[3];
  for (int i = 0; i < 3; i++)
```

```
    cin >> s[i];

  char p = proveriTablu(s);
  if (p == 'x' || p == 'o')
    cout << p << endl;
  else
    cout << "nereseno" << endl;

  return 0;
}
```

Задатак: Играчка

Аутор: Небојша Варњица

Такмичење: 2021/22. квалификације 2, VII разред, 3. задатак и VIII разред, 1. задатак

Играчка је облика квадрата странице 3 поља (3 реда и 3 колоне). У 8 од 9 поља је плочица са једним од бројева 1, 2, 3, …, 8 а преостало девето поље је празно. Потез се састоји у померању једне плочице у суседно лево, десно, доње или горње празно поље. Написати програм који за дато стање играчке одиграва један потез.

Опис улаза

Уноси се тренутно стање играчке (прва 3 реда) и потез који покушавамо да одиграмо (четврти ред).

Тренутно стање играчке се уноси тако што се у три реда уносе бројеви 1, 2, …, 8 (сваки тачно по једном) а на месту празног поља уноси се знак *. На пример:

```
245
*38
167
```

Потез чије се одигравање покушава може бити:

1 - плочица са бројем се помера у празно суседно поље са леве стране

2 - плочица са бројем се помера у празно суседно поље са десне стране

3 - плочица са бројем се помера у празно суседно поље са горње стране

4 - плочица са бројем се помера у празно суседно поље са доње стране

Опис излаза

Исписује се стање играчке након одиграног потеза. Испис треба да буде у облику у коме је унето почетно стање.

Ако потез није могуће одиграти исписује се учитано стање играчке.

Пример

Улаз	*Излаз*
245	245
*38	3*8
167	167
1	

Решење

Опис главног решења

Сваки ред играчке је један стринг. Померања лево и десно се своде на замену места унутар једне врсте а померања горе и доле на замену знакова између суседних редова.

```
#include <iostream>

using namespace std;

int main()
{
  // polje je predstavljeno pomocu tri niske
  string a[3];
  cin >> a[0] >> a[1] >> a[2];

  // pronalazimo koordinate zvezdice
  int pv = 0, pk = 0;
  for (int i = 0; i < 3; i++) {
    int j;
    if ((j = a[i].find("*")) != string::npos) {
      pv = i;
      pk = j;
      break;
    }
  }

  int op;
  cin >> op;
  switch (op) {
  case 1: // levo
    if (pk<2) {
      a[pv][pk] = a[pv][pk+1];
      a[pv][pk+1] = '*';
    }
    break;
  case 2: // desno
    if (pk>0) {
      a[pv][pk] = a[pv][pk-1];
      a[pv][pk-1] = '*';
    }
    break;
```

```
  case 3: // gore
    if (pv<2) {
      a[pv][pk] = a[pv+1][pk];
      a[pv+1][pk] = '*';
    }
    break;
  case 4: // dole
    if (pv>0) {
      a[pv][pk] = a[pv-1][pk];
      a[pv-1][pk] = '*';
    }
  }

  // ispisujemo polje nakon pomeranja
  cout << a[0] << endl;
  cout << a[1] << endl;
  cout << a[2] << endl;
  return 0;
}
```

Задатак: Судоку

Аутор: Небојша Варница

Такмичење: 2021/22. квалификације 3, VIII разред, 4. задатак

У квадрат са 16 поља треба уписати бројеве 1, 2, 3 и 4 (сваки по 4 пута) тако да у сваком хоризонталном и вертикалном реду буду различити бројеви. Када се тај квадрат преполови хоризонтално и вертикално добијају се 4 мања квадрата. Захтева се да и у тим малим квадратима буду различити бројеви. Напишите програм који за дати делимично попуњен квадрат одређује које бројеве можемо уписати у дато поље тако да сви захтеви буду испуњени.

Опис улаза

Са стандарног улаза се уноси 5 редова: Прва 4 реда дају стање делимично попуњеног квадрата – садрже бројеве 1, 2, 3, 4 и * на местима која нису попуњена. Пети ред садржи редни број врсте и колоне (без размака) поља које треба попунити. У свим тест примерима то поље ће бити непопуњено (тј. садржаће знак '*').

Опис излаза

На стандардном излазу, у једном реду исписати, без размака, све бројеве које је дозвољено ставити у описано поље. У свим тест примерима у захтевано поље моћи ће да се упише бар један број (тј. решење неће бити празно).

Пример

Улаз	*Излаз*	*Објашњење*

`1**3`
`**2*`
`2**4`
`*1**`
`43`
	`3`	Тражено поље (4. ред, 3. колона) не може да садржи 1 јер већ постоји у четвртом реду. Не може да садржи ни 2 јер већ постоји у трећој колони. Не може да садржи ни 4 јер већ постоји у доњем десном квадрату. Дакле у том пољу може да стоји само 3.

Решење

Прво учитавамо матрицу и координате поља (r, k) које треба попунити, а затим анализирамо једну по једну вредност од 1 до 4 и за сваку проверавамо да ли се може уписати на то поље. За сваку од те 4 вредности проверавамо да ли у реду r, колони k и квадрату који садржи поље (r, k) већ постоји та вредност (у том квадрату је довољно анализирати само оно поље које није у реду r и колони k). Ако не постоји ни у једном од та три скупа поља, онда је уписујемо у резултат.

```
#include <iostream>

using namespace std;

int main()
{
  // ucitavamo delimicno popunjenu tablu
  string t[4];
  for (int i = 0; i < 4; i++)
    cin >> t[i];
  // odredjujemo koordinate polja koje treba popuniti
  string x;
  cin >> x;
  int r = x[0] - '0' - 1;
  int k = x[1] - '0' - 1;

  // vrednosti koje se mogu upisati na polje (r, k)
  string rez;
  // analiziramo jednu po jednu vrednost
  for (int i = 1; i <= 4; i++) {
    // vrednost koju pokusavamo da upisemo
    char z = i + '0';

    // analiziramo vrstu r i kolonu k
    bool ok = true;
    for (int j = 0; j < 4; j++) {
      if (t[r][j] == z || t[j][k] == z) {
        ok = false;
        break;
      }
    }

    // analiziramo kvadrat u kom se nalazi polje (r, k)
    // tj. samo ono polje u kvadratu koje nije u redu r i koloni k
    int r1 = r / 2;
    int k1 = k / 2;
    int a = 1 - r % 2;
    int b = 1 - k % 2;
    if (t[2*r1+a][2*k1+b] == z)
      ok = false;
```

```
    // ako su svi testovi prosli, dodajemo vrednost z u skup mogucih vrednosti
    if (ok)
      rez += z;
  }

  // ispisujemo konacan rezultat
  cout << rez << endl;
  return 0;
}
```

Задатак: Шифра

Аутори: Душан Пајадић, Иван Дрецун

Такмичење: 2021/22. квалификације 3, V разред, 6. задатак

Сара је од своје симпатије примила шифровану поруку, али не уме да је протумачи. Она је замолила своју другарицу Милу, коју су увек занимале шифре, да јој помогне. Мила је одмах препознала шифорвање такозваним Магичним квадратом. Шифровање магичним квадратом ради на следећи начин: Конструише се квадрат димензија 26×26 који се затим попуни малим словима енглеске абецеде. Сви редови квадрата се обележе малим словима енглеске абецеде (први ред са a, други са b, трећи са c и тако даље до последњег који се обележава са z). На исти начин се обележе и колоне. Када се напише порука коју је потребно шифровати, свако слово из поруке се замењује комбинацијом два мала слова енглеске абецеде која означавају ред и колону у којима се у квадрату налази то посматрано слово. Пошто у квадрату може да се налази више истих слова, бира се било која комбинација ред/колона која садржи то слово јер то отежава дешифровање онима који не знају како изгледа квадрат. Мила је открила тачан квадрат који је коришћен за шифровање, али нема више времена, па је замолила тебе да, користећи њено откриће, дешифрујеш Сари поруку.

Опис улаза

У првих 26 редова се налази по 26 слова одвојених размаком. Сваки од 26 унетих редова представља по један ред квадрата који је Мила открила. У наредном реду се налази n ($1 \leq n \leq 1000$), број слова поруке коју је Сара примила. У наредних n редова се налазе по два слова енглеске абецеде која представљају једно слово у шифри.

Опис излаза

У једном реду стандардног излаза исписати дешифорвану поруку коју је Сара добила.

Пример

Улаз

```
t x b l h p p w i d j k l a z x p s c g x k s t h g
u t g a n n m e n r u r y d o c y g a c c n l p s u
r s a o l a d h t s y s k b b a a i a a v u k f r d
b n k d z m r p y f y v r u g y w u n x n r t g d o
u t m w l p e v a x x b t h m a t b q i h t g o s y
o f f v h f j d b t k x r e f b f j i k b s l x l r
p e e t n y z q c r u h w m f r m l m d b j z r k v
f f a e f c a d i t c y h d u v r b l t q v u d a m
k s t z o q p y j b i m l q d e u i a k n x d y a z
q i b p j c w t t y e s n t u j j z o o e e s d d l
g c y d y j z p z y x g x m r r u t v a p g i k s b
f d i k d p b o z o a z i y b a y g a k c a z e n n
j k z p w y m x a c q g l e f v q m z x f e q a l p
k k q d s j l q j k i b r b x x c d s h z g d w a h
a k w d u c g o g j q e e b v i f b o w e f e k q w
u n e y b q n h g j b b o z z k n u s t w u v a c h
o r p k a s r n o e b l h r i a l o j a u s n e z u
b e z v c s i f t o h f f b c f k b c c g z y v g a
y y x o v l d y s n v y v c a p v w i g q j q c c i
t h v j z e y t l f y c s n f w e l l e b f y a q x
n v v i g z a y g o c l q s x j p g f k f t y g d s
f w d f w k s g f j v a b t q k d j d j h x p q v k
e r k g g d h c d j g h m m j q d n r n m g r k g b
d v p d g m w n u n e i d t g w z f f j a s u i c z
k r f s h s u p n k i h r e p i n p h p h v v y k v
u a v t v r r k p l z b v y q i o k j k n y o d m r
4
cb
cc
zz
zb
```

Излаз

```
sara
```

Објашњење

Прво слово је шифровано са `cb`, дакле налази се у 3. реду (јер је `c` треће слово енглеске абецеде) и 2. колони (јер је `b` друго слово) магичног квадрата – то је слово `s`. Друго слово је шифровано са `cc`, дакле налази се у 3. реду и 3. колони квадрата – то је слово `a`. Треће слово је шифровано са `zz`, дакле налази се у 26. реду и 26. колони квадрата – то је слово `r`. Четврто слово је шифровано са `zb`, дакле налази се у 26. реду и 2. колони – то је поново слово `a`.

Решење

Прво је потребно учитати квадрат и запамтити га у матрицу. Након тога се учитавају шифровани знаци. На основу првог знака у комбинацији се одређује ред, а на основу другог колона у којој се у матрици налази тражено слово. Најбржи начин да се на основу знака одреди индекс реда или колоне је употреба ASCII тј. UNICODE табеле. У тим табелама се сви мали знакови енглеске абецеде налазе груписани: на месту 97 се налази `'a'`, на месту 98 `'b'`, на 99 `'c'` и тако даље до места 122 на ком се налази `'z'`. Наравно, није неопходно знати напамет ова места, довољно је знати да су узастопна. Индекс врсте/колоне се сада може добити тако што од ASCII

тј. UNICODE вредности учитаног карактера одузмемо ASCII тј. UNICODE вредност слова `'a'` (ако је учитано слово `'a'` тада ће индекс бити 97-97=0, ако је учитано слово `'b'` тада ће индекс бити 98-97=1 итд.). То можемо урадити на следећи начин: `index=ucitani_karakter-'a'` јер C++ одузимање карактера аутоматски тумачи као одузимање њихових ASCII вредности.

Када имамо потребне индексе, потребно је само из матрице прочитати тражено слово.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  char magicniKvadrat[26][26];
  for (int i = 0; i < 26; i++)
    for (int j = 0; j < 26; j++)
      cin >> magicniKvadrat[i][j];

  int n;
  cin >> n;

  string resenje(n, ' ');
  for (int i = 0; i < n; i++) {
    char a, b;
    cin >> a >> b;
    resenje[i] = magicniKvadrat[a - 'a'][b - 'a'];
  }

  cout << resenje << '\n';
  return 0;
}
```

Задатак: Дијагонале

Аутори: Милан Вуīделија, Филиī Марић

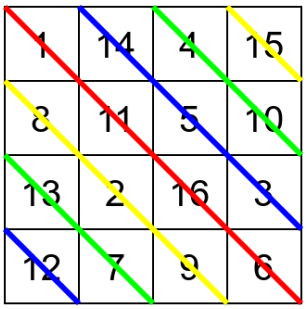
Такмичење: 2022/2023. квалификације 3, 6. разред, 6. задатак

Написати програм који за $n \times n$ бројева распоређених у квадрат проверава да ли је збир дуж сваке "главне изломљене дијагонале" једнак. Квадрат са овом особином зваћемо занимљив квадрат. На следећој слици је приказан један занимљив квадрат величине 4×4, а свака од његове четири главне изломљене дијагонале је означена по једном бојом.

На првој (црвеној) дијагонали, збир бројева је $1+11+16+6 = 34$, на другој (плавој) $14+5+3+12 = 34$, на трећој (зеленој) $4+10+13+7 = 34$, а на четвртој (жутој) $15+8+2+9 = 34$.

Опис улаза

У првој линији стандардног улаза налази се димензија квадрата n ($1 \leq n \leq 20$), а у свакој од n наредних линија по n природних бројева раздвојених по једном размаком, који представљају један ред квадрата. Вредности бројева у квадрату су између 1 и 2000.



Занимљив квадрат

Опис излаза

На стандардни излаз исписати n природних бројева, сваки у посебном реду. Ови бројеви треба да буду збирови елемената главних изломљених дијагонала редом. У $n+1$-вом реду исписати `da` ако су сви збирови једнаки, односно `ne` ако нису.

Пример 1

Улаз	*Излаз*
`4`	`34`
`1 14 4 15`	`34`
`8 11 5 10`	`34`
`13 2 16 3`	`34`
`12 7 9 6`	`da`

Пример 2

Улаз	*Излаз*
`4`	`34`
`16 9 2 7`	`26`
`6 3 12 13`	`34`
`11 14 5 4`	`42`
`1 8 15 10`	`ne`

Пример 3

Улаз	*Излаз*
`1`	`17`
`17`	`da`

Решење

Бројеве који формирају квадрат можемо да сместимо у матрицу a. Збир елемената главне дијагонале (у примеру са слике означене црвеном бојом) можемо да израчунамо као $S = a[0][0] + a[1][1] + ... + a[n-1][n-1]$.

Збир елемената следеће изломљене дијагонале (на слици плаве) добијамо као $S = a[0][1] + a[1][2] + ... + a[n-1][0]$. Видимо да је индекс колоне код свих ових елемената за један већи од индекса врсте, осим код последњег елемента, где је индекс колоне 0 (уместо n). Међутим, једнакост ће да важи и код последњег елемента ако индекс колоне узмемо по модулу n. Слично важи и код осталих дијагонала, само што је индекс колоне дуж целе дијагонале већи од индекса врсте за неку другу константну вредност. У примеру са слике, за зелену дијагоналу та вредност је 2, а за жуту 3. У општем случају та вредност је код сваке следеће дијагонале већа за један него код претходне.

Остаје да се израчуна и испише збир елемената сваке изломљене дијагонале, проверавајући успут да ли су сви ти збирови једнаки. Након просласка крос све дијагонале исписује се и да ли су сви збирови дијагонала једнаки.

```
#include <iostream>

using namespace std;

const int MAX = 50;
```

```
int main() {
    int n;
    cin >> n;
    int A[MAX][MAX];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> A[i][j];

    int zbir = 0;
    for (int i = 0; i < n; i++)
        zbir += A[i][i];

    bool zanimljiv = true;
    for (int d = 0; d < n; d++) {
        int zbir_d = 0;
        for (int i = 0; i < n; i++)
            zbir_d += A[i][(i+d) % n];

        cout << zbir_d << endl;
        if (zbir_d != zbir)
            zanimljiv = false;
    }
    cout << (zanimljiv ? "da" : "ne") << endl;
    return 0;
}
```

Задатак: Топови

Аутор: Владан Ковачевић

Такмичење: државно 2022/2023, VI разред, 3. задатак

Дата је шаховска табла димензије $n \times n$. Свако поље табле је или 0 (празно) или 1 (на пољу се налази топ). Одредити да ли на табли постоје два топа која се нападају. Два топа се нападају ако се налазе у истом реду или у истој колони.

Опис улаза

Са стандардног улаза уноси се димензија табле n ($1 \leq n \leq 10$), затим у наредних n редова по n бројева.

Опис излаза

На стандарни излаз исписати 1 ако постоје таква два топа или 0 ако не постоје.

Пример 1

Улаз	*Излаз*
3 0 1 0 0 0 1 1 0 0	0

Пример 2

Улаз	*Излаз*
4 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1	1

Решење

Да бисмо проверили да ли на табли постоје два топа која се нападају, довољно је да проверимо да ли у неком реду или у некој колони постоји више од једног топа.

За сваки ред (колону) можемо проласком кроз тај ред (колону) у петљи да избројимо колико постоји топова (поља чија је вредност 1). Дакле, довољна нам је једна двострука петља.

```
#include <iostream>
#include <vector>

using namespace std;

bool rooks(vector< vector<int>>& v) {
    int n = v.size();
    for (int i=0; i<n; i++) {
        int row1s = 0, col1s = 0;
        for (int j=0; j<n; j++) {
            if (v[i][j]) row1s++;
            if (v[j][i]) col1s++;
        }
        if (row1s >= 2 || col1s >= 2) return true;
    }
    return false;
}

int main() {
    int n;
    cin>>n;
    vector< vector<int>> v(n, vector<int>(n));
    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
            cin>>v[i][j];
    cout<<rooks(v)<<endl;

    return 0;
}
```

Глава 9

Теорија бројева

Задатак: Поклони у продавници

Аутор: Филип Марић

Такмичење: 2021/22. квалификације 3, VII разред, 4. задатак

У продавници се слави годишњица отварања. Сваки i-ти купац добија ваучер на 200 динара, сваки j-ти ваучер на 500 динара, а сваки k-ти купац ваучер на 1000 динара. Колика је укупна вредност ваучера који су подељени док први купац не добије ваучере у вредности од 1700 динара (укључујући и ваучере које он добије)?

Опис улаза

Са стандардног улаза се уносе вредности i, j и k (природни бројеви мањи од 10000).

Опис излаза

На стандардни излаз исписати тражени износ.

Пример

Улаз	*Излаз*	*Објашњење*

`20`
`30`
`50`
	`14000`	Тристоти купац ће добити сва три ваучера. До тада ће бити подељено 15 ваучера од по 200 динара, 10 ваучера од по 500 динара и 6 ваучера од по 1000 динара.

Решење

Груба сила

Могуће је обрађивати једног по једног купца и за сваког проверавати да ли добија неки ваучер (провером дељивости његовог редног броја са i, j и k). Укупна сума се увећава сваки пут када се додели неки ваучер, а петља се прекида први пут када се наиђе на купца који је добио сва три ваучера.

Сложеност овог поступка је $O(i \cdot j \cdot k)$ - овај број корака се извршава када су бројеви i, j и k узајамно прости. Без обзира на релативно мало ограничење бројева i, j и k, њихов производ може бити веома велики и овај алгоритам је неефикасан.

```
#include <iostream>

using namespace std;

typedef unsigned long long ull;

ull nzd(ull a, ull b) {
  if (b == 0) return a;
  return nzd(b, a % b);
}

ull nzs(ull a, ull b) {
  return (a / nzd(a, b)) * b;
}

int main() {
  ull i, j, k;
  cin >> i >> j >> k;
  int kupac = 1;
  int novac = 0;
  while (true) {
    if (kupac % i == 0)
      novac += 200;
    if (kupac % j == 0)
      novac += 500;
    if (kupac % k == 0)
      novac += 1000;
    if (kupac % i == 0 && kupac % j == 0 && kupac % k == 0)
      break;
    kupac++;
  }
  cout << novac << endl;
  return 0;
}
```

НЗС

Купац ће добити сва три поклона када је његов редни број дељив и са бројем i и са бројем j и са бројем k. Пошто тражимо првог таквог купца, потребно је да одредимо најмањи такав број n, а то је заправо НЗС бројева i, j и k. Број додељених ваучера од 200 динара је тада једнак n/i, оних од 500 динара је n/j, а оних од 100 динара је n/j.

НЗС два броја је најбоље одредити Еуклидовим алгоритмом.

Пошто је сложеност Еуклидовог алгоритма логаритамска, сложеност можемо одозго грубо ограничити са $O(\log i \cdot j \cdot k)$.

```
#include <iostream>

using namespace std;
```

```
typedef unsigned long long ull;

ull nzd(ull a, ull b) {
  if (b == 0) return a;
  return nzd(b, a % b);
}

ull nzs(ull a, ull b) {
  return (a / nzd(a, b)) * b;
}

int main() {
  ull i, j, k;
  cin >> i >> j >> k;
  ull n = nzs(nzs(i, j), k);
  cout << (n / i) * 200 + (n / j) * 500 + (n / k) * 1000 << endl;
  return 0;
}
```

Задатак: Бродови

Такмичење: окружно 2018/2019, VII разред, 3. задатак

Аутор: Филип Марић

Транспортна компанија успоставља бродске вожње између матичне луке и неколико (највише 5) других лучких градова. На свакој од тих линија циркулише тачно један брод тако што креће из матичне луке, превози путнике до свог одредишта, а затим друге путнике враћа назад у своју матичну луку. Ако сви бродови из матичне луке крећу истовремено и ако је за сваки од њих познато колико му је дана потребно да отпутује и да се врати назад у своју матичну луку (претпоставља се да се сваки транспорт и повратак обавља унутар једне календарске године), напиши програм који одређује после колико дана ће се теоријски сви бродови први пут поново сусрести у матичној луци.

Опис улаза

Са стандардног улаза се уноси број линија (највише 5), а затим за сваку од линија број дана потребних да брод оптутује и да се врати.

Опис излаза

На стандардни улаз исписати тражени број.

Пример

Улаз	*Излаз*
3 15 24 60	120

Решење

Брод се враћа у своју матичну луку у правилним умножцима свога периода путовања (на пример, ако је период путовања 15, брод се враћа у матичну луку после 15, 30, 45, 60 дана итд. Сви бродови су у луци у данима који су умножци свих њихових њихових периода путовања. Пошто нас занима први такав дан, тражимо најмањи број који је умножак унетих периода путовања тј. најмањи број који је њима дељив, а то је њихов најмањи заједнички садржалац. Њега рачунамо на уобичајени начин (применом Еуклидовог алгоритма).

```
#include <iostream>

using namespace std;

long long nzd(long long a, long long b) {
  while (b != 0) {
    long long ost = a % b;
    a = b;
    b = ost;
  }
  return a;
}

long long nzs(long long a, long long b) {
  return a / nzd(a, b) * b;
}

int main() {
  int n;
  cin >> n;
  long long broj = 1;
  for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    broj = nzs(broj, x);
  }
  cout << broj << endl;
  return 0;
}
```

Задатак: Квадрати максималне површине

Аутор: Нина Икодиновић

Такмичење: 2021/22. квалификације 1, V разред, 6. задатак

Васа је добио задатак да изреже лист папира димензије N пута M на квадрате максималне површине. Васа најпре исеца највећи могући квадрат тако да сече лист папира по најдужој страници (нпр. за лист димензије 3x7 највећи могући квадрат је 3x3). Потом Васа склони квадрат и над преосталим правоугаоником понови исецање квадрата највеће површине. Наставља исту операцију све док преостали правоугаоник не постане квадрат. Написати програм који ће израчунати

број квадрата који ће Васа добити након исецања на описани начин.

Опис улаза

У првом реду стандардног улаза налази се природан број M, а у другом природан број N. Бројеви M и N нису већи од 2^{60}.

Опис излаза

На стандардни излаз исписати број квадрата који се могу добити наведеним исецањем.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
3 7	5	23 6	9

Решење

Директно решење подразумева да се исеца један по један квадрат тј. да се мања страница правоугаоника одузима од веће и да се у сваком бројач квадрата увећава за 1. Поступак престаје када се добије квадрат тј. када се две димензије изједначе.

Иако коректан, овај поступак је неефикасан (нарочито у случајевима када је једна страна правоугаоника много мања од друге).

```
#include <iostream>

using namespace std;

int main() {
  long long m, n;
  cin >> m >> n;
  long long br = 1;
  while (n != m) {
    if (n < m)
      m = m - n;
    else
      n = n - m;
    br++;
  }
  cout << br << endl;
  return 0;
}
```

Нека су странице правоугаоника $M \times N$ и нека је $M < N$. Тада ће првих $\left\lfloor \frac{N}{M} \right\rfloor$ исечених квадрата бити димензије $M \times M$, а преостали правоугаоник ће бити димензије $N \bmod M$.

Уз мало додатне анализе, може се приметити да исто тврђење важи и када је $M = N$, па чак и када је $M > N$ (потврдите ово на примерима).

Одавде можемо да изведемо следећи поступак: док год је M позитивно, бројач исечених квадрата увећавамо за $\left\lfloor \frac{N}{M} \right\rfloor$, а бројеве M и N замењујемо редом са $N \bmod M$ и M.

По завршетку поступка M ће бити нула, што значи да је цео правоугаоник исечен на квадрате, а сви квадрати пребројани.

```cpp
#include <iostream>

using namespace std;

int main() {
  long long m, n, r;
  cin >> m >> n;
  long long br = 0;
  while (m > 0) {
    br += n / m;
    r = n % m;
    n = m;
    m = r;
  }
  cout << br;
  return 0;
}
```

Задатак: Ланац бројева

Такмичење: државно 2018/2019, VII и VIII разред, 4. задатак

Аутор: Јелена Хаџи-Пурић, *Аутор решења*: Филип Марић

Играмо игру ланац бројева. Замислимо природан број X и узмимо најмањи природан број који није његов делилац. Поступак поновимо за добијени број, и тако даље све док не дођемо до броја 2. Дужина ланца, $lanac(X)$ се дефинише као дужина добијеног низа бројева. На пример, за $X = 6$ добијамо низ $6, 4, 3, 2$ који се састоје од четири броја. Зато важи да $lanac(6) = 4$. За природне бројеве $A < B$ израчунајте збир дужина ланаца свих бројева од A до B, тј. израчунајте $lanac(A) + lanac(A + 1) + ... + lanac(B)$.

Опис улаза

У једином реду стандардног улаза дати су природни бројеви A и B ($3 \leq A < B < 10^{17}$).

Опис излаза

У једини ред стандардног излаза испишите тражени збир.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
4 6	9	10 20	29	1000 20000	49110

Решење

Израчунавањем дужине ланца по дефиницији за сваки број добило би се неефикасно решење. Кључна идеја је да за сваки број k израчунамо колико има ланаца чији је број k други елемент по реду.

- Јасно је да сви непарни бројеви имају ланац дужине 2, јер је најмањи природан број којим они нису дељиви број 2.

- Остали су парни бројеви. Парни бројеви који нису дељиви са 3 имају ланац дужине 3. На пример, $4, 3, 2$ или $8, 3, 2$.
- Остали су бројеви који су дељиви и са 2 и са 3 тј. бројеви који су дељиви са 6. Они од њих који нису дељиви са 4 имају ланац дужине 4. На пример, $6, 4, 3, 2$ или $18, 4, 3, 2$.
- Остали су бројеви који су дељиви са 2, 3 и 4 тј. који су дељиви са 12. Они који нису дељиви са 5 имају ланац дужине 3. На пример, $12, 5, 2$ или $24, 5, 2$.
- Остали су бројеви који су дељиви са 2, 3, 4 и 5 тј. који су дељиви са њиховим *NZS*, што је 60. Сви они су дељиви и са 6, па се онда анализира њихова дељивост са 7.

Поступак се даље наставља на исти начин.

Дакле, алгоритам може бити следећи. За сваки број k анализирамо колико је бројева у интервалу $[a, b]$ који нису дељиви са k, а који су дељиви са $NZS(1, 2, \ldots, k-1)$. Дужина њиховог ланца је $lanac(k) + 1$. Редом анализирамо све вредности k од 2, навише, све док не постанемо сигурни да смо исцрпли све бројеве из интервала $[a, b]$.

Број бројева у интервалу $[a, b]$ који су дељиви са неким бројем d_{da}, али нису дељиви са неким бројем d_{ne} можемо одредити тако што од броја бројева тог интервала дељивих са d_{da} одузме број бројева тог интервала који су дељиви и са d_{da} и са d_{ne}, што је тачно ако и само ако су дељиви са њиховим *NZS*, па се број бројева дељивих са d_{da} умањује за број бројева дељивих са $NZS(d_{da}, d_{ne})$.

Број бројева у интервалу $[a, b]$ дељивих неким бројем d се може одредити као $\left\lfloor \frac{b}{d} \right\rfloor - \left\lceil \frac{a}{d} \right\rceil + 1$. Наиме када се најмањи број интервала $[a, b]$ дељив са d подели са d добија се количник $\left\lceil \frac{a}{d} \right\rceil$, а када се највећи такав број подели са d добија се количник $\left\lfloor \frac{b}{d} \right\rfloor$. Сви количници при дељењу са d су елементи интервала $\left[\left\lceil \frac{a}{d} \right\rceil, \left\lfloor \frac{b}{d} \right\rfloor\right]$.

```
#include <iostream>
#include <map>

using namespace std;

// najveci zajednicki delilac brojeva a i b
long long nzd(long long a, long long b) {
  while (b != 0) {
    long long tmp = a % b;
    a = b;
    b = tmp;
  }
  return a;
}

// najmanji zajednicki sadrzalac brojeva a i b
long long nzs(long long a, long long b) {
  return a / nzd(a, b) * b;
}

// broj brojeva u intervalu [a, b] koji su deljivi sa d
long long brojDeljivih(long long a, long long b, long long d)
{
```

```
  long long a1 = (a + d - 1) / d; // ceil(a/d)
  long long b1 = b/d;             // floor(b/d)
  return b1 - a1 + 1;
}

// broj brojeva u intervalu [a, b] koji su deljivi sa dDa,
// a nisu deljivi su sa dNe
long long brojDeljivihNedeljivih(long long a, long long b,
                                 long long dDa, long long dNe) {
  // broj deljivih sa dDa umanjen za
  // broj onih koji su deljivi i sa dDa i sa dNe (pa i sa njihovim nzs)
  return brojDeljivih(a, b, dDa) - brojDeljivih(a, b, nzs(dDa, dNe));
}

// najmanji prirodan broj koji nije delilac broja x
long long najmanjiNedelilac(long long x) {
  long long d = 2;
  while (x % d == 0)
    d++;
  return d;
}

int main() {
  long long a, b;
  cin >> a >> b;

  // duzina lanca koji krece od datog broja
  map<long long, long long> lanac;
  lanac[2] = 1;

  // tekuci broj k (drugi element u lancu)
  long long k = 2;
  // nzs brojeva 1, 2, ..., k-1
  long long nzsPrethodnih = 1;
  // broj elemenata intervala [a, b] cija je duzina lanca odredjena
  long long ukupno = 0;
  // zbir duzina lanaca do sada resenih elemenata intervala [a, b]
  long long zbir = 0;

  // dok se ne odredi duzina lanca svih elemenata iz intevala [a, b]
  while (ukupno < b - a + 1) {
    // broj elemenata intervala [a, b] ciji je drugi element lanca k
    long long broj = brojDeljivihNedeljivih(a, b, nzsPrethodnih, k);
    zbir += broj * (lanac[k] + 1);
    ukupno += broj;
    // prelazimo na sledeci broj k
    k++;
    nzsPrethodnih = nzs(nzsPrethodnih, k-1);
    lanac[k] = lanac[najmanjiNedelilac(k)] + 1;
```

```
  }

  cout << zbir << endl;
  return 0;
}
```

Задатак: Множење и кореновање

Аутор: Филип Марић

Такмичење: 2021/22. квалификације 2, VII и VIII разред 4. задатак

Природни бројеви се могу трансформисати коришћењем следеће две операције:

- множење било којим другим природним бројем (може се применити увек)
- кореновање (може се применити само ако број потпун квадрат тј. ако је његов корен поново природан број).

Напиши програм који за дати број n одређује најмањи број који се може добити применом ове две операције.

Опис улаза

Са стандардног улаза се учитава природан број n ($1 \leq n \leq 10^12$).

Опис излаза

На стандардни излаз исписати тражени најмањи број.

Пример 1

Улаз	*Излаз*	*Објашњење*
`20`	`10`	Број 20 се може помножити бројем 5, а затим се може кореновати и тако добити број 10. Ниједан број мањи од 10 није могуће добити.

Пример 2

Улаз	*Излаз*	*Објашњење*
`540`	`30`	Број 540 се може помножити бројем 1500, а затим се може два пута кореновати и тако добити број 30. Ниједан број мањи од 30 није могуће добити.

Решење

Растављање на просте чиниоце

Кључ решења задатака је да се на основу основне теореме аритметике број представи као производ простих чинилаца $n = p_1^{k_1} \cdot ... \cdot p_m^{k_m}$.

Након множења неким бројем сви ови чиниоци остају присутни и у производу (а могу се евентуално појавити неки нови). Број се може кореновати ако и само ако су сви експоненти k_1 до k_m парни и тада се они смањују на пола, али ни при тој операцији ниједан прост чинилац p_i не може нестати из резултата кореновања.

Дакле, применом ових операција увек се добија резултат који садржи све просте чиниоце p_1 до p_m.

Са друге стране, применом операција је могуће добити број $p_1 \cdot \ldots \cdot p_m$. Заиста, нека је k најмањи степен двојке који је већи или једнак од свих експонената k_1 до k_m. Множењем полазног броја бројем $p_1^{k-k_1} \cdot \ldots \cdot p_m^{k-k_m}$ добијамо број $p_1^k \cdot \ldots \cdot p_m^k$. Пошто је k степен броја 2 кореновањем тог броја се експоненти смањују на пола, све док не стигну до 1.

Дакле, минимални број који се може добити је производ различитих простих фактора броја n и он се може одредити уобичајеним алгоритмом факторизације броја.

```
#include <iostream>

using namespace std;

typedef unsigned long long ull;

int main() {
  ull n;
  cin >> n;
  ull rez = 1;
  ull d = 2;
  while (d * d <= n) {
    if (n % d == 0) {
      rez *= d;
      while (n % d == 0)
        n /= d;
    }
    d++;
  }
  if (n > 1)
    rez *= n;
  cout << rez << endl;
  return 0;
}
```

Задатак: Срећан низ

Аутор: Марко Илић

Аутор решења: Филип Марић

Такмичење: СИО 2021/22. 1. задатак

Мали Перица је за свој рођендан, као и сва остала деца, пожелео један срећан низ. Низ је срећан ако је највећи заједнички делилац свих његових елемената већи од један. Његови другари су одлучили да му испуне рођенданску жељу и поклоне му један такав низ. Међутим, како и даље нису увежбали тражење најмањег заједничког делиоца, могуће је да њихов низ није одговарајући. Ипак, Перица је свакако прихватио поклон и сада од вас тражи помоћ. Потребно је да нађете срећан низ, такав да је збир апсолутних разлика елемената на истим позицијама у овом низу и низу који је Перица добио минималан.

Опис улаза

У првом реду стандардног улаза налази се број n ($1 \leq n \leq 10^3$), који означава број елемената низа. У наредном реду се налазе n бројева, који представљају елементе низа који је Перица добио. Гарантује се да је сваки елемент низа мањи или једнак 10^5.

Опис излаза

У једином реду стандардног улаза исписати n бројева, који представљају тражени низ. Уколико има више решења, исписати било које.

Пример

Улаз	*Излаз*
`5` `4 3 10 8 12`	`4 4 10 8 12`

Решење

Основна идеја решења је да, практично грубом силом, проверимо све могуће кандидате за НЗД елемената срећног низа. По условима задатка он мора да буде бар 2, а можемо закључити да нема потребе анализирати вредности НЗД веће од максималног елемента низа (јер ће бити потребне веће поправке да би се мањи елементи низа увећали до те вредности него до максимума низа).

За сваку вредност НЗД (обележимо је са d) одређујемо потребну поправку. Укупну поправку рачунамо као збир поправки за сваки елемент x низа a. За сваки елемент x низа a одређујемо њему најближи број дељив са тим d. Кандидати за тај број су $\lfloor \frac{a}{d} \rfloor \cdot d$ и $\lceil \frac{a}{d} \rceil \cdot d$. Растојање до тих бројева је $x \bmod d$ и $d - x \bmod d$, па је поправка једнака мањој од те две вредности.

Ако је M максимум низа a, сложеност овог алгоритма је $O(M \cdot n)$, јер испитујемо $O(M)$ кандидата за најбољи НЗД и за сваки поправку израчунавамо у времену $O(n)$.

Оптимизација претходног поступка се може добити ако се примети да ће поправка бити мања ако је НЗД прост број. Наиме, ако је НЗД сложен, онда је он облика $d = d_1 \cdot d'$. Бројеви дељиви са d_1 су "гушћи" од бројева дељивих са d, па је елементе низа потребно мењати мање да би се дошло до бројева дељивих са d' него до бројева дељивих са d (сваки број дељив са d је уједно дељив и са d', док обратно не мора да важи).

Зато прво Ератостеновим ситом одређујемо све просте бројеве мање од максимума низа, а онда приликом тражења највећег НЗД прескачемо све бројеве који нису прости.

Ератостеново сито захтева $O(M \log(\log M))$ корака. Простих бројева мањих од M има $O(M/logM)$, па је укупна сложеност $O(\frac{MN}{\log M})$, што је око $\log M$ пута брже од полазног алгоритма.

```
#include <iostream>
#include <vector>
#include <limits>
#include <algorithm>

#define ll long long
using namespace std;

int main() {
  // ucitavamo elemente niza
  int n;
```

```
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  // najveci element celog niza
  int maks = *max_element(a.begin(), a.end());

  // Eratostenovim sitom odredjujemo sve proste brojeve manje od
  // maksimalnog elementa niza
  vector<bool> prost(maks+1, true);
  prost[0] = prost[1] = false;
  for (ll i = 2; i <= maks; i++) {
    if (!prost[i]) continue;
    for (ll j = i*i; j <= maks; j += i)
      prost[j] = false;
  }

  // proveravamo sve proste brojeve koji su kandidati za NZD niza (od
  // 2 do maksimuma niza) i trazimo onaj koji zahteva najmanju
  // popravku do srecnog niza

  ll minPopravka = numeric_limits<int>::max();
  int najboljiNZD;
  for (int nzd = 2; nzd <= maks; nzd++) {
    if (!prost[nzd]) continue;
    ll popravka = 0;
    for (int x : a) {
      int ostatak = x % nzd;
      popravka += min(ostatak, nzd - ostatak);
    }
    if (popravka < minPopravka) {
      minPopravka = popravka;
      najboljiNZD = nzd;
    }
  }

  // rekonstruisemo srecan niz na osnovu poznatog NZD
  for (int x : a) {
    int ostatak = x % najboljiNZD;
    if (ostatak < najboljiNZD - ostatak)
      cout << x - ostatak;
    else
      cout << x + najboljiNZD - ostatak;
    cout << ' ';
  }

  return 0;
}
```

Задатак: Број цифара иза зареза

Аутор: Иван Дрецун

Такмичење: СИО 2022/23. 3. задатак

Дат је разломак облика $\frac{x_1 \cdot x_2 \cdot \cdots \cdot x_n}{y_1 \cdot y_2 \cdot \cdots \cdot y_m}$. Напиши програм који одређује број цифара иза зареза датог разломка када се он напише у децималном облику.

Опис улаза

Са стандардног улаза се уноси број n ($1 \leq n \leq 50000$) након чега се, у наредном реду, уноси n бројева x_i ($1 \leq x_i \leq 50000$) одвојених размаком. Затим се уноси број m ($1 \leq m \leq 50000$) након чега се, у наредном реду, уноси m бројева y_i ($1 \leq y_i \leq 50000$).

Опис излаза

На стандардни излаз исписати природан број који представља број цифара иза зареза, ако је тај број коначан. У супротном исписати `Beskonacno`.

Додатна ограничења

- У једној групи тест примера, вредној 40 поена важи $n, m \leq 300$.

Пример 1

Улаз	*Излаз*	*Објашњење*

`2`
`1 2`
`1`
`128`
	`6`	Бројилац разломка је $1 \cdot 2 = 2$, а именилац је 128. Када се тај разломак запише у децималном облику добије се $0,015625$ тако да је број цифара иза зареза 6.

Пример 2

Улаз	*Излаз*	*Објашњење*

`1`
`3`
`2`
`3 6`
	`Beskonacno`	Када се дати разломак $\frac{3}{18}$ запише у децимланом облику добије се $0,1666666\ldots$, односно број цифара иза зареза је бесконачан.

Решење

Број цифара после зареза било ког броја x можемо дефинисати као најмањи број k такав да је $10^k x$ цео број. На пример, ако је $x = 12.345$ можемо установити да је број цифара после зареза 3 зато што је $10^3 x = 1000x = 1000 \cdot 12.345 = 12345$ цео број, док то не важи за $k = 2$ јер је $10^2 x = 100x = 100 \cdot 12.345 = 1234.5$. Такође, ако је број цифара бесконачан, онда такав број k не постоји.

Проблем онда можемо преформулисати: потребно је да одредимо најмање k такво да је $\frac{10^k \cdot x_1 \cdot \cdots \cdot x_n}{y_1 \cdot \cdots \cdot y_m}$ цео број, односно да установимо да такво k не постоји.

Ако је на пример дат разломак $\frac{1}{20}$, важи $10 \cdot \frac{1}{20} = \frac{1}{2}$, а $100 \cdot \frac{1}{20} = 5$, па је решење 2. Ово можемо видети из факторизације броја $20 = 2 \cdot 2 \cdot 5$, пошто се број 2 појављује два пута испод разломачке црте, те множењем разломка бројем 10 једанпут није довољно - једна двојка остаје испод разломачке црте. Дакле, потребно је помножити разломак бројем 10 толико пута да се из факторизације елминишу сва појављивања бројева 2 и 5.

Посматрајмо још један пример где је дат разломак $\frac{1}{15}$. Посматрањем факторизације броја $15 = 3 \cdot 5$. Множењем разломка бројем 10 можемо уклонити 5 из имениоца, али број 3 никако не можемо уклонити. Другим речима, колико год пута да помножимо разломак бројем 10, увек ћемо имати тај број 3 испод разломачке црте. То значи да је број цифара бесконачан.

Задатак онда можемо решити одређивањем факторизације и бројиоца и имениоца разломка. На основу факторизације можемо једноставно скратити разломак и посматрати шта нам остаје испод разломачке црте. Уколико је остао било који прост број различит од 2 и 5, број цифара је бесконачан.Уколико није, онда број цифара одређујемо као већи међу бројевима појављивања фактора 2 и 5.

На пример, ако је дат разломак $\frac{15 \cdot 21 \cdot 4}{6 \cdot 25 \cdot 14}$, факторизација бројиоца је $2^2 \cdot 3^2 \cdot 5^1 \cdot 7^1$, а имениоца $2^2 \cdot 3^1 \cdot 5^2 \cdot 7^1$. Факторизација разломка је онда $2^{2-2} \cdot 3^{2-1} \cdot 5^{1-2} \cdot 7^{1-1} = 3^1 \cdot 5^{-1}$, па нам након скраћивања испод разломачке црте остаје број 5 са степеном 1, па је и решење овог примера 1.

Нека је, за последњи пример, дат разломак $\frac{10 \cdot 33}{28 \cdot 11}$. Факторизација бројиоца је $2^1 \cdot 3^1 \cdot 5^1 \cdot 11^1$, а имениоца $2^2 \cdot 7^1 \cdot 11^1$. Факторизација разломка је $2^{1-2} \cdot 3^{1-0} \cdot 5^{1-0} \cdot 7^{0-1} \cdot 11^{1-1} = 2^{-1} \cdot 3^1 \cdot 5^1 \cdot 7^{-1}$, што значи да након скраћивања испод разломачке црте остаје $2 \cdot 7$, па је због седмице број цифара бесконачан.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int n;
  cin >> n;

  vector<int> x(n);
  for (int i = 0; i < n; i++)
    cin >> x[i];

  int m;
  cin >> m;

  vector<int> y(m);
  for (int i = 0; i < m; i++)
    cin >> y[i];

  vector<int> faktorizacija(50001);

  for (auto xi : x) {
    for (int d = 2; d * d <= xi; d++)
      while (xi % d == 0) {
        faktorizacija[d]++;
        xi /= d;
      }
    if (xi != 1)
      faktorizacija[xi]++;
```

```
  }

  for (auto yi : y) {
    for (int d = 2; d * d <= yi; d++)
      while (yi % d == 0) {
        faktorizacija[d]--;
        yi /= d;
      }
    if (yi != 1)
      faktorizacija[yi]--;
  }

  int brojCifara = -min({0, faktorizacija[2], faktorizacija[5]});

  bool beskonacno = false;
  for (int i = 2; i < faktorizacija.size(); i++)
    if (i != 2 && i != 5 && faktorizacija[i] < 0)
      beskonacno = true;

  if (beskonacno)
    cout << "Beskonacno\n";
  else
    cout << brojCifara << '\n';

  return 0;
}
```

Задатак: Max НЗД

Аутор: Филип Марић

Такмичење: 2023/2024. квалификације 1, VII разред, 6. задатак

Напиши програм који на основу познатог производа два позитивна природна броја a и b одређује највећу могућу вредност њиховог највећег заједничког делиоца.

Опис улаза

Са стандардног улаза се учитава број $p = a \cdot b$ ($1 \leq p \leq 10^{18}$).

Опис излаза

На стандардни излаз исписати максималну могућу вредност за НЗД.

Пример 1

Улаз	*Излаз*	*Објашњење*
`600`	`10`	Највећи НЗД се добија када се број 600 представи као производ бројева 20 и 30.

Пример 2

Улаз	*Излаз*
`123456`	`8`

Решење

Груба сила

Решење грубом силом подразумева да број n на све начине разложимо на производ два броја $a \cdot b$, израчунамо њихов НЗД и одредимо максимум свих тако добијених бројева. Без губитка на општости можемо претпоставити да је $a \leq b$, па је довољно испитивати све вредности a од 1 до $\sqrt{n}$.

```
#include <iostream>
#include <algorithm>

using namespace std;

typedef unsigned long long ull;

ull nzd(ull a, ull b) {
  while (b != 0) {
    ull mod = a % b;
    a = b;
    b = mod;
  }
  return a;
}

int main() {
  ull p;
  cin >> p;
  ull f = 2;
  ull maxNzd = 1;
  while (f * f <= p) {
    if (p % f == 0)
      maxNzd = max(maxNzd, nzd(f, p / f));
    f++;
  }
  cout << maxNzd << endl;
  return 0;
}
```

Растављање на просте чиниоце

Ефикасније решење се добија ако се број растави на просте чиниоце: $p_1^{k_1} p_2^{k_2} \ldots p_m^{k_m}$. Највећи НЗД се добије ако се сваки чинилац p_i равномерно раздели у два броја a и b. Сваки од бројева ће садржати фактор $p_i^{\lfloor \frac{k_i}{2} \rfloor}$ (заиста, ако је k_i паран, оба ће имати тачно тај фактор, а ако је непаран, онда ће један од бројева садржати тај фактор, а други број ће садржати фактор $p_i^{\lfloor \frac{k_i}{2} \rfloor + 1}$).

На пример, важи $600 = 2^3 \cdot 3 \cdot 5^2$. Највећи НЗД се може добити ако бројеви a и b приме по једну двојку и по једну петицу (ирелевантно је како ће се распоредити преостала двојка и тројка).

```
#include <iostream>
```

```cpp
using namespace std;

int main() {
  unsigned long long p;
  cin >> p;
  unsigned long long max = 1;
  unsigned long long f = 2;
  while (f * f <= p) {
    int k = 0;
    while (p % f == 0) {
      p /= f;
      k++;
      if (k % 2 == 0)
        max *= f;
    }
    f++;
  }
  cout << max << endl;
  return 0;
}
```

Глава 10

Примена сортирања

Задатак: Најређе написана реч

Такмичење: државно 2018/2019, VII и VIII разред, 3. задатак

Аутор: Јелена Хаџи-Пурић, *Аутор решења*: Филип Марић

Ученици Милица и Ђорђе вежбају обраду ниски. Вежбање се одиграва у више рунди. У једној рунди Ђорђе напише слово, а потом Милица напише реч која почиње Ђолетовим словом. У свакој рунди, Милица бира речи поштујући додатна ограничења:

- постоји унапред дат списак дозвољених речи које Милица може да користи
- Милица увек бира ретке речи са списка (прецизније: речи које је до тада написала најмањи број пута).

Сматрајте да Милица увек може да изабере највише једну реч, јер ако постоји вишеструки избор, онда Милица бира ону реч која је лексикографски мања. Дакле Милица бира најређе записану реч која је у речнику организованом по абецеди била наведена раније. Додатно, можете да претпоставите да за свако слово које је Ђорђе задао, Милица може одабрати реч са списка.

Опис улаза

У првом реду стандардног улаза дата су два броја: K и N тако да $K (1 \leq K \leq 10^5)$ представља број различитих речи са списка, док број N $(1 \leq N \leq 10^5)$ означава да радимо са низом од N слова које задаје Ђорђе. У наредних K редова стандардног улаза дата је по једна реч са списка. Свака реч је састављена од малих слова енглеске абецеде. У наредних N редова налази се по једно мало слово енглеске абецеде које задаје Ђорђе.

Опис излаза

На стандардни излаз исписати речи које које је Милица исписивала као одговор на слова која је Ђорђе записивао.

Пример

Улаз	*Излаз*

`4 5`
`kmet`
`beba`
`kinez`
`bobina`
`k`
`b`
`b`
`k`
`k`
	`kinez`
	`beba`
	`bobina`
	`kmet`
	`kinez`

Објашњење

- Када Ђорђе напише слово `k` у 1. рунди, Милица са списка бира лексикографски најмању реч која почиње словом `k` и коју је до тада написала најмањи број пута, конкретно 0 пута, а то је реч `kinez`.
- Када Ђорђе напише слово `b` у 2. рунди, Милица са списка бира лексикографски најмању реч која почиње словом `b` и коју је до тада написала најмањи број пута, конкретно 0 пута, а то је реч `beba`.
- Када Ђорђе напише слово `b` у 3. рунди, Милица са списка бира лексикографски најмању реч која почиње словом `b` и коју је до тада написала 0 пута, а то је реч `bobina`.
- Када Ђорђе напише слово `k` у 4. рунди, Милица са списка бира лексикографски најмању реч која почиње словом `k` и коју је до тада написала 0 пута, а то је реч `kmet`.
- Када Ђорђе напише слово `k` у 5. рунди, Милица са списка бира лексикографски најмању реч која почиње словом `k` и коју је до тада написала 1 пут, а то је реч `kinez`.

Решење

Анализирајмо шта се догађа када се уноси једно почетно слово. На почетку се исписује лексикографски најмања реч која почиње тим словом (јер су све речи које почињу тим словом исписане 0 пута). Након тога је она исписана један, а остале нула пута, па се следећа исписује реч која је друга у лексикографском редоследу. Након тога су прва и друга реч у лексикографском редоследу исписане по једном, а остале нула пута, па се следећа исписује трећа реч у лексикографском редоследу. Поступак се понавља и редом се исписују све речи по лексикографском редоследу. Након тога су све речи исписане по једном и исти поступак креће из почетка.

Дакле, задатак решавамо тако што се за свако слово циклично исписују речи по лексикографском редоследу. Учитане речи ћемо груписати по првом слову и сортираћемо сваку групу речи. За свако слово памтићемо редни број речи коју наредну треба исписати.

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int k, n;
  cin >> k >> n;

  // za svako malo slovo ucitavamo sve reci na to slovo
  vector<vector<string>> a(26);
  for (int i = 0; i < k; i++) {
    string s;
    cin >> s >> ws;
    a[s[0] - 'a'].push_back(s);
  }
```

```
  // za svako slovo sortiramo sve reci leksikografski
  for (int i = 0; i < 26; i++)
    sort(begin(a[i]), end(a[i]));

  // redni broj tekuce reci za svako slovo (svi brojaci krecu od 0)
  vector<int> p(26, 0);

  // obradjujemo sve karaktere
  for (int i = 0; i < n; i++) {
    // ucitavamo karakter
    char c;
    cin >> c >> ws;
    c -= 'a';
    // ispisujemo tekucu rec na to slovo i prelazimo na narednu
    // kada ispisemo poslednju rec, vracamo se na prvu
    cout << a[c][p[c]] << endl;
    p[c] = (p[c] + 1) % a[c].size();
  }

  return 0;
}
```

Задатак: Клин

Аутор: Милан Вуīелија

Такмичење: 2021/22. квалификације 1, VII разред 4. задатак

Написати програм који утврђује да ли низ датих речи представља формацију, која се у енигматици назива клин.

Низ речи чини клин ако свака следећа реч може да се добије од претходне избацивањем једног слова и по потреби мењањем редоследа осталих слова. Последња реч у формацији клина треба да има једно слово.

Опис улаза

У првом реду стандардног улаза се налази цео позитиван број N, број речи у низу. У сваком од следећих N редова је по једна реч састављена од великих слова енглеске абецеде. Број N и дужине речи нису веће од 20.

Опис излаза

На стандардни излаз исписати само реч `DA` или реч `NE`.

Пример 1

Улаз	*Излаз*

`4`
`PLAV`
`VAL`
`LA`
`A`
	`DA`

Пример 2

Улаз	*Излаз*	*Објашњење*
3 VLAGA LAVA VAL	NE	Последња реч није дужине 1.

Пример 3

Улаз	*Излаз*	*Објашњење*
3 VLAGA LAV L	NE	Речи се не скраћују за по једно слово.

Пример 4

Улаз	*Излаз*	*Објашњење*
4 HLAD LAV LA L	NE	Друга реч се не добија од прве по описаном правилу.

Решење

Из поставке задатка је јасно да дужине речи морају да буду редом n, $n-1$, … 1. Због тога за сваку учитану реч прво проверавамо да ли је одговарајуће дужине, па ако није - даље провере могу да се зауставе јер знамо да је коначан одговор одречан.

Осим дужина, за сваку реч осим прве треба да проверимо да ли може да се добије од претходне на начин описан у поставци задатка. Да бисмо то што једноставније проверили, згодно је да приликом учитавања сваке речи слова у њој сортирамо растуће. Након сортирања, пролазимо кроз претходну и текућу реч упоредо. Када су слова у речима иста, напредујемо у обе речи, а када су различита, напредујемо само у претходној (дужој) речи и бројимо разлику. Да би тражени услов био испуњен, по проласку кроз две речи, број разлика треба да буде 1. Ако је добијени број разлика мањи или већи, даље провере могу да се прекину, јер је и у овом случају коначан одговор одречан.

Уколико по проласку кроз све речи нисмо наишли на неправилности, коначан одговор је потврдан.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
    int nPre;
    string sPre, sSled;
    cin >> nPre >> sPre;
    sort(sPre.begin(), sPre.end());
    bool nizJeKlin = true;
    if (sPre.size() != nPre)
        nizJeKlin = false;
```

```
    for (int nSled = nPre - 1; nSled > 0 && nizJeKlin; nSled--) {
        cin >> sSled;
        sort(sSled.begin(), sSled.end());
        if (sSled.size() != nSled)
            nizJeKlin = false;
        else {
            int iPre = 0, iSled = 0, brPreskocenih = 0;
            for (iPre = 0; iPre < nPre; iPre++) {
                if (iSled >= nSled || sPre[iPre] != sSled[iSled])
                    brPreskocenih++;
                else iSled++;
            }
            if (brPreskocenih != 1)
                nizJeKlin = false;
        }
        sPre = sSled;
        nPre = nSled;
    }

    if (nizJeKlin)
        cout << "DA" << endl;
    else
        cout << "NE" << endl;
   return 0;
}
```

Задатак: Сорт по парности

Аутор: Милан Вугделија

Такмичење: државно 2021/22., V и VI разред, 3. задатак

Дати низ има једнак број парних и непарних елемената. Исписати елементе низа у таквом редоследу да се парни елементи налазе на местима са парним индексом а непарни елементи на местима непарним индексом, и да при томе парни и непарни елементи чине два неопадајућа подниза.

Опис улаза

У првом реду стандардног улаза је паран природан број N, који није већи од 50000. Сваком од следећих N редова је по један елемент датог низа целих бројева, мањих од 10^9.

Опис излаза

На стандардни излаз у једном реду, раздвојене по једним размаком, исписати елементе датог низа у траженом редоследу.

Пример

Улаз	*Излаз*
8 4 1 8 6 2 11 5 9	2 1 4 5 6 9 8 11

Решење

Најједноставнији начин да се реши овај задатак је да се корите два додатна, краћа низа, један за парне а други за непарне бројеве. Када сваки од датих бројева сместимо у одговарајући низ, можемо да сортирамо сваки од ова два низа независно, а затим да њихове елементе наизменично убацујемо назад у велики, полазни низ.

Простор за полазни низ може и да се уштеди, јер бројеве можемо и одмах при учитавању да разврставамо у низове за парне и непарне бројеве, а након сортирања низова да наизменично исписујемо бројеве на стандардни излаз уместо да их враћамо у полазни низ.

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

int main() {
    int n, np, nn, i;
    cin >> n;
    vector<int> a(n);
    for (i = 0; i < n; i++)
        cin >> a[i];

    vector<int> parni(n/2);
    vector<int> neparni(n/2);
    np = 0;
    nn = 0;
    for (i = 0; i < n; i++) {
        if (a[i] % 2 == 0) {
            parni[np] = a[i];
            np++;
        } else {
            neparni[nn] = a[i];
            nn++;
        }
    }
    sort(parni.begin(), parni.end());
```

```
    sort(neparni.begin(), neparni.end());
    i = 0;
    for (int k = 0; k < n / 2; k++) {
        a[i] = parni[k];
        i++;
        a[i] = neparni[k];
        i++;
    }
    for (i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
}
```

Задатак: Највећи број од датих цифара

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 3, V и VI разред, 4. задатак и VII и VIII разред, 2. задатак

За дати број исписати највећи број који се пише истим цифрама.

Опис улаза

Са стандардног улаза се у првом реду уноси број N ($1 \leq N \leq 10^{30}$).

Опис излаза

На стандардни излаз исписати тражени број.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
`90123`	`93210`	`777`	`777`

Решење

Задатак једноставно решавамо тако што сортирамо цифре опадајуће. С обзиром на велики број цифара, најбоље је резултат представити ниском карактера.

```
#include <iostream>
#include <string>
#include <algorithm>
#include <functional>

using namespace std;

int main() {
  string cifre;
  cin >> cifre;
  sort(begin(cifre), end(cifre), greater<char>());
  cout << cifre << endl;
```

```
  return 0;
}
```

Резултат је могуће представити и низом цифара.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <functional>

using namespace std;

int main() {
  vector<int> cifre;
  char c;
  while (cin >> c && c != '\n')
    cifre.push_back(c - '0');
  sort(begin(cifre), end(cifre), greater<int>());
  for (size_t i = 0; i < cifre.size(); i++)
    cout << cifre[i];
  cout << endl;

  return 0;
}
```

Задатак: К најближих

Аутори: Владан Ковачевић, Иван Дрецун

Такмичење: окружно 2022/2023, VII разред, 4. задатак

Дат је низ од n целих бројева и један цео број x. Напиши програм који одређује k бројева из низа, најближих броју x (уколико су два броја на истој удаљености броју x, бира се мањи од њих). Под удаљеношћу се сматра апсолутна вредност разлике бројева.

Опис улаза

Из прве линије стандардног улаза се уносе бројеви n ($1 \leq n \leq 10^5$), k ($1 \leq k \leq n$) и x ($1 \leq x \leq 10^9$) раздвојени размацима. Затим се у наредној линији уноси n целих бројева a_i ($1 \leq a_i \leq 10^9$) који представљају елементе низа. Ови бројеви су, такође, раздвојени размацима.

Опис излаза

На стандардни излаз исписати тражених k бројева раздвојених размацима, уређених неопадајуће.

Пример 1

Улаз	*Излаз*	*Објашњење*
`7 4 4`	`2 3 4 5`	Од свих унетих бројева, 4 броја која су најближа броју 4 су 2, 3, 4 и 5.
`7 4 2 3 1 5 6`		Остали бројеви имају већу разлику у односу на број 4.

Пример 2

Улаз	*Излаз*
`5 3 2` `3 1 3 1 3`	`1 1 3`

Пример 3

Улаз	*Излаз*
`5 3 20` `5 3 2 4 1`	`3 4 5`

Решење

Опис главног решења

Приметимо да се након сортирања низа тражених k бројева налазе на узастопним позицијама. Претрагу таквих сегмената дужине k почињемо од левог краја сортираног низа. У сваком кораку сегмент померамо за по једно место удесно и заустављамо се онда када бисмо у сегмент на десни крај додали број који је даље од x него број који бисмо избацили са левог краја.

На пример, нека је соритрани низ `1 2 5 7 8 10 11`, $x = 6$ и $k = 3$. Тада у првом кораку претраге разматрамо сегмент `1 2 5`, затим `2 5 7`, затим `5 7 8` након чега се претрага зауставља. У следећем кораку бисмо разматрали сегмент `7 8 10` чији је десни крај (10) даље од $x = 6$ него леви крај претходног сегмента (5), што значи да на овом месту бројеви почињу да се удаљавају од траженог решења.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int n, k, x;
  cin >> n >> k >> x;

  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  sort(begin(a), end(a));

  int i = k;
  while (i < n && abs(a[i] - x) < abs(a[i - k] - x))
    i++;

  for (int j = i - k; j < i; j++)
    cout << a[j] << ' ';

  return 0;
}
```

Задатак: Атп листа

Аутор: Филип Марић

Такмичење: 2019/20 квалификације 3, VII и VIII разред, 4. задатак

Током године играју се многи тениски турнири на којима тенисери освајају поене. Поени се сабирају и на крају године објављује се завршна листа на којој су тенисери рангирани на основу укупног броја поена током те године. Напиши програм који на основу резултата свих турнира одређује k најбољих тенисера и њихове поене (претпоставити да ће сви тенисери имати различит број поена).

Опис улаза

Са стандардног улаза се учитава број турнира n ($1 \leq n \leq 1000$), а затим подаци о освојеним поенима на тим турнирима. За сваки турнир се учитава број m који представља број тенисера који су освајали поене, и након тога m линија које садрже податке о освојеним поенима тенисера на том турниру (презиме тенисера које има највише 50 карактера и број поена између 10 и 2000). На крају се уноси број k који одређује колико најбољих тенисера треба одредити.

Опис излаза

На стандардни излаз исписати презимена и укупан освојени број поена за k најбољих тенисера у опадајућем редоследу укупног освојеног броја поена (ако тенисери имају исти број поена, предност има онај чије је презиме веће у лексикографском редоследу).

Пример

Улаз	*Излаз*
3 3 Djokovic 1000 Nadal 800 Federer 600 3 Nadal 1000 Federer 800 Djokovic 600 3 Djokovic 1000 Cicipas 800 Nadal 600 3	Djokovic 2600 Nadal 2400 Federer 1400

Решење

У првој фази програма је потребно да израчунамо укупан број поена за сваког играча. Податке можемо чувати у мапи која пресликава име играча у његов број поена. Након тога треба одредити k највећих вредности у мапи. Један начин је да се сви елементи из мапе прекопирају у низ, да се низ сортира опадајуће и да се прочита првих k елемената тако сортираног низа.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <map>

using namespace std;

int main() {
```

```
  ios_base::sync_with_stdio(false);
  map<string, int> bodovi;

  int broj_turnira;
  cin >> broj_turnira;
  for (int i = 0; i < broj_turnira; i++) {
    int broj_tenisera;
    cin >> broj_tenisera;
    for (int j = 0; j < broj_tenisera; j++) {
      string teniser; int bodovi_na_turniru;
      cin >> teniser >> bodovi_na_turniru >> ws;
      bodovi[teniser] += bodovi_na_turniru;
    }
  }

  int k;
  cin >> k;
  vector<pair<string, int>> teniseri(bodovi.size());
  int i = 0;
  for (auto it : bodovi)
    teniseri[i++] = it;
  sort(begin(teniseri), end(teniseri),
       [](const pair<string, int>& p1, const pair<string, int>& p2) {
         return p1.second > p2.second ||
           (p1.second == p2.second && p1.first > p2.first);
       });
  for (int i = 0; i < k; i++)
    cout << teniseri[i].first << " " << teniseri[i].second << endl;

  return 0;
}
```

У језику C++ могуће је одредити k највећих елемената без сортирања целе колекције (што је донекле ефикасније). За колекције у којима је могуће произвољно мењати редослед елемената (као што је низ или вектор) можемо употребити функцију `partial_sort`. Пошто је редослед елемената у мапи фиксиран (мапа је уређена на основу кључева), за одређивање највећих елемената мапе можемо употребити библиотечку функцију `partial_sort_copy` која ће тих k највећих елемената прекопирати у нови низ (или вектор).

```
#include <iostream>
#include <algorithm>
#include <map>
#include <vector>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  map<string, int> bodovi;
```

```
  int broj_turnira;
  cin >> broj_turnira;
  for (int i = 0; i < broj_turnira; i++) {
    int broj_tenisera;
    cin >> broj_tenisera;
    for (int j = 0; j < broj_tenisera; j++) {
      string teniser; int bodovi_na_turniru;
      cin >> teniser >> bodovi_na_turniru >> ws;
      bodovi[teniser] += bodovi_na_turniru;
    }
  }

  int k;
  cin >> k;
  vector<pair<string, int>> najbolji(k);
  partial_sort_copy(bodovi.begin(), bodovi.end(),
                    najbolji.begin(), najbolji.end(),
                    [](const pair<string, int>& p1,
                       const pair<string, int>& p2) {
                      return p1.second > p2.second ||
                        (p1.second == p2.second && p1.first > p2.first);
                    });
  for (auto p : najbolji)
    cout << p.first << " " << p.second << endl;
  return 0;
}
```

Глава 11

Сложеност израчунавања 1

Задатак: Ивице аритметичког троугла

Такмичење: општинско 2018/2019, VIII разред, 4. задатак

Аутор: Филип Марић

Троугао има n врста и почетни елемент му је 0. Свака следећа врста има два елемента више него претходна, први елемент сваке врсте је за d_1 већи од првог елемента претходне врсте, док је у свакој врсти сваки наредни елемент за d_2 већи од претходног. На пример, за $n = 4$, $d_1 = 5$ и $d_2 = 2$ добијамо следећи троугао.

```
          0
        5  7  9
     10 12 14 16 18
  15 17 19 21 23 25 27
```

Напиши програм који исписује збир елемената дуж ивица троугла.

Опис улаза

Са стандардног улаза се учитавају бројеви d_1 ($1 \leq d_1 \leq 10$), d_2 ($1 \leq d_2 \leq 10$) и n ($2 \leq n \leq 1000$).

Опис излаза

На стандардни излаз исписати тражени збир.

Пример

Улаз	*Излаз*
5 2 4	189

Решење

Елементи на левој ивици троугла чине аритметички низ чији је први елемент 0 и разлика d_1. Можемо приметити и да елементи на десној ивици троугла чине аритметички низ. Заиста, у

врсти са редним бројем i има $2i + 1$ елемената, први елемент је $i \cdot d_1$, а последњи елемент је $i \cdot d_1 + 2 \cdot i \cdot d_2$. У наредној врсти последњи елемент је $(i+1) \cdot d_1 + 2 \cdot (i+1) \cdot d_2$, па је разлика суседних елемената на десној ивици троугла једнака $d_1 + 2d_2$, што је константа вредност. Са обе те ивице можемо узети по $n - 1$ елемент, при чему ћемо њихове последње елементе уврстити у збир елемената на доњој ивици. И елементи на доњој ивици представљају аритметички низ чији је први елемент једнак $(n - 1) \cdot d_1$, коме је разлика између два суседна елемента једнака d_2, при чему ћемо сабирати првих $2n - 1$ елемената тог низа.

Збир n елемената аритметичког низа чији је први члан a, разлика два суседна елемента једнака d једнака $\frac{(a+(a+(n-1)d))\cdot n}{2}$. Ако дефинишемо функцију која израчунатава тај збир, до коначног решења долазимо једноставно тако што је позовемо три пута.

```
#include <iostream>

using namespace std;

// zbir aritmetickog niza a, a+d, a+2d, ..., a+(n-1)d
int aritmeticki(int a, int d, int n) {
  return (a + (a + (n-1)*d)) * n / 2;
}

int main() {
  int d1, d2, n;
  cin >> d1 >> d2 >> n;
  int zbir = aritmeticki(0, d1, n-1) +          // leva ivica
             aritmeticki(0, d1 + 2*d2, n-1) +   // desna ivica
             aritmeticki((n-1)*d1, d2, 2*n-1);  // donja ivica
  cout << zbir << endl;
  return 0;
}
```

Задатак: Максимални принос

Аутор: Филип Марић

Такмичење: Ревијално такмичење 2019/2020., VII и VIII разред, 1. задатак

Фармер поседује њиву димензије $a \times b$ метара. Да би лакше парцелисао њиву, бројеви a и b су цели. На основу субвенције добио је могућности да продужи странице своје њиве укупно за c метара (али тако да њива остане целобројних димензија). Он жели да то уради тако да након продужења површина буде што већа, тако да може да оствари што већи укупан принос. Напиши програм који одређује највећу могућу површину њиве након продужења страница.

Опис улаза

Са стандардног улаза се учитавају природни бројеви $a, b, c \leq 10^4$, раздвојени са по једним размаком.

Опис излаза

На стандардни излаз исписати максималну површину након продужења страница.

Пример 1

Улаз	*Излаз*	*Објашњење*
`5 10 3`	`80`	Димензија након проширења ће бити 8×10.

Пример 2

Улаз	*Излаз*	*Објашњење*
`9 10 4`	`132`	Димензија након проширења ће бити 11×12.

Пример 3

Улаз	*Излаз*	*Објашњење*
`14 17 5`	`324`	Димензија након проширења ће бити 18×18.

Решење

Од свих правоугаоника датог фиксираног обима, највећу површину има квадрат. Заиста, ако је познат обим правоугаоника $O = 2(a + b)$, тада је познат и полуобим $a + b = s$. Површина $a \cdot b = a \cdot (s - a) = a \cdot s - a \cdot a = s^2/4 - (a - s/2)^2$. Пошто је $(a - s/2)^2 \geq 0$, површина не може бити већа од $s^2/4$, а једнака је тој вредности када је $a = b = s/2$. Зато увећање треба направити тако да се добије облик који је што сличнији квадрату.

Нека је $a \leq b$. Ако је $a + c \leq b$, тада се целокупан износ увећања c може додати на мању страницу a. У супротном се прво краћа страница a продужи тако да постане једнака дужој страници b, а затим се преостали износ увећања $(c - (b - a))$ подели што равномерније могуће (ако је то паран број може се добити квадрат, а ако није, тада се добија правоугаоник код којег је једна страница за један дужа од друге). У имплементацији тај ефекат можемо постићи тако што страницу b увећамо за $\left\lfloor \frac{c-(b-a)}{2} \right\rfloor$ и $\left\lceil \frac{c-(b-a)}{2} \right\rceil$.

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
  int a, b, c;
  cin >> a >> b >> c;
  if (a > b) swap(a, b);
  if (c <= b - a)
    a += c;
  else {
    int preostalo = c - (b - a);
    a = b + preostalo / 2;
    b = b + (preostalo + 1) / 2;
  }
  cout << a * b << endl;
  return 0;
}
```

Задатак: Ћуки

аутор: Душан Поп̄адић

Такмичење: 2021/22. квалификације 3, VI разред, 6. задатак

Пас звани Ћуки трчи по правој стази дужине L. На стази се на одређеним местима налазе звонца, а сваки пут када Ћуки пролази поред звонца, оно зазвони. Када Ћуки стигне до краја стазе он се окреће и трчи ка почетку, када стигне на почетак онда опет ка крају и тако даље. Сваки пут када се Ћуки окрене (ка почетку или ка крају), он скраћује дистанцу коју прелази до следећег окрета за d. Када дистанца коју треба да претрчи постане 0 (или мања од 0) Ћуки стаје. Одреди колико је пута свако звоно зазвонило током Ћукијевог трчања.

Опис улаза

У првом реду стандардног улаза се налазе три цела броја: дужина стазе L ($10 \leq L \leq 10^{18}$), скраћење D ($1 \leq D \leq L$, L је дељиво са D) и број звонца n ($1 \leq n \leq 10^6$). У наредних n редова се налази по један цео број x_i ($0 \leq x_i \leq L$) који представља удаљенсот i-тог звонца од почетка стазе.

Ограничења

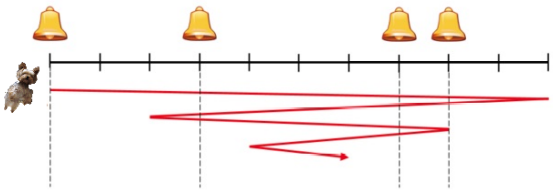
У тест примерима вредним 50 поена важи $\frac{L}{D} \leq 50$.

Опис излаза

У n редова стандардног излаза исписати колико пута је свако звонце зазвонило током Ћукијевог трчања и то редом којим су се уносила звонца.

Пример

Улаз	*Излаз*	*Објашњење*
`10 2 4`	`1`	На слици се може видети Ћукијева путања.
`0`	`3`	
`3`	`4`	
`7`	`3`	
`8`		



Решење

Решење симулацијом

Једно решење овог задатка је да се симулира Ћукијево кретање тако што се поставе лева и десна граница трчања које се после сваког окретања наизменично смањују за D. Треба имати посебан низ у који се бележи за свако звонце колико је пута зазвонило. При сваком окрету треба проћи кроз низ свих звонаца и убележити за свако које се налази између леве и десне границе да је зазвонило (његова вредност и низу се повећава за 1). За звонца код којих се Ћуки окреће, број пролазака треба умањити за 1.

Сложеност овог решења је $O(broj\_okreta \cdot broj\_zvonaca) = O(\frac{L}{D} \cdot n)$. Како однос $\frac{L}{D}$ може бити изузетно велик, ово решење доноси само 50 поена.

```
#include <iostream>
#include <vector>
```

```cpp
#include <cmath>

using namespace std;

int main()
{
  ios_base::sync_with_stdio(false);
  cin.tie(0);

  long long L, D;
  cin >> L >> D;

  // ucitavamo pozicije zvonca
  int n;
  cin >> n;
  vector<long long> zv(n);
  for (int i = 0; i < n; i++)
    cin >> zv[i];

  // broj zvonjenja svakog zvonca
  vector<long long> zvonjenja(n);
  long long l = 0;
  long long d = L;
  bool udesno = true;
  while (l <= d) {
    for (int i = 0; i < n; i++)
      if (l <= zv[i] && zv[i] <= d)
        zvonjenja[i]++;
    if (udesno)
      l += D;
    else
      d -= D;
    udesno = !udesno;
  }

  // broj zvonjenja zvonca kod kojih se Cuki okrece treba smanjiti za 1
  for (int i = 0; i < n; i++)
    if (zv[i] != 0 && zv[i] % D == 0)
      zvonjenja[i] -= 1;

  for (int i = 0; i < n; i++)
    cout << zvonjenja[i] << '\n';

  return 0;
}
```

Решење одређивањем формуле

Могуће је избећи симулацију тако што се примети да је за свако звонце могуће израчунати колико пута ће зазвонити извођењем математичке формуле. Посматрајмо слику из објашњења примера. Можемо приметити да поред свих звонца у левом делу слике Ћуки пролази непаран број пута, а поред свих звонца у десном делу слике Ћуки пролази паран број пута (изузеци су она звонца испред којих се Ћуки окреће и њих ћемо засебно размотрити за касније). Такође, што је звонце ближе средини стазе, то Ћуки више пута прође поред њега. Поред сваке две позиције између којих се Ћуки никад не окреће, он пролази исти број пута. Дакле кључно је посматрати тачке у којим се Ћуки окреће. Он креће од позиције 0, окреће се на позицији L, иде назад до позиције D на којој се окреће и иде до позиције $L-D$, затим се враћа до позиције $2D$ где се окреће и иде до позиције $L-2D$ итд. Низ тачака окрета је, дакле, 0, D, $2D$, $3D$, ..., $L-3D$, $L-2D$, $L-D$, L (овде су урачунате и почетна и крајња тачка Ћукијеве путање).

- Звонца у левој половини цртежа су таква да се Ћуки мање пута окрене лево од њих, него десно од њих и број пролазака Ћукија поред сваког таквог звонца је непаран број и једнак је $2\cdot(l_o-1)+1$, где је l_o број окрета Ћукија са леве стране звонца (укључујући и почетну тачку 0) тј. број тачака низа окрета које су мање од позиције звонца x (за сада претпостављамо да x не може бити једнак некој од тих вредности). Заиста за сваки елемент низа лево од x осим елемента 0 (а њих је l_o-1) Ћуки два пута прође поред звонца x (једном враћајући се налево ка тој тачки и други пут одлазећи од ње надесно), док за елемент 0 Ћуки само једном пролази поред тачке x (крећући се од тачке 0 надесно). Одредимо l_o. Тражимо највећу тачку облика kD лево од x. Највећи број k за који важи $kD<x$ је $k=\lfloor\frac{x}{D}\rfloor$, а тачака у низу закључно са том тачком има $k+1$, па је број окрета $l_o=\lfloor\frac{x}{D}\rfloor+1$ и зато је број пролазака поред звонца на позицији x једнак $2\lfloor\frac{x}{D}\rfloor+1$.
- Звонца у десној половини цртежа су таква да се Ћуки више пута окрене лево од њих, него десно од њих и број пролазака Ћукија поред сваког таквог звонца је паран број и једнак је $2\cdot d_o$, где је d_o број окрета Ћукија са десне стране тог звонца тј. број тачака низа окрета које су веће од позиције звонца x (за сада претпостављамо да x не може бити једнак некој од тих вредности). Заиста за сваки елемент низа десно од x Ћуки два пута прође поред звонца x (једном крећући се надесно ка тој тачки и други пут враћајући се од ње налево). Одредимо d_o. Тражимо најмању тачку облика $L-kD$ која је десно од x. Највећи број k за који важи $L-kD>x$ је највеће k за које важи $kD<L-x$, а то је $\lfloor\frac{L-x}{D}\rfloor$. Тачака у низу кренувши од тачке $L-kD$, закључно са тачком L има $k+1$, па је број окрета десно од x једнак $d_o=\lfloor\frac{L-x}{D}\rfloor+1$, док је број пролазака поред тачке x једнак $2\cdot d_o=2\cdot\left(\lfloor\frac{L-x}{D}\rfloor+1\right)$.
- У случају када је број тачака окрета исти са обе стране звонца, број пролазака поред звонца је непаран и треба применити формулу за звонца у левој половини цртежа.

Специјалан случај представљају звонца у самим тачкама окрета. Број окрета лево и десно од њих израчунавамо на потпуно исти начин као и за остала звонца. Међутим, пошто се Ћуки поред тих звонца окреће уместо да прође једном крећући се улево и једном крећући се удесно, потребно је смањити број пролазака који би се добио на основу горе изведених формула за 1.

Сложеност овог решења је $O(n)$ и оно доноси максималан број поена.

```
#include <iostream>
#include <cmath>

using namespace std;
```

```
int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    long long L, D, n;
    cin >> L >> D >> n;

    for (int i = 0; i < n; i++) {
        long long x;
        cin >> x;

        long long l_o = x / D + 1;
        long long d_o = (L - x) / D + 1;
        long long z;
        if (l_o <= d_o)
          z = 2*(l_o - 1) + 1;
        else
          z = 2*d_o;

        if (x != 0 && x % D == 0)
            z--;

        cout << z << "\n";
    }

    return 0;
}
```

Задатак: Подела књиге

Такмичење: државно 2018/2019, V и VI разред, 2. задатак

Аутор: Филип Марић

Низом a дате су дужине n поглавља једне књиге. Елементом a_0 је дат број страница за прво поглавље, елементом a_1 за друго и тако редом. Потребно је књигу поделити на два дела, тако да се укупни бројеви страна у та два дела најмање разликују. Подела се врши између поглавља. Ако постоје две могуће поделе са истом најмањом разликом, подела се врши након ранијег од два поглавља (тако да први део буде краћи). Напиши програм који одређује након ког поглавља треба извршити поделу.

Опис улаза

У првој линији стандардног улаза налази се природан број n ($1 < n \leq 50000$). У следећих n линија налазе се по један природан број (између 1 и 1000), бројеви представљају бројеве страница сваког поглавља.

Опис излаза

На стандарном излазу приказати у једној линији редни број поглавља (поглавља се броје од

нуле) после ког треба извршити поделу књиге.

Пример

Улаз	*Излаз*	*Објашњење*
5 100 120 50 150 70	1	Када се подела изврши после поглавља број 1 (то је друго поглавље), тада први део књиге садржи 220, а други 270 страна, што даје најмању могућу разлику од 50 страна. Ако би се подела извршила након поглавља број 2 (то је треће поглавље), први део би садржао 270 страна, а други 220, што би такође била разлика 50, међутим, по условима задатка у том случају подела се врши тако да први део књиге буде краћи, па је решење 1 (подела се врши после поглавља број 1).

Решење

Груба сила

Најдиректније решење задатка је грубом силом, где за свако поглавље p одређујемо број страница pre_d до тог поглавља, укључујући и то поглавље (збир елемената низа од позиције 0 закључно са позицијом p), и број страница књиге $posle_d$ после тог поглавља (збир елемената низа од позиције $d+1$ закључно са позицијом $n-1$) и одредимо за које поглавље се те две суме минимало разликују. Тражимо минималну вредност израза $|posle_p - pre_p|$, што радимо уобичајеним поступком.

Наивно решење у сваком кораку директно израчунава pre_p и $posle_p$. Ови збирови се директно могу израчунати применом алгоритма сабирање серије бројева (збир елемената постављамо на нулу и онда у петљи збир увећавамо за вредности у датом сегменту низа). Могуће је и коришћење библиотечке функције `accumulate`.

У оба случаја број корака потребан да се израчунају оба збира је линеарно пропорционалан укупном броју поглавља. Зато је број корака потребних за анализу свих поглавља пропорционалан квадрату броја поглавља (сложеност је $O(n^2)$, што је много неефикасније него што ћемо видети да је заиста потребно).

```
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>
using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  // ucitavamo brojeve strana u svakom poglavlju
  int n;
  cin >> n;
  vector<int> brojStrana(n);
  for (int i = 0; i < n; i++)
    cin >> brojStrana[i];

  // najmanja razlika i poglavlje posle kog je ona postignuta
  int minRazlika = numeric_limits<int>::max();
  int minPoglavlje = -1;
```

```
  for (int poglavlje = 0; poglavlje < n; poglavlje++) {
    // broj strana u prvom delu zakljucno sa poglavljem poglavlje
    // int pre = accumulate(begin(brojStrana),
    //                      next(begin(brojStrana), poglavlje+1), 0);
    int pre = 0;
    for (int i = 0; i <= poglavlje; i++)
      pre += brojStrana[i];

    // broj strana u drugom delu knjige nakon poglavlja poglavlje
    // int posle = accumulate(next(begin(brojStrana), poglavlje+1),
    //                        end(brojStrana), 0);
    int posle = 0;
    for (int i = poglavlje + 1; i < n; i++)
      posle += brojStrana[i];

    // razlika broja strana u dva dela knjige
    int razlika = abs(posle - pre);

    // ako je potrebno, azuriramo minimum
    if (razlika < minRazlika) {
      minRazlika = razlika;
      minPoglavlje = poglavlje;
    }
  }

  // ispisujemo rezultat
  cout << minPoglavlje << endl;

  return 0;
}
```

Инкрементално израчунавање броја страна

Парцијалне збирове од првог до неког поглавља, али и парцијалне збирове од неког до последњег поглавља можемо израчунавати инкрементално. Збир pre_p се може веома једноставно израчунати тако што се на на већ познати збир pre_{p-1} дода број страна које садржи поглавље p тј. елемент s_p (важи $pre_p = pre_{p-1} + s_p$). Слично, збир $posle_d$ се може добити ако се од познатог броја $posle_{d-1}$ одузме број страна које садржи поглавље p (важи $posle_p = posle_{p-1} - a_p$).

На основу овога можемо формулисати следећи алгоритам. Одржавамо број pre страна књиге закључно са поглављем p и број страна $posle$ након поглављ p. Њих иницијализујемо као да је подела извршена пре првог поглавља, тј. прву вредност иницијализујемо на нулу, а другу на укупну суму низа (коју израчунавамо било у петљи, било библиотечком функцијом `accumulate`

```
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>
using namespace std;
```

```
int main() {
  ios_base::sync_with_stdio(false);
  // ucitavamo broj strana za svako poglavlje
  int n;
  cin >> n;
  vector<int> brojStrana(n);
  for (int i = 0; i < n; i++)
    cin >> brojStrana[i];

  // minimalna vrednost razlike
  int minRazlika = numeric_limits<int>::max();
  // redni broj poglavlja za koje je ta razlika postignuta
  int minPoglavlje = -1;

  // broj strana u prvom delu knjige
  int stranaPrviDeo = 0;
  // broj strana u drugom delu knjige
  // int stranaDrugiDeo = accumulate(a.begin(), a.end(), 0);
  int stranaDrugiDeo = 0;
  for (int i = 0; i < n; i++)
    stranaDrugiDeo += brojStrana[i];

  for (int poglavlje = 0; poglavlje < n; poglavlje++) {
    // azuriramo broj strana tako sto strane u tekucem poglavlju prebacimo
    // iz drugog u prvi deo
    stranaPrviDeo  += brojStrana[poglavlje];
    stranaDrugiDeo -= brojStrana[poglavlje];
    // razlika broja strana u dva dela knjige
    int razlika = abs(stranaDrugiDeo - stranaPrviDeo);
    // provera da li je to najmanja razlika
    if (razlika < minRazlika) {
      minRazlika = razlika;
      minPoglavlje = poglavlje;
    } else
      break;
  }

  // ispisujemo redni broj poglavlja nakon kojeg je potrebno podeliti knjigu
  cout << minPoglavlje << endl;
  return 0;
}
```

Оптимизација заснована на монотоности

Још једна оптимизација долази од тога да се може приметити да, пошто су свако поглавље има позитиван број страна, разлика између броја страна у првом и другом делу страна монотоно расте од супротне вредности укупном броју страна у целој књизи (ако је подела извршена пре почетног поглавља), па све до укупног броја страна у целој књизи (ако је подела извршена после последњег поглавља). Ако посматрамо апсолутне вредности разлике оне монотоно опадају, до

вредности коју тражимо, а онда крећу монотоно да расту. У тренутку када раст крене, он се наставља до краја, тако да се петља може прекинути први пут када се наиђе да тренутна разлика не поправља до тада минималну.

Сложеност и након ове оптимизације остаје $O(n)$. Ако се претпостави да сва поглавља имају сличан број страна, може се очекивати да ће се на овај начин уштедети око половине израчунавања и алгоритам одређивања оптималног дана ће радити отприлике дупло брже (додуше учитавање и израчунавање почетног укупног броја страна књиге се овим не убрзава).

Задатак: Сервис камиона

Такмичење: државно 2018/2019, VII и VIII разред, 1. задатак

Аутор: Станка Матковић, *Аутор решења*: Филип Марић

Дате су планиране километраже које возач једним камионом треба да пређе у наредних n дана. Елементом a_0 је дата планирана километража за први од n дана, елементом a_1 за други и тако редом. Потребно је извршити сервисирање возила тако да се укупне суме пређених километара пре сервисирања и после сервисирања најмање разликују. Сервисирање се обавља на крају радног дана. Ако постоји два дана са истом најмањом разликом сервис се обавља у ранијем дану. Написати програм којим се одређује на крају ког дана треба извршити сервис.

Опис улаза

У првој линији стандардног улаза налази се природан број n ($1 < n \leq 50000$). У следећих n линија налазе се по један природан број (између 1 и 1000), бројеви представљају планиране километраже редом за сваки дан.

Опис излаза

На стандарном излазу приказати у једној линији редни број дана после ког треба вршити сервисирање камиона.

Пример

Улаз	*Излаз*
5 100 120 50 150 70	1

Решење

Груба сила

Најдиректније решење задатка је грубом силом, где за сваки потенцијални дан d одређујемо збир километара pre_d до тог дана, укључујући и тај дан (збир елемената низа од позиције 0 закључно са позицијом d), и збир километара $posle_d$ после тог дана (збир елемената низа од позиције $d + 1$ закључно са позицијом $n - 1$) и одредимо за који дан се те две суме минимало разликују. Тражимо минималну вредност израза $|posle_d - pre_d|$, што радимо уобичајеним поступком.

Наивно решење у сваком кораку директно израчунава pre_d и $posle_d$. Ови збирови се директно могу израчунати применом алгоритма сабирање серије бројева (збир елемената постављамо на нулу и онда у петљи збир увећавамо за вредности у датом сегменту низа). Могуће је и коришћење библиотечке функције `accumulate`.

У оба случаја број корака потребан да се израчунају оба збира је линеарно пропорционалан укупном броју дана. Зато је број корака потребних за анализу свих дана пропорционалан квадрату броја дана (сложеност је $O(n^2)$, што је много неефикасније него што ћемо видети да је заиста потребно).

```
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>
using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  // ucitavamo kilometrazu
  int n;
  cin >> n;
  vector<int> km(n);
  for (int i = 0; i < n; i++)
    cin >> km[i];

  // najmanja razlika i dan u kojem je ona postignuta
  int minRazlika = numeric_limits<int>::max();
  int minDan = 0;

  for (int dan = 0; dan < n; dan++) {
    // zbir kilometara zakljucno sa danom dan
    // int pre = accumulate(begin(km), next(begin(km), dan+1), 0);
    int pre = 0;
    for (int i = 0; i <= dan; i++)
      pre += km[i];

    // zbir kilometara nakon dana dan
    // int posle = accumulate(next(begin(km), dan+1), end(km), 0);
    int posle = 0;
    for (int i = dan + 1; i < n; i++)
      posle += km[i];

    // razlika kilometara pre i posle dana dan
    int razlika = abs(posle - pre);

    // ako je potrebno, azuriramo minimum
    if (razlika < minRazlika) {
      minRazlika = razlika;
      minDan = dan;
    }
```

```
  }

  // ispisujemo rezultat
  cout << minDan << endl;

  return 0;
}
```

Инкрементално израчунавање збирова километража

Парцијалне збирове од првог до неког дана, али и парцијалне збирове од неког до последњег дана можемо израчунавати инкрементално. Збир pre_d се може веома једноставно израчунати тако што се на на већ познати збир pre_{d-1} дода број километара које је потребно прећи у дану d тј. елемент km_d (важи $pre_d = pre_{d-1} + km_d$). Слично, збир $posle_d$ се може добити ако се од познатог броја $posle_{d-1}$ одузме број километара који су прећени у дану d (важи $posle_d = posle_{d-1} - a_d$).

На основу овога можемо формулисати следећи алгоритам. Одржавамо број pre прећених километара закључно са даном d и број километара $posle$ након дана d. Њих иницијализујемо као да је сервис извршен пре првог дана, тј. прву вредност иницијализујемо на нулу, а другу на укупну суму низа (коју израчунавамо било у петљи, било библиотечком функцијом `accumulate`).

Одржавамо и минималну вредност разлике између броја километара пре и после сервиса (њу иницијализујемо на вредност $+\infty$ тј. на највећи цео број који се може записати у оквиру типа `int`) и број дана када је та разлика остварена. Затим, у петљи пролазимо кроз све дане d од 0 до $n-1$, претпостављамо да је сервис извршен на крају дана d, ажурирамо вредности километраже тако што километре пре сервиса pre увећавамо за број километара у дану d, а километре после сервиса $posle$ умањујемо за број километара у дану d, рачунамо апсолутну вредност разлике између тих километража и ако је она мања од до тада најмање разлике, ажурирамо минимум и нотирамо да је најмања разлика постигнута баш у том дану.

Учитавање података и израчунавање збира свих километража је сложености $O(n)$. Инкременталношћу се поступак значајно убрзава јер је у сваком кораку потребно учинити само константан број операција (који не зависи од дужине низа n) тако да укупан број операција уместо квадратног $O(n^2)$ постаје линеаран $O(n)$.

```
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>
using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  // ucitavamo kilometraze
  int n;
  cin >> n;
  vector<int> km(n);
  for (int i = 0; i < n; i++)
    cin >> km[i];
```

```
  // minimalna vrednost razlike
  int minRazlika = numeric_limits<int>::max();
  // redni broj dana u kojem je ta razlika postignuta
  int minDan = 0;

  // zbir kilometara pre servisa ako bi se servris vrsio pre pocetnog dana
  int zbirPre = 0;
  // zbir kilometara posle servisa ako bi se servis vrsio pre pocetnog dana
  // int zbirPosle = accumulate(a.begin(), a.end(), 0);
  int zbirPosle = 0;
  for (int i = 0; i < n; i++)
    zbirPosle += km[i];

  for (int dan = 0; dan < n; dan++) {
    // azuriramo kilometraze tako sto kilometrazu u danu dan prebacimo
    // iz zbira posle servisa, u zbir pre servisa
    zbirPre   += km[dan];
    zbirPosle -= km[dan];
    // razlika kilometraza pre i posle dana dan
    int razlika = abs(zbirPosle - zbirPre);
    // provera da li je to najmanja razlika
    if (razlika < minRazlika) {
      minRazlika = razlika;
      minDan = dan;
    } else
      break;
  }

  // ispisujemo redni broj dana u kojem treba uraditi servis
  cout << minDan << endl;
  return 0;
}
```

Оптимизација заснована на монотоности

Још једна оптимизација долази од тога да се може приметити да, пошто су све дневне километраже позитивне, разлика између километража пре и после сервиса монотоно расте од супротне вредности укупној километражи у свим данима (ако је сервис извршен пре почетног дана), па све до укупне километраже у свим данима (ако је сервис извршен на крају последњег дана). Ако посматрамо апсолутне вредности разлике оне монотоно опадају, до вредности коју тражимо, а онда крећу монотоно да расту. У тренутку када раст крене, он се наставља до краја, тако да се петља може прекинути први пут када се наиђе да тренутна разлика не поправља до тада минималну.

Сложеност и након ове оптимизације остаје $O(n)$. Ако се претпостави да су километри који се прелазе сваког дана равномерно распоређени, може се очекивати да ће се на овај начин уштедети око половине израчунавања и алгоритам одређивања оптималног дана ће радити отприлике дупло брже (додуше учитавање и израчунавање почетног збира километража се овим не убрзава).

Задатак: Рутер

Аутор: Филип Марић

Такмичење: 2019/20 квалификације 2, V и VI разред, 7. задатак

Дуж једне улице су равномерно распоређене зграде (растојање између сваке две суседне је једнако). За сваку зграду је познат број корисника које нови добављач интернета треба да повеже. Одредити у коју од зграда треба поставити рутер тако да би укупна дужина оптичких каблова којим се сваки од корисника повезује са рутером била минимална (рачунати само дужину каблова од зграде до зграде и занемарити дужине унутар зграда).

Опис улаза

У првом реду стандардног улаза налази се број n ($0 \leq n \leq 50000$), а у наредном n природних бројева раздвојених размацима који представљају број корисника у свакој од n зграда.

Опис излаза

На стандардни излаз исписати минималну дужину каблова.

Пример

Улаз	*Излаз*	*Објашњење*
`6` `3 5 1 6 2 4`	`30`	Рутер треба поставити у четврту зграду слева и дужина каблова је тада једнака $3 \cdot 3 + 2 \cdot 5 + 1 \cdot 1 + 1 \cdot 2 + 2 \cdot 4 = 30$.

Решење

Наивно решење би подразумевало да се израчуна дужина каблова за сваку могућу позицију рутера и да се одабере најмањи. Да бисмо израчунали дужину каблова, ако је рутер у згради на позицији k, рачунамо заправо збир

$$\sum_{i=0}^{n-1} |k-i| \cdot a_i,$$

где је a_i број корисника у згради i. Тај тежински збир можемо израчунати у времену $O(n)$, па пошто се испитује n позиција, алгоритам би био сложености $O(n^2)$.

Много боље решење и линеарни алгоритам можемо добити ако применимо принцип инкременталности и избегнемо рачунање у сваком кораку из почетка. Размотримо како се дужина каблова мења када се рутер помера са зграде k на зграду $k+1$.

Дужину каблова за рутер у згради $k+1$ добијамо од дужине каблова за рутер у згради k тако што ту дужину увећамо за укупан број станара закључно са зградом k и умањимо је за укупан број станара почевши од зграде $k+1$. То је заправо интуитивно прилично јасно и без компликованог математичког извођења. Померањем рутера за дужину једне зграде надесно, сваком станару који живи закључно до зграде k дужина кабла се повећала за једно растојање између зграда, а свим станарима од зграде $k+1$ надесно се та дужина смањује за једно растојање између зграда.

Укупне бројеве станара пре и после дате зграде можемо такође рачунати инкрементално (при преласку на наредну зграду, први број се увећава, а други умањује за број станара текуће зграде).

Дакле, у програму можемо да памтимо три ствари: дужину каблова d_k ако је рутер на позицији k, укупан број станара pre_k пре зграде k и укупан број станара $posle_k$ од зграде k до краја. На

почетку, када је $k = 0$, први број d_0 морамо експлицитно израчунати као $\sum_{i=1}^{n-1} i \cdot a_i$ (за то нам је потребно време $O(n)$), други број треба иницијализовати на нулу $pre_0 = 0$, а трећи на укупан број свих станара $posle_k = \sum_{i=0}^{n-1} a_i$ (и за то нам је потребно време $O(n)$). Затим за свако k од 1 до $n-1$ рачунамо $pre_k = pre_{k-1} + a_{k-1}$, $posle_k = posle_{k-1} + a_{k-1}$ и $d_k = d_{k-1} + pre_k + posle_k$.

Интересантно, укупну почетну дужину каблова можемо израчунати у линеарној сложености и без множења, тако што на збир додамо број станара у последњој згради, затим број станара у последње две зграде, затим број станара у последње три зграде и тако даље, све док не додамо број станара у свим зградама осим прве.

Пошто је и за једну и за другу фазу потребно време $O(n)$, то је уједно сложеност овог алгоритма.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> stanara(n);
  for (int i = 0; i < n; i++)
    cin >> stanara[i];

  // krećemo od zgrade 0
  // ukupna dužina kablova ako je ruter u tekućoj zgradi
  long long duzina_kablova = 0;
  for (int i = 0; i < n; i++)
    duzina_kablova += stanara[i] * i;
  // broj stanara pre tekuće zgrade
  long long stanara_pre = 0;
  // broj stanara od tekuće zgrade do kraja
  long long stanara_posle = 0;
  for (int i = 0; i < n; i++)
    stanara_posle += stanara[i];

  // minimalna duzina kablova
  long long min_duzina_kablova = duzina_kablova;

  // obrađujemo sve zgrade od 1 do n-1
  for (int k = 1; k < n; k++) {
    // ažuriramo brojeve stanara
    stanara_pre += stanara[k-1];
    stanara_posle -= stanara[k-1];
    // ažuriramo duz
    duzina_kablova += stanara_pre - stanara_posle;
    if (duzina_kablova < min_duzina_kablova)
      min_duzina_kablova = duzina_kablova;
  }
```

```
  cout << min_duzina_kablova << endl;

  return 0;
}
```

Задатак: Збирови након поделе

Аутор: Иван Дрецун

Такмичење: 2022/2023. квалификације 1, 8. разред, 6. задатак

Дат је низ целих бројева дужине n. Низ је потребно пресећи на два дела. Након сечења одређује се колико постоји парова бројева таквих да је један број из левог, а други из десног дела и да је њихов збир паран број. Одредити максималан број таквих парова који је могуће добити сечењем низа.

Опис улаза

Са стандардног улаза се уноси цео број n ($2 \leq n \leq 50000$). Затим се у наредном реду уносе елементи низа a_i ($0 \leq a_i \leq 100$) раздвојени размаком.

- У примерима вредним 50 поена важи ограничење $2 \leq n \leq 100$.

Опис излаза

На стандардни излаз исписати један цео број који представља тражену вредност.

Пример 1

Улаз	*Излаз*	*Објашњење*
`8` `1 7 3 4 6 5 8 0`	`7`	Највећи број парова добија се поделом низа на делове `1 7 3 4 4` и `5 8 0`. Парови су $(1, 5)$, $(7, 5)$, $(3, 5)$, $(4, 8)$, $(4, 0)$, $(6, 8)$ и $(6, 0)$.

Пример 2

Улаз	*Излаз*	*Објашњење*
`7` `1 9 3 11 12 4 8`	`4`	Највећи број парова добија се поделом низа на делове `1 9` и `3 11 12 4 8`. Парови су $(1, 3)$, $(1, 11)$, $(9, 3)$ и $(9, 11)$.

Решење

Директно пребројавање

За свако пресецање можемо пребројати парове директно. Притом можемо памтити максималан до сада пронађен број парова.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int n;
  cin >> n;
```

```
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  int maxParova = 0;
  for (int i = 1; i < n; i++) {
    int brojParova = 0;
    for (int l = 0; l < i; l++)
      for (int r = i; r < n; r++)
        if ((a[l] + a[r]) % 2 == 0)
          brojParova++;
    maxParova = max(maxParova, brojParova);
  }

  cout << maxParova << '\n';
  return 0;
}
```

Примена комбинаторике

Уочимо да ако је збир бројева $a + b$ паран, тада су бројеви a и b исте парности. Сада, уместо директног пребројавања парова можемо пребројати колико има парних, односно непарних бројева са леве, односно десне стране пресека. Нека бројеви p_l, p_d, n_l, n_d редом означавају број парних бројева са леве стране, парних бројева са десне стране, непарних бројева са леве стране и непарних бројева са десне стране. Укупан број тражених парова је онда $p_l \cdot p_d + n_l \cdot n_d$.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int n;
  cin >> n;

  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];


  int maxParova = 0;
  for (int i = 1; i < n; i++) {
    int plevo = 0, pdesno = 0, nlevo = 0, ndesno = 0;
    for (int j = 0; j < i; j++)
      if (a[j] % 2 == 0)
        plevo++;
      else
        nlevo++;
    for (int j = i; j < n; j++)
      if (a[j] % 2 == 0)
```

```
      pdesno++;
    else
      ndesno++;
  maxParova = max(maxParova, plevo * pdesno + nlevo * ndesno);
 }

 cout << maxParova << '\n';
 return 0;
}
```

Инкрементално рачунање бројева парних и непарних

Вредности p_l, p_d, n_l, n_d не морамо сваки пут рачунати изнова када померимо позицију пресека низа за једно место. Уместо тога, те бројеве можемо директно ажурирати приликом померања - уколико је број који померањем пресека прелази из десног дела у леви део паран, тада можемо смањити p_d и повећати p_l за један. Уколико је тај број непаран, слично можемо смањити n_d и повећати n_l за један.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
 int n;
 cin >> n;

 vector<int> a(n);
 for (int i = 0; i < n; i++)
  cin >> a[i];

 int plevo = 0, pdesno = 0, nlevo = 0, ndesno = 0;
 for (int i = 0; i < n; i++)
  if (a[i] % 2 == 0)
   pdesno++;
  else
   ndesno++;

 int maxParova = 0;
 for (int i = 0; i < n - 1; i++) {
  if (a[i] % 2 == 0) {
   plevo++;
   pdesno--;
  }
  else {
   nlevo++;
   ndesno--;
  }
  maxParova = max(maxParova, plevo * pdesno + nlevo * ndesno);
 }
```

```
  cout << maxParova << '\n';
  return 0;
}
```

Задатак: Експеримент

Аутор: Иван Дрецун

Такмичење: 2021/22. квалификације 2, VIII разред, 5. задатак

Професор Прока је коначно направио времеплов! Сад је преостало само да га активира и отпутује t година у будућност. Времеплов се састоји од n гомила истоветних металних куглица поређаних у низ, сваке две суседне гомиле на размаку по 1 метар. За покретање времеплова потребно је поставити по један магнет у тачно две гомиле у том низу. Свака куглица која се налази између та два магнета ће одлетети до ближег и за њега се залепити. Времеплов ће професора одвести онолико година у будућност колики је укупан број метара који су све куглице прешле у збиру. Професор жели да изврши експеримент како би одредио где је потребно поставити магнете тако да отпутује што више година у будућност, али никако више од t година. Напиши програм који помаже професору да одреди где је потребно поставити магнете.

Опис улаза

Са стандардног улаза се уносе цели бројеви t ($1 \leq t \leq 10^{18}$) и n ($2 \leq n \leq 10^5$). Након тога се уноси n вредности s_i ($1 \leq s_i \leq 10^6$) које представљају број куглица у гомили i.

Опис излаза

У првом реду стандардног улаза исписати цео број који представља колико највише година у будућност професор може да отпутује. У наредном реду исписати две вредности i и j ($0 \leq i < j < n$) одвојене размаком, које представљају редне бројеве гомила где је потребно поставити магнете. Уколико постоји више могућих решења, исписати оно где је i најмање.

Пример

Улаз	*Излаз*	*Објашњење*
`16 6` `2 1 3 4 3 2`	`1 5`	Укупна удаљеност коју куглице прелазе је $0 \times 1 + 1 \times 3 + 2 \times 4 + 1 \times 3 + 0 \times 2$.

Решење

Означимо са $R_{x,y}$ укупно растојање које пређу куглице када су магнети на позицијама x и y.

Груба сила

Решење грубом силом подразумева да се испробају све могуће позиције магнета (оне се једноставно могу набројати унгежђеним петљама). За сваки пар позиција $0 \leq i < j < n$ изнова израчунавамо $R_{i,j}$ и ако је оно мање од задате границе, ажурирамо максимум ако је то потребно.

Сложеност овог решења је $O(n^3)$ и оно је недовољно ефикасно.

```
#include <iostream>
#include <vector>
```

```
using namespace std;

int main() {
  int64_t t;
  cin >> t;

  int n;
  cin >> n;

  vector<int64_t> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  int maxi = 0, maxj = 1;
  int64_t maxZbir = 0;
  for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
      int64_t zbir = 0;
      for (int k = i; k <= j; k++)
        zbir += a[k] * min(k - i, j - k);
      if (zbir <= t && zbir > maxZbir) {
        maxZbir = zbir;
        maxi = i;
        maxj = j;
      }
    }
  }

  cout << maxZbir << '\n';
  cout << maxi << ' ' << maxj << '\n';
  return 0;
}
```

Инкременталност

Боље решење се добија ако се приликом померања магнета збир не израчунава сваки пут изнова, већ се добије *инкрементално*, на основу претходне вредности. Заиста, ако знамо $R_{i,j}$ тада $R_{i,j+1}$ не морамо израчунати из почетка већ ажурирањем познате вредности $R_{i,j}$. То значи да за фиксирану позицију левог магнета i све вредности $R_{i,j}$ можемо рачунати инкрементално.

На пример, претпоставимо да је дат низ `1 5 2 2 4 3 2` и да смо израчунали збир пређених растојања куглица за сегмент низа `5 2 2 4`. Вредност тог збира је $s = 0 \cdot 5 + 1 \cdot 2 + 1 \cdot 2 + 0 \cdot 4$. Посматрајмо сада збир на сегменту `5 2 2 4 3` који добијамо померањем десне границе за једно место удесно. Вредност тог збира је $s' = 0 \cdot 5 + 1 \cdot 2 + 2 \cdot 2 + 1 \cdot 4 + 0 \cdot 3$. Разлика тих збирова је $s' - s = (0-0) \cdot 5 + (1-1) \cdot 2 + (2-1) \cdot 2 + (1-0) \cdot 4 + 0 \cdot 3 = 2 + 4$, што значи да вредност s' можемо да добијемо додавањем збира елемената десне половине сегмента `5 2 2 4`.

Овако нешто важи и у општем случају. Претпоставимо, да је познат $R_{i,j} = 0 \cdot a_i + 1 \cdot a_{i+1} + 2 \cdot a_{i+2} + ... + 2 \cdot a_{j-2} + 1 \cdot a_{j-1} + 0 \cdot a_j$. Када се магнет са позиције j помери на позицију $j+1$, тада је укупан пут једнак $R_{i,j+1} = 0 \cdot a_i + 1 \cdot a_{i+1} + 2 \cdot a_{i+2} + ... + 3 \cdot a_{j-2} + 2 \cdot a_{j-1} + 1 \cdot a_j + 0 \cdot a_{j+1}$

тј. увећава се у односу на претходни за $D_{i,j} = a_s + ... + a_{j-2} + a_{j-1} + a_j$, где је s прва позиција у десној половини дела низа између позиција i и j тј. $s = \lfloor \frac{i+j}{2} \rfloor + 1$ (ако тај део низа садржи непарни број елемената, лева половина је за један елемент дужа од десне).

Током рада алгоритма одржаваћемо у посебној променљивој збир $D_{i,j-1} = a_p + ... + a_{j-2} + a_{j-1}$, за $p = \lfloor \frac{i+j-1}{2} \rfloor + 1 = \lceil \frac{i+j}{2} \rceil$. За брзо одређивање вредности $D_{i,j-1}$ могуће је користити *низ збирова префикса*, али за тим нема потребе пошто је и тај збир могуће збир рачунати инкрементално, тј. на основу $D_{i,j-1}$ пре увећавања j можемо лако израчунати $D_{i,j}$ и затим $R_{i,j}$ увећати за ту ажурирану вредност да бисмо добили $R_{i,j+1}$. Непосредно пре повећања j познати збир $D_{i,j-1}$ увећавамо за a_j, проверавамо да ли је $i - j$ парно и ако јесте, тада ту увећану вредност $D[i, j-1]$ додатно умањујемо за $a_p = a_{\frac{i+j}{2}}$.

Нека је низ $2, 1, 3, 4, 3, 2$. Тада се збирови $R(0, j)$ инкрементално израчунавају на следећи начин.

i	j	[i, j]	p	[p, j)	D[i,j-1]	R[i,j]	s	[s, j]	D[i, j]
0	1	[2, 1]	1	[]	0	0	1	[1]	1
0	2	[2, 1, 3]	1	[1]	1=0+1	1=0+1	2	[3]	3
0	3	[2, 1, 3, 4]	2	[3]	3=1+3-1	4=1+3	2	[3, 4]	7
0	4	[2, 1, 3, 4, 3]	2	[3, 4]	7=3+4	11=4+7	3	[4, 3]	7
0	5	[2, 1, 3, 4, 3, 2]	3	[4, 3]	7=7+3-3	18=11+7	3	[4, 3, 2]	9

На основу претходне анализе је једноставно написати програмски код.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int64_t t;
  cin >> t;

  int n;
  cin >> n;

  vector<int64_t> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  int maxi = 0, maxj = 1;
  int64_t maxZbir = 0;
  for (int i = 0; i < n; i++) {
    int64_t zbir = 0, zbirDesno = 0;
    for (int j = i+1; j < n; j++) {
      if (zbir <= t && zbir > maxZbir) {
        maxZbir = zbir;
        maxi = i;
        maxj = j;
      }
      zbirDesno += a[j];
```

```
      if ((j - i) % 2 == 0)
        zbirDesno -= a[(i + j) / 2];
      zbir += zbirDesno;
    }
  }

  cout << maxZbir << '\n';
  cout << maxi << ' ' << maxj << '\n';
  return 0;
}
```

Техника два показивача

Задатак се ефикасније може решавати техником два показивача, слично задатку у коме се тражи проналажење сегмента узастопних елемената датог низа природних бројева чији је збир елемената једнак датом броју (тај задатак је често детаљно описан у литератури).

Означимо са $R(x, y)$ укупно растојање које пређу куглице када су магнети на позицијама x и y. Нека је један сегмент низа одређен позицијама a и b и неки је његов његов подсегмент одређен позицијама c и d тј. нека важи $a \leq c \leq d \leq b$. Пошто су сви бројеви у низу ненегативни, важи да је $R(c, d) \leq R(a, b)$. Због овога, ако је збир пређених растојања куглица на неком сегменту мањи или једнак t, нема потребе разматрати његове подсегменте зато што тражимо највећи такав збир. Аналогно, ако је збир на неком сегменту већи од t, нема потребе разматрати његове надсегменте зато што тражимо збир који је мањи или једнак t. Ово нам омогућава да решење оптимизујемо техником два показивача.

Дакле, пошто су сви бројеви у низу ненегативни, померањем десног магнета надесно тј. проширивањем обухваћеног сегмента се повећава укупно растојање које куглице пређу, док се померањем левог магнета надесно тј. скраћивањем обухваћеног сегмента смањује укупно растојање које куглице пређу.

Ако установимо да је $R_{i,j}$, за неко $i < j$ већи од дате границе t, тада нема потребе рачунати и проверавати $R(i, j')$ за $j < j'$, јер ће и у њима збир пређених растојања бити превелики.

Ако установимо да је $R_{i,j}$, за неко $i < j$ мањи или једнак од дате границе t, тада нема потребе рачунати и проверавати $R(i', j)$ за $i < i' < j$, јер је $R(i', j) \leq R_{i,j}$, па $R(i', j)$ не може да поправи максимум (а ако је збир једнак, тражи сегмент чији је леви крај мањи).

Зато одржавамо текући сегмент $[i, j]$ (иницијализујући га на $[0, 1]$) рачунамо збир пређених растојања куглица у том сегменту $R_{i,j}$ и ако је он мањи или једнак од дате границе померамо десни крај (ажурирајући пре тога максимум ако је то потребно), а ако је строго већи од дате границе померамо леви крај. При том одржавамо вредност максимума и позиције магнета на којима се он постиже.

Пошто се оба показивача i и j крећу у једном смеру кроз низ, број корака спољашње петље линеарно зависи од димензије низа n. У сваком кораку се збир изнова израчунава (поново у линеарној сложености), па је сложеност ове имплементације $O(n^2)$.

```
#include <iostream>
#include <vector>

using namespace std;
```

```
int main() {
  int64_t t;
  cin >> t;

  int n;
  cin >> n;

  vector<int64_t> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  // maksimalni zbir rastojanja i pozicije magneta na kojima se
  // postize taj maksimalni zbir
  int64_t maksZbir = 0;
  int maxi = i, maxj = j;

  // pozicije levog i desnog magneta - pomeraju se kuglice u intervalu [i, j]
  int i = 0, j = 1;
  while (j < n) {
    // izracunavamo ukupno predjeno rastojanje kada su magneti na
    // pozicijama [i, j]
    int64_t zbir = 0;
    for (int k = i; k <= j; k++)
      zbir += a[k] * min(k - i, j - k);

    if (zbir <= t) {
      // tekuci zbir je u granicama, pa azuriramo maksimum ako je to potrebno
      if (zbir > maksZbir) {
        maksZbir = zbir;
        maxi = i;
        maxj = j;
      }
      // zbir nije veci od ogranicenja, pa pomeramo desni kraj i
      // prosirujemo interval [i, j]
      j++;
    }
    else
      // zbir je veci od ogranicenja, pa pomeramo levi kraj i
      // skracujemo interval [i, j]
      i++;
  }

  cout << maksZbir << '\n';
  cout << maxi << ' ' << maxj << '\n';
  return 0;
}
```

Техника два показивача и инкременталност

Најефикасније решење се добија ако искомбинујемо технику два показивача и инкрементално рачунање збира. Већ је описано како се на основу познате вредности $R_{i,j}$ ефикасно израчунава вредност $R_{i,j+1}$. На сличан начин се може израчунати и вредност $R_{i+1,j}$ која нам је потребна када се леви магнет помера за једно место удесно.

Претпоставимо, да је познат $R_{i,j} = 0 \cdot a_i + 1 \cdot a_{i+1} + 2 \cdot a_{i+2} + ... + 2 \cdot a_{j-2} + 1 \cdot a_{j-1} + 0 \cdot a_j$. Када се магнет са позиције i помери на позицију $i+1$ тада је укупан пут једнак $R_{i+1,j} = 0 \cdot a_{i+1} + 1 \cdot a_{i+2} + 2 \cdot a_{i+3} ... + 2 \cdot a_{j-2} + 1 \cdot a_{j-1} + 0 \cdot a_j$ тј. умањује се у односу на претходни за $L_{i,j} = a_{i+1} + a_{i+2} + ... + a_s$, где је s последњи елемент леве половине дела низа између i и j тј. $s = \lfloor \frac{i+j}{2} \rfloor$ (ако тај део низа има непаран број елемената, лева половина је за један елемент дужа). Стога током рада алгоритма у посебној променљивој одржавамо додатно и збир елемената леве половине $L_{i,j} = a_{i+1} + a_{i+2} + ... + a_s$. И њега можемо ажурирати инкрементално. Да бисмо израчунали $L_{i+1,j} = a_{i+2} + ... + a_{\lfloor \frac{i+j+1}{2} \rfloor}$, непосредно након повећања i тј. одређивања вредности $i' = i + 1$ познати збир $L_{i,j}$ умањујемо за вредност $a_{i'} = a_{i+1}$, проверавамо да ли је $j - i' = j - (i + 1)$ паран број и ако јесте, тада збир додатно увећавамо за вредност $a_{\frac{i'+j}{2}} = a_{\frac{i+1+j}{2}}$.

Ако се инкременталност употреби у комбинацији са грубом силом, добија се алгоритам сложености $O(n^2)$.

Ако се инкременталност употреби у комбинацији са техником два показивача, добија се алгоритам сложености $O(n)$.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int64_t t;
  cin >> t;

  int n;
  cin >> n;

  vector<int64_t> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  // pozicije levog i desnog magneta - pomeraju se kuglice u intervalu [i, j]
  int i = 0, j = 1;

  // maksimalni zbir rastojanja i pozicije magneta na kojima se
  // postize taj maksimalni zbir
  int64_t maksZbir = 0;
  int maxi = i, maxj = j;

  //  ukupno predjeno rastojanje kada su magneti na pozicijama [i, j]
  int64_t zbir = 0, zbirLevo = 0, zbirDesno = 0;
```

```
  while (j < n) {
    if (zbir <= t) {
      // tekuci zbir je u granicama, pa azuriramo maksimum ako je to potrebno
      if (zbir > maksZbir) {
        maksZbir = zbir;
        maxi = i;
        maxj = j;
      }
      // zbir nije veci od ogranicenja, pa pomeramo desni kraj i
      // prosirujemo interval [i, j], azurirajuci pri tom zbir
      zbirDesno += a[j];
      j++;
      zbir += zbirDesno;
    } else {
      // zbir je veci od ogranicenja, pa pomeramo levi kraj i
      // skracujemo interval [i, j], azurirajuci pri tom zbir
      zbir -= zbirLevo;
      i++;
      zbirLevo -= a[i];
    }
    // prebacujemo jedan element iz desne u levu polovinu
    if ((j - i) % 2 == 0) {
      zbirLevo  += a[(i + j) / 2];
      zbirDesno -= a[(i + j) / 2];
    }
  }

  cout << maksZbir << '\n';
  cout << maxi << ' ' << maxj << '\n';
  return 0;
}
```

Задатак: Најмањи број инверзија

Аутор: Иван Дрецун

Такмичење: државно 2022/2023, VII разред, 3. задатак

Пера има низ природних бројева. Мика жели у тај низ да уметне свој број док Пера не гледа. Број је сумњив уколико има много бројева са његове леве стране који су већи од њега и бројева са његове десне стране који су мањи од њега. Напиши програм који помаже Мики да уметне број тако да буде што мање сумњив.

Опис улаза

Са стандардног улаза се уносе дужина низа n ($1 \leq n \leq 10^5$) и Микин природан број x ($1 \leq x \leq 10^6$), у истом реду одвојени размаком. Затим се из наредног реда уносе (природни) бројеви a_i ($1 \leq a_i \leq 10^6$) Периног низа одвојени размацима.

Опис излаза

На стандардни излаз исписати један природан број који представља позицију Микиног броја у низу након уметања. Бројање позиција почиње од 0. Уколико постоји више таквих позиција, исписати најмању.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`5 4` `1 8 3 2 5`	`4`	`6 5` `3 4 5 6 6 2`	`2`	`4 6` `7 5 4 5`	`4`

Решење

Израчунавање сумњивости на свим позицијама

Задатак је могуће решити директним израчунавањем сумњивости броја за сваку позицију. Временска сложеност оваквог решења је $O(n^2)$.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int n, x;
  cin >> n >> x;

  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  int brojInverzija = count_if (begin(a), end(a),
                                [x](int y) { return y < x; });
  int minInverzija = brojInverzija;
  int minIndeks = 0;
  for (int i = 0; i < n; i++) {
    brojInverzija = 0;
    for (int j = 0; j < n; j++)
      if (j <= i && a[j] > x)
        brojInverzija++;
      else if (i < j && x > a[j])
        brojInverzija++;

    if (brojInverzija < minInverzija) {
      minInverzija = brojInverzija;
      minIndeks = i + 1;
    }
  }
  cout << minIndeks << '\n';

  return 0;
}
```

Инкрементално рачунање

Уместо да се сумњивост сваки пут изнова рачуна за сваку позицију, могуће је приликом преласка на нареду позицију одредити на који начин се сумњивост мења у односу на претходну. Преласком са позиције i на позицију $i+1$ сада се број $a[i]$ налази са леве стране броја x. Дакле, уколико је $a[i]$ веће од x укупна сумњивост се повећава за 1. Уколико је $a[i]$ мање од x, укупна сумњивост се смањује за 1.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int n, x;
  cin >> n >> x;

  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  int brojInverzija = count_if (begin(a), end(a),
                                [x](int y) { return y < x; });
  int minInverzija = brojInverzija;
  int minIndeks = 0;
  for (int i = 0; i < n; i++) {
    if (a[i] < x) brojInverzija--;
    if (a[i] > x) brojInverzija++;
    if (brojInverzija < minInverzija) {
      minInverzija = brojInverzija;
      minIndeks = i + 1;
    }
  }

  cout << minIndeks << '\n';
  return 0;
}
```

Задатак: Двобојна огрлица

Аутор: Милан Вуīделија

Такмичење: државно 2022/2023, VIII разред, 3. задатак

Дата је огрлица од n перли, од којих је свака плава или зелена. Када једну перлу држимо, остале перле висе са леве или десне стране, а у случају да је укупан број перли паран, једна перла је тачно испод оне коју држимо (није ни лево ни десно).

Написати програм који за учитани распоред перли, за сваку перлу одговара на следеће питање: ако ту перлу држимо, да ли је више плавих перли са њене леве или десне стране?

Опис улаза

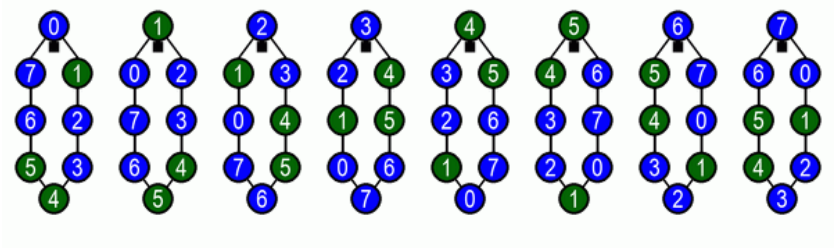
На стандардном улазу је ниска слова у једном реду, без размака. Свако од слова је или `P` или `Z`, где слова `P` означавају плаве, а слова `Z` зелене перле. Укупан број слова је мањи од 50000.

Опис излаза

На стандардни излаз исписати без размака онолико слова, колико их има на улазу. Свако од исписаних слова треба да буде `N`, `L` или `D`, при чему слово `N` значи да је број плавих перли са леве и десне стране подједнак, слово `L` значи да више плавих перли има са леве, а слово `D` да их више има са десне стране.

Пример 1

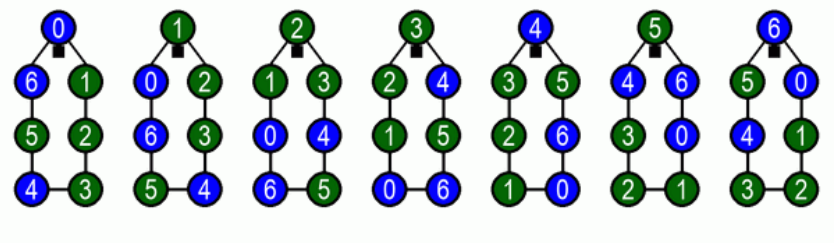
Улаз	*Излаз*	*Објашњење*
`PZPPZZPP`	`NLLLNDDD`	Огрлица има 8 перли и налазиће се у следећих 8 положаја редом.



На свакој од ових осам слика, горња перла је она коју држимо, по три перле су са леве и десне стране, а најнижа није ни са једне стране. Бројеви на перлама одговарају индексима слова из улазне ниске.
На првој слици имамо по две плаве перле са сваке стране, дакле подједнако, па је прво слово на излазу слово `N`. Да другој слици имамо три плаве перле са леве, а две са десне стране, дакле лево има више плавих па је друго слово на излазу слово `L` итд.

Пример 2

Улаз	*Излаз*	*Објашњење*
`PZZZPZP`	`LLLDDDN`	Огрлица има 7 перли и налазиће се у следећих 7 положаја редом.



У овом случају не постоји доња перла, тј. свака перла осим оне коју држимо је или са леве, или са десне стране. Из односа броја плавих перли са леве и десне стране добија се одгвоор.

Решење

Нека огрлица представљена ниском s има укупно n перли. Одредимо најпре број m перли са сваке стране горње перле, тј. перле коју држимо. За парно n важи $m = \frac{n-2}{2}$, а за непарно n је $m = \frac{n-1}{2}$. Користећи заокруживање наниже (целобројно дељење) ове две формуле можемо да објединимо у $m = \lfloor \frac{n-1}{2} \rfloor$.

Позиције перли на десној страни почињу од позиције 1, а пошто их има m, то су позиције од 1 до m. Позиције перли на левој страни се завршавају позицијом $n-1$, а пошто их има m, то су позиције од $n-m$ до $n-1$.

Када се на врху уместо перле 0 нађе перла i, десну страну ће да чине позиције од $1+i$ до $m+i$, а леву позиције од $n-m+i$ до $n-1+i$. Наравно, пошто је огрлица кружна, ове индексе почетка и краја леве и десне стране треба рачунати по модулу n. Један начин да избегнемо рачунање индекса по модулу n је да наредбом `s=s+s` поновимо ниску, тако да индекси од n до $2n-1$ постоје и за свако $k, 0 \leq k < n$ се на позицији $n+k$ у перли налази исто слово као и на позицији k. Тиме смо нешто једноставности и брзине (не рачунамо остатке при дељењу са n) у наставку програма платили већим заузећем меморије.

Да бисмо довршили решавање задатка, треба још да за свако i израчунамо број слова `P` на позицијама од $1+i$ до $m+i$ (десна страна), и на позицијама од $n-m+i$ до $n-1+i$ (лева страна), тј. да за свако i утврдимо који од ова два броја је већи.

Број слова `P` на левој и десној страни можемо да рачунамо за свако i независно од претходних.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    string s;
    cin >> s;
    int n = s.size();

    // ponavljamo vektor, da bismo izbegli racunanje indeksa po modulu n
    vector<int> a(2 * n);
    for (int i = 0; i < n; i++)
        a[i] = a[n + i] = (s[i] == 'P' ? 1 : 0);

    int m = (n - 1) / 2;

    string odg(n, ' ');
    for (int i = 0; i < n; i++) {
        int plavihLevo = 0, plavihDesno = 0;
        for (int k = 1; k <= m; k++) {
            plavihDesno += a[i + k];
            plavihLevo += a[n - m + i -1 + k];
        }

        if (plavihLevo > plavihDesno)
            odg[i] = 'L';
```

```
        else if (plavihLevo < plavihDesno)
            odg[i] = 'D';
        else
            odg[i] = 'N';
    }
    cout << odg << endl;
    return 0;
}
```

Оваквим поступком добијамо програм коме је потребно око $n \cdot 2m$, тј. око $n \cdot n$ операција да се изврши. То је за $n \approx 50000$ око 2.5 милијарде операција, а толики број операција не може да се изврши у датом временском року.

Боље решење је да само за први положај огрлице (са перлом 0 на врху) израчунамо бројеве плавих перли лево и десно као у претходном програму, а да при сваком померању огрлице нове бројеве плавих перли на свакој страни израчунамо тако што ажурирамо претходне бројеве, убацујући једну нову и избацујући једну стару перлу на свакој страни.

Већ смо рекли да, када је перла i на врху, десну страну чине позиције од $1 + i$ до $m + i$. При померању огрлице и преласку на следеће i, перла са позиције $1+i$ прелази на врх и више није на десној страни, а перла $m + i + 1$ се одоздо придружује десној страни. То значи да број плавих перли на десној страни треба да се увећа за један ако је на позицији $m + i + 1$ слово `P`, као и да се смањи за један ако је на позицији $1 + i$ слово `P`.

Број плавих перли на левој страни може да се ажурира на сличан начин. На леву страну прелази са врха перла са позиције i, а са леве стране испада перла са позиције $n - m + i$. Према томе, број плавих перли на левој страни треба да се увећа за један ако је на позицији i слово `P`, као и да се смањи за један ако је на позицији $n - m + i$ слово `P`. На овај начин добијамо програм коме је број потребних операција сразмеран са n, а не са $n \cdot n$.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    string s;
    cin >> s;
    int n = s.size();

    // ponavljamo vektor, da bismo izbegli racunanje indeksa po modulu n
    vector<int> a(2*n);
    for (int i = 0; i < n; i++)
        a[i] = a[n+i] = (s[i] == 'P' ? 1 : 0);

    int m = (n - 1) / 2;
    int plavihLevo = 0, plavihDesno = 0;
    for (int i = 1; i <= m; i++) {
        plavihDesno += a[i];
        plavihLevo += a[n - i];
    }
```

```
    string odg(n, ' ');
    for (int i = 0; i < n; i++) {
        if (plavihLevo > plavihDesno)
            odg[i] = 'L';
        else if (plavihLevo < plavihDesno)
            odg[i] = 'D';
        else
            odg[i] = 'N';
        // ubacujemo novu i izbacujemo staru perlu na desnoj strani
        plavihDesno += a[m + 1 + i] - a[1 + i];
        // ubacujemo novu i izbacujemo staru perlu na levoj strani
        plavihLevo += a[i] - a[n - m + i];
    }
    cout << odg << endl;
    return 0;
}
```

Задатак: 0-1 матрица

Аутор: Душан Пођадић

Такмичење: СИО 2022/23. 1. задатак

Дата је матрица димензија $n \times m$. Сваки ред ове матрице се састоји искључиво од нула и јединица тако да се у њему прво налазе нуле, па затим јединице. У сваком реду матрице се налази више или једнако нула у односу на претходни ред. Потребно је одговорити на q упита у којима се тражи да се одреди ред матрице (тј. индекс тог реда) у ком је број нула једнак унетом броју k_i. Ако постоји више таквих редова, исписати онај са мањим индексом. У случају да не постоји такав ред, исписати за колико најмање треба променити k_i (повећати или смањити) тако да би такав ред постојао - тражи се апсолутна вредност корекције, не треба исписивати знак.

Због величине матрице она неће бити учитана директно већ ће се за сваку колону матрице учитати број јединица у њој (приметите да због услова о форми матрице, ови подаци потпуно одређују матрицу).

Опис улаза

У првом реду стандардног улаза се налазе три броја n ($1 \leq n \leq 50000$), m ($1 \leq m \leq 50000$) и q ($1 \leq q \leq 50000$) међусобно раздвојена размаком. У наредном реду се налази m бројева одвојених размаком који редом означавају број јединица у колонама матрице. У наредних q редова се налази по један број k_i ($0 \leq k_i \leq 100000$).

Опис излаза

У q редова стандардног излаза исписати по један број: индекс траженог реда матрице или корекцију коју треба направити за k_i.

Ограничења

- У једној групи тест примера, вредној 30 поена важи $n, m, q \leq 200$.
- У другој групи тест примера, вредној 40 поена важи $n, m, q \leq 5000$.

- У трећој групи тест примера, вредној 30 поена нема додатних ограничења.

Пример 1

Улаз	*Излаз*	*Објашњење*
`5 7 3`	`3`	Унета матрица када се "развије" изгледа овако:
`1 1 2 3 3 4 5`	`1`	`1 1 1 1 1 1 1`
`5`	`0`	`0 0 1 1 1 1 1`
`4`		`0 0 0 1 1 1 1`
`0`		`0 0 0 0 0 1 1`
		`0 0 0 0 0 0 1`
		У реду са индексом 3 се налази тачно 5 нула, па је 3 одговор на први упит.
		У следећем упиту се пита у ком реду има 4 нуле, а пошто такав ред не постоји треба исписати тражену корекцију. Ред 2 има 3 нуле, а ред 3 има 5 нула, дакле k_i треба кориговати за 1 (небитно да ли повећавамо или смањујемо)
		У трећем упиту се тражи ред са 0 нула, што је 0. ред.

Пример 2

Улаз	*Излаз*	*Објашњење*
`3 9 3`	`1`	Када се матрица "развије" изгледа овако:
`0 0 1 1 1 3 3 3 3`	`7`	`0 0 1 1 1 1 1 1 1`
`3`	`1`	`0 0 0 0 0 1 1 1 1`
`12`		`0 0 0 0 0 1 1 1 1`
`5`		У 0. реду има 2 нуле, а у 1. има 5 нула, дакле у 0. реду је број нула најприближнији броју 3, па је корекција 1.
		Не постоји ред са 12 нула, а најближи ред има 5 нула, дакле корекција је 7.
		И у реду 1 и у реду 2 има по 5 нула што је тражени број, али исписујемо мањи индекс, тј. 1.

Решење

Наивно решење

Најједноставније решење овог задатка је да се на основу улаза направи матрица, а затим да се за сваки упит пролази кроз све редове, пребројава колико има нула у сваком реду и исписује ред у коме има тражени број нула или тражена корекција уколико такав ред не постоји.

Могуће је извршити благу оптимизацију при одређивању траженог реда матрице тако што се примети да број нула по колонама стално расте. Одатле следи да ће разлика између траженог броја нула и броја нула у редовима прво да опада (број нула се повећава са сваким редом па је све ближи траженом броју), а затим да расте (када број нула у неком реду премаши тражени број). Наравно, треба водити рачуна о специјалним случајевима - када је тражени број нула мањи од броја нула у првом реду или већи од броја нула у последњем реду.

У овом решењу правимо матрицу што је сложености $O(n \cdot m)$, а затим за свако q пролазимо кроз целу матрицу што је сложености $O(q \cdot m \cdot n)$, што је и укупна сложеност решења јер је $O(m \cdot n + q \cdot m \cdot n) = O(m \cdot n \cdot (q + 1)) = O(m \cdot n \cdot q)$.

Ово решење доноси 30 поена.

```cpp
#include <iostream>
#include <vector>

using namespace std;

int mat[5000][5000];

int brojNula(int i, int m) {
  int brn = 0;
  for (int j = 0; j < m; j++) {
    if (mat[i][j] == 0) brn++;
    else return brn;
  }
  return brn;
}

int main() {

  ios::sync_with_stdio(false);
  cin.tie(0);
  int n, m, q;
  cin >> n >> m >> q;

  vector <int> kolone(m);

  for (int j = 0; j < m; j++) {
    cin >> kolone[j];
  }

  // pravljenje matrice
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      if (kolone[j] <= i)
        mat[i][j] = 0;
      else
        mat[i][j] = 1;
    }
  }

  for (int i = 0; i < q; i++) {
    int k;
    cin >> k;

    int minRazlika = 150000;
    int r = 0;
    int brnR = -1;
    for (int j = 0; j < n; j++) {
      int brn = brojNula(j, m); // broj nula u j-tom redu
      int ar = abs(brn - k); // razlikua izmedju trazenog i izbrojanog broja nula
```

```
      if (ar < minRazlika) { // odredjivanje minimalne razlike
        minRazlika = ar;
        r = j;
        brnR = brn;
      } else if (ar > minRazlika) { // posto je broj nula po redovima rastuci razlika se
        break; // razlika se prvo smanjuje pa raste. Cim pocne da raste, prekidamo
      }
    }
    if (brnR == k) {
      cout << r << endl;
    } else {
      cout << abs(brnR - k) << endl;
    }
  }

  return 0;
}
```

Конструкција матрице и рачунање броја нула по колонама

Прво побољшање које ћемо посматрати је елиминисање кубне сложености из другог дела претходног решења. Поново ћемо направити матрицу као и у наивном решењу, али уместо да за сваки упит пролазимо кроз целу матрицу и бројимо нуле по редовима, можемо да унапред израчунамо и запамтимо број нула у сваком реду.

Први део решења (конструкција матрице) нисмо променили, док смо други побољшали тако што уместо обиласка целе матрице за сваки упит, пролазимо кроз низ у коме смо запамтили колико нула има у сваком реду. Овим смо сложеност свели на $O(n \cdot m + q \cdot n)$.

Ово решење доноси 70 поена.

```
#include <iostream>
#include <vector>

using namespace std;

int mat[5000][5000];

int main() {

  ios::sync_with_stdio(false);
  cin.tie(0);
  int n, m, q;
  cin >> n >> m >> q;

  vector <int> kolone(m);
  vector <int> brojNula(n);

  for (int j = 0; j < m; j++) {
    cin >> kolone[j];
```

```
  }

  // pravljenje matrice i brojanje nula
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      if (kolone[j] <= i) {
        mat[i][j] = 0;
        brojNula[i]++;
      } else
        mat[i][j] = 1;
    }
  }

  for (int i = 0; i < q; i++) {
    int k;
    cin >> k;

    int minRazlika = 150000;
    int r = 0;
    int brnR = -1;
    for (int j = 0; j < n; j++) {
      int brn = brojNula[j]; // broj nula u j-tom redu
      int ar = abs(brn - k); // razlikua izmedju trazenog i izbrojanog broja nula

      if (ar < minRazlika) { // odredjivanje minimalne razlike
        minRazlika = ar;
        r = j;
        brnR = brn;
      } else if (ar > minRazlika) { // posto je broj nula po redovima rastuci razlika se
        break; // razlika se prvo smanjuje pa raste. Cim pocne da raste, prekidamo
      }
    }
    if (brnR == k) {
      cout << r << endl;
    } else {
      cout << abs(brnR - k) << endl;
    }
  }

  return 0;
}
```

Приказано решење се може директно оптимизовати тако што се за сваки упит уместо линеарне користи бинарна претрага по низу $brojNula$. Међутим, пошто конструишемо матрицу ово побољшање нам само по себи не доноси већи број поена јер је сложеност $O(n \cdot m + q \cdot \log(n))$, што је и даље квадратно.

Директно одређивање броја нула по редовима и бинарна претрага

Да би се освојио максималан број поена очигледно је да не можемо да приуштимо да конструишемо матрицу јер та конструкција превише траје. Потребно је приметити да се низ $brojNula$ који је коришћен у претходном решењу заправо може директно добити на основу низа који учитавамо (број јединица по колонама) и да не морамо да конструишемо целу матрицу.

Треба приметити да у нашој матрици број јединица по колонама мора да буде растући низ, тј. улаз је сигурно растући низ. Даље, приметимо да ако у j-тој колони имамо x јединица, то значи да првих x редова (са индексима од 0 до $x-1$) имају највише j нула. Дакле, када учитамо прво x пролазимо кроз првих x редова и постављамо број нула у њима на 0. На пример, ако у нултој колони имамо 5 јединица, то значи да првих 5 редова нема ниједну нулу. Даље, учитавамо следеће x и попуњавамо број нула у свим редовима са индексима до $x-1$, а који нису већ попуњени са 1. Насатвљамо овај процес тако што увек попуњавамо број нула у свим редовима са идексом до $x_j - 1$ са j. Да би се боље разумео овај процес, препоручујемо да покушате да на овај начин попуните низ $brojNula$ ручно у примерима из текста задатка.

Када смо решили проблем квадратног учитавања матрице сада можемо за сваки упит применити бинарну претрагу по низу $brojNula$ као што је раније објашњено.

Први део решења (одређивање низа $brojNula$) је сада $O(n+m)$ јер имамо један пролаз по низу који се учитава и један пролаз по низу $brojNula$. Други део решења је сложености $O(q \cdot \log(n))$, па је укупна сложеност овог решења $O(n + m + q \cdot \log(n))$.

Ово решење доноси 100 поена.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int analiza(vector < int > & brojNula, int r, int k, int n) {

  if (r == n) { // trazeni broj je veci od svih u nizu
    return k - brojNula[n - 1];
  }
  if (brojNula[r] == k) { // nadjen je broj k u nizu
    return r;
  }
  if (r == 0) { // trazeni broj je manji od svih brojeva u nizu
    return brojNula[0] - k;
  }
  if (k - brojNula[r - 1] <= brojNula[r] - k) { // gornji red je blizi po broju nula
    return k - brojNula[r - 1];
  }
  return brojNula[r] - k; // donji red je blizi po broju nula
}

int main() {

  ios::sync_with_stdio(false);
```

```
  cin.tie(0);
  int n, m, q;
  cin >> n >> m >> q;

  vector <int> brojNula(n);

  // primetiti da se u ciklusu ispod i ne vraca na 0 svaki put,
  // dakle imamo jedan prolazak od 0 do m i jedan prolazak od 0 do na
  // pa je slozenost ovog dela koda O(m + n)
  int i = 0;
  for (int j = 0; j < m; j++) {
    int x;
    cin >> x;
    while (i < x) {
      brojNula[i] = j;
      i++;
    }
  }

  while (i < n) {
    brojNula[i++] = m;
  }

  for (int i = 0; i < q; i++) {
    int k;
    cin >> k;
    int r = lower_bound(brojNula.begin(), brojNula.end(), k) - brojNula.begin();

    cout << analiza(brojNula, r, k, n) << endl;
  }

  return 0;
}
```

Приметити да ако се користи овај начин одређивања низа $brojNula$, али се не користи бинарна претрага, решење и даље доноси 70 поена јер је други део квадратан.

Коришћење табеле решења

Још једна идеја за решавање задатка је да се унапред одреди решење за свако могуће унето k, сачува то решење у низ (табелу решења), и онда се прочита када се учита упит. Дакле, идеја је да се у табели на месту k налази решење за упит у ком се унесе k. Наравно, да би ово имало смисла, мора се користити директно одређивање низа $brojNula$.

Сложеност одређивања табеле решења је $O(m)$, а сложеност одговарања на један упит је $O(1)$ јер само читамо из табеле. Укупна сложеност је $O(n + m + q)$ јер је потребно одредити низ $brojNula$, одредити табелу решења и одговорити на сваки од упита.

Ово решење доноси 100 поена.

```
#include <iostream>
```

```cpp
#include <vector>
#include <algorithm>

using namespace std;

int main() {

  ios::sync_with_stdio(false);
  cin.tie(0);
  int n, m, q;
  cin >> n >> m >> q;

  vector <int> brojNula(n);

  // primetiti da se u ciklusu ispod i ne vraca na 0 svaki put,
  // dakle imamo jedan prolazak od 0 do m i jedan prolazak od 0 do na
  // pa je slozenost ovog dela koda O(m + n)
  int i = 0;
  for (int j = 0; j < m; j++) {
    int x;
    cin >> x;
    while (i < x) {
      brojNula[i] = j;
      i++;
    }
  }

  while (i < n) {
    brojNula[i++] = m;
  }

  vector <int> rez(m + 1); // tabela resenja (lookup table)

  // odredjivanje tabele resenja
  i = 0;
  for (int k = 0; k <= m; k++) {
    while (i < n && brojNula[i] < k) { // postavimo i tako da pokazuje na red u kom ima
      i++; // vise ili jednako nula u odnosu na k
    }

    if (i == n) // ako je k vece od broja nula u poslednjem redu
      rez[k] = k - brojNula[i - 1]; // upisati razliku u odnosu na taj broj nula
    else if (brojNula[i] == k)
      rez[k] = i; // ako je k jednak broju nula u i-tom redu
    else if (i == 0)
      rez[k] = brojNula[i] - k;
    else // ako je k izmedju broja nula u dva susedna reda
      rez[k] = min(brojNula[i] - k, k - brojNula[i - 1]); // ispisati razliku u odnosu na blizi
  }
```

```
  for (int i = 0; i < q; i++) {
    int k;
    cin >> k;
    if (k > m) // k je preveliko
      cout << k - brojNula[n - 1] << endl;
    else // iskoristi pripremljen odgovor
      cout << rez[k] << endl;
  }
  return 0;
}
```

Директно коришћење броја јединица по колонама

Могуће је директно користити унети низ (број јединица у свакој од колона) да би се на сваки упит одговорило у $O(1)$ сложености. Поново је потребно одредити низ $brojNula$ на начин који је описан раније. Када се учита неко k посматрамо три случаја:

Случај 1: Ако је $k = 0$ треба исписати разлику између броја нула у 0-том реду и k. Приметити да се лепо наместило да ако је та разлика заправо 0 и даље можемо да је испишемо је је тачан број нула баш у нултом реду.

Случај 2: Ако је $k > m$ или је број јединица у колони $k-1$ једнак n то значи да је k превелико, тј. k је веће од броја нула у последњем реду. Тада исписујемо разлику између k и броја нула у последњем реду. Приметити да услов да је број јединица у колони $k - 1$ једнак n заправо значи да је не постоји ниједна врста у којој је број нула већи до $k - 1$.

Случај 3 (нису успуњени претходни услови): Први ред који има бар k нула је заправо једнак броју јединица у колони $k - 1$ (назовимо тај ред r). Уколико је број нула у том реду једнак k, исписујемо r. Иначе, уколико је $r = 0$ то значи да је k мање од броја нула у било ком реду, па исписујемо разлику у односу на нулти ред. Иначе је k између $r - 1$-тог и r-тог реда, па исписујемо мању разлику у односу на број нула у та два реда.

Сложеност овог решења је $O(n + m + q)$ јер је потребно одредити низ $brojNula$ што знамо од раније да је сложености $O(n + m)$, а затим одговорити на сваки упит.

Ово решење доноси 100 поена.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {

  ios::sync_with_stdio(false);
  cin.tie(0);
  int n, m, q;
  cin >> n >> m >> q;

  vector <int> brojNula(n);
```

```
  vector <int> brojJedinica(m);

  // primetiti da se u ciklusu ispod i ne vraca na 0 svaki put,
  // dakle imamo jedan prolazak od 0 do m i jedan prolazak od 0 do na
  // pa je slozenost ovog dela koda O(m + n)
  int i = 0;
  for (int j = 0; j < m; j++) {
    int x;
    cin >> x;
    brojJedinica[j] = x;
    while (i < x) {
      brojNula[i] = j;
      i++;
    }
  }

  while (i < n) {
    brojNula[i++] = m;
  }


  for (int i = 0; i < q; i++) {
    int k;
    cin >> k;
    if (k == 0){
      cout << brojNula[0] - k << endl;
    }
    else if (k > m || brojJedinica[k-1] == n) // k je preveliko
      cout << k - brojNula[n - 1] << endl;
    else{
      int r = brojJedinica[k-1];
      if (brojNula[r] == k) // tacan broj nula
        cout << r << endl;
      else if (r == 0)
        cout << brojNula[0] - k << endl;
      else
        cout << min(brojNula[r] - k , k - brojNula[r-1]) << endl;
    }
  }
  return 0;
}
```

Задатак: Уравнотежени поднизови

Аутор: Владан Ковачевић

Такмичење: СИО 2022/23. 1. задатак

За низ ћемо рећи да је **уравнотежен** ако је збир његових елемената једнак његовој дужини

(броју елемената).

Дат је низ a дужине n, чији су елементи једноцифрени бројеви. Одредити колико он садржи уравнотежених сегмената (поднизова са узастопним елементима).

Опис улаза

У првом реду стандардног улаза је број n ($1 \leq n \leq 10^5$), а у другом n ненегативних једноцифрених бројева, раздвојених по једним размаком.

Опис излаза

На стандардни излаз исписати тражени број.

Пример

Улаз	*Излаз*	*Објашњење*
`5` `0 3 0 0 2`	`4`	Тражени сегменти су $[0,3,0]$, $[3,0,0]$, $[0,2]$ и $[0,3,0,0,2]$.

Решење

Груба сила

Решење грубом силом подразумева да се израчунају збирових свих сегмената и упореде се њиховом дужином. Ако се збирови рачунају инкрементално, решење ће бити сложености $O(n^2)$.

```
#include<iostream>
#include<vector>
#include<map>

using ll = long long;

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0);

  int n;
  cin >> n;

  vector<ll> v(n);
  for (int i = 0; i < n; i++)
    cin >> v[i];

  ll brojUravnotezenih = 0;
  for (int i = 0; i < n; i++) {
    ll zbir = 0;
    for (int j = i; j < n; j++) {
      zbir += v[j];
      if (zbir == j-i+1)
        brojUravnotezenih++;
    }
```

```
  }
  cout << brojUravnotezenih << endl;

  return 0;
}
```

Префиксни збирови

Да бисмо лакше конструисали ефикасно решење, проблем можемо мало преформулисати. Наиме, низ је уравнотежен ако и само ако је $a_i + ... + a_j = j - i + 1$, што је еквивалентно услову да је $(a_i - 1) + ... + (a_j - 1) = 0$. Дакле, број уравнотежених низова се може израчунати тако што се израчуна колико постоји сегмената чији је збир нула у низу који се од оригиналног добија умањењем свих елемената за 1. Ефикасно решење сада лако можемо добити ако израчунамо низ збирова префикса тог низа. Збир сваког сегмента $[a_i - 1, a_j - 1]$ се тада може добити као разлика између префиксног збира $P_i = (a_0 - 1) + ... + (a_{i-1} - 1)$ и префиксног збира $P_j = (a_0 - 1) + ... + (a_j - 1)$. За сваку позицију j треба да одредимо колико постоји позиција $i - 1 < j$ таквих да је $P_j - P_{i-1} = 0$ тј. да је $P_{i-1} = P_j$. Зато ћемо у мапи за сваки збир префикса памтити број пута који је виђен на претходним позицијама.

```
#include<iostream>
#include<vector>
#include<map>

using ll = long long;

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0);

  int n;
  cin >> n;
  // zbir prefiksa niza [a0 - 1, a1 - 1, ..., an-1 - 1]
  ll ps = 0;
  // broj pojavljivanja prefiksnog zbira
  map<ll, ll> m;
  // prvi prefiksni zbir je P0 = 0 i on se jednom javlja
  m[0] = 1;
  // broj uravnotezenih segmenata
  ll brojUravnotezenih = 0;
  for (int j = 0; j < n; j++) {
    // ucitavamo tekuci element
    int x;
    cin >> x;
    // izracunavamo novi prefiksni zbir modifikovanog niza
    ps += x - 1;
    // brojimo sve ranije prefiksne zbirove koji su mu jednaki
    brojUravnotezenih += m[ps];
    // azuriramo broj pojavljivanja tekuceg prefiksnog zbira
```

```
    m[ps]++;
  }
  cout << brojUravnotezenih << endl;

  return 0;
}
```

Задатак: Највећи поновљени елемент

Аутор: Филип Марић

Такмичење: Ревијално такмичење 2019/2020., VII и VIII разред, 2. задатак

Напиши програм који у низу бројева одређује највећи број који се појављује бар два пута у низу или констатује да такав број не постоји.

Опис улаза

Са стандардног улаза се учитава број n ($1 \leq n \leq 50000$), а затим у наредних n редова n целих бројева између 1 и 50000.

Опис излаза

На стандардни излаз исписати тражени број или текст `nema`, ако су сви елементи низа различити.

Пример 1

Улаз	*Излаз*
6	3
3	
8	
2	
2	
3	
5	

Пример 2

Улаз	*Излаз*
3	nema
1	
2	
3	

Пример 3

Улаз	*Излаз*
6	3
3	
3	
2	
2	
1	
1	

Решење

Решење грубом силом подразумева да се за сваки елемент провери да ли се понавља бар два пута у низу (нпр. тако што се преброје његова појављивања или тако што се изврши претрага за текућим елементом на свим позицијама осим на текућом) и да се пронађе максимум елемената који се појављују више пута. С обзиром на то да се анализира сваки од n елемената и да претрага (или пребројавање појављивања) у најгорем случају захтева обилазак скоро целог низа, сложеност овог алгоритма је $O(n^2)$, па је он недовољно ефикасан.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  // ucitavamo elemente u niz
  int n;
  cin >> n;
```

```
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  // najveci ponovljeni element
  int max = -1;

  // analiziramo jedan po jedan element niza
  for (int i = 0; i < n; i++) {
    // proveravamo da li se tekuci element ponavlja u nizu
    bool ponavljaSe = false;
    for (int j = 0; j < n; j++)
      if (i != j && a[i] == a[j]) {
        ponavljaSe = true;
        break;
      }
    // azuriramo maksimum ako je potrebno
    if (ponavljaSe && a[i] > max)
      max = a[i];
  }

  // prijavljujemo rezultat
  if (max != -1)
    cout << max << endl;
  else
    cout << "nema" << endl;

  return 0;
}
```

Алгоритам боље сложености се може добити ако се елементи низа претходно сортирају. Након тога могуће је елементе обрађивати са краја низа и за сваки елемент проверавати да ли је једнак претходном (ако се елемент понавља, након сортирања сва његова појављивања постају узастопна унутар низа). Пошто елемент обрађујемо у опадајућем редоследу, први елемент који се понавља уједно је и највећи такав елемент. Сложеношћу доминира сортирање, које, ако се ефикасно изврши сложености $O(n \log n)$.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  // ucitavamo element u niz
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
```

```
  // sortiramo ih
  sort(begin(a), end(a));

  // obilazimo elemente u opadajucem redosledu
  int k;
  for (k = n-1; k > 0; k--)
    // element je ponovoljen akko je jednak prethodnom
    if (a[k] == a[k-1])  {
      // prvi ponovljeni je ujedno i najveci takav
      cout << a[k] << endl;
      break;
    }

  // dosli smo do kraja i nismo nasli ponovljeni
  if (k == 0)
    cout << "nema" << endl;

  return 0;
}
```

Задатак можемо решити и употребом структура података. Ако учитане бројеве сместимо у скуп, тада ефикасно за сваки нови елемент можемо проверити да ли је поновљен или нови. Поновљене елементе можемо сместити у посебан низ и на крају максимум тог низа можемо одредити у линеарној сложености. Алтернативно, поновљене елементе бисмо могли скадиштити у сортираном скупу и тада бисмо максимум могли очитати у логаритамском времену (али по цену скупљег додавања елемената у колекцију поновљених). Ако претпоставимо да су операције над скупом логаритамске сложености, сложеност овог алгоритма је $O(n \log n)$.

```
#include <iostream>
#include <set>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int n;
  cin >> n;
  set<int> brojevi;        // skup svih ucitanih elemenata
  vector<int> ponovljeni;  // niz svih elemenata koji se ponavljaju
  for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    if (brojevi.find(x) != brojevi.end())  // element je vec bio ucitan
      ponovljeni.push_back(x);  // dodajemo ga medju ponovljene
    else                                  // element nije ranije bio ucitan
      brojevi.insert(x);        // dodajemo ga u skup svih elemenata
  }
  if (!ponovljeni.empty())
```

```
    // ispisujemo najveci ponovljeni element
    cout << *max_element(ponovljeni.begin(), ponovljeni.end()) << endl;
  else
    cout << "nema" << endl;
}
```

Задатак: Близанци

Аутор: Филип Марић

Такмичење: 2019/20 квалификације 1, V и VI разред, 7. задатак

Марија и Петар су близанци и желимо да свакоме од њих двоје купимо по једно одело као поклон за рођендан, али тако да се цене та два поклона што мање разликују (при томе није битно чији поклон ће бити скупљи).

Написати програм који учитава цене свих женских одела и свих мушких одела, а одређује и исписује најмању разлику између цена женског и мушког одела.

Опис улаза

Опис улаза: са стандардног улаза се учитава:

- у првом реду број мушких одела m ($1 \leq m \leq 50000$),
- у другом реду m целих бројева (цели бројеви између 1 и $2 \cdot 10^9$ раздвојени по једним размаком) - цене мушких одела
- у трећем реду број женских одела z ($1 \leq z \leq 50000$)
- у четвртом реду z целих бројева (цели бројеви између 1 и $2 \cdot 10^9$ раздвојени по једним размаком) - цене женских одела.

Опис излаза

На стандардни излаз исписати најмању вредност разлике цена мушког и женског одела.

Пример

Улаз	*Излаз*	*Објашњење*

`5`
`4680 2120 7940 11530 17820`
`4`
`850 13420 5770 6300`
	`1090`	Најмања разлика се постиже када се купе одела чије су цене 4680 и 5770 динара.

Решење

Наивно решење

Један могући приступ је да одредимо разлику (прецизније, апсолутну вредност разлике) у цени између сваког мушког и сваког женског одела, па од тих разлика нађемо најмању. Овакво решење ће дати тачан резултат у мањим примерима, али на већим примерима се неће извршити на време.

```
#include <iostream>
#include <vector>
#include <cmath>
```

```
#include <limits>
using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  int n1; cin >> n1;
  vector<int> a1(n1);
  for (int i = 0; i < n1; i++)
    cin >> a1[i];
  int n2; cin >> n2;
  vector<int> a2(n2);
  for (int i = 0; i < n2; i++)
    cin >> a2[i];

  int minRazlika = numeric_limits<int>::max();
  for (int i1 = 0; i1 < n1; i1++)
    for (int i2 = 0; i2 < n2; i2++)
      minRazlika = min(minRazlika, abs(a1[i1] - a2[i2]));

  cout << minRazlika << endl;

  return 0;
}
```

Упоредни пролаз кроз уређене низове

Ефикаснији приступ је да се низови цена најпре сортирају, а да се затим истовремено пролази кроз оба низа, рачунајући разлику текућих елемената и напредујући у оном низу у којем је цена тренутно мања. Успут се, наравно, по потреби ажурира најмања забележена разлика. Када се стигне до краја било којег низа, поступак је завршен и најмања забележена разлика је тада и укупно најмања.

Заиста, пошто су низови сортирани, када се упореде почетни елементи из оба низа, онај који је мањи од њих нема потребе упоређивати са осталим елементима низа коме он не припада, јер ће разлика моћи бити само већа (јер је тај низ сортиран). Тај елемент онда можемо избацити из даљег разматрања тако што ћемо у низу у ком се он налази прећи на следећи елемент. У специјалном случају када су почетни елементи оба низа једнаки, разлика је једнака нули, што је најмања могућа разлика, па нема потребе вршити даљу анализу.

На пример, нека су након сортирања вредности једнаке следећим.

```
1 14 28 33 45
8 21 22 41 56 68
```

- Прво поредимо елементе 1 и 8. Разлика је 7. Разлика између броја 1 и свих даљих бројева у доњем низу је већа од 7, па број 1 не морамо више анализирати.
- Након тога поредимо бројеве 14 и 8 и добијамо разлику 6. Разлика између броја 8 и свих бројева иза 14 је већа, па сада ни 8 не морамо више анализирати.
- Поредимо сада бројеве 14 и 21, разлика је 7, а 14 не морамо више да анализирамо.
- И разлика између 28 и 21 је 7, а број 21 не морамо више да анализирамо.

- Разлика између 28 и 22 је 6, а 22 не морамо да анализирамо даље.
- Разлика између 28 и 41 је 13, а 28 не морамо да анализирамо даље.
- Разлика између 33 и 41 је 8, а 33 не морамо да анализирамо даље.
- Разлика између 45 и 41 је 4, а 41 не морамо да анализирамо даље.
- Разлика између 45 и 56 је 11, а 45 не морамо да анализирамо даље. Пошто нема више елемената у горњем низу, поступак се завршава.

Можемо закључити да је најмања могућа разлика једнака 4 (за бројеве 41 и 45).

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <limits>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  int n1; cin >> n1;
  vector<int> a1(n1);
  for (int i = 0; i < n1; i++)
    cin >> a1[i];
  int n2; cin >> n2;
  vector<int> a2(n2);
  for (int i = 0; i < n2; i++)
    cin >> a2[i];

  sort(begin(a1), end(a1));
  sort(begin(a2), end(a2));
  int i1 = 0, i2 = 0;

  int minRazlika = numeric_limits<int>::max();
  while (i1 < n1 && i2 < n2)
    if (a1[i1] <= a2[i2]) {
      minRazlika = min(minRazlika, a2[i2] - a1[i1]);
      i1++;
    } else {
      minRazlika = min(minRazlika, a1[i1] - a2[i2]);
      i2++;
    }

  cout << minRazlika << endl;

  return 0;
}
```

Задатак: Поклони за родитеље

Аутор: Филип Марић

Такмичење: 2019/20 квалификације 1, VII и VIII разред, 7. задатак

Ненад жели да са путовања донесе поклоне својим родитељима. Пошто путује нискотарифном авио-компанијом и не жели да плаћа додатне трошкове, ограничена је маса коју може да понесе у свом ранцу. Оцу ће купити поклон у једној, а мајци у другој продавници. Написати програм који на основу познатих маса и цена свих поклона у обе продавнице помаже Ненаду да сваком од родитеља купи по један поклон, тако да поклони збирно буду што вреднији и да заједно не прелазе дато ограничење масе.

Опис улаза

са стандардног улаза се уносе следећи подаци:

- У првом реду број N_1 $(1 \leq N_1 \leq 10^5)$, број производа у првој продавници.
- У сваком од наредних N_1 редова по два цела броја, сваки између 1 и 10^9, који представљају масу и затим цену једног производа из прве продавнице.
- У следећем реду број N_2 $(1 \leq N_2 \leq 10^5)$, број производа у другој продавници.
- У сваком од наредних N_2 редова по два цела броја, сваки између 1 и 10^9, који представљају масу и затим цену једног производа из друге продавнице.
- У следећем и последњем реду цео број између 1 и $2 \cdot 10^9$, највећа дозвољена маса за пар поклона заједно.

Опис излаза

На стандардни излаз исписати највећи могући збир цена првог и другог поклона које је могуће понети.

Пример

Улаз	*Излаз*
3 2 4 3 6 4 5 3 2 3 3 2 4 5 6	9

Решење

Наивно решење

Можемо да проверимо сваки пар предмета (у коме је један предмет из једне, а други из друге продавнице). Ово решење је квадратне сложености, па може да донесе поене само за мале примере.

```
#include <iostream>
#include <vector>
#include <utility>
using namespace std;
```

```
typedef pair<int, int> Proizvod;
vector<Proizvod> ucitajProizvode() {
  int n;
  cin >> n;
  vector<pair<int, int>> proizvodi(n);
  for (int i = 0; i < n; i++) {
    int cena, tezina;
    cin >> cena >> tezina;
    proizvodi[i] = make_pair(cena, tezina);
  }
  return proizvodi;
}

int main() {
  vector<Proizvod> proizvodi1 = ucitajProizvode();
  vector<Proizvod> proizvodi2 = ucitajProizvode();
  int maksTezina;
  cin >> maksTezina;

  int maksCena = 0;
  for (const auto& p1 : proizvodi1) {
    if (p1.first >= maksTezina) continue;
    for (const auto& p2 : proizvodi2)
      if (p1.first + p2.first <= maksTezina &&
          p1.second + p2.second > maksCena)
        maksCena = p1.second + p2.second;
  }
  cout << maksCena << endl;
}
```

Сортирање и максимуми префикса

Решење које претендује на максималан број поена треба да буде ефикасније од квадратног. У ту сврху увек је корисно да подаци буду на неки начин сортирани. Питање је: како сортирати?

Ако оба низа уредимо тако да масе расту, можемо у једном низу да кренемо од почетка (од најлакшег предмета) а у другом од краја. На тај начин брзо долазимо до парова предмета који су у збиру испод лимита масе, а при томе најближи том лимиту. Ипак, ово није довољно да се задатак реши ефикасно, јер масе и цене не расту заједно, па предмет мање масе може бити вреднији. То значи да при прегледању свих парова који су кандидати за најбољи пар не бисмо избегли квадратно време.

Уређивање по цени није боље, јер бисмо тада за фиксиран предмет из једне продавнице морали међу онима из друге (који у збиру са првим) поправљју укупну вредност, да тражимо оне који се уклапају по маси и опет спадамо на квадратно време.

Оно што би нам овде помогло (након сортирања оба низа предмета по маси) је да можемо за сваки предмет P из једног низа да брзо одговоримо колико вреди највреднији предмет тог низа, чија маса није већа од масе предмета P. Такве одговоре можемо да припремимо након сортирања а пре испитивања парова, тако што фромирамо још један низ у коме ће се ти одговори

наћи. Конкретније, можемо у једном пролазу кроз низ сортиран по маси да за сваки префикс тог низа израчунамо цену највреднијег предмета у префиксу. Након тога првобитна идеја о истовременом проласку кроз један низ од почетка а кроз други од краја доводи до ефикасног решења.

Прикажимо рад овог алгоритма на једном примеру. Нека су цене и тежине мушких поклона дате у левој, а женских у десној колони, при чему смо поклоне сортирали по тежинама и нека је капацитет ранца једнак 20.

```
4   3               5   6         6
7   9               9   4         6
12  4               10  12        12
16  11              14  9         12
19  2               18  7         12
```

Покушавамо да за први мушки поклон (то је (4, 3)) пронађемо скуп поклона који се могу са њим упарити. Пошто је други низ сортиран, то ће бити неки поклони који се налазе на почетку тог низа. Крећемо од краја, елиминишемо последњи женски поклон јер је збир 4 + 18 већи од 20 и проналазимо да је последњи женски поклон који се може упарити са првим мушким онај који има масу 14. Потребно је пронаћи најскупљи женски поклон који има масу мању или једнаку 14. У томе нам помаже последња колона у којој су записани максимуми префикса. За поклон чија је маса 14 можемо очитати вредност 12, што значи да постоји неки женски поклон чија је маса мања или једнака 14 који има вредност 12 (то је поклон (10, 12)). Дакле, ако купимо први мушки поклон, тада је највећа цена коју можемо постићи једнака 3 + 12 = 15.

Прелазимо на други мушки поклон (то је (7, 9)) и одређујемо женске поклоне са којима се он може комбиновати. Веома важна напомена је то да се поклони који се нису могли укомбиновати са претходним мушким поклонима, не могу укомбиновати ни са текућим (јер је он још тежи од претходних). Дакле, довољно је само међу оним поклонима који су се могли укомбиновати са претходним поклоном наћи оне који се могу укомбиновати са текућим. Пошто је низ женских поклона сортиран по тежини, анализирамо и евентуално елиминишемо елементе са његовог краја (тј. краја оног дела низа где смо се претходно зауставили). Поклон масе 14 се не може комбиновати са поклоном масе 7 (јер им је укупна маса 21 већа од носивости ранца 20). Најтежи женски поклон који се може упарити са оним мушким масе 7 је онај чија је маса 10. Поново на основу низа максимума префикса очитавамо да је највреднији женски поклон чија маса не прелази 10 онај чија је вредност 12 (то је опет (10, 12)), па се његовим комбиновањем са мушким поклоном (7, 9) добија вредност 12 + 9 = 21, што је боље од претходне.

Прелазимо на следећи мушки поклон (то је (12, 4)), елиминишемо женске поклоне (10, 12) и (9, 4) јер се они не могу комбиновати са мушким поклоном масе 12 и закључујемо да је једини женски поклон који се може комбиновати са (12, 4) поклон (5, 6). Тиме добијамо вредност 4+6 = 10, што је лошије од претходне.

Преласком на следећи мушки поклон (то је (16, 11)), елиминишемо и женски поклон (5, 6) и закључујемо да се ни један мушки поклон који је тежак 16 или више не може укомбиновати ни са једним женским поклоном.

Дакле, оптимално упаривање је упаривање поклона (7, 9) и (10, 12).

```
#include <iostream>
#include <vector>
#include <algorithm>
```

```cpp
using namespace std;


typedef pair<int, int> Proizvod;

vector<Proizvod> ucitajProizvode() {
  int n;
  cin >> n;
  vector<pair<int, int>> proizvodi(n);
  for (int i = 0; i < n; i++) {
    int cena, tezina;
    cin >> cena >> tezina;
    proizvodi[i] = make_pair(cena, tezina);
  }
  return proizvodi;
}

int main() {
  vector<Proizvod> proizvodi1 = ucitajProizvode();
  vector<Proizvod> proizvodi2 = ucitajProizvode();
  int maksTezina;
  cin >> maksTezina;

  sort(begin(proizvodi1), end(proizvodi1));
  sort(begin(proizvodi2), end(proizvodi2));
  int n1 = proizvodi1.size(), n2 = proizvodi2.size();

  vector<int> maksCenaDo2(n2 + 1);
  maksCenaDo2[0] = 0;
  for (int i = 1; i <= n2; i++)
    maksCenaDo2[i] = max(maksCenaDo2[i-1], proizvodi2[i-1].second);

  int maksCena = 0;
  int i = 0, j = n2-1;
  while (i < n1 && proizvodi1[i].first < maksTezina) {
    while (j >= 0 && proizvodi1[i].first + proizvodi2[j].first > maksTezina)
      j--;
    if (j >= 0)
      maksCena = max(maksCena, proizvodi1[i].second + maksCenaDo2[j+1]);
    i++;
  }
  cout << maksCena << endl;

  return 0;
}
```

Сортирање, максимуми префикса и бинарна претрага

Претходна идеја се може мало и изменити. Довољно је да сортирамо само други низ и израчунамо максимуме његових префикса, као у претходном решењу. Након тога можемо за сваки предмет неуређеног првог низа да бинарном претрагом нађемо предмет највеће масе из другог низа, који се може понети заједно са првим, а онда (користећи максимуме префикса другог низа) да нађемо и вредност највреднијег предмета у другом низу, који се може понети заједно са текућим предметом из првог низа.

```
#include <iostream>
#include <utility>
#include <algorithm>
#include <vector>
using namespace std;

typedef pair<int, int> Proizvod;

vector<Proizvod> ucitajProizvode() {
  int n;
  cin >> n;
  vector<pair<int, int>> proizvodi(n);
  for (int i = 0; i < n; i++) {
    int cena, tezina;
    cin >> cena >> tezina;
    proizvodi[i] = make_pair(cena, tezina);
  }
  return proizvodi;
}

int main() {
  // ucitavamo podatke o proizvodima i maksimalnu tezinu koja moze da
  // stane u ranac
  vector<Proizvod> proizvodi1 = ucitajProizvode();
  vector<Proizvod> proizvodi2 = ucitajProizvode();
  int maksTezina;
  cin >> maksTezina;

  // sortiramo proizvode iz druge prodavnice po tezini
  sort(begin(proizvodi2), end(proizvodi2));

  // za svaku poziciju u sortiranom nizu proizvoda iz druge prodavnice
  // odredjujemo maksimalnu cenu proizvoda striktno pre te pozicije
  vector<int> maksCenaDo(proizvodi2.size() + 1);
  maksCenaDo[0] = 0;
  for (size_t i = 1; i <= proizvodi2.size(); i++)
    maksCenaDo[i] = max(maksCenaDo[i-1], proizvodi2[i-1].second);

  // maksimalni zbir cena dva proizvoda
  int maksCena = 0;
  // analiziramo jedan po jedan proizvod iz prve prodavnice
```

```
  for (const auto& p1 : proizvodi1) {
    // odredjujemo najvecu poziciju pre koje se svi proizvodi iz druge
    // pozicije mogu staviti u ranac sa tekucim proizvodom iz prve
    // prodavnice
    auto p2Granica = upper_bound(begin(proizvodi2), end(proizvodi2),
                                 maksTezina - p1.first,
                                 [](int tezina, const Proizvod& pj) {
                                   return tezina < pj.first;
                                 });
    int pj = distance(begin(proizvodi2), p2Granica);
    // ako ima bar neki takav proizvod
    if (pj > 0)
      // kombinujemo tekuci proizvod iz prve prodavnice sa najskupljim
      // proizvodom iz druge pre te pozicije (on se sigurno moze
      // iskombinovati sa tekucim proizvodom iz prve prodavnice)
      maksCena = max(maksCena, p1.second + maksCenaDo[pj]);
  }

  // ispisujemo pronadjeni zbir
  cout << maksCena << endl;

  return 0;
}
```

Задатак: Што сличнија тројка

Аутори: Филип Марић, Душан Пошадић

Такмичење: 2022/2023. квалификације 3, 7. разред, 6. задатак

Три школске програмерске екипе треба да пошаљу своје представнике на информатички турнир. Ако се зна процена рејтинга сваког програмера, напиши програм који ће помоћи да се одреде три представника што сличнија по рејтингу (да би турнир био што неизвеснији и интерсантнији). Дакле, треба одабрати по једног представника из прве, друге и треће екипе, тако да је разлика између најјачег од та три представника и најслабијег од њих што мања. Програм треба да одреди најмању могућу вредност те разлике.

Опис улаза

Са стандардног улаза се учитава број n ($1 \leq n \leq 10^5$) који представља број чланова сваке екипе, а затим се учитавају рејтинзи чланова прве екипе, затим чланова друге екипе и на крају рејтинзи чланова треће екипе (сваки рејтинг је природан не већи од 10^9).

Опис излаза

На стандардни излаз исписати најмању могућу разлику између најбољег и најлошијег члана одабране тројке.

Пример 1

Улаз	*Излаз*	*Објашњење*

`5`
`19 7 13 17 10`
`20 9 1 4 12`
`2 14 8 29 23`
	`2`	Најсличније тројке се добијају ако се из прве екипе узме 10, из друге 9, из треће 8 или ако се из прве узме 13, из друге 12, а из треће 14. У оба случаја разлика између најбољег и најлошијег члана је 2.

Пример 2

Улаз	*Излаз*

`7`
`9 18 4 3 2 17 9`
`5 17 3 1 9 20 6`
`52 44 37 17 4 52 55`
	`0`

Решење

Груба сила

Решење грубом силом подразумева да се испитају све могуће тројке ученика и одреди минимална разлика. Ово се може постићи тако што се користе три угнежђене петље помоћу којих се проверавају све могуће тројке (један члан прве, један члан друге и један члан треће екипе), а за сваку изабрану тројку се рачуна њихова разлика и чува се најмања.

Сложеност овог решења је лако израчунати јер се пролази кроз све могуће тројке којих има $n \cdot n \cdot n$, дакле сложеност алгоритма је $O(n^3)$. Пошто је ограничење $n \leq 10^5$, ово решење не доноси максималан број поена.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int trojka(int x, int y, int z) {
  return max({x, y, z}) - min({x, y, z});
}

int main() {
  int n;
  cin >> n;
  vector<int> a(n), b(n), c(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
  for (int i = 0; i < n; i++)
    cin >> b[i];
  for (int i = 0; i < n; i++)
    cin >> c[i];

  int najbliza_trojka = trojka(a[0], b[0], c[0]);
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      for (int k = 0; k < n; k++)
```

```
        najbliza_trojka = min(najbliza_trojka, trojka(a[i], b[j], c[k]));

  cout << najbliza_trojka << endl;

  return 0;
}
```

Два показивача

Ефикасније решење се добија техником "два показивача" (мада ће за решење овог задатка бити потребно три показивача), алгоритмом сличном оном који се користи за обједињавање сортираних низова (енгл. merge).

Погледајмо следећи пример: екипа A има чланове са рејтинзима 7, 9 и 12, екипа B има чланове са рејтинзима 2, 5 и 6, а екипа C са рејтинзима 1, 3 и 17. Ако проверимо тројку (9, 5, 3) која има разлику 6, јасно је да нема потребе да проверавамо тројку (12, 5, 3), јер повећавањем рејтинга представника прве екипе (који је већ имао највећи рејтинг у тројци) можемо само да повећамо тренутну разлику. Слично нема смисла да смањујемо рејтинг представника екипе C јер ће се на тај начин разлика тројке повећавати. Када смо ово приметили, лако долазимо до идеје да пре било какве провере сортирамо сва три низа. Дакле, прво сортирамо сва три низа, а затим поставимо три показивача на њихове почетне елементе. У сваком тренутку показивачи указују на нека три елемента низа. Размислимо како је могуће смањити разлику између најмањег и највећег елемента од та три, померањем показивача надесно. Јасно је да се померањем показивача који указује на највећи од та три елемента разлика може само повећати (јер је тај низ сортиран). Слично, померањем показивача који указује на средњи по величини елемент разлика ће неко време остати иста, док се у том низу не дође до елемента који је већи од највећег у тој тројци, након чега ће разлика кренути да се повећава. Дакле, комбиновањем најмањег од та три елемента са било којим од елемената остала два низа иза текућих показивача у тим низовима не може се добити разлика која је мања од текуће. Стога је за најмањи елемент одређена минимална разлика тројке у којој он учествује, тако да је тај елемент могуће елиминисати из даљег разматрања тј. померити показивач који на њега указује, чиме се добија проблем истог облика, али мање димензије. Поступак се наставља све док су сва три показивача унутар граница низа.

Сложеност овог решења је заправо сложеност сортирања низа, дакле $O(n \log(n))$. Ово решење доноси максималан број поена.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int trojka(int x, int y, int z) {
  return max({x, y, z}) - min({x, y, z});
}

int main() {
  int n;
  cin >> n;
  vector<int> a(n), b(n), c(n);
  for (int i = 0; i < n; i++)
```

```
    cin >> a[i];
  for (int i = 0; i < n; i++)
    cin >> b[i];
  for (int i = 0; i < n; i++)
    cin >> c[i];

  sort(begin(a), end(a));
  sort(begin(b), end(b));
  sort(begin(c), end(c));
  int najbliza_trojka = trojka(a[0], b[0], c[0]);
  int i = 0, j = 0, k = 0;
  while (i < n && j < n && k < n) {
    najbliza_trojka = min(najbliza_trojka, trojka(a[i], b[j], c[k]));
    if (a[i] <= b[j] && a[i] <= c[k])
      i++;
    else if (b[j] <= a[i] && b[j] <= c[k])
      j++;
    else if (c[k] <= a[i] && c[k] <= b[j])
      k++;
  }

  cout << najbliza_trojka << endl;

  return 0;
}
```

Задатак: Кртице и заставице

Аутор: Филип Марић

Такмичење: окружно 2021/22., VII разред, 4. задатак

Дуж једне ливаде укопане су заставице (за сваку заставицу је позната целобројна x-координата). Кртице су се због тога изнервирале и кренуле су у акцију и свака од њих је ископала рупу на неком месту дуж те ливаде (за сваку рупу је позната целобројна x-координата). Око сваке рупе земља постаје нестабилна и све заставице које су на растојању највише d од неке рупе се урушавају (укључујући и оне на растојању d). Напиши програм који одређује колико је заставица преостало након копања свих рупа.

Опис улаза

У првом реду стандардног улаза је број заставица m ($1 \leq m \leq 10^5$), а у другом целобројне координате свих заставица (цели бројеви између 0 и 10^9) раздвојене по једним размаком.

У трећем реду је број рупа n ($1 \leq n \leq 10^5$), а у четвртом целобројне координате свих рупа (цели бројеви између 0 и 10^9), такође раздвојене по једним размаком. На крају, у петом реду, налази се број d ($1 \leq 0 \leq 10^9$).

Опис излаза

На стандардни излаз исписати број преосталих заставица.

Пример

Улаз	*Излаз*
3 4 5 8 4 10 9 1 8 2	2

Решење

Груба сила

Решење грубом силом подразумева да се за сваку заставицу провери да ли постоји нека рупа која се налази на растојању мањем или једнаком d од те заставице.

Ако је број кртица m, а број заставица n, сложеност овог решења је $O(mn)$, што је неефикасно.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  int m;
  cin >> m;
  vector<int> zastavice(m);
  for (int i = 0; i < m; i++)
    cin >> zastavice[i];
  int n;
  cin >> n;
  vector<int> rupe(n);
  for (int i = 0; i < n; i++)
    cin >> rupe[i];
  int d;
  cin >> d;

  int preostalo = 0;
  for (int z : zastavice) {
    bool ostaje = true;
    for (int r : rupe)
      if (z-d <= r && r <= z+d) {
        ostaje = false;
        break;
      }
    if (ostaje)
      preostalo++;
  }
  cout << preostalo << endl;
  return 0;
```

```
}
```

Бинарна претрага

Задатак можемо решити тако што за сваку заставицу проверимо да ли постоји кртица која је пробила рупу на растојању које је мање или једнако d. То ефикасно можемо урадити применом бинарне претраге. Сортирамо низ позиција кртица, и за сваку заставицу на позицији z проверавамо да ли постоји кртица која је на растојању највише d од ње. То можемо урадити бинарном претрагом тако што проналазимо прву кртицу чија је позиција већа или једнака од $z - d$ и проверавамо да ли таква кртица постоји и да ли јој је позиција мања или једнака од $z + d$. Ако таква кртица не постоји, увећавамо бројач преосталих заставица.

За сортирање низа рупа потребно је време $O(n \log n)$. Након тога се за сваку од m заставица спроводи претрага низа рупа што доноси додатних $O(m \log n)$ корака. Укупна сложеност је стога $O((m + n) \log n)$.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  int m;
  cin >> m;
  vector<int> zastavice(m);
  for (int i = 0; i < m; i++)
    cin >> zastavice[i];
  int n;
  cin >> n;
  vector<int> rupe(n);
  for (int i = 0; i < n; i++)
    cin >> rupe[i];
  int d;
  cin >> d;

  sort(begin(rupe), end(rupe));
  int preostalo = 0;
  for (int z : zastavice) {
    // da li postoji rupa u intervalu [z-d, z+d]?
    auto it = lower_bound(begin(rupe), end(rupe), z-d);
    if (it == end(rupe) || *it > z + d)
      preostalo++;
  }
  cout << preostalo << endl;
  return 0;
}
```

Два показивача

Уместо бинарне претраге могуће је употребити и технику два показивача којима се пролази кроз сортиране низове рупа и заставица.

Сортирање оба низа се врши у сложености $O(m \log m + n \log n)$. Након тога се техника два показивача извршава у сложености $O(m + n)$.

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

int main(){
  ios_base::sync_with_stdio(false);
  int br_z;
  cin >> br_z;
  vector<int> zastavice(br_z);
  for (int i = 0; i < br_z; i++)
    cin >> zastavice[i];
  int br_r;
  cin >> br_r;
  vector<int> rupe(br_r);
  for (int i = 0; i < br_r; i++)
    cin >> rupe[i];
  int d;
  cin >> d;

  sort(begin(rupe), end(rupe));
  sort(begin(zastavice), end(zastavice));

  int i = 0, j = 0, sruseno = 0;
  while (i < br_z && j < br_r)
    if (zastavice[i] + d < rupe[j])
      i++;
    else if (zastavice[i] - d > rupe[j])
      j++;
    else {
      sruseno += 1;
      i++;
    }

  cout << br_z - sruseno << endl;
}
```

Задатак: Рале и Спале

Аутор: Милан Вугделија

Такмичење: државно 2021/22., VII и VIII разред, 3. задатак

Рале и Спале су веома талентовани ловачки керови, који често иду у лов на зечеве. У последње време, Рале је постао мало неозбиљан и дешава му се да залаје без разлога док се Спале прикрада зечевима. У тренутку када Рале залаје, сви зечеви и све рупе се налазе на једној правој линији. Чим чује лавеж, сваки зец потрчи ка најближој рупи истом, јединичном брзином. У сваку рупу може да стане произвољан број зечева и они могу истовремено да улазе у исту рупу. Пошто је Спале веома брз, нада се да и под овим околностима може да ухвати неког зеца, па га занима колико је времена потребно зечевима од тренутка када Рале залаје да сви стигну до рупа. Помозите Спалету да одреди ово време.

Опис улаза

У првом реду стандардног улаза налази се број рупа R, $1 \leq R \leq 50000$. У другом реду је R целих бројева, који представљају координате рупа. У трећем реду се налази број зечева Z, $1 \leq Z \leq 50000$. У четвртом реду је Z целих бројева, који представљају координате зечева. Координате рупа и зечева су цели бројеви из интервала $[-10^{18}, 10^{18}]$. Могуће је да постоји више рупа са истом координатом, више зечева са истом координатом, као и да неки зечеви и неке рупе имају исту координату.

Опис излаза

На стандардни излаз исписати један цео број, број секунди који је потребан да се сви зечеви нађу у рупама.

Пример

Улаз	*Излаз*
3 7 3 8 2 11 4	3

Решење

Имајући у виду могућу величину улазних података и временско ограничење, важно је да се избегне алгоритам који би одређивао растојање сваке рупе од сваког зеца, јер би време његовог извршавања било сразмерно производу броја рупа и броја зечева, што је сувише споро.

Задатак може да се ефикасно реши на више начина. Један начин је да се сортира низ рупа, а затим да се у њему за сваког зеца бинарном претрагом нађе најближа рупа. За то може да се искористи библиотечка функција lower_bound, која даје позицију прве рупе са истом или већом координатом од координате зеца. Ако та позиција не постоји (координате свих рупа су мање од координате зеца), користимо последњу позицију уместо ње. Најближа рупа мора да буде или на тој позицији, или на претходној (ако претходна посзиција не постоји, користимо почетну позицију уместо ње). Док за сваког зеца одређујемо растојање до најближе рупе, можемо успут да израчунамо и максимум тих растојања, а управо тај максимум је и тражени одговор.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;
```

```
int main() {
    int r, z;
    cin >> r;
    vector<long long> rupe(r);
    for (int i = 0; i < r; i++)
        cin >> rupe[i];
    sort(rupe.begin(), rupe.end());
    cin >> z;
    vector<long long> zecevi(z);
    for (int i = 0; i < z; i++)
        cin >> zecevi[i];
    sort(zecevi.begin(), zecevi.end());
    long long dMax = 0;
    for (long long zec : zecevi) {
        int i = lower_bound(rupe.begin(), rupe.end(), zec) - rupe.begin();
        long long rupa1 = (i < r) ? rupe[i] : rupe[r-1];
        long long rupa2 = (i > 0) ? rupe[i-1] : rupe[0];
        long long d = min(abs(zec-rupa1), abs(zec-rupa2));
        dMax = max(dMax, d);
    }
    cout << dMax << endl;
    return 0;
}
```

Задатак: Брана

Аутор: Милан Вугделија

Такмичење: 2019/20 квалификације 3, V и VI разред, 7. задатак

Даброви граде брану да би подигли ниво воде изнад улаза у њихова склоништа. Дабра Животу боли зуб, па он данас уместо да ради, процењује колико још има посла. Живота је прво измерио висину сваког улаза у склониште, а затим за неколико околних стабала оценио колико би она подигла ниво воде ако би била уграђена у брану.

Живота жели да за сваки могући ниво воде одреди колико би улаза у склоништа остало на сувом (те улазе би требало затрпати). Улаз је на сувом ако је његова висина строго већа од нивоа воде.

Опис улаза

Са стандардног улаза учитава се број улаза n ($0 \leq n \leq 50000$), а затим и висине улаза (цели бројеви), задати у сортираном редоследу од највишег до најнижег. Након тога се учитава број m ($1 \leq m \leq 50000$) који представља број могућих нивоа воде, а затим и m бројева који представљају нивое за које треба одговорити колико би улаза у склоништа остало на сувом, када би вода достигла тај ниво. Сваки број се налази у посебном реду.

Опис излаза

На стандардни излаз за сваки ниво воде исписати тражени број непотопљених улаза, сваки у посебном реду.

Пример 1

Улаз	*Излаз*
5	0
79	4
63	3
63	5
46	
13	
4	
85	
40	
60	
0	

Пример 2

Улаз	*Излаз*
4	3
30	0
29	1
29	0
28	4
5	
28	
30	
29	
31	
27	

Решење

У задатку је потребно ефикасно одредити број елемената сортираног низа који су већи од датог броја. Ако нађемо позицију првог елемента који је већи од датог броја, тада број таквих елемената можемо одредити тако што израчунамо разлику између укупног броја чланова низа и те позиције.

Најједноставнији начин да нађемо позицију првог елемента који је већи од датог броја је да применимо линеарну претрагу и да редом проверавамо један по један елемент све док не дођемо или до краја низа или до тражене позиције. Сложеност овакве претраге је $O(n)$, па би укупна сложеност решења била $O(m \cdot n)$, што је превише имајући у виду ограничења дата у задатку.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> visine_ulaza(n);
  for (int i = 0; i < n; i++)
    cin >> visine_ulaza[i];

  int m;
  cin >> m;
  for (int i = 0; i < m; i++) {
    int nivo_vode;
    cin >> nivo_vode;
    int broj = 0;
    for (int visina : visine_ulaza)
      if (visina > nivo_vode)
        broj++;
    cout << broj << endl;
  }
```

```
  return 0;
}
```

Позиција се ефикасно може пронаћи применом алгоритма бинарне претраге. Најједноставније је употребити библиотечку имплементацију. У језику C++ можемо употребити функцију `upper_bound` која враћа итератор који указује на тражену позицију или на крај низа (позицију непосредно иза последњег елемента) ако тражени елемент не постоји). Број елемената можемо одредити израчунавајући растојање од те позиције до краја низа (најбоље функцијом `distance`). Сложеност једне бинарне претраге је $O(\log n)$, па је укупна сложеност алгоритма $O(m \log n)$.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iterator>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false); cin.tie(0);
  int n;
  cin >> n;
  vector<int> visine_ulaza(n);
  for (int i = n-1; i >= 0; i--)
    cin >> visine_ulaza[i];

  int m;
  cin >> m;
  for (int i = 0; i < m; i++) {
    int nivo;
    cin >> nivo;
    cout << distance(upper_bound(begin(visine_ulaza), end(visine_ulaza), nivo),
                     end(visine_ulaza)) << '\n';
  }

  return 0;
}
```

Наравно, позицију првог елемента који је већи од датог можемо пронаћи и ручно имплементираном бинарном претрагом.

```
#include <iostream>
#include <vector>

using namespace std;

int prvi_veci(const vector<int>& a, int x) {
  int l = 0, d = a.size()-1;
  while (l <= d) {
    int s = l + (d - l) / 2;
    if (a[s] <= x)
      l = s + 1;
```

```
    else
      d = s - 1;
  }
  return d + 1;
}

int main() {
  ios_base::sync_with_stdio(false); cin.tie(0);
  int n;
  cin >> n;
  vector<int> visine_ulaza(n);
  for (int i = n-1; i >= 0; i--)
    cin >> visine_ulaza[i];

  int m;
  cin >> m;
  for (int i = 0; i < m; i++) {
    int nivo_vode;
    cin >> nivo_vode;
    cout << n - prvi_veci(visine_ulaza, nivo_vode) << '\n';
  }

  return 0;
}
```

Задатак: Зграде

Аутор: Душан Пејадић

Такмичење: 2021/22. квалификације 1, VI разред 6. задатак

За потребе научног експеримента научницима је потребно да у Њујорку пронађу две зграде на чијим ће крововима обавити одређена мерења притиска. Да би резултати били прецизни потребно је да разлика у висинама између тих зграда буде велика, али због неких услова везаних за апаратуру коју користе, та разлика не сме бити ни превелика. Максимална дозвољена разлика у висинама зграда је утврђена и износи h. Пошто зграда у Њујорку има много, научницима је врло тешко да пронађу одговарајуће зграде и због тога су замолили тебе за помоћ. Дат је низ од n зграда разних висина у коме је потребно да пронађеш две зграде чија је разлика у висинама што већа, али мања или једнака h.

Опис улаза

У првом реду стандардног улаза налазе се два броја: максимална дозвољена разлика у висини h (h је природан број, $1 \leq h \leq 2 \cdot 10^9$) и број зграда n ($2 \leq n \leq 10^5$) одвојени размаком. У наредних n редова се налазе природни бројеви (мањи или једнаки од $2 \cdot 10^9$) који представљају висине зграда.

Напомена: Гарантује се да ће постојати барем две зграде чија је разлика у висинама мања или једнака h.

Ограничења

- У тест примерима вредним 50 поена важи $2 \leq n \leq 1000$.
- У осталим тест примерима нема додатних ограничења.

Опис излаза

У једином реду стандардног улаза исписати прво висину мање зграде, а затим висину веће зграде које ће бити коришћене за експеримент. Бројеве одвојити размаком. У случају да постоји више решења, исписати оно код ког су зграде ниже (ако је једно решење 2 и 6, а друго 4 и 8, треба исписати прво).

Пример 1

Улаз	*Излаз*
5 6 12 4 20 16 22 17	12 17

Пример 2

Улаз	*Излаз*
8 5 13 4 20 11 57	4 11

Решење

Решење грубом силом

Најједноставније решење је да се са две угнежђене петље прође кроз сваки пар зграда и провери се да ли им је разлика у висинама мања од h. Уколико јесте тада проверавамо да ли је разлика у висинама већа од до сада највеће запамћене разлике и уколико јесте памтимо је као нову највећу разлику и памтимо посматрани пар зграда. С обзиром да се у случају више решења тражи оно у коме се појављују мање зграде треба и о томе водити рачуна, па ћемо памтити у случају да се појави пар зграда који постиже исту разлику као до сада најбољи пар, тај нови пар упамтити уколико су му зграде ниже, а у супротном задржавамо стари пар.

Потребно проћи кроз све парове зграда којих има $\frac{n \cdot (n-1)}{2}$. Како је максималан број зграда $n_{max} = 10^5$, максималан број парова је нешто мањи од $5 \cdot 10^9$ тако да кроз толики број парова не можемо проћи за $1s$ колико је временско ограничење. Како је у задатку речено да ће у тест примерима вредним 50п важити да је n до 1000, у том случају имамо око $5 \cdot 10^5$ парова и можемо их све обићи за $1s$. Дакле, ово решење нам може донети 50п на овом задатку.

Сортирање и техника два показивача

Претходно решење има такозвану квадратну сложеност ($\frac{n \cdot (n-1)}{2} = O(n^2)$), а нама је потребна сложеност која је боља од тога: линеарне $O(n)$ или сублинеарне $O(n \log n)$. Алгоритми сублинеарне сложености ће бити извршени за мање $1s$ када је $n\_max = 10^5$ јер је $n_{max} \log n_{max}$ реда величина једног милиона.

Пошто алгоритми сублинеарне сложености пролазе временско ограничење, то значи да можемо да приуштимо сортирање низа и треба размислити да ли нам сортирање нешто доноси. У овом задатку се испоставља да доноси.

Прво је потребно сортирати низ висина зграда растуће (уз могућност појаве више зграда исте висине). Сада ћемо поставити два показивача (два цела броја која представљају индексе у низу)

на прва два елемента у низу (рецимо i на први и j на други елемент). Прво ћемо померати показивач j ка крају низа докле год је разлика у висинама зграда на коју показују i и j мања или једнака h. Све време памтимо и највећу до сада постигнуту разлику, као и пар зграда који ју је постигао. Када j дође довољно далеко да је разлика у виснама зграда премашила h, тада померамо i док разлика не буде поново у дозвољеном оквиру. Након тога понављамо поступак (померамо j док разлика не буде већа од h, па померамо i док се разлика не врати у дозвољени оквир) док j не стигне до краја низа.

Треба уочити да када померамо i ка крају низа, није потребно да поново проверавамо зграде које су на позицијама мање од j. Ово се може видети на следећем примеру:

Нека је низ висина зграда: 1, 3, 6, 7, 8, 9, 15 и нека је $h = 6$.

На почетку постављамо i да показује на нулту позицију у низу (на којој се налази висина 1), а j на прву (висина 3).

Сада померамо j удесно и памтимо сваки пут нову највећу разлику. Када j стигне до позиције 3 (висина 7), максимална разлика коју ћемо упамтити је 6.

Сада се j помера на позицију 4 (висина 8) и разлика је сада $8 - 1 = 7$ што премашује h. Сада ћемо померити i за једно место удесно. Показивач i сада показује на позицију 1 (висина 3).

Оно што је потребно уочити је да није потребно проверавати разлику у висинама на позицији 1 (висина 3) и на позицији 3 (висина 7) јер смо већ проверавали разлику на позицијама 0 и 3, а та разлика је сигурно већа (или једнака) разлици висина на позицијама 1 и 3 пошто је зграда на позицији 0 нижа од зграде на позицији 1 (на почетку смо сортирали низ). Слично нема потребе проверавати разлику у висинама између зграде на позицији 1 и било које зграде лево од позиције 3. Дакле нема потребе враћати j уназад након померања i.

Такође, треба напоменути да уколико наиђемо на пар који има исту разлику у висинама као до сада запамћени најбољи пар, нећемо памтити нови пар јер нам се тражи нижи пар. Пошто пролазимо кроз растуће сортиран низ, сигурно је претходно запамћени пар нижи од новог.

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0);
  int n;
  int h;
  cin >> h >> n;
  vector<int> zgrade(n);
  for (int i = 0; i < n; i++)
    cin >> zgrade[i];
  sort(begin(zgrade), end(zgrade));
  int i = 0, j = 1;
  int maxD = 0;
  int in = 0, iv = 0;
  while (j < n) {
```

```
    if (zgrade[j] - zgrade[i] > h)
      i++;
    else {
      if (zgrade[j] - zgrade[i] > maxD) {
        maxD = zgrade[j] - zgrade[i];
        in = i;
        iv = j;
      }
      j++;
    }
  }
  cout << zgrade[in] << " " << zgrade[iv];

  return 0;
}
```

Задатак: Слаткиши за сав новац

Аутор: Филип Марић

Такмичење: 2019/20 квалификације 3, VII и VIII разред, 5. задатак

Дуж једне улице деца продају слаткише. Јована има G динара и жели да их потроши тако што ће кренути да се шета дуж улице и од сваког детета, редом, ће купити тачно један слаткиш (ни једно дете неће прескочити). Ако су познате цене свих слаткиша и ако је позната позиција са које Јована креће у куповину, одредити број слаткиша које ће Јована на тај начин купити.

Опис улаза

Са стандардног улаза се уноси број деце који продају слаткише n ($1 \leq n \leq 5 \cdot 10^4$), а затим и низ који садржи цене слаткиша (позитивни природни бројеви мањи од 100). Након тога се уноси број упита k ($1 \leq k \leq 5 \cdot 10^4$), а затим у k наредних редова упити који садрже позицију првог детета које ће Јована обићи (позиције се броје од нуле), а затим број динара које Јована има.

Опис излаза

На стандардни излаз за сваки упит у посебном реду исписати број слаткиша које ће Јована купити.

Пример

Улаз	*Излаз*
7 3 5 1 2 3 1 4 4 3 1 2 5 2 13 0 10	0 2 5 3

Објашњење

- У првом упиту Јована има један динар, а креће од детета које продаје слаткиш за 2 динара, па не може да купи ни један слаткиш.
- У другом упиту Јована има пет динара и купује слаткише који коштају 1 и 2 динара.
- У трећем упиту Јована има 13 динара и купује слаткише који коштају 1, 2, 3, 1 и 4 динара.
- У четвртом упиту Јована има 10 динара и купује слаткие који коштају 3, 5 и 1 динар.

Решење

Решење грубом силом подразумева да се за сваку почетну позицију са које Јована креће редом сабирају цене колача, све док је збир мањи од новца које Јована има. Сложеност таквог решења је $O(n \cdot q)$, што је с обзиром на ограничења недопустиво споро.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> cena(n);
  for (int i = 0; i < n; i++)
    cin >> cena[i];
  int Q;
  cin >> Q;
  for (int q = 0; q < Q; q++) {
    int p, novac;
    cin >> p >> novac;
    int kupljeno = 0;
    while (p < n && novac >= cena[p]) {
      novac -= cena[p];
      kupljeno++;
      p++;
    }
    cout << kupljeno << endl;
  }
}
```

Ефикасније решење можемо постићи ако уместо низа цена колача одржавамо низ збирова префикса низа цена (чувамо низ z такав да је $z_k = 0$ и $z_k = \sum_{i=0}^{k-1} c_k$). Пошто су цене позитивне, низ збирова префикса је сортиран растуће и може се претраживати бинарном претрагом. Ако Јована креће са позиције p а последњи колач купује на позицији p', тада се цена коју ће платити може израчунати као $z_{p'+1} - zp$. Дакле, бинарном претрагом ћемо наћи највећу вредност p' такву да је $z_{p'+1} - z_p \leq N$ (где је N износ новца који Јована има).

Низ префиксних збирова можемо израчунати у сложености $O(n)$ приликом учитавања низа цена, након чега вршимо q бинарних претрага, свака је сложености $O(\log n)$. Дакле, укупна сложеност је $O(n + q \log n)$.

Нагласимо да је ово један од задатака у ком се одговара на упите тј. у ком се наизменично читају подаци са стандардног улаза и исписују на стандардни излаз. Ако се не обрати посебна пажња, пуно времена може отићи на синхронизацију између улаза и излаза.

У језику C++ прво што треба урадити да би се убрзао улаз и излаз је искључивање сихнронизације са C-овском библиотеком за улаз и излаз (то се постиже наредбом `ios_base::sync_with_stdio(false)`). Када се то уради веома важно је да се нигде у програму не позивају функције из библиотеке `<cstdio>`. Након тога, важно је да спречимо пражњење излазног бафера при сваком испису. Прво је да искључимо аутоматско пражњење излазног бафера пре учитавања података са улаза, што радимо наредбом `cin.tie(0)`. Након

тога програм неће лепо радити у интерактивном режиму. На крају, потребно је да осигурамо да не користимо `endl`, који по својој дефиницији врши пражњење излазног бафера и да га заменимо исписом карактера `\n`. И ова измена доводи до тога да се у интерактивном тестирању програм не понаша исправно, али то нам не смета, јер је циљ да максимално убрзамо програм за аутоматско тестирање.

Друго решење (које додуше троши више меморије) је да резултате све сместимо у низ и да на крају програма испишемо тај низ.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false); cin.tie(0);
  int n;
  cin >> n;
  vector<int> zbir_prefiksa_cena(n+1);
  zbir_prefiksa_cena[0] = 0;
  for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    zbir_prefiksa_cena[i+1] = zbir_prefiksa_cena[i] + x;
  }
  int q;
  cin >> q;
  for (int i = 0; i < q; i++) {
    int p, novac;
    cin >> p >> novac;
    auto poc = next(begin(zbir_prefiksa_cena), p);
    auto kraj = end(zbir_prefiksa_cena);
    auto it = upper_bound(poc, kraj, zbir_prefiksa_cena[p] + novac);
    cout << distance(poc, it) - 1 << '\n';
  }
  return 0;
}
```

Задатак: Тачке

Аутор: Душан Пођадић

Такмичење: 2021/22. квалификације 1, VII и VIII разред 5. задатак

Дат је низ затворених интервала на x-оси координатног система (интервал је ограничен почетном и крајњом тачком које се задају) и низ тачака. За сваки од интервала одредити колико му унетих тачака припада.

Напомена: Уколико тачка има x-координату која одговара самој граници интервала, сматра се да та тачка припада интервалу. Интервали се могу преклапати што може довести до тога да

се једна тачка налази у више интервала.

Опис улаза

У првом реду стандардног улаза уноси се број тачака m ($1 \leq m \leq 10^5$), а у наредних m редова x-координата сваке од тачака (природни бројеви $1 \leq x_i \leq 2 \cdot 10^9$). У наредном реду се уноси број затворених интервала n ($1 \leq n \leq 10^5$), а у наредних n редова се уносе по два природна броја ($1 \leq l_i < d_i \leq 2 \cdot 10^9$) која означавају границе тих интервала (уноси се прво лева па десна граница).

Ограничења

- У тест примерима вредним 50 поена важи $1 \leq n, m \leq 1000$.
- У осталим тест примерима нема додатних ограничења.

Опис излаза

У n редова стандардног излаза исписати број тачака које припадају сваком од интервала (по редоследу уношења интервала).

Пример

Улаз	*Излаз*
5 3 6 7 1 4 3 1 3 2 6 8 9	2 3 0

Решење

Решење грубом силом

Најједноставније решење је да се за сваки интервал прође кроз низ тачака и провери за сваку тачку да ли припада интервалу. Уколико тачка припада интервалу треба увећати бројач и када се провере све тачке треба исписати вредност бројача.

Пошто је максималан број и интервала и тачака 10^5, угњежђене петље ће направити укупно $10^5 \cdot 10^5 = 10^10$ итерација што је превише да би се постигло за временско ограничење од $1s$. Како у тест примерима вредности 50 поена важи да је и број интервала и број тачака до 1000, у тим тест примерима број итерација ће бити највише $1000 \cdot 1000 = 10^6$, што је довољно мало да задовољи временско ограничење. Дакле, ово решење доноси 50 поена на задатку.

Решење бинарном претрагом

Да бисмо убрзали претрагу тачака које припадају интервалу можемо сортирати низ тачака. Сада пролазимо кроз сваки интервал и уместо да пролазимо кроз све тачке, довољно је да нађемо прву и последњу тачку у низу које припадају интервалу. Означимо границе интервала са A и B, а низ тачака са T. Сада је потребно пронаћи најлевљу тачку T_i која се налази у интервалу, односно за коју важи $A \leq T_i$. Такође, потребно је наћи најдешњу тачку T_j која припада

интервалу, односно за коју важи $T\_j \leq B$. Пошто је низ тачака сортиран све тачке између ће, такође, припадати посматраном интервалу. Тражене тачке се могу наћи бинарном претрагом која је модификована тако да не тражи конкретну вредност у низу него прву вредност која је већа или једнака односно мања или једнака од задате вредности. Након тога је лако да се, на основу добијених индекса тражених тачака, одреди колико тачака има између и самим тим колико тачака припада посматраном интервалу. За потребе бинарне претраге могу се на пример користити функције `upper_bound` и `lower_bound` из C++ стандардне библиотеке.

Сложеност сортирања низа тачака је $O(m \log m)$. Сложеност бинарне претраге је $O(\log m)$, а пошто је потребно извршити бинарне претраге за сваки од интервала, сложеност тог дела решења је $O(n \log m)$. Укупна сложеност целог решења је $O(m \log m + n \log m) = O((m + n) \log m)$. Како ни n ни m сигурно нису већи од 10^5, ово решење је довољно ефикасно да задовољи временско ограничење од $1s$.

Потребно је обратити пажњу да није довољно да се само лева граница интервала тражи бинарном претрагом, а да се онда, почевши од ње, линеарно тражи десна зато што интервали могу бити веома велики тако да се ефективно за сваки интервал прође кроз скоро цео низ тачака, а то се своди на већ коментарисано лошије решење.

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    int m;
    cin >> m;
    vector<int> tacke(m);
    for (int i = 0; i < m; i++)
        cin >> tacke[i];

    sort(tacke.begin(), tacke.end());

    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        int lg, dg;
        cin >> lg >> dg;
        auto ub = upper_bound(tacke.begin(), tacke.end(), dg);
        auto lb = lower_bound(tacke.begin(), tacke.end(), lg);
        cout << distance(lb, ub) << "\n";
    }

   return 0;
}
```

Задатак: Пуно фигурица

Аутор: Филип Марић

Такмичење: 2019/20 квалификације 2, VII и VIII разред, 6. задатак

Друштвену игру игра k играча и сваки играч игру почиње са по k фигура, при чему све фигуре једног играча морају бити исте јачине, док су јачине фигура различитих играча различите. Фигуре су доступне у неограниченим количинама, при чему је познат низ јачина доступних фигура. Напиши програм који одређује највећи број играча k који могу играти игру тако да разлика између укупне јачине свих фигура било која два играча не пређе задату границу.

Опис улаза

Са стандардног улаза се учитава број n ($1 \leq n \leq 10^5$), а затим у наредном реду n различитих расположивих јачина фигура (природни бројеви између 1 и 10^5). Наредни ред садржи границу (природни број између 1 и 10^{12}).

Опис излаза

На стандардни излаз исписати два броја раздвојена размаком – број k и најмању разлику укупних јачина фигура када игра k играча.

Пример

Улаз	*Излаз*	*Објашњење*

`5`
`5 4 2 7 3`
`15`
	`4 12`	Ако играчи узму по четири фигуре јачине $5, 4, 2$ и 3 највећа разлика јачина фигура играча биће $4 \cdot 5 - 4 \cdot 2 = 12$. Ако би играло 5 играча, морали би да узму и фигуре јачине 7 и највећа разлика би била $5 \cdot 7 - 5 \cdot 2 = 25$, што је веће од дозвољене границе.

Решење

Наивно решење

Задатак се може решити тако што се за свако k између 2 и n провери да ли је могуће игру одиграти са k играча.

Централно питање је како за дато k изабрати k играча, тако да разлика између укупне јачине фигура најјачег и најслабијег међу њима буде што мања. Ефикасан алгоритам се може направити ако се низ јачина фигура претходно сортира. Играчи тада треба да узимају фигуре чије су јачине узастопних k елемената низа (ако би неки играч заменио фигуре, добила би се већа разлика између најјачег и најслабијег играча). Зато је довољно проверити све узастопне k-точлане поднизове низа (којих има $n - k$), одузети последњи од првог члана и помножити резултат са k (јер сваки играч узима по k фигура).

Пошто се са порастом броја играча разлика може само повећати, претрагу можемо прекинути када први пут нађемо вредност k такву да игру не могу играти k играча. Сложеност таквог алгоритма, суштински заснованог на линеарног претрази је $O(n^2)$.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <limits>

using namespace std;
```

```
long long najmanja_razlika_k(const vector<int>& a, int k) {
  int min = numeric_limits<int>::max();
  for (size_t i = 0; i + k - 1 < a.size(); i++)
    if (a[i+k-1] - a[i] < min)
      min = a[i+k-1] - a[i];
  return min;
}

int main() {
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
  long long granica;
  cin >> granica;
  sort(begin(a), end(a));
  int k = 2;
  while (k <= n && k * najmanja_razlika_k(a, k) <= granica)
    k++;
  cout << k-1 << " " << (k-1) * najmanja_razlika_k(a, (k-1)) << endl;
  return 0;
}
```

Бинарна претрага

Брже решење можемо добити ако оптималну вредност k тражимо бинарном претрагом (то је могуће, јер вредности разлика расту са порастом k). Сложеност таквог алгоритма је $O(n \log n)$.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <limits>
#include <cassert>

using namespace std;

long long najmanja_razlika_k(const vector<int>& a, int k) {
  int min = numeric_limits<int>::max();
  for (size_t i = 0; i + k - 1 < a.size(); i++)
    if (a[i+k-1] - a[i] < min)
      min = a[i+k-1] - a[i];
  return min;
}

int main() {
  int n;
  cin >> n;
  vector<int> a(n);
```

```
  for (int i = 0; i < n; i++)
    cin >> a[i];
  long long granica;
  cin >> granica;
  sort(begin(a), end(a));
  int lK = 2, dK = n;
  while (lK <= dK) {
    int k = lK + (dK - lK) / 2;
    if (k * najmanja_razlika_k(a, k) <= granica)
      lK = k+1;
    else
      dK = k-1;
  }
  cout << dK << " " << dK * najmanja_razlika_k(a, dK) << endl;
  return 0;
}
```

Два показивача

Још једно брзо решење можемо да добијемо техником два показивача. За сваки леви крај сегмента одређујемо највећу могућу вредност десног краја која не прелази границу. Након сортирања користимо сегмент $[poc, kraj]$ који представља изабране фигуре. Овај сегмент је на почетку $[0, 0]$. Ако је разлика између укупне јачине најјачих и најслабијих фигура из сегмента довољно мала, продужавамо сегмент надесно (повећавамо $kraj$), а ако је већа од дозвољене, онда скраћујемо сегмент слева (повећавамо poc). Сигурни смо да након повећања левог краја, десни крај не морамо смањивати (размисли зашто). Успут памтимо највећу дужину сегмента и оптималну разлику при тој дужини сегмента. Када прођемо цео низ јачина фигура, исписујемо коначне вредности највеће дужине сегмента и оптималне разлике. Сложеност оваквог решења је $O(n \log n)$ у почетној фази сортирања, а након тога, у другој фази је линеарна $O(n)$.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <limits>
#include <cassert>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
  long long granica;
  cin >> granica;

  sort(begin(a), end(a));
  int poc = 0, kraj = 0;
```

```
  int opt_k = 1;
  long long opt_razlika = 0;
  while (kraj < n) {
    long long k = kraj - poc + 1;
    long long razlika = k * (a[kraj] - a[poc]);
    if (razlika > granica)
      poc += 1;
    else {
      if (k > opt_k) {
        opt_k = k;
        opt_razlika = razlika;
      } else if (k == opt_k)
        opt_razlika = min(opt_razlika, razlika);
      kraj += 1;
    }
  }
  cout << opt_k << " " << opt_razlika << endl;
  return 0;
}
```

Задатак: Допуна мејлова

Аутор: Филип Марић

Такмичење: 2022/2023. квалификације 1, 7. разред, 6. задатак и 8. разред 5. задатак

Апликације за слање мејлова чувају раније коришћене мејл адресе а онда помажу корисницима тако што на основу њих допуњавају започет унос нове мејл адресе (тзв. опција аутоматског допуњавања, енгл. aucomplete). Напиши програм који ефикасно имплементира ову функционалност.

Опис улаза

Са стандардног улаза се учитава број мејл адреса n ($1 \leq n \leq 10^5$), а затим у n наредних редова по једна мејл адреса. Након тога се уноси број различитих упита (започетих мејл адреса) m ($1 \leq m \leq 10^5$), а након тога m упита.

Опис излаза

На стадардни излаз за сваки упит исписати број мејл адреса које почињу тим упитом.

Пример

Улаз	*Излаз*
9 pera@gmail.com lidija.djokic@yahoo.com andrija@yahoo.com laza@gmail.com ana.petrovic@mail.ru lazica@hotmail.com milica@gmail.com larisa@zoho.com anaconda@python.org 3 laz milica@ a	2 1 3

Решење

Линеарна претрага

Задатак се може решити тако што се за сваки префикс адресе линеарном претрагом целог низа пронађу оне адресе које почињу тим префиксом.

Ако постоји n елемената низа адреса, сложеност линеарне претраге је $O(n)$, па ако се испитује m префикса, укупна сложеност је $O(mn)$.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <tuple>

using namespace std;

pair<int, int> dopune(const vector<string>& mejlovi,
                      const string& prefiks) {
  vector<string> result;
  int donja_granica = lower_bound(begin(mejlovi), end(mejlovi), prefiks) - begin(mejlovi);
  string sledeci_prefiks = prefiks;
  sledeci_prefiks.back()++;
  int gornja_granica = lower_bound(begin(mejlovi), end(mejlovi), sledeci_prefiks) - begin(mejlovi);
  return make_pair(donja_granica, gornja_granica);
}


int main() {
  ios_base::sync_with_stdio(false);
  int n;
  cin >> n;
  vector<string> mejlovi(n);
  for (int i = 0; i < n; i++)
```

```
    cin >> mejlovi[i];

  int m;
  cin >> m;
  for (int i = 0; i < m; i++) {
    string prefiks;
    cin >> prefiks;
    int broj = 0;
    for (const string& str : mejlovi)
      if (str.compare(0, prefiks.size(), prefiks) == 0)
        broj++;
    cout << broj << endl;
  }
  return 0;
}
```

Бинарна претрага

Задатак се ефикасније може решити тако што се адресе сортирају, а затим се за сваки префикс бинарном претрагом проналазе адресе које почињу тим префиксом. Прва таква адреса је најмања адреса (у лексикографском поретку) која је већа или једнака од унетог префикса. Иако се може помислити да се након њеног проналаска, остале адресе могу пронаћи у петљи која наставља даље све док адресе почињу тим префиксом, то решење је потенцијално неефикасно (јер може бити пуно таквих адреса). Зато је боље новом бинарном претрагом одредити прву адресу која не почиње тим префиксом, што се може урадити тако што се последњи карактер тог префикса увећа и пронађе позиција адресе која је већа или једнака од тако трансформисаног префикса.

Ако постоји n елемената низа адреса, сложеност иницијалног сортирања је $O(n \log n)$, а сложеност сваке од ове две бинарне претраге је $O(\log n)$, па ако се испитује m префикса, укупна сложеност је $O((n+m) \log n)$.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <tuple>

using namespace std;

pair<int, int> dopune(const vector<string>& mejlovi,
                      const string& prefiks) {
  vector<string> result;
  int donja_granica = lower_bound(begin(mejlovi), end(mejlovi), prefiks) - begin(mejlovi);
  string sledeci_prefiks = prefiks;
  sledeci_prefiks.back()++;
  int gornja_granica = lower_bound(begin(mejlovi), end(mejlovi), sledeci_prefiks) - begin(mejlovi);
  return make_pair(donja_granica, gornja_granica);
}
```

```
int main() {
  ios_base::sync_with_stdio(false);
  int n;
  cin >> n;
  vector<string> mejlovi(n);
  for (int i = 0; i < n; i++)
    cin >> mejlovi[i];
  sort(begin(mejlovi), end(mejlovi));

  int m;
  cin >> m;
  vector<string> prefiksi(m);
  for (int i = 0; i < m; i++) {
    string prefiks;
    cin >> prefiks;
    int a, b;
    tie(a, b) = dopune(mejlovi, prefiks);
    cout << b - a << endl;
  }
  return 0;
}
```

Задатак: Фабрика

Аутор: Душан Пођадић

Такмичење: окружно 2021/22., VIII разред, 4. задатак

Марко и Деа се баве енергетиком. Деа је за фабрику аутомобилских гума проценила да ће у наредном периоду трошити P киловата електричне енергије. Да би се одржала равнотежа у енергетском систему, потребно је изабрати тачно три електране за напајање ове фабрике чија ће укупна производња електричне енергије бити тачно P. Марко је за n електрана проценио њихову будућу производњу у киловатима, а сада је на теби нађеш три одговарајуће.

Опис улаза

У првом реду стандардног улаза налазе се природан број P који представља процењену потрошњу фабрике у наредном периоду и природан број $n (3 \leq n \leq 2500)$ који представља укупан број електрана које је Марко посматрао (та два броја су одвојена размаком). У другом реду се налази n природних бројева који представљају производњу електрана. Свих n бројева ће бити мањи или једнаки $5 \cdot 10^8$. Гарантује се да решење постоји.

Опис излаза

У једином реду стандардног излаза исписати три броја који представљају производњу оне три електране које се могу искористити за напајање фабрике. У случају више решења исписати било које.

Пример

Улаз	*Излаз*
800 7 130 780 20 70 600 400 400	130 70 600

Решење

У питању је чувени проблем 3SUM чије је решење детаљно описано у литератури.

Груба сила

Решење грубом силом подразумева да се употребе три угнежђене петље и тако испитају све тројке фабрика и оно је веома неефикасно.

Сложеност овог решења је $O(n^3)$.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int t, n;
  cin >> t >> n;

  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  sort(begin(a), end(a));

  for (int i = 0; i < n; i++){
    if (3 * a[i] > t)
      break;
    for (int j = i + 1; j < n; j++){
      if (a[i] + 2 * a[j] > t)
        break;
      for (int k = j + 1; k < n; k++){
        if (a[i] + a[j] + a[k] > t)
          break;
        if (a[i] + a[j] + a[k] == t) {
          cout << a[i] << ' ' << a[j] << ' ' << a[k] << '\n';
          return 0;
        }
      }
    }
  }
  return 0;
}
```

Техника два показивача

Техником два показивача могуће је да се једним проласком кроз низ пронађе пар елемената који има дати збир. Наш проблем се може свести на овај потпроблем, тако што се за сваки елемент

a_i проверава да ли међу осталим елементима постоји пар елемената који има збир $s - a_i$, где је s тражени збир елемената тројке.

Сложеност овог решења је $O(n^2)$.

```
#include <bits/stdc++.h>
typedef long long ll;
using namespace std;

bool three_sum(const vector<int>& v, int sum) {
  vector<int> a = v;
  sort(begin(a), end(a));
  int n = a.size();
  for (int i = 0; i < n-2; i++) {
    int l = i+1;
    int r = n-1;
    while (l < r) {
      if (a[i]+a[l]+a[r] == sum) {
        cout << a[i] << " " << a[l] << " " << a[r] << endl;
        return true;
      }
      else if (a[i]+a[l]+a[r] > sum)
        r--;
      else
        l++;
    }
  }
  return false;
}

int main() {
  int sum, n;
  cin >> sum >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  three_sum(a, sum);
  return 0;
}
```

Бинарна претрага

Проверу да ли постоји пар елемената датог збира је могуће извршити и бинарном претрагом (тако што се фиксира један елемент и бинарном претрагом провери да ли у сортираном низу постоји његова допуна до датог збира).

Сложеност овог решења је $O(n^2 \log n)$.

```
#include <bits/stdc++.h>
typedef long long ll;
```

```
using namespace std;

bool three_sum(const vector<int>& v, int sum) {
  vector<int> a = v;
  sort(begin(a), end(a));
  int n = a.size();
  for (int i = 0; i < n; i++)
    for (int j = i+1; j < n; j++)
      if (binary_search(next(begin(a), i+1), end(a), sum - a[i] - a[j])) {
        cout << a[i] << " " << a[j] << " " << sum - a[i] - a[j] << endl;
        return true;
      }
  return false;
}

int main() {
  int sum, n;
  cin >> sum >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  three_sum(a, sum);
  return 0;
}
```

Скуп

За сваки пар позиција (i, j), за $0 \leq i < j < n$ можемо проверити да ли се у делу низа на позицијама из интервала (i, j) налази елемент $s - a_i - a_j$. Да би та претрага била ефикасна, елементе из тог интервала можемо чувати у скупу.

У зависности од тога како је имплементиран скуп, сложеност овог решења може бити $O(n^2 \log n)$ или чак $O(n^2)$.

```
#include <iostream>
#include <set>
#include <vector>

using namespace std;

bool three_sum(const vector<int>& a, int sum) {
  int n = a.size();
  for (int i = 0; i < n; i++) {
    set<int> values;
    for (int j = i + 1; j < n; j++){
      if (values.count(sum - a[i] - a[j])) {
        cout << a[i] << " " << a[j] << " " << sum - a[i] - a[j] << endl;
        return true;
      }
```

```
      values.insert(a[j]);
    }
  }

  return false;
}

int main() {
  int sum, n;
  cin >> sum >> n;

  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  three_sum(a, sum);

  return 0;
}
```

Задатак: Што више сличних

Аутор: Филип Марић

Такмичење: окружно 2022/2023, VIII разред, 3. задатак

Треба одабрати што више елемената датог низа бројева тако да је разлика између највећег и најмањег одабраног елемента највише k (може да буде и k).

Опис улаза

Са стандардног улаза се учитава дужина низа n ($1 \leq n \leq 10^5$), затим елементи низа (природни бројеви мањи од 10^9) и на крају природан број k.

Опис излаза

На стандардни излаз исписати тражени највећи број одабраних елемената низа.

Пример 1

Улаз	*Излаз*	*Објашњење*
12 34 7 31 17 24 40 3 5 20 53 38 29 10	4	Максимално се могу узети четири елемента низа (на пример, 34, 31, 38 i 29), а да је разлика између највећег и најмањег члана ($38 - 29$) manja ili jednaka 10.

Пример 2

Улаз	*Излаз*
11 32 50 45 40 5 10 15 20 25 35 30 20	6

Решење

Ефикасно решење се може добити ако се низ прво сортира, а затим се примени техника два показивача.

Након сортирања, крећемо од првог елемента у низу и померамо десни показивач све док не дођемо до првог елемента који није више сличан том првом елементу низа (већи је од њега за више од k) или док се евентуално не дође до краја низа. Разлика између десног и левог показивача представља број елемената који су слични првом елементу низу.

Након тога леви показивач померамо на следећу позицију, а десни показивач ажурирамо. Можемо приметити да десни показивач не треба да се помера налево (јер ако су a_i и a_j на растојању највише k, пошто је низ сортиран, то исто важи и за a_{i+1} и a_j). Са друге стране, десни показивач можда треба померити удесно и то се ради све док се не пронађе први елемент који му није више сличан или док се евентуално не дође до краја низа. Када се то деси, знамо број елемената сличних оном на позицији левог показивача и ако је тај број већи од дотадашњег максимума, ажурира се максимум.

Поступак се понавља све док десни показивач не дође до краја низа.

Уместо технике два показивача, након сортирања је могуће употребити и бинарну претрагу.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
  int k;
  cin >> k;
  sort(begin(a), end(a));
  int i = 0, j = 0;
  int maks_duzina = 1;
  while (j < n) {
    while (j < n && a[j] - a[i] <= k)
      j++;
    maks_duzina = max(maks_duzina, j - i);
    i++;
  }
  cout << maks_duzina << endl;
  return 0;
}
```

Задатак: Блиски

Аутори: Владан Ковачевић и Иван Дрецун

Такмичење: државно 2022/2023, VI разред, 4. задатак

Бројеви a и b су блиски ако се разликују за мање од d. Дата су два низа бројева дужине n и број d. Одредити колико елемената првог низа има близак број у другом низу.

Опис улаза

Са стандардног улаза се уносе бројеви n ($1 \leq n \leq 1000000$) и d ($1 \leq d \leq 100000$), затим елементи првог низа, а онда елементи другог низа.

Опис излаза

На стандардни излаз исписати тражени број.

Пример

Улаз	*Излаз*	*Објашњење*

`6 2`
`5 1 7 -5 10 13`
`12 0 3 -2 7 9`
	`4`	Из првог низа бројеви који имају близак у другом низу су 1, 7, 10 и 13, а њихови блиски бројеви из другог низа су редом 0, 7, 9, 12.

Решење

Груба сила

Наивно решење подразумева да за сваки елемент у првом низу проверимо да ли је неки елемент у другом низу њему близак, тако што пролазимо кроз други низ у петљи - елемент по елемент. Овакав начин тражења одговарајућег елемента је *линеарна претрага* која подразумева да се у најгорем случају прође кроз цео други низ (нпр. ако у другом низу не постоји такав елемент).

Дакле, ако први низ има n елемената и други низ има n елемената, онда за сваки елемент првог низа вршимо линеарну претрагу у другом низу тј. n пута извршавамо петљу која се у најгорем случају извршава n пута, па је укупна сложеност овог решења квадратна, што је веома неефикасно.

```
#include <iostream>
#include <vector>

using namespace std;

int bliski(int a, int b, int d) {
  return abs(a-b) < d;
}

int ima_bliskih(int x, const vector<int>& v, int d) {
  for (int el : v)
    if (bliski(el, x, d))
      return 1;
  return 0;
}
```

```
int main() {
  int n, d;
  cin >> n >> d;

  vector<int> v1(n);
  vector<int> v2(n);
  for (int i = 0; i < n; i++)
    cin >> v1[i];
  for (int i = 0; i < n; i++)
    cin >> v2[i];

  int broj_bliskih = 0;
  for (int el : v1)
    if (ima_bliskih(el, v2, d))
      broj_bliskih++;

  cout << broj_bliskih << endl;

  return 0;
}
```

Сортирање и бинарна претрага

Како бисмо убрзали тражење одговарајућег елемента у другом низу, можемо уместо линеарне претраге да користимо *бинарну претрагу*, која је знатно бржа. Да бисмо користили бинарну претрагу, потребно је прво да сортирамо други низ, што није проблем, јер је сортирање (адекватним алгоритмом) знатно брже од двоструке петље. Дакле, сортирање другог низа и примена бинарне претраге n пута ће бити брже од грубе силе и задовољиће ограничења дата у задатку.

```
#include<iostream>
#include<vector>
#include<algorithm>

using namespace std;

int bliski(int a, int b, int d) {
  return abs(a-b) < d;
}

int ima_bliskih(int x, vector<int>& v, int d) {
  int l = 0;
  int r = v.size()-1;

  while (l <= r) {
    int m = (l+r)/2;

    if (bliski(x, v[m], d))
      return 1;
    else if (x > v[m])
      l = m + 1;
```

```
    else
      r = m - 1;
  }

  return 0;
}

int ima_bliskih2(int x, vector<int>& v, int d) {
  auto lower = lower_bound(v.begin(), v.end(), x);

  if (lower == v.begin())
    return bliski(x, v.front(), d);

  if (lower == v.end())
    return bliski(x, v.back(), d);

  return bliski(x, *lower, d) || bliski(x, *(lower-1), d);
}

int main() {
  int n, d;
  cin >> n >> d;
  vector<int> v1(n);
  vector<int> v2(n);
  for (int i=0; i<n; i++)
    cin >> v1[i];
  for (int i=0; i<n; i++)
    cin >> v2[i];

  sort(v1.begin(), v1.end());
  sort(v2.begin(), v2.end());

  int broj_bliskih = 0;
  for (int el : v1)
    if (ima_bliskih(el, v2, d))
      broj_bliskih++;

  cout << broj_bliskih << endl;
  return 0;
}
```

Задатак: Околина

Аутори: Иван Дрецун, Владан Ковачевић и Филип Марић

Такмичење: државно 2022/2023, VII разред, 4. задатак

Дат је скуп од n целих бројева и природан број d. Кажемо да је елемент a у околини елемента b ако им је разлика мања или једнака d. Напиши програм који одређује који елемент скупа има

највише елемената у својој околини као и колико их има.

Опис улаза

Са стандардног улаза се уносе бројеви n ($1 \leq n \leq 2 \cdot 10^5$) и d ($1 \leq n \leq 2 \cdot 10^8$), одвојени размаком. Из наредне линије се уноси n различитих целих бројева x_i ($-10^9 \leq x_i \leq 10^9$) одвојених размаком, који представљају елементе скупа.

Опис излаза

На стандардни излаз исписати два природна броја одвојена размаком. Прва вредност представља елемент скупа који има највећу околину (ако има више таквих, исписати најмањи), а друга вредност представља број елемената те околине.

Пример 1

Улаз	*Излаз*
`7 4` `4 -6 3 12 1 -4 9`	`1 3`

Пример 2

Улаз	*Излаз*
`4 1` `3 5 1 2`	`2 3`

Пример 3

Улаз	*Излаз*
`5 2` `-4 -2 0 1 2`	`0 4`

Решење

Груба сила

Решење грубом силом подразумева да за сваки елемент пребројимо елементе у његовој околини тако што ћемо тај елемент упоредити са свим осталим. Дакле, сваки од n елемената се пореди са свих n низа, па је укупан број поређења n^2, што доводи до неефикасног програма.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int n, d;
  cin >> n >> d;

  vector<int> v(n);
  for (int i = 0; i < n; i++)
    cin >> v[i];

  int sol = 0, solv = v[0];
  for (int i = 0; i < n; i++) {
    int uOkolini = 0;
    for (int j = 0; j < n; j++) {
      if (abs(v[i] - v[j]) <= d)
        uOkolini++;
    }
    if (uOkolini > sol || (uOkolini == sol && v[i] < solv)) {
      sol = uOkolini;
      solv = v[i];
    }
  }
```

```
  cout << solv << ' ' << sol << endl;

  return 0;
}
```

Сортирање и бинарна претрага

Ефикасније решење се добија тако што претходно сортирамо низ. У сортираном низу бинарном претрагом ефикасно можемо да одредимо број елемената у околини било ког датог елемента a_i. Довољно је пронаћи индекс првог елемента који је строго већи од a_i+d и индекс првог елемента који је већи или једнак $a_i - d$. Разлика између та два индекса представља број елемената у околини елемента a_i.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int n, d;
  cin >> n >> d;

  vector<int> v(n);
  for (int i = 0; i < n; i++)
    cin >> v[i];

  sort(begin(v), end(v));

  int sol = 0, solv = v[0];
  for (int i = 0; i < n; i++) {
    int s = distance(lower_bound(begin(v), end(v), v[i] - d),
                     upper_bound(begin(v), end(v), v[i] + d));
    if (s > sol) {
      sol = s;
      solv = v[i];
    }
  }

  cout << solv << ' ' << sol << endl;

  return 0;
}
```

Сортирање и два показивача

Уместо бинарне претраге након сортирања можемо употребити и технику два показивача.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int n, d;
  cin >> n >> d;

  vector<int> v(n);
  for (int i = 0; i < n; i++)
    cin >> v[i];

  sort(begin(v), end(v));

  int l = 0, r = 0, sol = 0;
  int solv = v[0];
  for (int i = 0; i < n; i++) {
    while (v[i] - v[l] > d)
      l++;
    while (r < n && v[r] - v[i] <= d)
      r++;
    if (r - l > sol) {
      sol = r - l;
      solv = v[i];
    }
  }

  cout << solv << ' ' << sol << endl;

  return 0;
}
```

Задатак: Wifi снага

Аутор: Филип Марић

Такмичење: 2023/2024. квалификације 1, VII и VIII разред, 5. задатак

Познате су локације кућа дуж једне улице (њихове целобројне x-координате) и локације WiFi предајника који су постављени дуж те улице (поново њихове целобројне x-координате). Одредити минималну снагу сигнала коју је потребно подесити свим предајницима (свима се мора подесити иста снага) да би свака кућа имала сигнал. Ако је снага предајника на позицији x једнака d, тада се њиме обухватају куће на позицијама из интервала $[x - d, x + d]$.

Опис улаза

Са стандардног улаза се учитава број кућа m ($1 \leq m \leq 10^5$), а затим локације кућа (бројеви између 0 и 10^6). Затим се учитава број предајника n ($1 \leq n \leq 10^5$), а затим локације предајника

(бројеви између 0 и 10^6).

Опис излаза

На стандардни излаз исписати минималну снагу потребну да се све куће покрију.

Пример 1

Улаз	*Излаз*	*Објашњење*
4 1 2 3 4 2 1 5	2	**Објашњење** Ако је снага предајника 2, тада први обухвата интервал $[-1, 3]$, а други покрива интервал $[3, 7]$, што покрива све куће. Ако би снага била 1, тада би први обухватао интервал $[0, 2]$, а други $[4, 6]$, па кућа на позицији 3 не би била покривена.

Пример 2

Улаз	*Излаз*
10 13 4 18 9 16 38 25 42 7 19 5 2 16 33 26 10	9

Решење

Груба сила

Решење грубом силом подразумева да мало по мало повећавамо снагу све док све куће не буду покривене.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

bool sveKucePokrivene(const vector<int>& kuce,
                      const vector<int>& predajnici,
                      int snaga) {
  for (int k : kuce) {
    bool pokrivena = false;
    for (int p : predajnici) {
      if (p-snaga <= k && k <= p+snaga) {
        pokrivena = true;
        break;
      }
    }
    if (!pokrivena)
      return false;
  }
  return true;
}

int main() {
  int m;
```

```
  cin >> m;
  vector<int> kuce(m);
  for (int i = 0; i < m; i++)
    cin >> kuce[i];
  int n;
  cin >> n;
  vector<int> predajnici(n);
  for (int i = 0; i < n; i++)
    cin >> predajnici[i];

  int snaga = 0;
  while (!sveKucePokrivene(kuce, predajnici, snaga))
    snaga++;

  cout << snaga << endl;
  return 0;
}
```

Сортирање и техника два показивача

Можемо одредити најмању потребну снагу за сваку кућу и одредити максимум тих вредности. За сваку кућу одређујемо најближи предајник (анализирајући најближи предајник који је испред ње и најближи предајник који је иза ње) и најмања потребна снага за ту кућу је растојање до њој најближег предајника.

Ако сортирамо низ кућа и низ предајника до решења можемо доћи техником два показивача.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int m;
  cin >> m;
  vector<int> kuce(m);
  for (int i = 0; i < m; i++)
    cin >> kuce[i];
  int n;
  cin >> n;
  vector<int> predajnici(n);
  for (int i = 0; i < n; i++)
    cin >> predajnici[i];

  sort(begin(kuce), end(kuce));
  sort(begin(predajnici), end(predajnici));

  int snaga = 0;
  int p = 0;
```

```
  for (int k = 0; k < m; k++) {
    while (p < n && predajnici[p] < kuce[k])
      p++;
    int potrebno;
    if (p == 0)
      potrebno = predajnici[p] - kuce[k];
    else if (p == n)
      potrebno = kuce[k] - predajnici[p-1];
    else
      potrebno = min(predajnici[p] - kuce[k], kuce[k] - predajnici[p-1]);
    snaga = max(snaga, potrebno);
  }

  cout << snaga << endl;
  return 0;
}
```

Сортирање и бинарна претрага

Можемо одредити најмању потребну снагу за сваку кућу и одредити максимум тих вредности. За сваку кућу одређујемо најближи предајник (анализирајући најближи предајник који је испред ње и најближи предајник који је иза ње) и најмања потребна снага за ту кућу је растојање до њој најближег предајника.

Ако сортирамо само низ предајника, најближи предајник иза куће можемо наћи бинарном претрагом (тражимо најмањи број у низу који је већи или једнак од дате вредности, што се може урадити библиотечком функцијом).

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  int m;
  cin >> m;
  vector<int> kuce(m);
  for (int i = 0; i < m; i++)
    cin >> kuce[i];
  int n;
  cin >> n;
  vector<int> predajnici(n);
  for (int i = 0; i < n; i++)
    cin >> predajnici[i];

  int snaga = 0;
  sort(begin(predajnici), end(predajnici));
  for (int kuca : kuce) {
    int p = distance(begin(predajnici),
```

```
                    lower_bound(begin(predajnici), end(predajnici), kuca));
    int potrebno;
    if (p == n)
      potrebno = kuca - predajnici[n-1];
    else if (p == 0)
      potrebno = predajnici[0] - kuca;
    else
      potrebno = min(kuca - predajnici[p-1], predajnici[p] - kuca);

    snaga = max(snaga, potrebno);
  }

  cout << snaga << endl;

  return 0;
}
```

Бинарна претрага по решењу

До оптималног решења се може доћи и процесом бинарне преграге по решењу. Знамо да оно лежи у интервалу $[0, R]$, где је R распон између најближе и најдаље куће. Половимо тај интервал, уз коришћење помоћне функције којом се проверава да ли је текућа снага довољна да све куће буду покривене.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

bool sveKucePokrivene(const vector<int>& kuce,
                      const vector<int>& predajnici,
                      int snaga) {
  for (int k : kuce) {
    auto it = lower_bound(begin(predajnici), end(predajnici), k - snaga);
    if (it == end(predajnici) || *it > k + snaga)
      return false;
  }
  return true;
}

int main() {
  int m;
  cin >> m;
  vector<int> kuce(m);
  for (int i = 0; i < m; i++)
    cin >> kuce[i];
  int n;
  cin >> n;
  vector<int> predajnici(n);
```

```
  for (int i = 0; i < n; i++)
    cin >> predajnici[i];
  sort(begin(predajnici), end(predajnici));

  int l = 0;
  int d = abs(*min_element(begin(kuce), end(kuce)) -
              *max_element(begin(kuce), end(kuce)));

  while (l <= d) {
    int snaga = l + (d - l) / 2;
    if (sveKucePokrivene(kuce, predajnici, snaga))
      d = snaga - 1;
    else
      l = snaga + 1;
  }

  cout << l << endl;
  return 0;
}
```

Задатак: Арматура

Аутор: Филип Марић

Такмичење: СИО 2021/22. 2. задатак

На градилишту се налази арматура (дугачке челичне шипке) коју грађевинци желе да употребе за изградњу носећег стуба. Постоји n комада арматуре, потенцијално неједнаке дужине. Арматура се може скраћивати и од дугачких комада може да се направи више краћих, међутим, из безбедносних разлога арматура се не може продужавати и од краћих комада арматуре није допуштено правити дуже. Познато је да стуб мора да у себи садржи бар k комада арматуре исте дужине. Написати програм који одређује највећу могућу дужину стуба који се може изградити од постојећих комада арматуре.

Опис улаза

Са стандардног улаза се учитава број n ($1 \leq n \leq 10^5$), а затим у наредном реду n позитивних природних бројева који представљају дужину постојеће арматуре у центиметрима. У трећем реду се налази број k ($1 \leq k \leq 10^6$).

Опис излаза

На стандардни излаз исписати висину највећег могућег стуба, такође у центиметрима.

Пример 1

Улаз	*Излаз*
5 100 80 60 90 70 10	33

Пример 2

Улаз	*Излаз*
2 40 40 100	0

Пример 3

Улаз	*Излаз*
10 10 20 30 40 50 60 70 80 90 100 20	22

Решење

За дату висину стуба једноставно можемо проверити да ли имамо довољно комада жице да би се тај стуб направио. Ако је висина стуба v, а дужина жице d, тада се од те жице може направити $\left\lfloor \frac{d}{v} \right\rfloor$ мањих жица које се могу уградити у стуб. Дакле, стуб се може направити ако и само ако је

$$\sum_{i=0}^{n-1} \left\lfloor \frac{d_i}{v} \right\rfloor \geq k,$$

где су d_i дужине n жица које имамо на располагању, v је висина стуба, а k број потребних комада арматуре дужине v.

Задатак онда можемо решити бинарном претрагом по решењу тако што пронађемо највећу висину стуба v за коју можемо исећи довољно комада арматуре. Интервал који бинарно претражујемо на почетку можемо иницијализовати на вредности $[1, \max_i d_i]$, јер знамо да се стуб висине 0 увек може направити као и да се не може направити стуб који је строго веће висине од дужине најдуже жице.

Бинарна претрага је применљива јер знамо да до одређених висина стуба имамо довољно жице, а да је од одређене висине стуба немамо довољно жице, тако да заправо тражимо преломну тачку, тј. највећу висину стуба за коју имамо довољно жица тј. последњи елемент низа који задовољава услов. Приметимо да у овом случају немамо вредности смештене у низ, већ их рачунамо по потреби. Зато бинарну претрагу имплементирамо ручно.

Сложеност једне провере да ли се стуб може направити је $O(n)$. Ширина почетног интервала који се бинарно претражује је $\max_i d_i$. Зато је укупна сложеност једнака $O(n \log(\max_i d_i))$.

```
bool mozeStub(const vector<int>& duzineZica, int visinaStuba, int potrebnoZica) {
  long long brojZica = 0;
  for (int duzinaZice : duzineZica) {
    brojZica += duzinaZice / visinaStuba;
    if (brojZica >= potrebnoZica)
      return true;
  }
  return false;
}

int maksVisinaStuba(const vector<int>& duzineZica, int potrebnoZica) {
  int l = 1, d = *max_element(begin(duzineZica), end(duzineZica));
  while (l <= d) {
    int s = l + (d - l) / 2;
    if (mozeStub(duzineZica, s, potrebnoZica))
      l = s + 1;
    else
      d = s - 1;
  }
  return d;
}
```

Задатак: Подморница

Аутори: Иван Дрецун, Душан Пођадић

Такмичење: 2021/22. квалификације 2, VI разред, 6. задатак и 7. разред, 5. задатак

Подморница се састоји из n сегмената исте ширине и различитих висина, при чему је дно подморнице равно. Притисак извршен на један сегмент се рачуна као удаљеност између врха тог сегмента и површине воде, ако је врх тог сегмента испод површине, а иначе је нула. Укупан притисак на подморницу се рачуна као збир притиска извршеног на сваки сегмент. Познато је да помдорница не може да издржи притисак већи од p. Написати програм који одређује максималну дубину до које подморница може да иде. Дубина подморнице се рачуна као удаљеност између дна подморнице и површине воде.

Опис улаза

У првом реду стандардног улаза налазе се два цела броја, n ($1 \leq n \leq 100000$) и p ($0 \leq p \leq 10^9$). У другом реду је n целих бројева h_i ($1 \leq h_i \leq 10^9$) који представљају висине сегмената подморнице.

Опис излаза

На стандардни излаз исписати један цео број који представља максималну дубину до које подморница може да иде.

Пример

Улаз	*Излаз*	*Објашњење*
`6 8` `2 1 3 3 4 1`	`3`	На слици је приказана подморница на дубини 3. Испод сваког сегмента написан је притисак који је на њега извршен.

```
____x_
..xxx.
x.xxx.
xxxxxx
120002
```

Решење

Инкрементално потапање једног по једног сегмента

У задатку је потребно одредити максималну дубину коју подморница може да достигне, а да притисак на њу не буде преко дозвољене границе. Некада је решење такво да је цела подморница потопљена, а некада да је потопљен само један њен део. У сваком случају је пожељно да сортирамо низ висина сегмената и да онда извршимо анализу. Идеја је да прођемо кроз низ сегмената и покушамо да потопимо подморницу на дубину тако да је врх посматраног семгента тачно на површини и успут рачунамо притисак који је извршен на тако потопљену подморницу.

Обрађујемо један по један сегмент покушавајући да га у потпуности потопимо. Одржавамо текућу дубину подморнице d и текући притисак p на тој дубини. Ако је подморница након потапања i сегмената била на дубини d, тада се би се потпуним потапањем сегмента на позицији i висине h_i дубина повећала за $\Delta d = h_i - d$, док се притисак повећао за $\Delta p = i \cdot \Delta d$ и добија би се нови притисак $p' = p + \Delta p = p + i \cdot \Delta d$ (приметимо да нови притисак израчунавамо инкрементално на основу старог, без потребе за анализом појединачних сегмената).

Ако добијени притисак p' не прелази границу p_{max}, текућу дубину ажурирамо на h_i, текући притисак ажурирамо на p' и прелазимо на наредни сегмент.

Ако добијени притисак прелази границу p_{max}, тада није могуће у потпуности потопити сегмент висине h_i. Зато одређујемо највеће могуће повећање дубине Δd такво да је нови притисак $p' = p + i \cdot \Delta d$ и даље одржив тј. да је $p' \leq p_{max}$. Зато тражимо максимално Δd такво да је $\Delta d \leq \frac{p'-p}{i}$, а то је $\Delta d = \left\lfloor \frac{p'-p}{i} \right\rfloor$. Висину увећавамо за тако одређену разлику Δd, а притисак за $\Delta p = i \cdot \Delta d$. Пошто се повећањем дубине макар за 1 прелази граница притиска, овим је одређена највећа могућа дубина, па се петља која потапа један по један сегмент може прекинути.

Сложеношћу решења доминира иницијално сортирање $O(n \log n)$, док се касније потапање једног по једног сегмента врши у сложености $O(n)$. Ако би се притисак сваки пут рачунао изнова, уместо инкрементално, добило би се неефикасно решење сложености $O(n^2)$.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0);

  int n;
  int64_t maxPritisak;
  cin >> n >> maxPritisak;
  vector<int64_t> h(n);
  for (int i = 0; i < n; i++)
    cin >> h[i];

  sort(begin(h), end(h));

  // tekuca dubina i pritisak na toj dubini
  int64_t dubina = 0, pritisak = 0;
  for (int i = 0; i < n; i++) {
    // ako segment visine h[i] jos nije ceo potopljen
    if (h[i] > dubina) {
      // pritisak koji bi nastalo potpunim potapanjem segmenta visine h[i]
      int64_t noviPritisak = pritisak + i * (h[i] - dubina);
      if (noviPritisak > maxPritisak) {
        // segment nije moguce potpuno potopiti pa racunamo koliko se
        // najvise moze potopiti
        int64_t deltaDubina = (maxPritisak - pritisak) / i;
        dubina += deltaDubina;
        pritisak += deltaDubina * i;
        break;
      }
      // potapamo ceo segment visine h[i]
```

```
      dubina = h[i];
      pritisak = noviPritisak;
    }
  }

  // ako je cela podmornica potopljena tako da joj je maksimalni
  // segment tik ispod povrsine, racunamo koliko se jos moze potopiti
  // dok se ne dostigne maksimalni pritisak
  dubina += (maxPritisak - pritisak) / n;

  cout << dubina << endl;
  return 0;
}
```

Бинарна претрага

За дату дубину d можемо једноставно израчунати притисак и видети да ли он превазилази задату границу. Повећањем дубине се повећава и притисак, па је за одређивање максималне дубине могуће применити бинарну претрагу (тзв. бинарну претрагу по решењу). Тражимо највећу вредност дубине за коју је притисак мањи од дате границе. Ако је дубина 0, притисак је 0, па је услов сигурно испуњен и вредност 0 можемо узети за иницијалну леву границу бинарне претраге. Означимо са h_{max} максималну висину сегмента подморнице. Ако је подморница на дубини d њен притисак је сигурно већи или једнак од $n \cdot (d - h_{max})$. Зато је притисак сигурно изнад границе ако је $(d - h_{max}) \cdot n > p_{max}$ тј. ако је $d > h_{max} + \frac{p_{max}}{n}$. Стога иницијалну вредност десне границе у бинарној претрази можемо поставити на вредност $h_{max} + \lfloor \frac{p_{max}}{n} \rfloor + 1$.

Бинарном претрагом тражимо највећу дозвољену дубину на следећи начин: проверићемо дубину која је на средини између леве и десне границе. Уколико је могуће спустити подморницу на ту дубину, померамо леву границу претраге на нађену дубину + 1. Уколико није могуће спустити подморницу на ту дубину, онда померамо десну границу на дубина - 1. Након овога понављамо поступак док се лева и десна граница не мимоиђу.

Остаје нам још само да за задату дубину проверимо да ли је могуће подморницу спустити дотле. То можемо урадити тако што прођемо кроз низ висина свих сегмената и рачунамо притисак на сваки сегмент појединачно и на крају одредимо укупан притисак као збир. Уколико је укупан притисак мањи или једнак највећем дозвољеном притиску, онда је могуће спустити подморницу на ту дубину, иначе није могуће.

Пошто се за фиксирану вредност дубине провера притиска може извршити у времену $O(n)$, укупна сложеност је $O(n \log (\frac{p_{max}}{n} + h_{max}))$. Како ни p_{max} ни h_{max} не могу бити већи од 10^9, тај логаритам је највише око 30, а величина низа је највише 10^5, па ово решење успешно завршава посао у оквиру задатог временског ограничења.

```
#include <iostream>
#include <vector>
#include <limits>
#include <cmath>
#include <algorithm>
using namespace std;

// provera da li se podmornica moze potopiti do date dubine tako da izdrzi dati
```

```cpp
// maksimalni pritisak
bool pritisakOK(const vector<int>& h, int dubina, int pMax) {
  long long pritisak = 0;
  for (int hh : h){
    pritisak += max(dubina - hh, 0);
    if (pritisak > pMax)
      return false;
  }
  return true;
}

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0);

  int n, pMax;
  cin >> n >> pMax;
  vector<int> h(n);
  for (int i = 0; i < n; i++)
    cin >> h[i];

  // binarna pretraga optimalnog resenja
  int lg = 0;
  int dg = *max_element(h.begin(), h.end()) + pMax / n + 1;
  while (lg <= dg) {
    int dubina = lg + (dg - lg) / 2;
    if (pritisakOK(h, dubina, pMax))
      lg = dubina + 1;
    else
      dg = dubina - 1;
  }
  cout << lg - 1 << endl;
  return 0;
}
```

Задатак: Благо у пећини

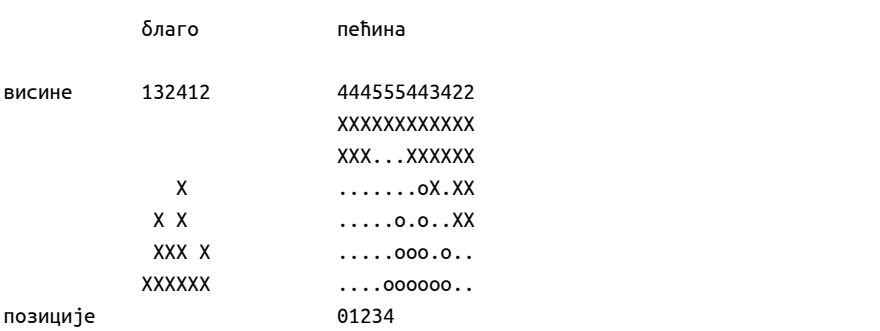
Аутор: Милан Вутделија

Такмичење: 2021/22. квалификације 3, VII и VIII разред, 6. задатак

Капетан Кукица је сакупио велико благо, које се састоји од n предмета једнаких ширина и различитих висина. Предмети морају да остану у датом редоследу, јер Кукица иначе не би могао да се сети колико је који предмет вредан. Сада Кукица жели да сакрије благо што даље у дубокој и узаној пећини, али у томе му сметају сталактити, који висе са сваког дела пећинског свода. Ширина сваког од m сталактита једнака је ширини једног предмета.

Кукица зна висине свих n предмета и висине простора испод свих m сталактита. Пошто не жели да се заглави са делимично сакривеним благом, Кукицу интересује докле би стигао ако почне да гура низ предмета у пећину. Прецизније, ако позицију првог сталактита означи са 0, дтугог са 1

итд, Кукица жели да зна која је највећа позиција, до које може да стигне почетак низа предмета. На пример, ако су висине предмета редом `1 3 2 4 1 2`, а висине свода у пећини редом `4 4 4 5 5 5 4 4 3 4 2 2`, онда благо може да се унесе најдаље до позиције 4 (положај означен кружићима).



Опис улаза

У првом реду стандардног улаза налази се природан број n, такав да $1 \leq n \leq 50000$. У другом реду се налази n природних бројева, висине предмета редом. Ови бројеви нису већи 10^9. У трећем реду се налази природан број m, такав да $n \leq m \leq 50000$. У четвртом реду се налази m природних бројева, висине простора у пећини редом. Ови бројеви нису већи 10^9.

Опис излаза

На стандардни излаз исписати највећу позицију на којој може да се нађе први предмет. Овај број може да буде и негативан, ако није могуће да се комплетно благо смести у пећину.

Пример 1

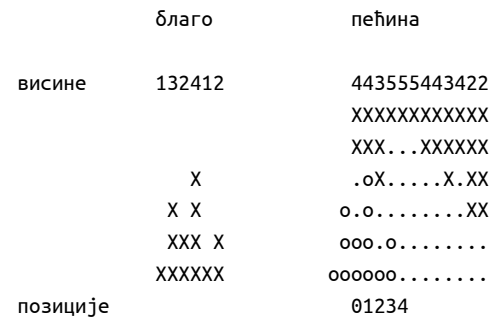
Улаз	*Излаз*

`6`
`1 3 2 4 1 2`
`12`
`4 4 4 5 5 5 4 4 3 4 2 2`
	`4`

Пример 2

Улаз	*Излаз*	*Објашњење*

`6`
`1 3 2 4 1 2`
`12`
`4 4 3 5 5 5 4 4 3 4 2 2`
	`-2`	Резултат -2 значи да почетак блага остаје два корака испред улаза у пећину. Шематски приказ те ситуације изгледа овако:



Решење

Анализом примера можемо да приметимо да од висине свода једнаке 4, која долази након висине једнаке 3 нема користи, јер је сваки предмет који стигне до те висине 4 пре тога морао да прође испод свода висине 3. Према томе, решење се неће променити ако уместо висина свода 3, 4 (тим редом) имамо висине 3, 3. Када уопштимо ово размишљање, закључујемо да сваку висину свода која је виша од било које претходне, можемо да смањимо на најмању од свих висина које јој претходе. Ово може да се уради у једном пролазу кроз низ висина сводова.

Слично претходном, висину сваког предмета који је нижи од било ког наредног, можемо да повећамо на висину највишег од свих предмета који следе у низу (тј. раније улазе у пећину, па су већ прошли испод свода под којим остаје текући предмет). Ова трансфомација може да се уради у једном пролазу (сдесна налево) кроз низ висина предмета.

Након ових трансформација улазних података које не утичу на решење, оба низа су монотоно нерастућа.

Сада можемо да тражимо најдубље место у пећини за сваки предмет, почевши од последњег предмета и последњег места у пећини. Када нађемо довољно високо место на коме последњи предмет може да стоји, настављамо да тражимо место за претпоследњи предмет, крећући се и даље ка почетку пећине. Овај поступак настављамо док не сместимо све предмете или док не дођемо до улаза у пећину. У сваком од ова два случаја се лако одређује позиција почетка низа предмета након што сваки предмет нађе своје најдаље место.

```cpp
#include <iostream>
#include <vector>

using namespace std;


int main() {
  int n;
  cin >> n;
  vector<int> predmeti(n);
  for (int i = 0; i < n; i++)
    cin >> predmeti[i];

  int m;
  cin >> m;
  vector<int> svod(m);
  for (int i = 0; i < m; i++)
    cin >> svod[i];

  for (int i = n - 2; i >= 0; i--)
    predmeti[i] = max(predmeti[i], predmeti[i + 1]);
  for (int i = 1; i < m; i++)
    svod[i] = min(svod[i], svod[i - 1]);

  int pr = n - 1, sv = m - 1;
  for (sv = m - 1; sv >= 0; sv--) {
    if (predmeti[pr] <= svod[sv])
```

```
      pr--;

    if (pr < 0)
      break;
  }
  if (pr < 0)
    cout << sv << endl;
  else
    cout << -pr - 1 << endl;

  return 0;
}
```

Решење бинарном претрагом по решењу

Након што су улазни низови трансформисани у нерастуће, важе следећа тврђења:

- благо може да се угура у пећину до позиције x, ако и само ако може да стоји на позицији x;
- ако благо може да стоји на позицији x, онда може да стоји и на било којој мањој позицији;
- ако благо не може да стоји на позицији x, онда не може да стоји ни на било којој већој позицији;

Ово нам омогућава да тражену максималну позицију блага одредимо бинарном претрагом по решењу. Благо увек може да се остави на позицији $-n$, што би значило да ни један предмет није унет у пећину (предмет са највећим индексом је пред самом пећином). Такође знамо да благо није могуће унети до позиције $m+1-n$, јер пећина није толико дубока. Уведимо ознаке $levo = -n$, $desno = m + 1 - n$. Овим смо формирали један интервал $[levo, desno]$, такав да је лева граница могућ положај блага, а десна немогућ. Сада је још потребно да бинарном претрагом сужавамо овај интервал чувајући особине граница, све док *levo* и *desno* не постану два суседна броја. Тада ће вредност *levo* бити највећа за коју је положај блага могућ, а то је управо тражена вредност.

```
#include <iostream>
#include <vector>

using namespace std;

vector<int> predmeti, svod;
int n, m;

bool MozeDaStane(int poz) {
  int iPoc = max(0, -poz);
  int iKraj = min(n, m - poz);
  for (int i = iPoc; i < iKraj; i++)
    if (predmeti[i] > svod[i + poz])
      return false;

  return true;
}
```

```
int main() {
  cin >> n;
  predmeti.resize(n);
  for (int i = 0; i < n; i++)
    cin >> predmeti[i];

  cin >> m;
  svod.resize(m);
  for (int i = 0; i < m; i++)
    cin >> svod[i];

  for (int i = n - 2; i >= 0; i--)
    predmeti[i] = max(predmeti[i], predmeti[i + 1]);
  for (int i = 1; i < m; i++)
    svod[i] = min(svod[i], svod[i - 1]);

  int levo = -n, desno = m + 1 - n;
  while (levo < desno - 1) {
    int sredina = levo + (desno - levo) / 2;
    if (MozeDaStane(sredina))
      levo = sredina;
    else
      desno = sredina;
  }
  cout << levo << endl;

  return 0;
}
```

Глава 12

Рекурзија и бектрекинг

Задатак: Абацаба

Аутор: Милан Вугделија, *Аутор решења*: Филип Марић

Такмичење: Ревијално такмичење 2019/2020., V и VI разред, 4. задатак

Низ слова $ABACABADABACABAEABACABADABACABAFABACA...$ се може формирати на следећи начин:

1. Низ је на почетку празан.
2. На низ се допише прво велико слово енглеског алфабета које се не појављује у формираном делу низа, а иза тог слова се понове сва слова која су се појавила пре њега.
3. Корак 2 се понови потребан број пута

Тако после прве примене корака 2 добијамо низ A, после друге низ ABA, после треће низ $ABACABA$ итд.

Одредити слово које се појављује на n-том месту у низу, бројећи места од 1. Редослед слова у енглеском алфабету је ABCDEFGHIJKLMNOPQRSTUVWXYZ.

Опис улаза

Један природан број мањи од $67\,108\,864$.

Опис излаза

Једно велико слово енглеског алфабета.

Пример 1		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`8`	`D`	`65`	`A`	`100`	`C`

Решење

Решење грубом силом је да се у меморији креира цео низ карактера и да се затим прочита карактер са одговарајуће позиције. Ово решење троши превише меморије, а и времена док се дугачак низ карактера не изгради.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  unsigned n;
  cin >> n;
  string s = "";
  char slovo = 'A';
  while (s.size() <= n - 1) {
    s = s + slovo + s;
    slovo++;
  }
  cout << s[n-1] << endl;
}
```

Задатак се може решити без креирања дугачке ниске карактера, ако пажљиво проучимо правилност по којој се слова појављују. Прво слово `A` се налази на месту 1, прво слово `B` на месту `2`, прво слово `C` на месту `4`, прво слово `D` на месту `8` итд., па се може наслутити да се прва појављивања слова налазе на местима која су степени броја 2.

Заиста, ако дужину низа карактера пре уметања новог слова абецеде обележимо са d_k, важи да је $d_0 = 0$ (пре уметања слова `A` не налази се ниједан карактер) и да је $d_{k+1} = 2d_k + 1$ (пре уметања новог слова налази се ниска која је добијена тако што је претходно слово спојено са два појављивања ниске која се појављивала пре тог претходног слова). Зато је $d_k = 2^k - 1$ (важи да је $2^0 - 1 = 0$ и да је $2^{k+1} - 1 = 2 \cdot (2^k - 1) + 1$), док је прва позиција слова k (ако се слова броје од нуле) једнака 2^k.

Дакле, ако је унети број n неки степен двојке $n = 2^k$, онда је у питању место на ком се слово први пут појављује и важи да је $k = \log_2 n$ ($\log_2 n$ означава број k такав да је $2^k = n$, нпр. $\log_2 32 = 5$, јер је $2^5 = 32$). Знајући k слово лако можемо одредити сабирајући k са ASCII/UNICODE кодом слова `A`.

У супротном проблем можемо свести на проблем мање димензије. Нека је $2^k < n < 2^{k+1}$, тј. нека је k највећи степен двојке мањи од n. Карактер који тражимо налази се, дакле, иза позиције 2^k у делу низа који је добијен копирањем дела низа испред позиције 2^k. Зато је n-ти карактер у целом низу једнак $(n - 2^k)$-том карактеру у копираном делу, а пошто део који се копира почиње на почетку низа, једнак је $(n - 2^k)$-том карактеру у целом низу.

Овим смо описали рекурзивни поступак којим ефикасно можемо доћи до решења.

$$f(n) = \begin{cases} \log_2 n, & n = 2^k \\ f(n - 2^k), & 2^k < n < 2^{k+1} \end{cases}.$$

На пример, $f(23) = f(23 - 16) = f(7) = f(7 - 4) = f(3) = f(3 - 2) = f(1) = 0$, па се на месту 23 налази слово `A`. Слично, на пример, важи да је $f(40) = f(40 - 32) = f(8) = 3$, па се на месту 40 налази слово `D`.

На основу претходне дефиниције би се могла имплементирати рекурзивна функција (за то би било потребно испитати да ли је дати број степен броја 2, наћи логаритам таквог броја, и наћи

највећи степен броја 2 од ког је дати број већи или једнак). Ипак, за тим нема потребе. Анализирајући рад такве рекурзивне функције можемо унапред закључити шта ће њен резултат бити, без потребе за њеним извршавањем. Претпоставимо да знамо бинарни запис броја n тј. да знамо да се број n представља као збир неких степенова двојке $2^{s_m} + 2^{s_{m-1}} + ... + 2^{s_0}$. Током рекурзије од броја ће се одузимати један по један степен двојке, све док не остане само 2^{s_0} и тада ћемо знати да је $k = s_0$. Дакле важи да је резултат једнак најмањем степену двојке који учествује разлагању броја на збир степенова двојке тј. да је тражени број k једнак позицији најдешње јединице у бинарном запису броја (претпостављамо да се позиције броје од 0, здесно). До траженог резултата је могуће доћи дељењем броја са 2_0^s, тј. узастопним дељењем броја са 2 све док последњи сабирак не постане 1, тј. све док је број паран (то ће се догодити тачно s_0 пута).

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  int k = 0;
  while (n % 2 == 0) {
    n /= 2;
    k++;
  }
  cout << (char)('A' + k) << endl;
  return 0;
}
```

Сличан резултат можемо добити и баратајући директно са интерним записом броја у рачунару, коришћењем битовских оператора. Позицију крајње десне јединице једноставно можемо израчунати шифтовањем (померањем) бинарног записа броја удесно за једно место све док последња цифра не постане једнака 1 (последњу цифру можемо испитати битовском конјункцијом са 1).

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  int k = 0;
  while ((n & 1) == 0) {
    n >>= 1;
    k++;
  }
  cout << "ABCDEFGHIJKLMNOPQRSTUVWXYZ"[k] << endl;
  return 0;
}
```

Савремени хардвер често поседује и инструкцију *count trailing zeroes* којом се одређује број

нула иза последње јединице у бинарном запису (што је баш позиција последње јединице). Иако програсмки језици обично не стандардизују приступ овој операцији, неки компилатори нуде подршку за то (на пример, ако се користи GCC, може се употребити функција `__builtin_ctz`, а ако се користи Microsoft Visual C++, може се употребити функција `_BitScanReverse`).

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  cout << (char)('A' + __builtin_ctz(n)) << endl;
  return 0;
}
```

Задатак: Не садрже цифру 3

Аутор: Филип Марић

Такмичење: Ревијално такмичење 2019/2020., VII и VIII разред, 4. задатак

Напиши програм који одређује колико природних бројева из интервала $[0, n]$ не садрже цифру 3 у свом декадном запису.

Опис улаза

Прва линија стандарног улаза садржи природан број n ($n \leq 2 \cdot 10^9$).

Опис излаза

У првој линији стандардног излаза приказати тражени резултат.

Пример 1

Улаз	*Излаз*	*Објашњење*
15	14	У интервалу $[0, 15]$ постоји 16 бројева, а бројеви 3 и 13 једини садрже цифру 3.

Пример 2

Улаз	*Излаз*
999	729

Пример 3

Улаз	*Излаз*
12345	8262

Решење

Бројање бројева који не садрже цифру 3

Задатак можемо решити тако што за сваки број од 0 до n проверимо да ли садржи цифру 3, и ако не садржи увећамо бројач бројева који не садрже цифру 3 (тај бројач у почетку иницијализујемо на нулу). У том решењу примењује се алгоритам одређивања броја елемената серије који задовољавају дати услов, тј. алгоритам бројања филтриране серије. Проверу да ли број садржи цифру 3 можемо реализовати у посебној функцији.

```
#include <iostream>
```

```
using namespace std;

bool sadrziCifru3(int n) {
  do {
    if (n % 10 == 3)
      return true;
    n /= 10;
  } while (n > 0);
  return false;
}

int main() {
  int n, br = 0;
  cin >> n;
  for (int i = 0; i <= n; i++)
    if (!sadrziCifru3(i))
      br++;
  cout << br << endl;
  return 0;
}
```

Ефикасније израчунавање броја бројева

Задатак можемо решити и на много ефикаснији начин (али је решење у том случају доста комплексније). Нека $f(a, b)$ означава број бројева из интервала $[a, b]$ који у свом декадном запису не садрже цифру 3, а $f_0(n)$ број таквих бројева из интервала $[0, n]$. Размотримо пример у коме желимо да израчунамо вредност $f_0(4251)$. Све бројеве из интервала $[0, 4251]$ можемо поделити у неколико група тј. подинтервала. Важи да је

$$[0, 4251] = [0, 999] \cup [1000, 1999] \cup [2000, 2999] \cup [3000, 3999] \cup [4000, 4251].$$

Зато је

$$f_0(4251) = f(0, 999) + f(1000, 1999) + f(2000, 2999) + f(3000, 3999) + f(4000, 4251).$$

Важи да је $f(0, 999) = f_0(999)$. Такође, важи и да је $f(1000, 1999)$ једнак броју $f(0, 999)$ тј. $f_0(999)$. Заиста, између интервала $[0, 999]$ и $[1000, 1999]$ може се успоставити бијекција таква да слика у свом декадном запису садржи цифру 3 ако и само ако њен оригинал у свом декадном запису садржи цифру 3. Слично, важи и да је $f(2000, 2999) = f_0(999)$, док је $f(3000, 3999) = 0$ јер сви бројеви у интервалу $[3000, 3999]$ садрже цифру 3. На крају, важи и да је $f(4000, 4251)$ једнако $f(0, 251)$ тј. $f_0(251)$. Зато је

$$f_0(4251) = 3 \cdot f_0(999) + f_0(251).$$

Дакле, ако знамо бројеве $f_0(999)$ и $f_0(251)$ тада можемо израчунати и број $f_0(4251)$.

Иста техника се може применити и на израчунавање бројева $f_0(999)$ и $f_0(251)$.

Важи да је

$$[0, 999] = [0, 99] \cup [100, 199] \cup ... \cup [900, 999],$$

па је

$$f_0(999) = f(0, 99) + f(100, 199) + ... + f(900, 999).$$

Важи да је $f(0, 99) = f(100, 199) = f(200, 299) = f(400, 499) = ... = f(900, 999) = f_0(99)$, а да је $f(300, 399) = 0$. Стога је

$$f_0(999) = 8 \cdot f_0(99) + f_0(99) = 9 \cdot f_0(99).$$

Слично, важи да је

$$f_0(99) = 9 \cdot f_0(9),$$

док је $f_0(9) = 9$ (јер у интервалу $[0, 9]$ који има 10 елемената, једино елемент 3 садржи цифру 3).

Важи и да је

$$[0, 251] = [0, 99] \cup [100, 199] \cup [200, 251],$$

па је

$$f_0(251) = 2 \cdot f_0(99) + f_0(51).$$

Пошто је

$$[0, 51] = [0, 9] \cup [10, 19] \cup [20, 29] \cup [30, 39] \cup [40, 49] \cup [50, 51],$$

важи да је

$$f_0(51) = 4 \cdot f_0(9) + f_0(1).$$

Важи и да је $f_0(1) = 2$ јер су оба елемента интервала $[0, 1]$ такви да не садрже цифру 3.

Дакле, $f_0(4251) = 3 \cdot f_0(999) + f_0(251) = 3 \cdot (9 \cdot f_0(99)) + 2 \cdot f_0(99) + f_0(51) = 3 \cdot (9 \cdot (9 \cdot f_0(9))) + 2 \cdot (9 \cdot f_0(9)) + 4 \cdot f_0(9) + f_0(1) = 3 \cdot 9 \cdot 9 \cdot 9 + 2 \cdot 9 \cdot 9 + 4 \cdot 9 + 2 = 2387$.

Функцију f_0 је могуће рекурзивно дефинисати. Ако се број n разлаже на почетну цифру c и суфикс n' тада се $f_0(n)$ може израчунати на следећи начин, у зависности од цифре c (претпостављамо да број $9 ... 9$ има онолико деветки колико цифара има број n').

- Ако је $c < 3$ тада је $f_0(n) = c \cdot f_0(9 ... 9) + f_0(n')$,
- Ако је $c = 3$ тада је $f_0(n) = c \cdot f_0(9 ... 9)$,

- Ако је $c > 3$ тада је $f_0(n) = (c-1) \cdot f_0(9 \ldots 9) + f_0(n')$.

Излаз из рекурзије може бити и само случај $f_0(0) = 1$ (0 је једини број у интервалу $[0, 0]$ и он не садржи цифру 3).

Проблем са имплементацијом овакве рекурзивне функције је то што се цифре одвајају слева надесно, што је компликованије него здесна налево, када број n лако разлажемо на $n \operatorname{div} 10$ и $n \bmod 10$. Стога пре уласка у рекурзију можемо обрнути цифре броја. Такође, за дати број n потребно је одредити одговарајући број који се састоји само од деветки. То једноставно можемо решити тако што конструишемо број који се од броја n добија заменом свих цифара цифром 9 и ако тај број прослеђујемо као други параметар рекурзије (тај број у сваком позиву садржи само деветке и то онолико деветки колико цифара има број n).

Приметимо да се од једног рекурзивног позива за број са k цифара најчешће добијају два рекурзивна позива за бројеве са $k-1$ цифара, што указује на то да је сложеност експоненцијална (за основу 2), што је допустиво, јер је број цифара мали. Ипак, с обзиром на то да се исти рекурзивни позиви преклапају (пре свега они облика $f_0(9 \ldots 9)$), имплементација се може убрзати динамичким програмирањем или тако што се се примети да је да је $f_0(\underbrace{9 \ldots 9}_{k}) = 9^k$, па би се ови рекурзивни позиви могли специјализовати.

```
#include <iostream>

using namespace std;

int f(int n, int d) {
  if (n == 0)
    return 1;
  int c = n % 10;
  if (c < 3)
    return c * f(d / 10, d / 10) + f(n / 10, d / 10);
  else if (c == 3)
    return c * f(d / 10, d / 10);
  else
    return (c - 1) * f(d / 10, d / 10) + f(n / 10, d / 10);
}


int f(int n) {
  int nObratno = 0;
  int devetke = 0;
  do {
    nObratno = nObratno * 10 + n % 10;
    devetke = devetke * 10 + 9;
    n /= 10;
  } while (n > 0);
  return f(nObratno, devetke);
}

int main() {
  int n;
```

```
  cin >> n;
  cout << f(n) << endl;
  return 0;
}
```

Уместо рекурзије која ради одозго наниже, решење можемо синтетизовати одоздо навише.

Обрађиваћемо једну по једну цифру броја n, здесна налево и мало по мало ћемо проширивати обрађени суфикс n' broja n. Уведимо променљиву, нпр. br, која током итерације чува вредности $f_0(n')$ за суфиксе n' броја n које током итерације обрађујемо, и променљиву, нпр. t, која чува вредности бројева $f_0(9_0 \ldots 9) = 9^k$, за бројеве $9 \ldots 9$ који имају k цифара 9, колико укупно цифара има у тренуном суфиксу n'.

Променљиве t и br иницијализујемо на 1 (претпостављамо да је пре петље обрађен празан суфикс који одговара броју 0 и да је $9^0 = 1$). Затим у петљи обрађујемо цифру по цифру броја n, кренувши од цифре јединица. Променљиву br ажурирамо на следећи начин, у зависности од текуће цифре c.

- Ако је $c < 3$ тада је $br = c \cdot t + br$,
- Ако је $c = 3$ тада је $br = c \cdot t$,
- Ако је $c > 3$ тада је $br = (c - 1) \cdot t + br$.

Променљиву t ажурирамо на вредност $9 \cdot t$.

Размотримо поново пример израчунавања $f_0(4251)$ и применимо претходно описани алгоритам на број 4251.

Претпоставимо да на почетку променљиве t и br имају вредност 1.

Прво обрађујемо последњу (прву с десна) цифру броја n, а то је цифра 1. Пошто је она мања од 3, ажурирамо br на вредност $c \cdot t + br = 1 \cdot 1 + 1 = 2$, а вредност t на вредност $9 \cdot t = 9$. Након овога, променљива br има вредност $f_0(1)$, а променљива t има вредност $f_0(9)$.

Наредна цифра је 5 и пошто је она већа од 3, ажурирамо вредност br на $(c - 1) \cdot t + br = 4 \cdot 9 + 2 = 38$, а вредност t на редност $9 \cdot t = 81$. Након овога, променљива br има вредност $f_0(51)$, а променљива t има вредност $f_0(99)$.

Наредна цифра је 2 и пошто је она мања од 3, ажурирамо вредност br на $c \cdot t + br = 2 \cdot 81 + 38 = 200$, а вредност t на редност $9 \cdot t = 9 \cdot 81 = 729$. Након овога, променљива br има вредност $f_0(251)$, а променљива t има вредност $f_0(999)$.

На крају, почетна цифра је 4 и пошто је она већа од 4, ажурирамо вредност br на $(c-1) \cdot t + br = 3 \cdot 729 + 200 = 2387$. Вредност t се ажурира на $9 \cdot 729 = 6561$, али се та вредност даље не користи. Након овога, променљива br има вредност $f_0(4251)$, а променљива t има вредност $f_0(9999)$.

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  int t = 1, br = 1;
  while (n > 0) {
```

```
    int c = n % 10;
    if (c < 3)
      br = c*t + br;
    else if (c == 3)
      br = c*t;
    else
      br = (c-1)*t + br;
    t = 9*t;
    n /= 10;
  }
  cout << br << endl;
  return 0;
}
```

Задатак: Боје у кругу

Аутор: Иван Дрецун

Такмичење: 2021/22. квалификације 1, VII разред 6. задатак

Перица и другари седе у кругу и смишљају коју ће игру следеће да играју. Договорили су се да ће се поделити у тачно три екипе (плаву, зелену и, наравно, црвену), али тако да деца која седе једна до другог морају бити у различитим екипама. Перицу и другаре занимају сви начини да направе поделу по екипама. Пошто још увек нису смислили коју ће игру следеће играти, замолили су тебе да им помогнеш и напишеш програм који ће исписати све поделе по екипама.

Опис улаза

Са стандардног улаза се уноси цео број n ($2 \leq n \leq 16$) који означава са колико другара се Перица игра.

Опис излаза

На стандардни излаз исписати онолико редова колико постоји могућих подела. Сваки ред треба да се састоји из тачно $n + 1$ карактера који редом означавају Перичину екипу и затим екипу сваког следећег друга удесно. Плаву екипу представити малим словом `p`, зелену малим словом `z` и црвену малим словом `c`. Редове исписати уређене по абецедном редоследу.

Пример

Улаз	*Излаз*
2	cpz
	czp
	pcz
	pzc
	zcp
	zpc

Решење

Приметимо да је потребно исписати сва бојења дужине $n + 1$ која испуњавају услове.

Задатак је могуће решити формирањем свих распореда боја и исписивањем само оних који испуњавају услове задатка (појављују се све три боје, сваке две суседне се разликују). Распо-

реди се могу формирати рекурзивном претрагом - прво формирањем свих распореда бојења који почињу словом c, затим оних који почињу словом p и коначно оних који почињу словом z. Сложеност оваквог решења је $O(n3^n)$ и не осваја максималан број поена.

Брже решење је могуће постићи уколико формирамо само она бојења која немају две суседне једнаке боје. Можемо модификовати рекурзивну претрагу претходног решења тако да уместо генерисања бојења која почињу сваком бојом генеришемо само бојења која почињу бојом различитом од претходне. На пример, уколико је у претходном кораку претраге фиксирана боја p, довољно је проверити сва бојења којима је следећа боја c или z. Овим се сложеност смањује на $O(n2^n)$ што је довољно за максималан број поена.

```
#include <iostream>

using namespace std;

void generisiBojenja(int i, string& boje, bool sveBoje) {
  if (i == boje.size()) {
    if (sveBoje && boje[i - 1] != boje[0])
      cout << boje << '\n';
    return;
  }

  for (auto boja : {'c', 'p', 'z'})
    if (boje[i - 1] != boja) {
      boje[i] = boja;
      generisiBojenja(i + 1, boje,
                      sveBoje || (boja != boje[0] && boja != boje[1]));
    }
}

void generisiBojenja(int n) {
  string boje(n, ' ');
  for (auto boja : {'c', 'p', 'z'}) {
    boje[0] = boja;
    generisiBojenja(1, boje, false);
  }
}

int main() {
  int n;
  cin >> n;

  generisiBojenja(n + 1);

  return 0;
}
```

Задатак: Шпанска серија

Аутор: Иван Дрецун

Такмичење: 2021/22. квалификације 1, VIII разред 6. задатак

Писци шпанске серије “Игра пресолца” одлучили су да у последњој епизоди направе велико окупљање на које ће бити позвани неки од n ликова из серије. Наравно, како то обично бива у шпанским серијама, постоји списак тројки ликова који се међусобно не подносе. Задатак за писце је да одаберу које од n ликова треба позвати тако да из сваке тројке са списка **тачно једна** особа буде позвана. Напиши програм који рачуна колико највише људи може бити позвано поштујући задате услове и исписује један такав списак званица.

Опис улаза

У првом реду стандардног улаза се уносе цели бројеви n ($3 \leq n \leq 14$) и k ($1 \leq k \leq 140$) који редом представљају број ликова у серији и број тројки ликова који се међусобно неподносе. У наредних k редова уносе се по три различита броја који означавају три лика од којих тачно један треба бити позван. Ликови су означени редним бројевима између 1 и n.

Опис излаза

У првом реду стандардног излаза исписати број позваних људи. У другом реду исписати редне бројеве позваних људи одвојене размаком. Уколико постоји више решења, исписати било које. Уколико не постоји списак званица који задовољава све услове, у првом реду стандардног излаза исписати 0.

Пример 1

Улаз	*Излаз*
6 2 1 2 3 1 4 5	3 2 4 6

Пример 2

Улаз	*Излаз*
4 4 1 2 3 1 2 4 1 3 4 2 3 4	0

Решење

Потребно је испробати све подскупове ликова и за сваки подскуп проверити да ли испуњава задати низ услова, односно да ли се из сваке тројке појављује тачно једна особа у подскупу. Приликом претраге подскупова потребно је памтити највећи до сада пронађен подскуп који испуњава услове - по завршетку испитивања свих подскупова тај ће бити решење.

Подскупове је могуће генерисати рекурзивним поступком: испитивањем прво свих подскупова који не садрже једног лика, а затим испитивањем свих подскупова који га садрже. Подскупове је могуће генерисати и употребом алгоритма за одређивање наредног подскупа.

Сложеност поскупка је $O(k2^n)$, што је у реду за дата ограничења променљивих.

```
#include <iostream>
#include <vector>

using namespace std;

struct trojka { int a, b, c; };
```

```
int max_velicina = 0;
vector<bool> max_podskup;

bool ispunjavaUslove(vector<bool>& podskup, vector<trojka>& uslovi) {
  for (auto t : uslovi) {
    int broj_ljudi = 0;
    broj_ljudi += podskup[t.a - 1];
    broj_ljudi += podskup[t.b - 1];
    broj_ljudi += podskup[t.c - 1];
    if (broj_ljudi != 1)
      return false;
  }
  return true;
}

void sviPodskupovi(int i, vector<bool>& podskup, int velicina, vector<trojka>& uslovi) {
  if (i == podskup.size()) {
    if (velicina > max_velicina && ispunjavaUslove(podskup, uslovi)) {
      max_velicina = velicina;
      max_podskup = podskup;
    }
    return;
  }

  podskup[i] = false;
  sviPodskupovi(i + 1, podskup, velicina, uslovi);

  podskup[i] = true;
  sviPodskupovi(i + 1, podskup, velicina + 1, uslovi);
}

int main() {
  int n, k;
  cin >> n >> k;

  vector<trojka> trojke(k);
  for (int i = 0; i < k; i++)
    cin >> trojke[i].a >> trojke[i].b >> trojke[i].c;

  vector<bool> podskup(n);
  sviPodskupovi(0, podskup, 0, trojke);

  cout << max_velicina << '\n';
  if (max_velicina > 0)
    for (int i = 0; i < n; i++)
      if (max_podskup[i])
        cout << (i + 1) << ' ';
  cout << '\n';
  return 0;
```

```
}
```

Подскупове можемо представити и помоћу бинарног записа неозначених целих бројева.

```
#include <iostream>
#include <vector>

using namespace std;

struct trojka { int a, b, c; };

int brojPozvanih(unsigned pozvani) {
  return __builtin_popcount(pozvani);
}

bool pozvan(unsigned pozvani, int o) {
  return pozvani & (1u << o);
}

int main() {
  int n, k;
  cin >> n >> k;

  vector<trojka> trojke(k);
  for (int i = 0; i < k; i++)
    cin >> trojke[i].a >> trojke[i].b >> trojke[i].c;

  int max_podskup = 0;
  int max_pozvanih = 0;
  for (int podskup = 0; podskup < (1 << n); podskup++) {
    if (brojPozvanih(podskup) <= max_pozvanih)
      continue;
    bool sat = true;
    for (auto t : trojke) {
      int count = 0;
      count += (int)pozvan(podskup, t.a - 1);
      count += (int)pozvan(podskup, t.b - 1);
      count += (int)pozvan(podskup, t.c - 1);
      if (count != 1) {
        sat = false;
        break;
      }
    }
    if (sat) {
      max_podskup = podskup;
      max_pozvanih = brojPozvanih(podskup);
    }
  }

  cout << max_pozvanih << '\n';
```

```
  if (max_pozvanih > 0)
    for (int o = 1; o <= n; o++)
      if (pozvan(max_podskup, o-1))
        cout << o << ' ';

  cout << '\n';
  return 0;
}
```

Задатак: Сва кретања резултата

Аутор: Милан Вуīделија

Такмичење: државно 2022/2023, VIII разред, 4. задатак

Познато нам је да се фудбалска утакмица завршила резултатом $a : b$, али нам није познат редослед постизања голова. Написати програм, који исписује све могуће редоследе постизања голова. У случају да постоји више од 1000 различитих редоследа, исписати првих 1000.

Опис улаза

У првом и једином реду стандардног улаза су два цела неозначена броја, раздвојена једним размаком, бројеви a и b, који нису већи од 100. У примерима вредним 40% поена бар један од ова два броја ће бити мањи од 4.

Опис излаза

На стандардни излаз исписати тражене редоследе постизања голова, сваки у посебном реду (најмање 1 а највише 1000 редова). Сваки редослед постизања голова приказати као низ узастопних резулатата, који почиње са `0:0`, а завршава се коначним резултатом. Постигнуте голове у једном резултату раздвојити знаком `:`, а узастопне резултате у низу резултата једним размаком.

Сами низови резултата треба да су у лексикографском (азбучноим, абецедном) поретку. На пример, ако је $a = 3, b = 2$, низ резултата `0:0 1:0 1:1 2:1 2:2 3:2` треба исписати пре низа `0:0 1:0 2:0 2:1 2:2 3:2`, јер је `1:1` по лексикографском поретку пре `2:0`.

Пример 1

Улаз	*Излаз*
`3 2`	`0:0 0:1 0:2 1:2 2:2 3:2`
	`0:0 0:1 1:1 1:2 2:2 3:2`
	`0:0 0:1 1:1 2:1 2:2 3:2`
	`0:0 0:1 1:1 2:1 3:1 3:2`
	`0:0 1:0 1:1 1:2 2:2 3:2`
	`0:0 1:0 1:1 2:1 2:2 3:2`
	`0:0 1:0 1:1 2:1 3:1 3:2`
	`0:0 1:0 2:0 2:1 2:2 3:2`
	`0:0 1:0 2:0 2:1 3:1 3:2`
	`0:0 1:0 2:0 3:0 3:1 3:2`

Пример 2

Улаз	*Излаз*
`0 6`	`0:0 0:1 0:2 0:3 0:4 0:5 0:6`

Пример 3

Улаз	*Излаз*
`2 1`	`0:0 0:1 1:1 2:1`
	`0:0 1:0 1:1 2:1`
	`0:0 1:0 2:0 2:1`

Решење

Све могуће редоследе постизања голова можемо да добијемо помоћу рекурзивне функције. Функција добија два целобројна параметра, x и y, који представљају тренутни резултат (тренутни број погодака домаће и гостујуће екипе редом), при чему се подразумева да су у низ `rez` уписани претходни резултати, почевши од `0:0`, до резултата који претходи тренутном. Задатак ове функције је да генерише све могуће завршетеке започетог низа резултата и да испише све генерисане низове (водећи рачуна да их не испише више од 1000).

Кораци функције `f(x, y)` треба да буду следећи:

- ако је већ исписано 1000 редоследа, завршити са радом,
- додати тренутни резултат $x : y$ у низ `rez`,
- ако је $x : y$ коначан резултат, исписати садржај листе у једном реду, бројати исписани редослед и завршити са радом
- ако је $y < b$, позвати `f(x, y+1)` (ако гостујућа екипа још није постигла све своје голове, генерисати наставке у којима је први следећи гол постигла гостујућа екипа)
- ако је $x < a$, позвати `f(x+1, y)` (ако домаћа екипа још није постигла све своје голове, генерисати наставке у којима је први следећи гол постигла домаћа екипа)

```
#include <iostream>
#include <vector>

using namespace std;

vector<string> rez;
int a, b, n, br;

void f(int x, int y) {
    if (br >= 1000)
        return;

    rez[x+y] = to_string(x) + ":" + to_string(y);
    if (x+y==n) {
        br++;
        for (int i = 0; i <= n; i++)
            cout << rez[i] << (i==n ? "\n" : " ");
        return;
    }

    if (y<b) f(x, y+1);
    if (x<a) f(x+1, y);
}

int main() {
    cin >> a >> b;
    n = a+b;
    br = 0;
    rez.resize(n+1);
    f(0, 0);
    return 0;
```

```
}
```

За ограничене вредности a и b, задатак може да се реши и помоћу m угнежђених петљи, где је $m = min(a, b)$. Нека је, на пример, $m = b = 2, a \geq 2$. То значи да се, почевши од $0 : 0$, број голова домаће екипе, број x, повећава a пута, а број голова гостујуће екипе, број y се повећава само два пута. Према томе, у низу од укупно $n + 2$ гола, треба на све начине изабрати позиције за два гола гостујуће екипе, а то може да се уради помоћу две уметнуте петље.

Ово решење не решава општи случај, али доноси поене на неким тест-примерима.

```
#include <iostream>

using namespace std;

void prikazi(string golovi) {
    int x = 0, y = 0;
    cout << "0:0";
    for (char ekipa : golovi) {
        if (ekipa == 'a')
            x++;
        else
            y++;
        cout << " " << x << ":" << y;
    }
    cout << endl;
}

int main() {
    int a, b, n, br;
    cin >> a >> b;
    n = a + b;
    br = 0;
    string golovi(n, (a > b ? 'a' : 'b'));
    if (a == 0)
        prikazi(golovi);
    else if (b == 0)
        prikazi(golovi);
    else if (a == 1)
        for (int i = n-1; i >= 0 && br < 1000; i--) {
            br++;
            golovi[i] = 'a';
            prikazi(golovi);
            golovi[i] = 'b';
        }
    else if (b == 1)
        for (int i = 0; i < n && br < 1000; i++) {
            br++;
            golovi[i] = 'b';
            prikazi(golovi);
            golovi[i] = 'a';
```

```
        }
    else if (a == 2)
        for (int i = n - 1; i >= 0 && br < 1000; i--) {
            for (int j = n - 1; j > i && br < 1000; j--) {
                br++;
                golovi[i] = golovi[j] = 'a';
                prikazi(golovi);
                golovi[i] = golovi[j] = 'b';
            }
        }
    else if (b == 2)
        for (int i = 0; i < n && br < 1000; i++) {
            for (int j = i + 1; j < n && br < 1000; j++) {
                br++;
                golovi[i] = golovi[j] = 'b';
                prikazi(golovi);
                //cout << i << " " << j << endl;
                golovi[i] = golovi[j] = 'a';
            }
        }
    else if (a == 3)
        for (int i = n - 1; i >= 0 && br < 1000; i--) {
            for (int j = n - 1; j > i && br < 1000; j--) {
                for (int k = n - 1; k > j && br < 1000; k--) {
                    br++;
                    golovi[i] = golovi[j] = golovi[k] = 'a';
                    prikazi(golovi);
                    golovi[i] = golovi[j] = golovi[k] = 'b';
                }
            }
        }
    else if (b == 3)
        for (int i = 0; i < n && br < 1000; i++) {
            for (int j = i + 1; j < n && br < 1000; j++) {
                for (int k = j + 1; k < n && br < 1000; k++) {
                    br++;
                    golovi[i] = golovi[j] = golovi[k] = 'b';
                    prikazi(golovi);
                    golovi[i] = golovi[j] = golovi[k] = 'a';
                }
            }
        }
    else
        cout << "-" << endl;

    return 0;
}
```

Задатак: Највеће језеро

Аутор: Милан Вугделија

Такмичење: СИО 2021/22. 3. задатак

Написати програм који за дату мапу одређује површину највећег језера на мапи.

Мапа је задата матрицом карактера у којој сваки знак `.` (тачка) представља воду, а сваки знак `X` (велико латинично икс) копно. Површина језера је једнака броју елемената матрице од којих се језеро састоји. Језеро чини група међусобно повезаних тачака, од којих ниједна није ивична (вода уз ивице матрице се сматра океаном). Две тачке су повезане ако се од једне до друге може стићи у коначном броју корака, прелазећи у сваком кораку са једне тачке на неку суседну тачку. Две тачке су суседне ако се налазе у истој врсти и суседним колонама, или истој колони и суседним врстама матрице.

Опис улаза

У првом реду стандардног улаза се налазе два природна броја V и K, број врста (редова) и број колона матрице, раздвојени једним размаком. У наредних V редова налази се по K карактера без размака, од којих је сваки `X` или `.`.

Опис излаза

На стандардни излаз исписати један неозначен цео број, број тачака од којих се састоји највеће језеро на мапи.

Ограничења

- У тест примерима вредним 80 поена важи $1 \leq V, K \leq 200$.
- У свим тест примерима важи $1 \leq V, K \leq 1200$.

Пример 1

Улаз	*Излаз*	*Објашњење*

`4 5`
`XXXXX`
`X..XX`
`XXX.X`
`XXXXX`
	`2`	На мапи су два језера, једно величине 2, друго величине 1.

Пример 2

Улаз	*Излаз*	*Објашњење*

`4 5`
`.....`
`.XXX.`
`XXXXX`
`..X..`
	`0`	На мапи нема језера, свака тачка је део неког океана.

Пример 3

Улаз	*Излаз*
`19 40`	`48`

```
........................XXX.............
......XXXXXX..........XXXXXXXX..........
XXXXXXXXXXXXXXXXX.....XXXXXXXXXXX.......
XXXXXX.....XXXXXXX...XXXX....XXXX.......
XXX.....X.....XXX....XXXX......XXXXX....
XXXX........XXXXXXX...XXXX....XXXXX.....
XXXXXX...XXXX....XX.....XXXXXXXXXX......
XXXXXX...........XXX....XXXXXXXXX.......
XXXXXXXX.......XXXXX....................
XXXXXXXXXXXXXXXXXXXX...XXXXXXXXX.......
XXXXXXXXXXXXXXXXXXX....XX....XXX........
XXXXXXXXX.......XXXX...XXX....XXX.......
..XXXXXX.......XXXXXX....XXXXXXXX.......
....XXXX.......XXXXXX.........XX.....XX.
.....XXXXXXXXXXXXX......XXX........XXXX
XXX..................XXXXXXXXXXXXXXXXX
XXXXX..............XXXXXXXXXXX.....XXX
XXXXXXXXXXX......XXXXXXXXXXXXXXX.....X
XXXXXXXX..........XXXXXXXXXXXXXXXXX....
```

Објашњење

Од четири језера на мапи, највеће је горе лево и оно је величине 48.

Решење

Решење се добија класичном претрагом матрице у дубину или у ширину. Уобичајене варијанте претраге матрице у оваквим задацима су:

- Рекурзивна претрага у дубину
- Претрага у дубину помоћу стека
- Претрага у ширину помоћу реда

У двострукој петљи пролазимо кроз поља матрице. Када наиђемо на тачку која до тада није посећена, започињемо претрагу компоненте повезаних тачака. Претрагу вршимо тако што проверимо која од највише 4 суседна поља су непосећене тачке, означавамо те тачке као посећене и и зависно од варијанте претраге коју смо изабрали, додајемо их у стек или ред, односно рекурзивно настављамо претрагу из тих тачака. У свакој варијанти је потребно да се поља приликом посете означе као посећена, да се не бисмо враћали на њих и кретали се у круг по матрици.

У овом задатку је потребно додатно водити рачуна о још једном детаљу. За свако поље означено тачком знамо да припада или океану или неком језеру. Зато при започињању претраге сваке повезане компоненте постављамо логичку променљиву `jezero` на `true`, а ако током посећивања повезаних тачака наиђемо на тачку уз неку од ивица матрице, поставићемо променљиву `jezero` на `false`. По завршеној претрази компоненте повезаних тачака, величину компоненте узимамо у обзир за одређивање максимума само ако је та компонента језеро, тј. ако је вредност променљиве `jezero` остала `true`. Алтернативно, може да се користи логичка променљива `okean` са обрнутим вредностима.

Претрага помоћу стека

У овом решењу сваки пар координата стављамо на стек, а приликом скидања са стека проверавамо да ли су то координате поља (тј. да ли су у границама матрице) и да ли је то поље непосећена тачка. Ако јесте непосећена тачка, мењамо јој вредност у неки други знак (овде доњу црту), чиме означавамо да је посећена, бројимо је и настављамо претрагу из ње.

```
#include <iostream>
#include <vector>
#include <stack>

using namespace std;

int main() {
    int nv, nk;
    cin >> nv >> nk;
    vector<string> a(nv);
    for (int v = 0; v < nv; v++)
        cin >> a[v];

    stack<pair<int, int>> st;
    int pMax = 0;
    for (int v = 0; v < nv; v++) {
        for (int k = 0; k < nk; k++) {
            st.push(pair<int, int>(v,k));
            int p = 0;
            bool jezero = true;
            while (!st.empty()) {
                pair<int, int> polje = st.top();
                st.pop();
                int i = polje.first, j = polje.second;
                if (i >= 0 && j >= 0 && i < nv && j < nk && a[i][j] == '.') {
                    a[i][j] = '_';
                    p++;
                    if (i == 0 || j == 0 || i == nv - 1 || j == nk - 1)
                        jezero = false;
                    st.push(pair<int, int>(i - 1, j));
                    st.push(pair<int, int>(i, j - 1));
                    st.push(pair<int, int>(i + 1, j));
                    st.push(pair<int, int>(i, j + 1));
                }
            }
            if (jezero && pMax < p)
                pMax = p;
        }
    }
    cout << pMax << endl;
    return 0;
}
```

Претрага помоћу реда

Једна мала разлика у коду у односу на претходно решење је да се уместо стека користи ред, што утиче на редослед посећивања поља. Независно од тога, овде смо направили још једну промену, а то је да парове координата прво проверимо, па тек ако испуњавају услов, означавамо их и додајемо у ред.

```
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

int main() {
    int nv, nk;
    cin >> nv >> nk;
    vector<string> a(nv);
    for (int v = 0; v < nv; v++)
        cin >> a[v];

    queue<pair<int, int>> q;
    int pMax = 0;
    for (int v = 0; v < nv; v++) {
        for (int k = 0; k < nk; k++) {
            if (a[v][k] == '.') {
                a[v][k] = '_';
                q.push(pair<int, int>(v,k));
                int p = 0;
                bool jezero = true;
                while (!q.empty()) {
                    pair<int, int> polje = q.front();
                    q.pop();
                    p++;
                    int i = polje.first, j = polje.second;
                    if (i == 0 || j == 0 || i == nv - 1 || j == nk - 1)
                        jezero = false;
                    if (i > 0 && a[i - 1][j] == '.') {
                        a[i-1][j] = '_';
                        q.push(pair<int, int>(i - 1, j));
                    }
                    if (j > 0 && a[i][j - 1] == '.') {
                        a[i][j - 1] = '_';
                        q.push(pair<int, int>(i, j - 1));
                    }
                    if (i < nv - 1 && a[i + 1][j] == '.') {
                        a[i + 1][j] = '_';
                        q.push(pair<int, int>(i + 1, j));
                    }
                    if (j < nk - 1 && a[i][j + 1] == '.') {
                        a[i][j + 1] = '_';
```

```
                    q.push(pair<int, int>(i, j + 1));
                }
            }
            if (jezero && pMax < p)
                pMax = p;
        }
    }
}
cout << pMax << endl;
return 0;
}
```

Претрага помоћу симулираног реда

Ред можемо да имплементирамо и помоћу вектора, уместо да користимо баш структуру `queue` која имплементира ред. У том случају нећемо стварно да избацујемо елементе са почетка вектора (то би веома успорило програм због премештања осталих елемената), него памтимо позицију са које читамо следећи елемент (променљива `i` у програму).

Такође, уместо да користимо 4 `if` наредбе за суседна поља, можемо да направимо низове `dx` и `dy` који садрже помераје до суседних поља у смеру координатних оса. Тиме омогућавамо да поменуте 4 `if` наредбе заменимо једном `if` наредбом у `for` петљи кроз коју се пролази 4 пута.

```
// Test
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int ozn[1200][1200];
int dx[4] = { 1, -1, 0, 0 };
int dy[4] = { 0, 0, 1, -1 };
vector<string> mat;
int n, m;

int ivica(int x, int y) {
    if (x < 0 || y < 0 || x >= n || y >= m)
        return -1;
    if (x == 0 || y == 0 || x == n - 1 || y == m - 1)
        return 0;
    return 1;
}

int povrsina(int x, int y) {
    vector<pair<int, int>> q(n * m);
    bool okean = ivica(x, y) == 0;
    int i = 0, j = 1;
    q[0] = pair<int, int>(x, y);
    ozn[x][y] = 1;
```

```
    while (i < j) {
        for (int k = 0; k < 4; k++) {
            int xn = q[i].first + dx[k];
            int yn = q[i].second + dy[k];
            int iv = ivica(xn, yn);
            if (iv < 0 || ozn[xn][yn] == 1 || mat[xn][yn] == 'X')
                continue;
            if (iv == 0)
                okean = true;
            ozn[xn][yn] = 1;
            q[j++] = pair<int, int>(xn, yn);
        }
        i++;
    }
    if (okean)
        return 0;
    return i;
}


int main() {
    cin >> n >> m;
    mat.resize(n);
    for (int i = 0; i < n; i++)
        cin >> mat[i];
    int maxp = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (!ozn[i][j] && mat[i][j] == '.') {
                int p = povrsina(i, j);
                if (p > maxp)
                    maxp = p;
            }
        }
    }
    cout << maxp;
}
```

Рекурзивна претрага

У овом решењу уместо да користимо свој експлицитан стек, ослањамо се на системски стек који користи рекурзија приликом позива у дубину.

```
// Test
#include <iostream>
#include <string>
#include <vector>

using namespace std;
```

```
int ozn[1200][1200];
int dx[4] = {1, -1, 0, 0};
int dy[4] = {0, 0, 1, -1};

int ivica(int x, int y, int n, int m){
    if (x < 0 || y < 0 || x >= n || y >= m)
        return - 1;
    if (x == 0 || y == 0 || x == n - 1 || y == m - 1)
        return 0;
    return 1;
}

int povrsina(vector<string>& mat, int x, int y, int n, int m, bool& okean){
    int iv = ivica(x, y, n, m);
    if (iv < 0 || ozn[x][y] || mat[x][y] == 'X')
        return 0;
    ozn[x][y] = 1;
    int p = 1;
    okean |= iv == 0;
    for (int k = 0; k < 4; k++)
        p += povrsina(mat, x + dx[k], y + dy[k], n, m, okean);
    return p;
}


int main(){
    int n, m;
    cin >> n >> m;
    vector<string> mat(n);
    for (int i = 0; i < n; i++)
        cin >> mat[i];
    int maxp = 0;
    for (int i = 0; i < n; i++){
        for (int j = 0; j < m; j++){
            if (!ozn[i][j] && mat[i][j] == '.'){
                bool okean = false;
                int p = povrsina(mat, i, j, n, m, okean);
                if (p > maxp && !okean)
                    maxp = p;
            }
        }
    }
    cout << maxp;
}
```

Глава 13

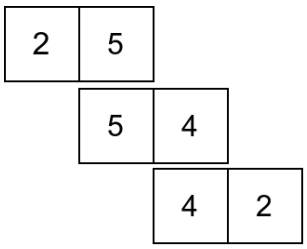
Динамичко програмирање

Задатак: Постављање домина

Аутор: Милан Вуīделија

Такмичење: 2021/22. квалификације 1, VIII разред 4. задатак

У игри домина прва домина се поставља на празан сто, а свака следећа се надовезује на један или други крај низа раније постављених домина. Да би домина могла да се надовеже на неки крај, један од њена два броја мора да буде једнак броју на том крају. Приликом надовезивања, једнаки бројеви се стављају један до другог и остају у унутрашњости низа, а преостали број на домини постаје нови крај низа. На пример, ако је на почетку игре на сто стављена домина са бројевима 5 и 4, на њу је могла да буде надовезана домина са бројевима 5 и 2, а затим и она са бројевима 4 и 2, као што је приказано. После оваквог надовезивања, на крајевима низа се налазе бројеви 2 и 2.



Постављање домина

Написати програм, који за дати низ домина исписује да ли су оне могле да буду одигране и надовезане на низ редом у коме су наведене.

Опис улаза

У првом реду стандардног улаза је цео број n ($1 \leq n \leq 100$), број домина. У сваком од следећих n редова су по два једноцифрена природна броја, који представљају једну домину. Редослед два броја који представљају једну домину није битан (свака домина може да се окрене, ако је то потребно због надовезивања).

Опис излаза

На стандардни излаз исписати само реч `da` или реч `ne`.

Пример 1

Улаз	*Излаз*
4 2 1 2 1 2 1 1 3	da

Пример 2

Улаз	*Излаз*
4 2 1 1 3 4 2 5 6	ne

Решење

Прву домину увек можемо да поставимо. Током учитавања следећих домина, треба памтити два броја, који се налазе на крајевима претходно формираног низа и на које може да се надовеже нова домина.

Код сваке нове домине, потребно је да се за сваки крај низа и сваку страну домине провери да ли се уклапају, што је укупно 4 начина да се домина постави. Ако ниједан од та 4 начина није могућ, коначан одговор је `NE`. Ако домина може да се стави само на један крај низа, тај крај низа добија вредност другог краја домине и поступак се наставља.

Остаје да се размотри ситуација када домина може да се стави на било који крај низа. На пример, на крајевима низа су бројеви 2 и 5, а и домина саджи управо бројеве 2 и 5. Домину можемо да ставимо тако, да после стављања на крајевима буду бројеви 2 и 2, или бројеви 5 и 5. Размислимо шта све може да се догоди са појављивањем следеће домине.

- Ако следећа домина нема ниједан од бројева 2 и 5, коначан одговор је `NE`.
- Ако следећа домина има само један од бројева 2 и 5, нпр. 5 и 4, претходну домину је требало ставити тако да крајеви низа буду 5 и 5, а нову тако да крајеви постану 5 и 4.
- Ако следећа домина има управо бројеве 2 и 5, све једно је како смо ставили претходну домину, јер у оба случаја после стављања нове домине на крајевима низа ће бити бројеви 2 и 5.

У сва три случаја нејднозначност из претходног потеза се одмах разрешава. Према томе, довољно је да употребимо једну логичку променљиву (у програму је то променљива `dupli`), која означава да је наступила ова привремена нејднозначност. То значи да се од два броја које памтимо као крајеве низа, један налази на оба краја, а други ни на једном. Који је који од та два, одлучујемо већ након читања следеће домине и нејднозначност престаје.

```
#include <iostream>

using namespace std;

int main() {
  int n, x, y, a, b;
  cin >> n >> x >> y;
  bool dupli = false, moze = true;
  for (int i = 1; i < n; i++) {
    cin >> a >> b;
    if (a == x && b == y)
```

```
      dupli = !dupli;
    else if (a == y && b == x)
      dupli = !dupli;
    else {
      if (a == x)
        if (dupli) y = b; else x = b;
      else if (a == y)
        if (dupli) x = b; else y = b;
      else if (b == x)
        if (dupli) y = a; else x = a;
      else if (b == y)
        if (dupli) x = a; else y = a;
      else
        moze = false;

      dupli = false;
    }
  }
  if (moze)
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  return 0;
}
```

Поред овог елегантног, елементарног решења, задатак је могуће решавати и грубом силом, тј. провером свих могућих начина постављања наредне домине. Најједноставнији начин да се то имплементира је рекурзивна претрага. У сваком тренутку претраге памтимо бројеве `kraj1` и `kraj2` који означавају тренутне вредности на крајевима тренутно постављеног низа домина (можемо их иницијализовати на вредности са прве задате домине) и покушавамо да поставимо наредну домину (k-ту у низу свих домина) на све начине на које је то могуће. Излаз из рекурзије је успешан када су постављене све домине, а неуспешан ако наредну домину није могуће додати у низ.

```
#include <iostream>
#include <vector>
#include <utility>
using namespace std;

bool mogu_se_postaviti(const vector<pair<int, int>>& domine, int k, int kraj1, int kraj2) {
  if (k == domine.size())
    return true;
  if (domine[k].first == kraj1 && mogu_se_postaviti(domine, k+1, domine[k].second, kraj2))
    return true;
  if (domine[k].second == kraj1 && mogu_se_postaviti(domine, k+1, domine[k].first, kraj2))
    return true;
  if (domine[k].first == kraj2 && mogu_se_postaviti(domine, k+1, kraj1, domine[k].second))
    return true;
  if (domine[k].second == kraj2 && mogu_se_postaviti(domine, k+1, kraj1, domine[k].first))
    return true;
```

```
  return false;
}

bool mogu_se_postaviti(const vector<pair<int, int>>& domine) {
  return mogu_se_postaviti(domine, 1, domine[0].first, domine[0].second);
}

int main() {
  int n;
  cin >> n;
  vector<pair<int, int>> domine(n);
  for (int i = 0; i < n; i++)
    cin >> domine[i].first >> domine[i].second;
  cout << (mogu_se_postaviti(domine) ? "da" : "ne") << endl;
}
```

Пошто током рекурзивне претраге може доћи до понављања истих рекурзивних позива, програм је потребно убрзати неком техником динамичког програмирања (на пример, мемоизацијом).

```
#include <iostream>
#include <vector>
#include <array>
#include <utility>
#include <set>
using namespace std;

pair<int, int> sortiran_par(int a, int b) {
  return make_pair(min(a, b), max(a, b));
}

bool mogu_se_postaviti(const vector<pair<int, int>>& domine, int k,
                       int kraj1, int kraj2, vector<array<array<int, 10>, 10>>& memo) {
  if (k == domine.size()) {
    memo[k][kraj1][kraj2] = 1;
    return true;
  }

  if (memo[k][kraj1][kraj2] != -1)
    return memo[k][kraj1][kraj2] == 1;

  set<pair<int, int>> mogucnosti;
  if (domine[k].first == kraj1)
    mogucnosti.insert(sortiran_par(domine[k].second, kraj2));
  if (domine[k].second == kraj1)
    mogucnosti.insert(sortiran_par(domine[k].first, kraj2));
  if (domine[k].first == kraj2)
    mogucnosti.insert(sortiran_par(kraj1, domine[k].second));
  if (domine[k].second == kraj2)
    mogucnosti.insert(sortiran_par(kraj1, domine[k].first));
```

```
  for (auto p : mogucnosti)
    if (mogu_se_postaviti(domine, k+1, p.first, p.second, memo)) {
      memo[k][kraj1][kraj2] = 1;
      return true;
    }
  memo[k][kraj1][kraj2] = 0;
  return false;
}

bool mogu_se_postaviti(const vector<pair<int, int>>& domine) {
  int n = domine.size();
  vector<array<array<int, 10>, 10>> memo(n+1);
  for (int i = 0; i < n; i++)
    for (int j = 0; j < memo[i].size(); j++)
      memo[i][j].fill(-1);
  return mogu_se_postaviti(domine, 1, domine[0].first, domine[0].second, memo);
}

int main() {
  int n;
  cin >> n;
  vector<pair<int, int>> domine(n);
  for (int i = 0; i < n; i++)
    cin >> domine[i].first >> domine[i].second;
  cout << (mogu_se_postaviti(domine) ? "da" : "ne") << endl;
}
```

Задатак: Сечење

Аутори: Иван Дрецун, Душан Пођадић

Такмичење: 2021/22. квалификације 2, VII разред, 6. задатак

Дат је штап дужине n метара. Потребно је пресећи га на неколико места тако да добијени делови буду целобројне дужине и ниједан од добијених делова не буде дужи од k метара. Написати програм који одређује на колико начина је могуће извршити овакво сечење.

Опис улаза

Са стандардног улаза се уносе бројеви n ($1 \leq n \leq 1000$) и k ($1 \leq k \leq 300$).

Опис излаза

На стандардни излаз исписати број могућих сечења. Како тај број може бити велик, исписати његов остатак при дељењу са $10^9 + 9$.

Пример

Улаз	*Излаз*	*Објашњење*

`4 2`	`5`	Могућа су следећа сечења представљена дужинама добијених делова штапа:
		`1 1 1 1`
		`2 1 1`
		`1 2 1`
		`1 1 2`
		`2 2`

Решење

Основно решење

Обележимо број могућих сечења штапа дужине d са *broj*(d). Штап дужине нула се може исећи на један начин. Посматрајмо штап дужине $d > 0$. Ако фиксирамо дужину последњег исеченог дела на 1, остатак штапа је дужине $d - 1$ и њега можемо исећи на *broj*$(d - 1)$ начина. Наравно, дужина последњег дела не мора бити 1. Ако фиксирамо дужину последњег исеченог дела на 2, остатак штапа је дужине $d - 2$ и њега можемо исећи на *broj*$(d - 2)$ начина. Слично можемо фиксирати дужину последњег дела на 3, на 4 итд. Дужина последњег дела може бити највише k по условима задатка, па је тада дужина остатка штапа $d - k$, а број могућности за сечење је *broj*$(d - k)$. Укупан број начина да исечемо штап дужине d је збир броја могућности када је последњи део 1, броја могућности када је последњи део 2 и тако даље. Наравно, уколико је дужина d мања од k, онда се сабирање не врши до k, него само до d. Дакле можемо записати

$$broj(d) = \sum_{i=1}^{\min(k,d)} broj(d-i).$$

Директно рекурзивно решење на основу ове рекурентне формуле би било веома неефикасно, због понављања рекурзивних позива, те стога примењујемо технику динамичког програмирања. У низу *broj* ћемо чувати број сечења за сваку могућу дужину штапа. Потребно проћи кроз све вредности дужина штапа од 1 до n и на основу претходно добијених резултата које чувамо у низу лако можемо израчунати број могућности за тренутну дужину.

Треба водити рачуна да се при сваком сабирању и множењу бројева, узима остатак при дељењу бројем $10^9 + 9$ као што је сугерисано у задатку, да не би дошло до прекорачења.

Сложеност овог алгоритма је $O(n \cdot k)$ јер за сваку од дужина треба проћи кроз k (или мање за првих k дужина) претходних резултата и сабрати их. Како важи $k \leq n \leq 10^6$, ово решење се не може извршити у датим временским ограничењима (1,5 секунди) на свим тест-примерима.

```
#include <iostream>
#include <vector>

using namespace std;

constexpr int mod = 1000000009;

int main()
{
  int n, k;
```

```
  cin >> n >> k;

  vector<int> broj(n + 1);
  broj[0] = 1;
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= k && j <= i; j++)
      broj[i] = (broj[i] + broj[i - j]) % mod;

  cout << broj[n] << '\n';
  return 0;
}
```

Оптимизовано решење

Претходно решење можемо побољшати тако што користимо инкрементално рачунање збира претходних k елемената. Када рачунамо решење за $broj(d)$, ми пролазимо кроз све чланове низа од $broj(d-k)$ до $broj(d-1)$. Када рачунамо решење за $broj(d+1)$, пролазимо кроз све чланове од $broj(d-k+1)$ до $broj(d)$, односно тако израчунат збир се разликује од збира који смо рачунали за $broj(d)$ само у члановима $broj(d-k)$ и $broj(d)$. Ако ово уочимо, јасно нам је да не морамо сваки пут да пролазимо кроз свих k последњих чланова низа, већ можмо да памтимо претходни збир и да је једноставно мењамо у сваком кораку. Наравно, опет треба водити рачуна за дужине штапа које су мање од k.

Сложеност овог решења је $O(n)$ и оно доноси максималан број поена.

```
#include <iostream>
#include <vector>

using namespace std;

const int mod = 1000000009;

int main()
{
  int n, k;
  cin >> n >> k;

  vector<int> broj(n + 1);
  broj[0] = 1;
  int zbirPoslednjihK = 1;
  for (int i = 1; i <= n; i++) {
    broj[i] = zbirPoslednjihK;
    zbirPoslednjihK = (zbirPoslednjihK + broj[i]) % mod;
    if (i >= k)
      zbirPoslednjihK = (zbirPoslednjihK + mod - broj[i - k]) % mod;
  }

  cout << broj[n] << '\n';
  return 0;
}
```

Задатак: Инвестициони фонд

Аутор: Филип Марић

Такмичење: 2019/20 квалификације 3, VII и VIII разред, 6. задатак

Инвестициони фонд који се бави продајом криптовалута је објавио очекивани профит за сваки могући уложени износ у биткоинима. Никола је студент-програмер који је био веома вредан током прошле године, зарадио је N биткоина и жели да их распореди у неколико улога, тако да добије што је већи могући профит. Пошто тренутно мора да спрема испите и не може да програмира, ти му помози тако што ћеш написати програм који одређује максимални профит.

Опис улаза

Са стандардног улаза се уноси број биткоина N ($1 \leq N \leq 5000$), које Никола има на располагању а затим N позитивних природних бројева мањих од 100 раздвојених са по једним размаком који представљају очекивану добит за сваки уложени износ од 1 до N у биткоинима.

Опис излаза

На стандардни излаз исписати највећи могући износ биткоина који Никола може добити након улагања.

Пример 1

Улаз	*Излаз*	*Објашњење*
`6` `1 2 3 4 5 6`	`6`	Ако Никола уложи свих 6 биткоина имаће профит 6. Исти профит би могао да добије и за друге комбинације улагања (на пример, да два пута уложи по 3 биткоина).

Пример 2

Улаз	*Излаз*	*Објашњење*
`6` `10 21 32 41 51 61`	`64`	Ако Никола два пута уложи по 3 биткоина имаће профит 32+32 = 64.

Пример 3

Улаз	*Излаз*	*Објашњење*
`10` `1 5 8 9 10 17 17 20 24 26`	`27`	Ако Никола уложи 6 биткоина и два пута по 2 биткоина, имаће профит 17 + 5 + 5 = 27.

Решење

Задатак решавамо тако што за сваки могући новчани улог n између 1 и N израчунавамо зараду која се може добити ако се уложи баш тај улог. Добит од улога n је дата на улазу, и након тога на преостаје $N - n$ новца. Тиме смо за сваки улаз n полазни проблем свели на потпроблем истог облика, али мање димензије, који се може решити рекурзијом. Ако обележимо са $f(n)$ највећи профит који се може добити улагањем n динара, важе следеће рекурентне релације:

$$\begin{aligned} f(0) &= 0 \\ f(N) &= \max_{1 \leq n \leq N} (d(n) + f(N - n)) \end{aligned}$$

На основу овога је веома једноставно дефинисати рекурзивну функцију.

```
#include <iostream>
#include <vector>

using namespace std;

int zarada(int N, const vector<int>& dobit) {
  if (N == 0)
    return 0;
  int maks = 0;
  for (int n = 1; n <= N; n++)
    maks = max(maks, dobit[n] + zarada(N-n, dobit));
  return maks;
}

int main() {
  int N;
  cin >> N;
  vector<int> dobit(N+1);
  dobit[0] = 0;
  for (int n = 1; n <= N; n++)
    cin >> dobit[n];
  cout << zarada(N, dobit) << endl;
  return 0;
}
```

Пошто у претходном решењу долази до понављања рекурзивних позива, потребно је применити динамичко програмирање. Један начин је да се изврши мемоизација.

```
#include <iostream>
#include <vector>

using namespace std;

int zarada(int N, const vector<int>& dobit, vector<int>& memo) {
  if (memo[N] != -1)
    return memo[N];

  if (N == 0)
    return 0;
  int maks = 0;
  for (int n = 1; n <= N; n++)
    maks = max(maks, dobit[n] + zarada(N-n, dobit, memo));
  return memo[N] = maks;
}

int zarada(int N, const vector<int>& dobit) {
  vector<int> memo(N+1, -1);
  return zarada(N, dobit, memo);
}
```

```
int main() {
  int N;
  cin >> N;
  vector<int> dobit(N+1);
  dobit[0] = 0;
  for (int n = 1; n <= N; n++)
    cin >> dobit[n];
  cout << zarada(N, dobit) << endl;
  return 0;
}
```

Рекурзије се можемо ослободити и задатак можемо решити динамичким програмирањем навише.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int N;
  cin >> N;
  vector<int> dobit(N+1);
  dobit[0] = 0;
  for (int n = 1; n <= N; n++)
    cin >> dobit[n];
  vector<int> dp_zarada(N+1);
  dp_zarada[0] = 0;
  for (int n = 1; n <= N; n++) {
    dp_zarada[n] = 0;
    for (int ulog = 1; ulog <= n; ulog++) {
      int c = dobit[ulog] + dp_zarada[n-ulog];
      if (c >= dp_zarada[n])
        dp_zarada[n] = c;
    }
  }
  cout << dp_zarada[N] << endl;
  return 0;
}
```

Глава 14

Грамзиви алгоритми

Задатак: Мали поштар

Аутор: Филип Марић

Такмичење: 2019/20 квалификације 3, V и VI разред, 5. задатак и VII и VIII разред, 3. задатак

Јовица зарађује џепарац тако што доноси пакете својим комшијама. Поделу креће од своје куће и потребно је да пакете разнесе у друге куће распоређене дуж улице и да се врати назад у своју кућу. За сваку кућу познато је растојање од почетка улице. Најкраћи пут би прешао ако би пакете сложио тако да их редом дели комшијама дуж улице. Пошто Јовица жели да буде у доброј физичкој форми, он током поделе пакета трчи и жели да пакете уреди тако да пређе што већи пут. Напиши програм који одређује највећи пут који може да пређе.

Опис улаза

Са стандардног улаза се уноси број кућа у које треба донети пакете (међу њима се налази и Јовицина кућа), а затим и растојања тих кућа од почетка улице.

Опис излаза

На стандардни излаз исписати највеће растојање које Јовица може прећи током поделе пакета.

Пример 1

Улаз	*Излаз*	*Објашњење*
`5` `7 3 6 10 2`	`24`	Постоји више начина да Јовица претрчи 24 дужне јединице. На пример, ако је његова кућа на позицији 3, он може да редом обилази куће 3, 7, 2, 10, 6, 3.

Пример 2

Улаз	*Излаз*
`7` `3 5 11 4 2 17 9`	`56`

Решење

Груба сила

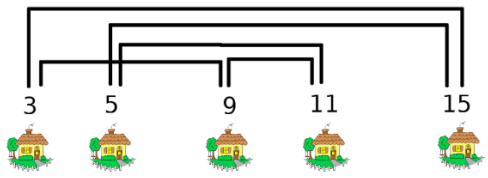
Решење грубом силом подразумева да се провере сви могући редоследи обиласка n кућа, тј. свих $n!$ пермутација датих бројева и да се утврди која од њих даје највећу могућу вредност пређеног пута.

Алгоритам заснован на провери свих пермутација је изразито неефикасан, његова сложеност је $O(n!)$ и може се применити само на веома, веома мале улазе (практично, само за $n \leq 10$ програм може да реши задатак у датом временском ограничењу).

```
int zbirApsRazlika(const vector<int>& a) {
  int zbir = 0;
  for (size_t k = 1; k < a.size(); k++)
    zbir += abs(a[k] - a[k-1]);
  zbir += abs(a[a.size()-1] - a[0]);
  return zbir;
}

int najduziPut(vector<int>& a) {
  int maks = 0;
  do {
    maks = max(zbirApsRazlika(a), maks);
  } while (next_permutation(begin(a), end(a)));
  return maks;
}
```

Грамзива стратегија

Интуитивно нам је јасно да ће се пуно претрчати ако се стално трчи са једног на други крај улице. Један грамзиви приступ је да се прво обиђе крајња лева кућа, па затим крајња десна, па друга слева, па друга здесна и тако "цик-цак".



Цик-цак обилазак: креће се из 3, затим се иде у 15, па у 5, па у 11, па у 3 и на крају назад у 3

Докажимо да је ово грамзиво решење исправно.

За дати распоред $x_0, \ldots x_{n-1}$ одређујемо суму апсолутних вредности разлика елемената тј. покушавамо да максимизујемо суму

$$|x_0 - x_1| + |x_1 - x_2| + \ldots + |x_{n-2} - x_{n-1}| + |x_{n-1} - x_0|.$$

Сваки елемент се у суми јавља тачно два пута. У зависности од међусобног односа бројева неки

елементи ће бити узети са знаком $+$, а неки са знаком $-$, и то тако да се тачно n елемената узима са знаком $+$ и тачно n елемената узима са знаком $-$. Циљ нам је да елементи који се узимају са знаком $+$ буду што већи, а да ови са знаком $-$ буду што мањи. Распоред који иде цик-цак постиже да се за знаком $+$ узме $\frac{n}{2}$ већих елемената низа, а са знаком $-$ узме $\frac{n}{2}$ мањих елемената низа (ако их је непаран број, тада се средњи узима једном са знаком $-$, а једном са знаком $+$). Заиста, ако, на пример, имамо 6 елемената $a_0 \leq a_1 \leq a_2 \leq a_3 \leq a_4 \leq a_5$, цик-цак распоред даје вредност

$$|a_0 - a_5| + |a_5 - a_1| + |a_1 - a_4| + |a_4 - a_2| + |a_2 - a_3| + |a_3 - a_0|,$$

што је једнако

$$(a_5 - a_0) + (a_5 - a_1) + (a_4 - a_1) + (a_4 - a_2) + (a_3 - a_2) + (a_3 - a_0),$$

тј.

$$2 \cdot (a_5 + a_4 + a_3) - 2 \cdot (a_2 + a_1 + a_0)$$

Јасно је да се не може добити више од овога, а ово се, видели смо, увек може експлицитно достићи баш цик-цак распоредом.

У овом примеру смо коректност грамзиве стратегије доказали тако што смо израчунали теоријско ограничење функције која се оптимизује и затимо смо доказали да се грамзивом стратегијом достиже то теоријско ограничење.

Елементе низа можемо експлицитно сортирати, па затим распоредити елементе у нови низ по цик-цак редоследу и за тај нови низ израчунати збир апсолутних вредности разлика суседних елемената.

Сложеношћу алгоритма доминира фаза сортирања, чија је сложеност $O(n \log n)$. Распоређивање елемената у помоћни низ и израчунавање дужне врши се у сложености $O(n)$.

```
int najduziPut(vector<int>& a) {
  int n = a.size();
  sort(begin(a), end(a));
  vector<int> b(n);
  int i = 0, j = n-1;
  for (int k = 0; k < n; k++)
    if (k % 2 != 0)
      b[k] = a[i++];
    else
      b[k] = a[j--];

  return zbirApsRazlika(b);
}
```

Помоћни низ се може веома једноставно избећи у имплементацији.

```
int najduziPut(vector<int>& a) {
  int n = a.size();
  sort(begin(a), end(a));
  int i = 0, j = n-1;
```

```
  int zbir = 0;
  bool paran = true;
  while (i < j) {
    zbir += abs(a[j]-a[i]);
    if (paran)
      i++;
    else
      j--;
    paran = !paran;
  }
  zbir += abs(a[0] - a[i]);
  return zbir;
}
```

Ако пажљиво размотримо доказ коректности, примећујемо да распоред у коме идемо од прве до последње, па до друге, затим претпоследње куће итд., није једини који даје максимални пређени пут. Довољно је само да наизменично узимамо елементе из прве и друге половине низа (у било ком редоследу). Ово инспирише још једноставнији алгоритам за израчунавање траженог максимума (саберемо $\frac{n}{2}$ бројева са почетка, одузмемо $\frac{n}{2}$ бројева са краја низа и помножимо са 2).

```
int najduziPut(vector<int>& a) {
  int n = a.size();
  sort(begin(a), end(a));

  int zbir = 0;
  for (int i = 0; i < n/2; i++) {
    zbir += a[n-1-i];
    zbir -= a[i];
  }
  return zbir * 2;
}
```

Задатак: Прорeђен скуп

Аутор: Иван Дрецун

Такмичење: окружно 2021/22., VIII разред, 3. задатак

Дат је скуп целих бројева дужине n. Написати програм који одређује величину највећег подскупа таквог да не садржи два елемента чија је разлика строго мања d.

Опис улаза

Са стандардног улаза се уносе бројеви n ($1 \leq n \leq 10^5$) и d ($0 \leq d \leq 10^9$), одвојени размаком. У наредном реду се уноси n различитих целих бројева одвојених размаком.

Опис излаза

На стандардни излаз исписати један број који представља величину траженог подскупа.

Пример

Улаз	*Излаз*	*Објашњење*
`7 3` `8 2 4 1 9 6 5`	`3`	Постоји више подскупова величине 3 који испуњавају тражени услов, а не постоји такав подскуп величине 4. Један такав подскуп је $\{8, 2, 5\}$, зато што се свака два његова елемента разликују за најмање 3.

Решење

Задатак решавамо тако што низ прво сортирамо, а затим прорeђен скуп иницијализујемо на први елемент низа и у сваком кораку га проширујемо најмањим елементом низа који је на растојању већем или једнаком од d у односу на највећи елемент до тада изграђеног скупа.

На пример, низ дат у примеру и сортирамо чиме добијамо низ $1, 2, 4, 5, 6, 8, 9$. Скуп иницијализујемо на $\{1\}$, а затим га проширујемо редом елементима 4 и 8, добијајући најдужи могући прорeђени скуп $\{1, 4, 8\}$.

Докажимо да се овом суштински грамзивом стратегијом добија оптимално решење. Претпоставимо да постоји неки скуп $x = \{x_0, x_1, ..., x_{m-1}\}$ максималне дужине m, а нека се нашом стратегијом добија скуп $y = \{y_0, y_1, ...\}$. Ако скуп x не садржи најмањи елемент низа a_0 (који је на основу наше стратегије једнак y_0), тада најмањи елемент x_0 тог скупа можемо заменити најмањим елементом низа y_0 и скуп ће и даље остати прорeђен. Заиста, пошто је најмањи елемент низа y_0 сигурно мањи од најмањег елемента x_0 тог скупа, након замене ће се разлика између најмања два елемента скупа $x_1 - y_0$ само повећати, а пошто је почетна разлика $x_1 - x_0$ била већа или једнака d таква ће бити разлика $x_1 - y_0$ након замене. Ако сада скуп x не садржи други елемент који би био додат нашом грамзивом стратегијом (нека је то y_1), могли бисмо извршити замену другог најмањег елемента x_1 у скупу x елементом y_1, уз очување прорeђености. Заиста, пошто би га наша стратегија додала, y_1 је сигурно већи или једнак од најмањег елемента низа y_0. Са друге стране, пошто је $x_1 - y_0 \geq d$, а y_1 је најмањи елемент низа за који важи да је $y_1 - y_0 \geq d$, важи да је $y_1 \leq x_1$. Пошто је низ x био прорeђен, важи да је разлика $x_2 - x_1 \geq d$, па је и $x_2 - y_1 \geq d$. На сличан начин можемо извршити замену једног по једног елемента низа и на тај начин од низа x добити скуп y, задржавајући све време исти број елемената и прорeђеност. Пошто је низ x био оптималан и садржао је максимални број елемената, такав мора бити и низ y.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);

  int n, d;
  cin >> n >> d;

  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];

  sort(begin(a), end(a));
```

```
  int prethodni = 0;
  int velicina  = 1;
  for (int i = 1; i < n; i++) {
    if (a[i] - a[prethodni] >= d) {
      prethodni = i;
      velicina++;
    }
  }

  cout << velicina << endl;
  return 0;
}
```

Задатак: Завршно гласање

Аутор: Небојша Варница,

Аутори решења: Милан Вугделија, Филип Марић

Такмичење: СИО 2021/22. 1. задатак

На такмичењу за најбољу песму учествује n земаља, свака са по једном песмом. Након извођења свих песама представник сваке од земаља учесница се укључује у програм и саопштава колико поена даје свакој од осталих земаља: земљи коју сматра најбољом даје $n-1$ поен, следећој $n-2$ поена и тако до краја уз услов да не може дати ниједан поен земљи чији је представник.

Гласање је при крају и имамо укупан број поена који је освојила свака од земаља. Преостала је последња земља која управо треба да саопшти своју одлуку. Ваш програм треба да одреди најбољи резултат једне одабране земље.

Опис улаза

- Број земаља учесница - цео број n;
- Тренутни укупан број поена сваке од земаља (бројеви раздвојени размаком, сортирано нерастуће) - n целих бројева;
- Редни број земље чији нас успех занима - цео број $1 \leq a \leq n$;
- Редни број земље која треба да гласа - цео број $1 \leq b \leq n$.

Опис излаза

Цео број који представља колико најмање земаља мора да има више поена од земље са редним бројем i

Ограничења

- У тест примерима вредним 50 поена важи $4 \leq n \leq 100$.
- У свим тест примерима важи $4 \leq n \leq 45000$.

Пример 1

Улаз	*Излаз*	*Објашњење*
`4`	`0`	Пошто гласа земља број 1 она не може себи дати поене па је за земљу број 2 најбоље да се гласа на следећи начин: 1. 0 2. 3 3. 2 4. 1 јер би онда укупан број поена био: 1. 7 2. 8 3. 6 4. 3 тј. ниједна земља не би имала више поена од земље са редним бројем 2.
`7 5 4 2`		
`2`		
`1`		

Пример 2

Улаз	*Излаз*	*Објашњење*
`4`	`1`	Пошто гласа земља број 3 она не може себи дати поене па је за земљу број 4 најбоље да се гласа на следећи начин: 1. 2 2. 1 3. 0 4. 3 јер би онда укупан број поена био: 1. 7 2. 6 3. 5 4. 6 тј. само једна земља би морала да има више поена од четврте.
`5 5 5 3`		
`4`		
`3`		

Пример 3

Улаз	*Излаз*
`10`	`0`
`46 44 44 43 42 41 40 39 35 31`	
`6`	
`4`	

Решење

Задатак се ефикасно може решити грамзивим алгоритмом.

Разликујемо случајеве када гласа земља чији нас успех занима (земља a) и када она не гласа.

Ако гласа земља a, она не може себи дати ни један поен, па је њен коначни број поена једнак тренутном. Све остале земље ће добити бар један поен, што значи да ће земље које се у низу налазе испред ње сигурно остати испред ње. Треба да испитамо колико земаља које су иза земље a могу да остану иза ње. Пролазимо земље редом и свакој покушавамо да доделимо што већи број поена (који још нисмо доделили) тако да та земља нема већи број поена од земље a. Ако то не успемо, и та земља ће сигурно претећи земљу a.

Ако не гласа земља чији нас успех занима, онда је најбоље да њој доделимо максималан број поена. Земље које су биле у низу иза земље a ће добити мање поена од ње и остаће иза ње.

Зато анализирамо редом земље које су у низу испред земље a и свакој од њих покушавамо да доделимо што већи број поена (који још нисмо доделили) тако да та земља нема већи број поена од земље a. Ако то не успемо, и та земља ће сигурно претећи земљу a. Поступак можемо прекинути када дођемо до прве земље која и пре доделе поена има више поена него земља a, јер ће и она и све земље које су испред ње у низу имати и касније више поена него земља a. Посебну пажњу треба обратити на земљу која гласа, јер она себи не може да додели поене.

За сваку од $n-1$ земаља обилазимо низ недодељених поена дужине $O(n)$, да бисмо пронашли највећи број поена који можемо додати текућој земљи, па је сложеност алгоритма када се користи низ недодељених поена $O(n \log n)$.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> poeni(n);
  for (int i = 0; i < n; i++)
    cin >> poeni[i];

  int zemljaA, zemljaGlasa;
  cin >> zemljaA >> zemljaGlasa;
  // prelazimo na indeksiranje od nule
  zemljaA--;
  zemljaGlasa--;

  // trenutni broj poena zemlje A
  int poeniZemljeA = poeni[zemljaA];
  // broj zemalja boljih od zemlje A
  int brojBoljihOdA;

  // poeni koje treba dodeliti svim zemljama
  vector<int> poeniZaDodelu(n-1);
  for (int i = 1; i <= n-1; i++)
    poeniZaDodelu[i-1] = i;

  if (zemljaGlasa == zemljaA) {
    // svi u nizu ispred zemljeA ce dobiti bar 1 poen i bice sigurno
    // bolji od zemlje A
    brojBoljihOdA = zemljaA;

    // analiziramo zemlje koje su pre glasanja iza zemlje A
    for (int zemlja = zemljaA+1; zemlja < n; zemlja++) {
      // trazimo maksimalni broj poena koji mozemo dodeliti trenutnoj
      // zemlji tako da ne prestigne zemlju A
      int i = poeniZaDodelu.size() - 1;
      while (i >= 0 && poeni[zemlja] + poeniZaDodelu[i] > poeniZemljeA)
```

```
      i--;
    // ako takav broj poena postoji, dodeljujemo ga tekucoj zemlji i
    // ona ostaje iza zemlje A a u suprotnom i ona prestize zemlju A
    if (i >= 0)
      poeniZaDodelu.erase(poeniZaDodelu.begin() + i);
    else
      brojBoljihOdA++;
  }
} else {
  // zemljiA dodeljujemo maksimalni broj poena (n-1)
  poeniZemljeA += (n-1);
  poeniZaDodelu.pop_back();

  // jos nismo nasli zemlje bolje od A
  brojBoljihOdA = 0;

  // sve zemlje u nizu koje su bile iza zemlje A ce dobiti manje od
  // n-1 poena i neće prestići zemlju A

  // analiziramo samo zemlje u nizu koje su bile ispred zemlje A
  int zemlja;
  for (zemlja = zemljaA-1; zemlja >= 0 && poeni[zemlja] <= poeniZemljeA; zemlja--) {
    if (zemlja == zemljaGlasa) {
      // zemlja koja glasa dobija 0 poena i gledamo da li je ona
      // bolja od zemljeA
      if (poeni[zemljaGlasa] > poeniZemljeA)
        brojBoljihOdA++;
    } else {
      // trazimo maksimalni broj poena koji mozemo dodeliti trenutnoj
      // zemlji tako da ne prestigne zemlju A
      int i = poeniZaDodelu.size() - 1;
      while (i >= 0 && poeni[zemlja] + poeniZaDodelu[i] > poeniZemljeA)
        i--;
      // ako takav broj poena postoji, dodeljujemo ga tekucoj zemlji i
      // ona ostaje iza zemlje A a u suprotnom i ona prestize zemlju A
      if (i >= 0)
        poeniZaDodelu.erase(poeniZaDodelu.begin()+i);
      else
        brojBoljihOdA++;
    }
  }

  // sve preostale zemlje koje su bile bolje od zemlje A ce i dalje biti
  // bolje od zemlje A
  brojBoljihOdA += (zemlja+1);
}

// ispisujemo rezultat
cout << brojBoljihOdA << endl;
```

```
  return 0;
}
```

Ефикасније решење се постиже ако се недодељени поени чувају у уређеном скупу. Бинарном претрагом, у сложености $O(\log n)$ можемо одредити најмањи број недодељених поена којим текућа земља строго престиже земљу A. Ако се то дешава већ за најмањи број недодељених поена, онда та земља мора да претекне земљу A. У супротном је претходни елемент скупа највећи број недодељених поена којим текућа земља не престиже строго земљу A, па њега додељујемо и уклањамо из скупа (у сложености $O(\log n)$. Дакле, уз коришћење уређеног скупа сложеност решења је $O(n \log n)$.

```
#include <iostream>
#include <vector>
#include <set>

using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> poeni(n);
  for (int i = 0; i < n; i++)
    cin >> poeni[i];

  int zemljaA, zemljaGlasa;
  cin >> zemljaA >> zemljaGlasa;
  // prelazimo na indeksiranje od nule
  zemljaA--;
  zemljaGlasa--;

  // trenutni broj poena zemlje A
  int poeniZemljeA = poeni[zemljaA];
  // broj zemalja boljih od zemlje A
  int brojBoljihOdA;

  // poeni koje treba dodeliti svim zemljama
  set<int> poeniZaDodelu;
  for (int i = 1; i <= n-1; i++)
    poeniZaDodelu.insert(i);

  if (zemljaGlasa == zemljaA) {
    // svi u nizu ispred zemljeA ce dobiti bar 1 poen i bice sigurno
    // bolji od zemlje A
    brojBoljihOdA = zemljaA;

    // analiziramo zemlje koje su pre glasanja iza zemlje A
    for (int zemlja = zemljaA+1; zemlja < n; zemlja++) {
      // trazimo maksimalni broj poena koji mozemo dodeliti trenutnoj
      // zemlji tako da ne prestigne zemlju A
```

```
    auto it = poeniZaDodelu.upper_bound(poeniZemljeA - poeni[zemlja]);
    // ako takav broj poena postoji, dodeljujemo ga tekucoj zemlji i
    // ona ostaje iza zemlje A a u suprotnom i ona prestize zemlju A
    if (it != poeniZaDodelu.begin()) {
      it--;
      poeniZaDodelu.erase(it);
    } else
      brojBoljihOdA++;
  }
} else {
  // zemljiA dodeljujemo maksimalni broj poena (n-1)
  poeniZemljeA += (n-1);
  poeniZaDodelu.erase(n-1);

  // jos nismo nasli zemlje bolje od A
  brojBoljihOdA = 0;

  // sve zemlje u nizu koje su bile iza zemlje A ce dobiti manje od
  // n-1 poena i neće prestići zemlju A

  // analiziramo samo zemlje u nizu koje su bile ispred zemlje A
  int zemlja;
  for (zemlja = zemljaA-1; zemlja >= 0 && poeni[zemlja] <= poeniZemljeA; zemlja--) {
    if (zemlja == zemljaGlasa) {
      // zemlja koja glasa dobija 0 poena i gledamo da li je ona
      // bolja od zemljeA
      if (poeni[zemljaGlasa] > poeniZemljeA)
        brojBoljihOdA++;
    } else {
      // trazimo maksimalni broj poena koji mozemo dodeliti trenutnoj
      // zemlji tako da ne prestigne zemlju A
      auto it = poeniZaDodelu.upper_bound(poeniZemljeA - poeni[zemlja]);
      // ako takav broj poena postoji, dodeljujemo ga tekucoj zemlji i
      // ona ostaje iza zemlje A a u suprotnom i ona prestize zemlju A
      if (it != poeniZaDodelu.begin()) {
        it--;
        poeniZaDodelu.erase(it);
      } else
        brojBoljihOdA++;
    }
  }

  // sve preostale zemlje koje su bile bolje od zemlje A ce i dalje biti
  // bolje od zemlje A
  brojBoljihOdA += (zemlja+1);
}

// ispisujemo rezultat
cout << brojBoljihOdA << endl;
```

```
  return 0;
}
```

Задатак: Последња штанглица

Аутор: Огњен Нешковић

Такмичење: 2022/2023. квалификације 3, 7. разред, 3. задатак и 8. разред 2. задатак

Познато је да је древни народ Маја изузетно вредновао и користио какао у многим церемонијама и ритуалима. Оно што историја још увек није потврдила, али је сигурно тачно је да су још богови Маја знали да направе чоколаду и јако су волели да је једу. Древно божанство, бог трговине и какаоа Ek Chuah је био веома вешт у прављењу чоколаде. Једном му је остала само једна штанглица и баш у том тренутку се појавио бог муње K'awiil који, као и сви богови, јако воли чоколаду. Сада морају да се боре за преосталу чоколаду.

Штанглица има n коцкица. Богови су одлучили да ће играти једну игру у којој ће наизменично сећи и узимати делове штанглице. Бог Ek Chuah игра први и сече штанглицу једним резом на два дела. Затим бог K'awiil бира који од та два дела жели за себе.

Штангла чоколаде је већ подељена на коцкице, богови ће сећи по коцкицама, неће пресећи неку коцкицу на пола. Ek Chuah и K'awiil наизменично секу и узимају делове док не остане само једна коцкица, тај део иде ономе ко је на реду да узима следећи.

Ek Chuah и K'awiil обоје желе да добију што већи број коцкица чоколаде. Ви посматрате ову борбу, али пошто штанглица може бити веома дугачка и имати до 10^9 коцкица (што је скоро $\frac{3}{4}$ обима Земље) постали сте нестрпљиви. *Занима вас да одредите колико коцкица ће добити древно божанство Ek Chuah, ако и Ek Chuah и K'awiil играју оптимално.*

Опис улаза

У једном реду стандардног улаза се уноси природан број $n \leq 10^9$.

Опис излаза

На стандардни излаз исписати један природан број који представља број парчића које ће древно божанство Ek Chuah добити.

Пример 1

Улаз	*Излаз*	*Објашњење*

`7`	`2`	Један начин на који се игра може одвити је следећи - штанглица је на почетку:
		`#######`
		Ek Chuah прави рез, а K'awiil узима леви део.
		`KKKK|###`
		Сада је преостало за поделу:
		`###`
		K'awiil прави рез, а Ek Chuah узима десни део.
		`#|EE`
		Сада је преостало за поделу:
		`#`
		Преостала је 1 коцкица. Ek Chuah треба да направи рез, а K'awiil узима ову коцкицу. Ek Chuah је, дакле, укупно узео 2 дела, а K'awiil је узео 5.
		Приметимо да на пример следеће не би могло да се деси јер K'awiil игра оптимално: `KK|#####` -> `EEEE|#` -> `#` (овде би Ek Chuah добио 4 коцкице, а K'awiil 3)

Пример 2		**Пример 3**		**Пример 4**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`13`	`4`	`2`	`1`	`1`	`0`

Решење

Решење се добија грамзивим алгоритмом. Оптимална стратегија оба играча је да рез праве што је ближе могуће половини, јер на тај начин противнику препуштају што мањи број коцкица.

```
#include <iostream>

using namespace std;

int main()
{
  long long n; cin >> n;
  // broj parcica koje su Ana i Bojan uzeli
  long long a = 0, b = 0;
  while (n > 0) {
    // tortu delimo što bliže sredini
    long long l = n / 2, r = n - n / 2;
    // Bojan uzima veći deo
    b += max(l, r);
    // preostao je manji deo
    n = min(l, r);

    // tortu ponovo delimo što bliže sredini
    l = n / 2, r = n - n / 2;
    // Ana uzima veći deo
    a += max(l, r);
    // preostao je manji deo
    n = min(l, r);
```

```
  }
  cout << a << endl;
  return 0;
}
```