Садржај

Глава 1

Сезона 2022/23.

1.1 Квалификације – први круг

Први круг квалификација.

Задатак: Гориво

Аутор: Филип Марић

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.

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) {
  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;
}
```

Ово је након задатка.

Задатак: Катанац

```
int okret(int a, int b) {
  return min((a - b + 10) % 10, (b - a + 10) % 10);
}
```

Ово је након другог задатка.

Задатак: Квиз

Аутор: Милан Вутделија

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*
3 5 12	11

Објашњење

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

```
7
15
4
```

Излаз

```
-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: Гориво

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види текст задатка.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

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.

```cpp
#include <iostream>

using namespace std;

int gorivo(int potrosnja, int rastojanje) {
  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: Какуро провера

Аутор: Филип Марић

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 **Пример 2**

Улаз *Излаз*

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

```
da
```

Улаз *Излаз*

```
11 9
17 3
9 8
3 1
```

```
ne
```

Објашњење

Збир елемената прве колоне није 11 већ 12, а збир елемената друге врсте није 3 већ 4.

Пример 3

Улаз

```
11 10
17 4
9 8
2 2
```

Излаз

```
ne
```

Објашњење

Иако се сви збирови поклапају, у другој врсти је два пута употребљена цифра 2.

Пример 4

Улаз

```
16 6
14 8
10 4
6 2
```

Излаз

```
ne
```

Објашњење

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: Списак производа

Аутор: Филип Марић

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

Улаз	*Излаз*
32 30 45 jjkllkl	259

Пример 2

Улаз	*Излаз*
10 20 30 jkl	60

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: Мајстор

Аутор: Иван Дрецун

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 промењливама $\min_1$ (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 промењливе.

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).

```cpp
#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).

```cpp
#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: Зрак

Аутор: Филип Марић

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

```
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
```

Излаз

```
2
```

Објашњење

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

```
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
```

Излаз

```
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: Број дана одмереног тренинга

Аутор: Филип Марић

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: Словарица

Аутор: Милан Вуџелија

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.

```
#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: Ћ++

Аутор: Душан Попадић

Пас Ћуки је најпаметнији пас на свету. Поред тога што уме да направи разне акробатске трикове, он је и изврстан програмер, па је измислио нови програмски језик који је назвао Ћ++.

Свака линија програмског кода у језику Ћ++ се састоји од тачно m целих бројева раздвојених размацима. Пошто је Ћ++ веома напредан језик, потребно га је превести тако да буде разумљив данашњим рачунарима.

У процесу превођења је у једном тренутку потребно да се свака линија кода трансформише тако да садржи m истих целих бројева. Линија се трансформише тако што се сваки број повећава или смањује за 1 док се не дође до жељеног броја, а цена трансформације једног броја у други је број повећавања/смањивања.

Рецимо да желимо да трансформишемо линију “1 4 9” у “3 3 3”.

- Број 1 треба да повећамо 2 пута да бисмо добили број 3, па је цена те трансформације 2.
- Број 4 треба да смањимо једном, па је цена трансформације 1.
- Број 9 треба да смањимо 6 пута да бисмо добили жељени број 3, па је цена трансформације тог броја 6.

Укупна цена трансформације линије је 2 + 1 + 6 = 9.

Како би се уштедели ресурси при превођењу, потребно је трансформисати код тако да укупна цена трансформације буде што мања. За једну учитану линију је потребно одредити у који ће се број трансформисати да би цена трансформације била најмања могућа. У случају да је линију могуће трансформисати у више различитих бројева тако да цена остане иста, врши се трансформација у највећи такав број.

Опис улаза

У првом реду стандардног улаза се налази природан број m ($1 \leq m \leq 10^5$) који означава колико бројева се налази у линији која се учитава. У наредном реду се налази низ од m целих бројева (између -1000 и 1000) одвојених размацима који представљају једну линију у Ћ++-у.

Додатна ограничења: * У тест примерима вредним 49 поена додатно важи $m \leq 100$.

Опис излаза

У првом реду стандардног излаза исписати један број који означава у који број ће се трансформисати бројеви у датој линији. У другом реду стандардног излаза исписати најмању цену трансформације дате линије програмског кода.

Пример 1

Улаз	*Излаз*
6	6
1 8 6 -4 0 7	24

Објашњење

Најмања цена трансформације постиже се ако се сви бројеви трансформишу у број 6. Цена трансформације линије је: $|1-6|+|8-6|+|6-6|+|-4-6|+|0-6|+|7-6| = 24$. Приметите да би се иста цена добила и ако бисмо трансформисали линију у број 1, али је број 6 већи, па бирамо њега.

Пример 2

Улаз

```
7
-5 -4 -3 -2 -1 0 1
```

Излаз

```
-2
12
```

Решење

Хајде да замислимо да је низ бројева у линији кода сортиран и да смо те бројеве поређали на бројевну праву. Хајде сада да претпоставимо да је x тражени најбољи број за трансформацију и хајде да и њега поставимо на исту бројевну праву. Уколико x померимо удесно за један број на бројевној правој, тада смо повећали растојање тог броја од свих њему левих бројева, а смањили његово растојање од свих њему десних бројева. Овим смо заправо повећали цену за 1 свим бројевима који су мањи од x, а смањили цену за 1 свим бројевима који су већи од x. Одавде можемо лако да закључимо да нам се исплати да померамо x удесно докле год има мање бројева лево од x него десно од x (укупну цену ћемо тако смањивати јер смо је више смањили него повећали). Потпуно аналогно размишљање примењујемо и ако x померамо улево уместо удесно. Дакле x треба да буде такво да се са његове леве и десне стране налази исти број бројева, тј. да x треба да буде средњи елемент (медијана) овог низа. Средњи елемент лако можемо одредити тако што сортирамо низ и онда узмемо елемент који је тачно на пола сортираног низа. Када одредимо x, укупну цену лако рачунамо као збир одстојања свих бројева од x.

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

using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    int n, m;
    cin >> m;
    n = 1;
        int cena = 0;
    for(int i = 0; i < n; i++){
        vector<int> linija(m);
        for(int j = 0; j < m; j++){
            cin >> linija[j];
        }
        sort(begin(linija), end(linija));
        int r = linija[m / 2];
```

```
        cout << r << endl;
        for(int j = 0; j < m; j++){
            cena += abs(linija[j] - r);
        }
    }
    cout << cena;

    return 0;
}
```

Задатак: Допуна мејлова

Аутор: Филип Марић

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 (n), па ако се испитује m префикса, укупна сложеност је (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 $(n \log n)$, а сложеност сваке од ове две бинарне претраге је $(\log n)$, па ако се испитује m префикса, укупна сложеност је $((n+m)\log n)$.

```
#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: Кувар

Аутор: Милан Вуīделија

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*
200 250 150 2200 1400 890	10

Објашњење

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

```
150
```

```
200
100
6100
5100
1000
```

Излаз

```
35
```

Објашњење

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: Стубићи од коцкица

Аутор: Милан Вуīделија

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*
10 20	20
18 28	
21 31	
1 10	

Објашњење

Објашњење

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

Пример 2

Улаз

```
7 4
9 13
14 12
3 2
```

Излаз

```
15
```

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: Ноте

Аутор: Филип Марић

Тонови се у музициређају по октавама, које се састоје од по 7 основних тонова (беле дирке на клавиру) и 5 повишених/снижених тонова (црне дирке на клавиру). Основни тонови су C, D, E, F, G, A, H, док се повишени/снижени тонови налазе између тонова C и D (то је тон C# тј. D♭), затим између D и E (то је тон D# тј. E♭), између F и G (то је тон F# тј. G♭), између G и A (то је тон G# тј. A♭) и између A и H (то је тон A# тј. H♭, који се некада обележава и са B). Договоримо се да ћемо све међутонове обележавати као повишене (коришћењем ознаке #). Тада се сви тонови ређају на следећи начин:

`C, C#, D, D#, E, F, F#, G, G#, A, A#, H, C, C#, D, D#, E, F, ....`

Транспоновање неке мелодије значи њено померање за одређени број тонова навише или наниже. Напиши програм који транспонује унету мелодију тј. унети низ тонова.

Опис улаза

Са стандардног улаза се уноси ниска од највише 1000 карактера која садржи низ тонова које треба транспоновати, а затим и број између -11 и 11 који означава за колико тонова је потребно транспоновати тај низ тонова.

Опис излаза

На стандардни излаз транспоновани низ тонова.

Пример 1

Улаз	*Излаз*
GDGHDCAF#D 5	CGCEGFDHG

Објашњење

Померањем од тона G за 5 места удесно у односу на редослед

C, C#, D, D#, E, F, F#, G, G#, A, A#, H, C, C#, D, D#, E, F, ...

добија се тон C. Померањем од тона D за 5 места удесно добија се тон G итд.

Пример 2

Улаз

```
C#F#AG
-3
```

Излаз

```
A#D#F#E
```

Решење

Један од најважнијих делова задатака је поделити учитану ниску на низ појединачних нота. То радимо тако што за свако учитано слово проверавамо да ли је праћено карактером #. Ако није, онда се нота састоји само од тог слова, а у супротном од тог слова и повисилице која га прати. Да бисмо могли да транспонујемо, потребно је да у једном низу чувамо редослед свих нота. Да бисмо за сваку учитану ноту могли ефикасно да одредимо позицију у том низу, згодно је да направимо мапу тј. речник у коме ноте пресликавамо у њихове редне бројеве. Редни број сваке учитане броје сабирамо са померајем, по модулу 12 (што је укупан број нота). Пошто померај може бити и негативан, пре одређивања модула увећавамо збир за 12 (да бисмо избегли дељење негативних бројева).

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

using namespace std;

int main() {
  vector<string> skala{"C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "H"};
  map<string, int> rbr;
  for (int i = 0; i < skala.size(); i++)
    rbr[skala[i]] = i;

  string melodija;
  cin >> melodija;
  int pomeraj;
  cin >> pomeraj;
  for (int i = 0; i < melodija.size(); i++) {
    string nota;
    if (i + 1 < melodija.size() && melodija[i+1] == '#') {
      nota = melodija.substr(i, 2);
      i++;
    } else {
      nota = melodija.substr(i, 1);
    }
    cout << skala[(rbr[nota] + pomeraj + skala.size()) % skala.size()];
  }
  cout << endl;
  return 0;
}
```

Задатак: Датум са највећом зарадом

Аутор: Филип Марић

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

Да бисмо ефикасно израчунали зараду за сваки дан (дневни пазар), можемо 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 <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;
}
```

Задатак: Збирови након поделе

Аутор: Иван Дрецун

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*
`8` `1 7 3 4 6 5 8 0`	`7`

Објашњење

Објашњење: Највећи број парова добија се поделом низа на делове `1 7 3 4` и `4 5 8 0`. Парови су $(1, 5)$, $(7, 5)$, $(3, 5)$, $(4, 8)$, $(4, 0)$, $(6, 8)$ и $(6, 0)$.

Пример 2

Улаз

```
7
1 9 3 11 12 4 8
```

Излаз

```
4
```

Објашњење

Објашњење: 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;
}
```

1.2 Квалификације – други круг

Задатак: Шљиве

Аутор: Милан Вуџелија

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 $u = s \operatorname{div} 5 = \left\lfloor \frac{s}{5} \right\rfloor$, 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: Флаше

Аутор: Душан Пођадић

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*
7 6	4 0
24	2 1
13	1 0
6	1 4
10	2 5
17	3 5
23	6 4
40	

Објашњење

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: Катанац

Овај задатак је поновљен у циљу увежбавања различитих техника решавања. Види текст задатка.

Покушај да задатак урадиш коришћењем техника које се излажу у овом поглављу.

Решење

Разлика по модулу

Број окрета да би се од цифре a дошло до цифре b ако се прстен окреће ка већим цифрама једнак је разлици $b-a$ по модулу 10, а ако се прстен окреће ка мањим цифрама једнак је разлици $a-b$ по модулу 10. Разлика два броја x и y по модулу 10 се може израчунати формулом $(x-y+10) \bmod 10$. На пример, ако је $a=2$ и $b=8$ тада се на први начин прави $(8-2+10) \bmod 10 = 6$ окрета, а на други начин прави $(2-8+10) \bmod 10 = 4$ окрета. Треба узети мањи од ова два броја, а онда тај поступак поновити за сваку од 4 цифре. Да бисмо издвојили цифре броја најлакше је да те бројеве учитамо као ниске карактера.

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

using namespace std;

int okret(int a, int b) {
  return min((a - b + 10) % 10, (b - a + 10) % 10);
}


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

  int broj = 0;
  for (int i = 0; i < 4; i++)
    broj += okret(a[i] - '0', b[i] - '0');
  cout << broj << endl;

  return 0;
}
```

Апсолутна разлика

Број потребан да би се од цифре a дошло до цифре b једнак је $|a-b|$ ако се прстен окреће у смеру од a до b, тј. $10-|a-b|$ ако се прстен окреће у супротном смеру. На пример, ако је $a=2$, $b=8$ и ако прстен окрећемо од a ка b тј. од 2 ка 8, потребно нам је $|2-8|=6$, а ако прстен окрећемо у супротном смеру потребно нам је $10-|2-8|=4$ окрета.

```
int okret(int a, int b) {
  return min(abs(a - b), 10 - abs(a - b));
}
```

Задатак: Фудбалска група табела

Аутор: Филип Марић

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: Највише одличних

Аутор: Милан Вуīелија

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: Долине

Аутор: Иван Дрецун

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 уноси 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*
`8` `5 3 3 7 4 4 4 8`	`2`

Објашњење

Долине су `5 3 3 7` и `7 4 4 4 8`.

Пример 2

Улаз

```
5
4 3 1 2 4
```

Излаз

```
1
```

Објашњење

Једина долина је `3 1 2`.

Пример 3

Улаз

```
11
2 2 3 4 4 2 2 2 5 4 5
```

Излаз

```
2
```

Објашњење

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.

```
#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: Последњи меч

Аутор: Филип Марић

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

Улаз	*Излаз*
6	ne
4	
4	
0	

Пример 2

Улаз	*Излаз*
6	0
4	
0	
4	

Пример 3

Улаз	*Излаз*
9	1
0	
1	
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: Распон крила

Аутор: Душан Попадић

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*
4 3 1 1 5 2 10	10

Објашњење

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

```
3
1
6
2 1
3 2
```

Излаз

```
6
```

Објашњење

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: Мешалица

Аутор: Иван Дрецун

На располагању нам је мешалица која може да меша редослед елемената низа. Мешалица елементе увек меша на исти начин. У мешалицу је убачен низ a чијим је мешањем добијен низ m. Напиши програм који учитава низове a и m, као и низ b и исписује како изгледа низ b након мешања.

Опис улаза

Са стандардног улаза се уноси цео број n ($1 \leq n \leq 50$) који представља дужине низова. У наредној линији се уноси n различитих елемената низа a, одвојених размаком. У наредној линији се уноси n различитих елемената низа m, одвојених размаком. У последњој линији се уноси n различитих елемената низа b, одвојених размаком. Сви елементи низова су цели бројеви из интервала $[0, 100]$.

Опис излаза

На стандардни излаз у једној линији исписати n елемената низа добијеног мешањем низа b, одвојене размаком.

Пример 1

Улаз	*Излаз*

`5`
`4 3 1 0 2`
`1 3 2 4 0`
`3 2 0 4 1`
	`0 2 1 3 4`

Пример 2

Улаз	*Излаз*

`4`
`1 2 5 7`
`5 2 1 7`
`3 4 1 5`
	`1 4 3 5`

Пример 3

Улаз	*Излаз*

`5`
`2 6 8 1 3`
`3 1 8 6 2`
`5 4 3 2 1`
	`1 2 3 4 5`

Решење

Посматрајмо пример из текста задатка:

a је низ `4 3 1 0 2`. Након мешања се добија `1 3 2 4 0`. b је низ `3 2 0 4 1`. Приметимо да се мешањем низа a број 4, који је био на позицији 0, преместио на позицију 3. Дакле, након мешања низа b ће се број 3, који се налази на позицији 0, преместити на позицију 3. Слично важи и за остале вредности низа b.

До решења долазимо ако за сваку вредност $b[i]$ низа b одредимо на коју позицију ће се та вредност преместити мешањем. Како бисмо ово урадили, довољно је да видимо шта се дешава са вредношћу низа a на позицији i након мешања. Ако је након мешања низа a вредност $a[i]$ доведена на позицију j, значи да ће мешањем низа b вредност $b[i]$ бити доведена на позицију j.

За одређивање позиције елемента $a[i]$ у промешаном низу a можемо користити функцију `find`.

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

using namespace std;

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

  vector<int> a(n), pa(n), b(n), pb(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
```

```
  for (int i = 0; i < n; i++)
    cin >> pa[i];

  for (int i = 0; i < n; i++)
    cin >> b[i];

  for (int i = 0; i < n; i++) {
    auto it = find(begin(pa), end(pa), a[i]);
    pb[it - begin(pa)] = b[i];
  }

  for (int x : pb)
    cout << x << ' ';
  cout << '\n';

  return 0;
}
```

Задатак: Грешка у тексту

Аутор: Милан Вуīделија

Порука од 4 велика слова је послата три пута, али сваки пут је због сметњи током преноса ТАЧНО ЈЕДАН карактер поруке погрешно пренесен као неко друго велико слово. Написати програм који на основу три примљене поруке исписује реконструисану поруку. Реконструисана порука је исте дужине као и примљене поруке и састоји се од слова и звездица. На позицијама на којима на основу примљених порука СА СИГУРНОШЋУ МОЖЕ да се установи слово оригиналне поруке, реконструисана порука има слово оригиналне поруке, а на осталим местима звездице.

Опис улаза

У сваком од три реда стандардног улаза налазе се по 4 велика слова енглеске абецеде.

Опис излаза

У први и једини ред стандардног излаза исписати 4 карактера. На месту сваког реконструисаног слова исписати то слово, а на месту на коме слово није могло да буде реконструисано исписати звездицу.

Пример 1

Улаз	*Излаз*
AXCD ABYD ABCZ	ABCD

Пример 2

Улаз	*Излаз*
AXCD AYCD AZCD	A*CD

Решење

Да бисмо реконструисали текст онолико колико је то могуће, треба да анализирамо случајеве. Први корак у анализи може да буде налажење позиција на којима се неки од текстова разликују. Индексе позиција на којима се појављују разлике чувамо у вектору `pozRaz`. Упоредо са тражењем позиција са разликама, попуњавамо ниску `rez`, у којој ће се на крају наћи резултујући текст, реконструисан колико је могуће. На позицијама на којима се дати текстови не разликују, у `rez` се иницијално уписује заједничко слово из сва три текста, а на осталим позицијама звездица. Ово уписивање слова и звездица у резултат ће накнадно бити или оправдано, или измењено у складу са закључцима анализе.

Након проналажења позиција са разликама, разликујемо следеће случајеве:

- Ако постоје три позиције са разликама, на свакој од њих само један од текстова може да буде измењен (у противном би укупан број измена био већи од 3). У овом случају комплетан текст може да се реконструише, тако што у резултат на свакој позицији упишемо слово које се у улазним текстовима на тој позицији појављује два пута. Ово радимо у функцији `sredi`.
- Ако постоје две позиције са разликама, није могуће да на обе позиције имамо по три различита слова (јер би тада укупан број измена био већи од 3). Према томе, бар на једној позицији су два слова иста. Разликујемо два подслучаја:

  - Ако су на другој позицији са разликама сва слова различита, на тој позицији су две измене, а на оној са истим словима једна, па слово на обе позиције може да се реконструише. На пример, ако су текстови `.A.C`, `.A.D` и `.B.E` (где тачке означавају нека неизмењена слова), реконструисани текст је `.A.E`.
  - Ако су и на другој позицији са разликама два слова иста, онда се на некој од позиција догодила иста измена у два текста и реконструкција није могућа ни за једну од позиција, јер постоје две могућности за полазни текст. На пример, ако су текстови `.A.C`, `.A.C` и `.B.D` (где тачке означавају нека неизмењена слова), оригиналан текст би могао да буде `.B.C` или `.A.D`.
- Ако постоји само једна позиција са разликама, све измене су на тој позицији (у противном би укупан број измена био већи од 3). Реконструкција тог слова није могућа.
- Коначно, ако нема позиција са разликама, ни једно слово није сигурно и сва их треба у резултату представити звездицама.

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

using namespace std;

const int n = 4;
string s[3];
char rez[n];
int Sredi(int i) {
    if (s[0][i] == s[1][i] && s[1][i] != s[2][i]) {
        rez[i] = s[0][i];
        return 2;
    }
    if (s[1][i] == s[2][i] && s[2][i] != s[0][i]) {
        rez[i] = s[1][i];
        return 0;
    }
    if (s[2][i] == s[0][i] && s[0][i] != s[1][i]) {
        rez[i] = s[2][i];
        return 1;
    }
    return -1;
}

int main() {
    cin >> s[0] >> s[1] >> s[2];
    vector<int> pozRaz;
    for (int i = 0; i < n; i++) {
        if (s[0][i] != s[1][i] || s[1][i] != s[2][i] || s[2][i] != s[0][i]) {
            pozRaz.push_back(i);
            rez[i] = '*';
        }
        else rez[i] = s[0][i];
    }
    if (pozRaz.size() == 3)
        for (int p = 0; p < 3; p++)
            Sredi(pozRaz[p]);
    else if (pozRaz.size() == 2) {
        int razlicit0 = Sredi(pozRaz[0]);
        int razlicit1 = Sredi(pozRaz[1]);
        if (razlicit0 >= 0 && razlicit1 >= 0) {
            rez[pozRaz[0]] = '*';
            rez[pozRaz[1]] = '*';
        }
        else if (razlicit0 >= 0)
            rez[pozRaz[1]] = s[razlicit0][pozRaz[1]];
```

```
        else // if (razlicit1 >= 0)
            rez[pozRaz[0]] = s[razlicit1][pozRaz[0]];
    }
    else if (pozRaz.size() == 0)
        for (int i = 0; i < n; i++)
            rez[i] = '*';

    for (int i = 0; i < n; i++)
        cout << rez[i];
    cout << endl;

    return 0;
}
```

Задатак: Број троцифрених парних

Аутор: Милан Вуђелија

Написати програм који за дати низ цифара исписује број различитих троцифрених парних бројева, који могу да се саставе од датих цифара. Приликом формирања сваког троцифреног парног броја, сваки елемент низа цифара може да се употреби највише једном.

Опис улаза

У првом реду стандардног улаза се налази број цифара N, такав да $3 \leq N \leq 50000$. У другом реду је N цифара раздвојених по једним размаком.

Опис излаза

На стандардни излаз исписати само један број, тражени број троцифрених бројева.

Пример 1

Улаз *Излаз*

```
4
2 4 2 2
```

```
4
```

Објашњење

Од цифара 2, 4, 2, 2 могу да се саставе 4 различита троцифрена парна броја, а то су 222, 224, 242, 422.

Пример 2

Улаз

```
3
0 1 0
```

Излаз

```
1
```

Објашњење

Од цифара 0, 1, 0 може да се састави само 1 троцифрен паран број, а то је 100.

Пример 3

Улаз

```
3
4 1 2
```

Излаз

```
4
```

Објашњење

Од цифара 4, 1, 2 могу да се саставе 4 различита троцифрена парна броја, а то су 124, 142, 214, 412.

Решење

Задатак може да се реши тако што за сваку тројку цифара из улазног низа проверимо да ли је она запис троцифреног парног броја, а пребројимо оне тројке које то јесу. Ово решење захтева број операција сразмеран са n^3, где је n дужина улазног низа, а то може да буде преспоро.

```
#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];

  vector<bool> napravljen(1000, false);
  for (int c1 = 0; c1 < n; c1++)
    if (a[c1] != 0)
      for (int c2 = 0; c2 < n; c2++)
        if (c1 != c2)
          for (int c3 = 0; c3 < n; c3++)
            if ((c1 != c3) && (c2 != c3) && (a[c3] % 2 == 0))
              napravljen[a[c1] * 100 + a[c2] * 10 + a[c3]] = 1;

  int br = 0;
  for (int i = 0; i < 1000; i++)
    if (napravljen[i])
      br++;
  cout << br << endl;

  return 0;
}
```

Решење са провером сваке тројке бројева можемо да побољшамо ако приметимо да ниједна цифра не може да учествује у формираном броју више од три пута. Према томе, из полазног низа можемо да задржимо до три појављивања сваке цифре (а остала појављивања да одбацимо), а затим да проверимо сваку тројку цифара из тако скраћеног низа. Такав, скраћени низ може да има највише 30 елемената (сваку цифру по три пута), па чак и да анализирамо сваку тројку, број операција у том делу програма не прелази $30^3 = 27000$.

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

using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> ulaz(n);
  for (int i = 0; i < n; i++)
    cin >> ulaz[i];

  vector<int> a;
  for (int cifra = 0; cifra < 10; cifra++) {
    int m = count(ulaz.begin(), ulaz.end(), cifra);
    if (m > 3)
      m = 3;
```

```
    for (int i = 0; i < m; i++)
      a.push_back(cifra);
  }

  vector<bool> napravljen(1000, false);
  n = a.size();
  for (int c1 = 0; c1 < n; c1++)
    if (a[c1] != 0)
      for (int c2 = 0; c2 < n; c2++)
        if (c1 != c2)
          for (int c3 = 0; c3 < n; c3++)
            if ((c1 != c3) && (c2 != c3) && (a[c3] % 2 == 0))
              napravljen[a[c1] * 100 + a[c2] * 10 + a[c3]] = true;

  int br = 0;
  for (int i = 0; i < 1000; i++)
    if (napravljen[i])
      br++;
  cout << br << endl;
  return 0;
}
```

Веома ефикасно решење добијамо када пребројимо појављивања сваке цифре, а затим за сваки паран троцифрен број проверимо да ли он може да се формира од цифара које су на располагању.

```
#include <iostream>

using namespace std;

int frek[10];
int main() {
  int n, c;
  cin >> n;
  for (int i = 0; i < n; i++) {
    cin >> c;
    frek[c]++;
  }
  int br = 0;
  for (int c100 = 1; c100 < 10; c100++) {
    if (frek[c100] > 0) {
      frek[c100]--;
      for (int c10 = 0; c10 < 10; c10++) {
        if (frek[c10] > 0) {
          frek[c10]--;
          for (int c1 = 0; c1 < 10; c1 += 2)
            if (frek[c1] > 0)
              br++;
          frek[c10]++;
        }
      }
      frek[c100]++;
    }
  }
  cout << br << endl;
  return 0;
}
```

Задатак: Пуж

Аутор: Душан Пејадић

Немања има 6 година и почео је да учи да чита и пише. Од стрица је добио вежбанку за учење слова у којој се налази вежбица “Пуж”. Наиме у овој вежбици постоји табела диманзија nxm у коју је потребно уписати неку реченицу на следећи начин: креће се из горњег левог угла табеле и уписују се слова ка десној ивици табеле докле може. Када се стигне до ивице онда се наставља уписивање на доле до ивице, затим на десно до ивице, па на горе док има места, онда опет на десно и тако док се не попуни цела табела (слика). У табелу се уписују само слова, игноришу се размаци, знакови интерпункције и слично. Пошто је Немања сувише мали за ову вежбу, табелу је попунио стриц. Немању сада занима да ли се у реченици која је коришћена да се попуни табела налази име његовог омиљеног суперхероја. Пошто Немања још не зна сва слова, помозите му да ово открије.

Опис улаза

У првом реду стандардног улаза се налазе два цела броја n и m ($2 \leq n, m \leq 50$) која редом представљају број редова и број колона ове табеле. У наредних n редова се налази по m карактера (мала слова енглеске абецеде) који сви заједно представљају попуњену табелу. У наредном реду се налази ниска карактера (дужине до 30) која представља име Немањиниог омиљеног суперхероја (такође мала слова енглеске абецеде, без размака и знакова интерпункције).

Опис излаза

У јединoм реду стандардног излаза исписати један цео број који представља позицију у реченици од које почиње име траженог суперхероја. У случају да име не постоји исписати -1. Уколико се име појављује више пута, исписати почетну позицију првог појављиавања.

Пример 1

Улаз	*Излаз*
`4 5`	`11`
`mnogo`	
`jdrmv`	
`aaneo`	
`psmil`	
`spajdrmen`	

Објашњење

Када се табела развије добија се ниска карактера mnogovolim**spajdrmen**a, тако да се реч spajdrmen појављује почевши од 11. места.

Пример 2

Улаз

```
5 5
dalis
rnuui
ccudt
oivoi
adelg
crnaudovica
```

Излаз

```
-1
```

Објашњење

Иако се у реченици прича о Црној удовици, низ карактера `crnaudovica` се не појављује јер је њено име у реченици у падежу (акузативу).

Решење

У овом задатку је потребно обићи табелу на описан начин и преписати слова из табеле у ниску карактера, а затим применити функцију `find` да би се пронашао тражени индекс. Треба приметити да се табела увек прво обилази удесно док се не дође до њене десне ивице, затим надоле, па улево и онда нагоре, затим опет удесно и тако у круг док се не обиђу сва поља. Дефинишемо два низа `di` и `dj` који представљају помераје у табели по врстама и колонама (прво је померај (`di`=0, `dj`=1) - дакле не померамо се по врстама, а расте

нам индекс колона за 1, тј. идемо удесно). Променљива `s` означава индекс у овим низовима, односно смер кретања кроз табелу. Када дођемо до ивице онда мењамо смер, тј. повећавамо `s` за 1 по модулу 4, да бисмо ишли кружно. Наравно, док се крећемо и границе до којих можемо да идемо у табели се сужавају, тј. када се нпр. крећемо удесно и ударимо у десну ивицу (након чега крећемо надоле), тада сужавамо табелу одозго, тј. повећавамо горњу границу за 1. Слично је и за остале смерове кретања. Заустављамо се када смо обишли сва поља, односно када се лева и десна или горња и доња граница мимоиђу.

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

using namespace std;

int main() {
  int n, m;
  cin >> n >> m;
  vector<string> tabela(n);

  for(int i = 0; i < n; i++)
    cin >> tabela[i];

  string rec;
  cin >> rec;

  // pomeraji kroz tabelu: desno, dole, levo, gore
  vector<int> di = {0, 1, 0, -1};
  vector<int> dj = {1, 0, -1, 0};

  int gornjaGranica = 0;
  int donjaGranica = n - 1;
  int levaGranica = 0;
  int desnaGranica = m - 1;

  int s = 0;

  string recenica(n*m,' ');

  int i = 0, j = 0;
  // trenutna duzina recenice
  int duz = 0;
  while(donjaGranica >= gornjaGranica && levaGranica <= desnaGranica){
    recenica[duz++] = tabela[i][j];

    int ni = i + di[s];
    int nj = j + dj[s];
    if (ni < gornjaGranica || ni > donjaGranica || nj < levaGranica || nj > desnaGranica) {
      switch(s){
      case 0:
        gornjaGranica++;
        break;
      case 1:
        desnaGranica--;
        break;
      case 2:
        donjaGranica--;
        break;
      case 3:
        levaGranica++;
        break;
```

```
      }

      s = (s + 1) % 4;
      i += di[s];
      j += dj[s];
    }
    else{
      i = ni;
      j = nj;
    }
  }

  int p = recenica.find(rec);

  if (p >= 0)
    cout << p + 1; // trazi se pozicija koja pocinje od 1, a ne od 0
  else
    cout << -1;

  return 0;
}
```

Задатак: Тачан израз

Аутор: Душан Пођадић

У израз $a\_b\_c = d$ уместо доњих црта треба убацити две рачунске операције тако да израз буде тачан. Рачунске операције које се могу користити су сабирање ($+$), одузимање ($-$), множење ($*$) и целобројно дељење ($/$). Важе стандардни приоритети рачунских операција.

Опис улаза

У сваком од четири реда стандардног улаза се уноси по један цео број (редом се уносе a, b, c и d). Сви бројеви су у опсегу $[1, 1000]$.

Опис излаза

На стандардни излаз исписати сва могућа решења (у произвољном редоследу), свако у посебном реду. Једно решење се исписује у форми 2 знака који представљају рачунске операција које је тим редом потребно убацити да би израз био тачан. Уколико није могуће постићи тачан израз, исписати nemoguce.

Пример 1

Улаз	*Излаз*
2 3 4 2	* - + / - /

Објашњење

- $*-$ је решење јер важи $2 * 3 - 4 = 2$
- $+/$ је решење јер важи $2 + 3/4 = 2$ (приметити да овде имамо целобројно дељење које је већег приоритета од сабирања и чији је резултат 0)
- $-/$ је решење јер важи $2 - 3/4 = 2$ (приметити да овде имамо целобројно дељење које је већег приоритета од сабирања и чији је резултат 0)

Пример 2

Улаз

```
8
3
2
12
```

Излаз

```
* /
```

Пример 3

Улаз

```
2
2
9
3
```

Излаз

```
nemoguce
```

Пример 4

Улаз

```
2
2
7
4
```

Излаз

```
nemoguce
```

Решење

У овом задатку је потребно генерисати све могуће комбинације рачунских операција, а затим те рачунске операције применити на учитане бројеве. У две угнежђене петље се прво генеришу сви могући парови рачунских операција и за сваки пар се рачуна вредност израза, та вредност се упоређује са траженим решењем и ако се поклапа, онда се испише тражени пар операција.

У функцији за рачунање израза је потребно водити рачуна о приоритетима операција. У случају да је прва операција + или -, а друга * или /, мора се прво извршити друга операција над својим операндима, па тек онда прва.

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

using namespace std;

// primeni operaciju op na operande a i b
int operacija(int a, int b, char op){
    if(op == '+')
        return a + b;
    if(op == '-')
        return a - b;
    if(op == '*')
        return a * b;
    if(op == '/')
        return a / b;
}

// Za zadati vektor od 3 broja i zadati vektor od 2 operacije resi izraz
int resi_izraz(vector<int>& s, vector<char>& ops){
  int r;
  if ((ops[0] == '+' || ops[0] == '-') &&
      (ops[1] == '*' || ops[1] == '/')) {
    // mnozenje i deljenje imaju prioritet
    // prvo primeni drugu operaciju na drugi i treci operand
    r = operacija(s[1], s[2], ops[1]);
```

```
    // zatim primeni prvu operaciju na prvi operand i gornji rezultat
    r = operacija(s[0], r, ops[0]);
  } else {
    // prvo primeni prvu operaciju na prvi i drugi operand
    r = operacija(s[0], s[1], ops[0]);
    // zatim primeni drugu operaciju na gornji rezultat i treci operand
    r = operacija(r, s[2], ops[1]);
  }
  return r;
}

int main(){
  vector<int> s(4);
  cin >> s[0] >> s[1] >> s[2] >> s[3];

  vector<char> ops(2);

  bool moguce = false;

  vector<char> op = {'+', '-', '*', '/'};

  for (int i = 0; i < op.size(); i++) {
    ops[0] = op[i];
    for (int k = 0; k < op.size(); k++) {
      ops[1] = op[k];
      int resenje = resi_izraz(s, ops);
      if (resenje == s[3]) {
        cout << ops[0] << " " << ops[1] << endl;
        moguce = true;
      }
    }
  }
  if (!moguce)
    cout << "nemoguce";

  return 0;
}
```

Задатак: Најмањи добитак

Аутор: Милан Вуīелија

Замислимо једну огромну матрицу попуњену нулама. Сваким потезом (a_i, b_i, c_i), сви елементи у првих a_i редова и првих b_i колона матрице се повећавају за по c_i. Која је минимална позитивна вредност у тој матрици после n потеза?

Опис улаза

У првом реду стандардног улаза је број потеза $n (1 \leq n \leq 50000)$. У сваком од наредних n редова су по три природна броја, a_i, b_i, c_i, раздвојени по једном размаком. Вредности a_i, b_i нису веће од милијарду, а збир свих c_i не прелази две милијарде. Сматрати да матрица има више од милијарду редова и више од милијарду колона.

Опис излаза

На стандардни излаз исписати један цео број, тражену минималну позитивну вредност.

Пример 1

Улаз	*Излаз*
4	3
2 3 1	
4 5 2	
1 2 1	
4 5 1	

Објашњење

Након сва 4 потеза, ова "бесконачна" матрица изгледа овако:

```
5 5 4 3 3 0 0 ...
4 4 4 3 3 0 0 ...
3 3 3 3 3 0 0 ...
3 3 3 3 3 0 0 ...
0 0 0 0 0 0 0 ...
0 0 0 0 0 0 0 ...
...
```

Најмања позитивна вредност у матрици је 3.

Пример 2

Улаз

```
2
3 7 3
4 5 2
```

Излаз

```
2
```

Објашњење

Након оба потеза, бесконачна матрица изгледа овако:

```
5 5 5 5 5 3 3 0 0 ...
5 5 5 5 5 3 3 0 0 ...
5 5 5 5 5 3 3 0 0 ...
2 2 2 2 2 0 0 0 0 ...
0 0 0 0 0 0 0 0 0 ...
0 0 0 0 0 0 0 0 0 ...
...
```

Најмања позитивна вредност у матрици је 2.

Решење

Редове и колоне поменуте велике матрице ћемо да бројимо од 0, тако да је горњи леви елемент матрице на позицији $(0, 0)$. Нека је S_i скуп елемената матрице, који се повећавају за c_i потезом (a_i, b_i, c_i). Први од тих елемената је $(0, 0)$, а последњи $(a_i - 1, b_i - 1)$.

Ако је S_j прави подскуп од S_i, онда потез (a_j, b_j, c_j) може да се игнорише, јер је елемент на позицији $(a_i - 1, b_i - 1)$ сигурно мањи бар за c_j од свих елемената скупа S_j. Да бисмо лакше утврдили које потезе можемо да изоставимо из разматрања, корисно је да након учитавања сортирамо потезе лексикографски (као торке, тј. растуће по броју редова, а за исти број редова растуће по броју колона). Када, напредујући кроз сортирани низ потеза стигнемо до потеза (a_i, b_i, c_i), сви претходни потези (a_j, b_j, c_j) имају исто или мање редова, тј. важи $a_j \leq a_i$ (тако смо сортирали потезе). Од тих претходних потеза, све оне за које важи $b_j \leq b_i$ можемо да изоставимо из разматрања, с тим да у случају да је баш $a_i = a_j, b_i = b_j$, j-ти потез можемо да спојимо са потезом (a_i, b_i, c_i) у $(a_i, b_i, c_i + c_j)$, након чега потез (a_j, b_j, c_j) може да се изостави из даљег разматрања.

На основу досадашње анализе можемо да закључимо да од претходних потеза треба да сачувамо само оне за које је број колона већи од текућег, тј. $b_j > b_i$. Такве потезе можемо да памтимо у посебном низу, а можемо и у првих k елемената истог низа, на позицијама од 0 до $k - 1$. Након избацивања свих небитних потеза, доњи десни углови преосталих потеза нису "прекривени" другим потезима, па је најмањи позитиван број у матрици једнак неком (јасно, најмањем) од бројева c_j међу преосталим потезима.

Поставља се још питање како можемо да одржавамо групу од k битних потеза на ефикасан начин. Довољно је да приликом обраде новог (i-тог) потеза крај $k-1$ почетног дела низа померамо улево све док је потезу на крају тог дела број колона мањи од b_i, а затим да текући потез допишемо на крај тог дела (упишемо га на позицију k и повећавамо k за 1). Поступајући на тај начин, на првих k позиција стално имамо низ потеза коме број колона опада, а број редова расте. Описани алгоритам је линеарне сложености, јер избацивање потеза из групе од првих k може да се изврши укупно највише онолико пута, колико пута се укупно и додаје потез у ту групу, а то је n (сваки потез се само једном додаје на крај групе).

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

using namespace std;

struct Pravougaonik {
  int w, h, n;
  bool operator <(Pravougaonik& p) {
    if (w < p.w) return true;
    if (w > p.w) return false;
    if (h < p.h) return true;
    return false;
  }
};

int main() {
  int n;
  cin >> n;
  vector<Pravougaonik> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i].w >> a[i].h >> a[i].n;

  sort(a.begin(), a.end());
  int k = 1;
  for (int i = 1; i < n; i++) {
    while (k > 0 && a[k - 1].h <= a[i].h) {
      k--;
      if (a[k].h == a[i].h && a[k].w == a[i].w)
        a[i].n += a[k].n;
    }
    a[k] = a[i];
    k++;
  }
  int m = a[0].n;
  for (int i = 1; i < k; i++)
    m = min(m, a[i].n);
  cout << m << endl;
  return 0;
}
```

1.3 Квалификације – трећи круг

Задатак: AM/PM

Аутори: Филип Марић, Владимир Кузмановић

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 променљиве 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 $[11, 0, 1, \ldots, 11, 0, ..., 10]$. Заиста, да не бисмо размишљали о дељењу негативних бројева, можемо прво уве-

ћати бројеве за 12, да бисмо добили интервал $[11, 24]$, чији су остаци редом заиста $[11, 0, 1, \ldots, 11, 0, ..., 10]$. 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: Из бајке у басну

Аутор: Милан Вуīделија

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*
170 60 20 6	124

Објашњење

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

```
85
90
14
15
```

Излаз

`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: Тениски резултат гем

Аутори: Филип Марић, Владимир Кузмановић

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		**Пример 2**		**Пример 3**		**Пример 4**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
`3` `1`	`40:15`	`0` `2`	`0:30`	`5` `6`	`40:ad`	`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: Врхови

Аутор: Иван Дрецун

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 n који представља величину низа. У наредном реду се уноси n 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

Улаз	*Излаз*
7 1 3 4 3 3 1 6	2

Пример 2

Улаз	*Излаз*
4 8 3 7 1	2

Пример 3

Улаз	*Излаз*
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;
```

```cpp
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: По три цифре

Аутори: Филип Марић, Владимир Кузмановић

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.

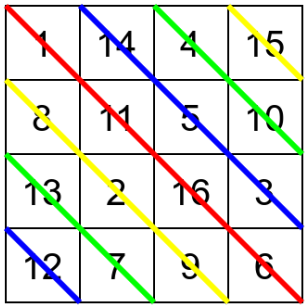
Описани поступак је приказан у решењу.

```
#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: Дијагонале

Аутори: Милан Вуђелија, Филип Марић

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.



Занимљив квадрат

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.

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 $S = a[0][0] + a[1][1] + \ldots + a[n-1][n-1]$.

Збир елемената следеће изломљене дијагонале (на слици плаве) добијамо као $S = a[0][1] + a[1][2] + \ldots + a[n-1][0]$. 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: Додата цифра стотина

Аутори: Филип Марић, Владимир Кузмановић

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`	`28`	`6`	`60`	`3`	`ne`
`26`		`11`		`27`	

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: Наизменична подела

Аутор: Оњен Нешковић

Анастасија и Бојан желе да поделе правоугаону торту димензија $1 \times n$, па су одлучили да играју једну игру.

Анастасија игра прва и сече торту вертикалним резом на два дела. Бојан затим узима део који жели за себе. Затим Бојан сече преостали део торте на два такође једним вертикалним резом. Затим Анастасија бира који од два дела жели за себе.

Торта је већ подељена на квадратиће 1×1*, па ће Анастасија и Бојан увек сећи по квадратићима, неће пресећи неки квадратић на пола.* Анастасија и Бојан наизменично секу и узимају делове док торта не постане димензија 1×1, тај део иде ономе ко је на реду да узима следећи. Анастасија и Бојан обоје желе да добију што већи број квадратића торте. Ваш задатак је да одредите колико највише парчића ће Анастасија добити ако и она и Бојан играју оптимално.

Опис улаза

У једном реду стандардног улаза се уноси природан број $n \leq 10^{10}$.

Опис излаза

На стандардни излаз исписати један природан број који представља број парчића које ће Анастасија добити.

Пример 1

Улаз	*Излаз*
7	2

Објашњење

Један начин на који се игра може одвити је следећи - торта је на почетку

#######

Анастасија прави рез и Бојан узима леви део.

BBBB|###

Сада је преостало за поделу:

###

Бојан прави рез и Анастасија узима десни део.

#|AA

Сада је преостало за поделу:

#

Торта је димензије 1×1, Анастасија је последња узела део, дакле сада је Бојан на реду и он узима ово парче. Анастасија је укупно узела 2 дела, а Бојан је узео 5.

Приметимо да на пример следеће не би могло да се деси јер Бојан игра оптимално:

BB|##### → AAAA|# → #

Овде би Анастасија добила 4 парчета, а Бојан 3.

Пример 2

Улаз

13

Излаз

4

Пример 3

Улаз

2

Излаз

1

Пример 4

Улаз

```
1
```

Излаз

```
0
```

Решење

```
#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;
}
```

Задатак: Обртања

Аутор: Милан Вуīделија

Дат је број N и број његових обртања r. Исписати да ли је број који се добија после r обртања броја N једнак броју N.

Обртањем броја се добија број који се пише истим цифрама у обрнутом редоследу. На пример, обртањем броја 310 се добија број 13, а обртањем броја 2227 се добија број 7222.

Опис улаза

У првом реду стандардног улаза се налази број N, а у другом број r. Оба броја су неозначени и цели, и важи $0 \leq N < 10^{50\ 000}$, $0 \leq r < 1\ 000\ 000$.

У тестовима вредним 70% поена важи $0 \leq N < 200\ 000$, $0 \leq r < 15$.

Опис излаза

На стандардни излаз исписати само реч `DA` или реч `NE`.

Пример 1

Улаз	*Излаз*
123999999999999999999999321 787657	DA

Пример 2

Улаз	*Излаз*
100000002000000300000004 458698	DA

Пример 3

Улаз	*Излаз*
10000000200000003000000040 2	NE

Решење

Обртањем се добија полазни број ако је испуњен бар један од следећих услова:

- број обртања је 0;
- број обртања је паран и број N се не завршава нулом;
- запис броја N је палиндром;

Ако ниједан од ових услова није испуњен, онда се обртањем не добија полазни број.

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

using namespace std;

int main() {
    string n, r;
    cin >> n >> r;

    string nObr = n; reverse(nObr.begin(), nObr.end());
    bool neGubiCifre = (n == "0") || n[n.size() - 1] != '0';
    bool brojObrtanjaJeParan = ((r[r.size() - 1] - '0') % 2 == 0);
    if (r == "0" || (neGubiCifre && brojObrtanjaJeParan) || n == nObr)
        cout << "DA" << endl;
    else
        cout << "NE" << endl;
    return 0;
}
```

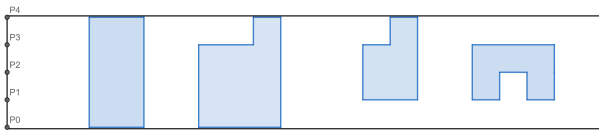
Задатак: Сеча дрва

Аутор: Оћьен Нешковић

Ана и Биљана спремају дрво за огрев за зиму. Ана је насекла комаде дрва разних величина и облика и истоварила их на траку. На траци се налази тестера која сече комаде дрвета на два дела. Тестера се поставља на неку позицију на почетку, затим се укључује трака и дрва се крећу ка тестери. Када комад дрвета дође до тестере он бива пресечен на два дела, један део пада са леве, други са десне стране траке. *У току сечења се тестера не може померати.*

Биљана треба да одлучи где да постави тестеру тако да се сво дрво подели што боље. Ана и Биљана су одлучиле да након што се заврши сечење, *Ана ће узети све што је пало са леве стране, а Биљана све што је пало са десне стране. Биљана хоће да постави тестеру тако да она и Ана добију што сличнију количину дрвета.*

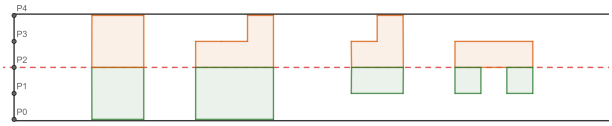
Биљана има прилично напредну машину за сечење која је сликала све комаде дрвета кад су били постављени и забележила облике приближно као квадратиће.



Пример података са Биљанине машине

Биљана може поставити тестеру на позиције: $0, 1, 2, ..., k$. На пример, на претходној слици Биљана може поставити тестеру на позиције $0, 1, 2, 3$ и 4. Уколико постави тестеру на позицију 2 дрво ће бити подељено на следећи начин:

Ана ће у овом случају добити $4 + 4 + 3 + 3 = 14$ комада дрвета, а Биљана: $4 + 6 + 2 + 2 = 14$.



Подељени комади дрвета

Помозите Биљани и напишите програм који ће за дате облике дрвета пронаћи најбоље место за тестеру тако да је апсолутна разлика комада дрвета које Ана и Биљана добију што мања.

Опис улаза

У првом реду стандардног улаза уноси се природан број $n \leq 5 \cdot 10^5$ - број комада дрвета на траци.

У другом реду стандардног улаза уноси се природан број $k \leq 5 \cdot 10^5$ - највећа димензија дрвета.

Затим се уноси n матрица димензије $k \times k$. Свака од матрица сачињена је од k редова од по k бројева (0 - машина није детектовала дрво на том месту или 1 - машина је детектовала дрво на том месту). У свим тест примерима важи $n \cdot k \cdot k \leq 5 \cdot 10^5$.

Опис излаза

У једом реду стандардног излаза исписати минималну апсолутну разлику између комада дрвета које ће Ана и Биљана добити ако се тестера постави оптимално.

Пример 1

Улаз	*Излаз*
4	0
4 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 1 1 1 0 1 1 1 0 1 1 1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0	

Објашњење

Ово је пример из текста задатка.

Пример 2

Улаз

```
4
3
1 1 1
1 0 1
0 0 0
1 0 1
1 0 1
1 1 0
0 0 0
```

```
0 1 1
0 0 1
0 0 0
1 1 0
1 1 0
```

Излаз

```
8
```

Објашњење

Разлика 8 се постиже ако се тестера постави на позицију 1 или на позицију 2.

Пример 3

Улаз

```
3
5
1 0 0 0 0
1 1 0 0 1
0 0 0 0 1
0 0 0 0 1
0 0 0 0 0
0 1 1 0 1
0 1 0 0 0
0 0 0 0 0
0 1 0 0 0
0 0 0 0 0
0 1 1 1 0
0 0 0 0 0
0 0 1 1 0
0 0 1 1 1
0 0 0 1 0
```

Излаз

```
2
```

Објашњење

Разлика 2 се постиже ако се тестера постави на позицију 3.

Решење

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

using namespace std;

int saberi_red(const vector<vector<int>>& mat, int r)
{
  int suma = 0;
  for (int kol = 0; kol < mat[0].size(); kol++)
    suma += mat[r][kol];
  return suma;
}

int main()
{
  int n, k; cin >> n >> k;
  vector<vector<int>> mat(k, vector<int>(n*k));
  int ukupno_drveta = 0;
  for (int i = 0; i < n; i++)
```

```
    for (int r = 0; r < k; r++)
      for (int c = 0; c < k; c++) {
        cin >> mat[r][i*k+c];
        ukupno_drveta += mat[r][i * k + c];
      }

  int kolicina_a = 0;
  int min_razlika = ukupno_drveta;
  for (int poz = 0; poz < k; poz++) {
    kolicina_a += saberi_red(mat, poz);
    int kolicina_b = ukupno_drveta - kolicina_a;
    min_razlika = min(min_razlika, abs(kolicina_a - kolicina_b));
  }
  cout << min_razlika << endl;
  return 0;
}
```

Задатак: Што сличнија тројка

Аутори: Филип Марић, Душан Пејадић

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

```
5
19 7 13 17 10
20 9 1 4 12
2 14 8 29 23
```

Излаз

```
2
```

Објашњење

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

```
7
9 18 4 3 2 17 9
5 17 3 1 9 20 6
52 44 37 17 4 52 55
```

Излаз

```
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 (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 $(n \log(n))$. 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: Ко први игра

Аутор: Иван Дрецун

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: Контролна цифра

Аутор: Филип Марић

Кредитне картице имају 16 цифара, од којих се последња користи као контролна и израчунава се на основу осталих 15 на основу следећег алгоритма. Цифре се деле у две групе: 8 цифара које су на парним позицијама

(када се позиције броје од нуле, слева надесно) и 7 цифара које су на непарним позицијама. Свака од цифара на парним позицијама се множе са два, па ако се тако добије двоцифрени број, онда се сабирају цифре тог двоцифреног броја. Цифре на непарним позицијама се не мењају. На крају се израчунава збир свих цифара са непарних позиција и свих трансформисаних цифара са парних позиција. Контролна цифра се одређује тако да се када се она сабере са добијеним збиром добије број који је дељив са 10.

Опис улаза

Са стандардног улаза се уноси низ од 15 цифара.

Опис излаза

На стандардни излаз исписати контролну цифру.

Пример 1

Улаз	*Излаз*
`430239720231900`	`7`

Објашњење

Цифре на парним позицијама су `4.0.3.7.0.3.9.0`, а на непарним позицијама су `.3.2.9.2.2.1.0.`Цифре на парним позицијама се трансформишу и добија се `8.0.6.5.0.6.9.0` (множењем са 2 броја 7 добија се 14, па се сабирањем његових цифара добија 5, док се множењем са 2 броја 9 добија 18, па се сабирањем његових цифара добија 9). Затим израчунавамо збир цифара на непарним позицијама и трансформисаних цифара на парним позицијама и добијамо $3+2+9+2+2+1+0$ и $8+0+6+5+0+6+9+0$ и тако добијамо збир $19+34=53$. Одатле следи да контролна цифра мора да буде 7 да би се њеним додавањем на 53 добио број дељив са 10.

Пример 2

Улаз

`450439730231920`

Излаз

`0`

Решење

Алгоритам описан у овом задатку се назива Лунов алгоритам и заиста се користи за контролу кредитних картица.

Број картице учитавамо као стринг тј. ниску карактера (тај број нас не занима као целина, већ нас занимају само његове појединачне цифре, не може да стане у опсег целобројног типа, а може да почне и нулом, па је сасвим природно репрезентовати га ниском карактера).

Затим се пролази кроз све цифре и рачуна се збир на начин који је описан у тексту задатка.

Прво се свака цифра (која је карактер у ниски) претвара у своју бројну вредност коју добијамо умањивањем карактера за ASCII/UNICODE код цифре `0`, користећи израз: `s[i] - '0'`.

Уколико је цифра била на парном месту потребно је да се она помножи са 2 и да се онда саберу цифре новодобијеног броја, иначе се цифра не мења. Нова цифра се додаје на збир.

Када се израчуна збир, треба одредити која цифра се може додати на тај збир да би он био дељив са 10. То је најлакше урадити тако што се од 10 одузме последња цифра збира. Потребно је обратити пажњу на специјалан случај када је оригинални збир већ дељив са 10 јер ће онда одузимањем последње цифре (0) од 10 добити 10. Пошто треба да одредимо контролну цифру, а 10 није цифра, контролна цифра ће у том случају бити 0. Ово се може имплементирати помоћу гранања, али се може користити и пречица тако што се добијена крајња цифра узме по модулу 10. Ако је добијена контролна цифра мања од 10 онда се ништа не мења, а ако је тачно 10 (тј. није заправо цифра), онда ће резултат бити 0 што смо и желели.

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

using namespace std;
```

```
int main() {
  string s;
  cin >> s;
  int zbir = 0;
  for (int i = 0; i < s.size(); i++) {
    int c = s[i] - '0';
    if (i % 2 == 0) {
      c = c * 2;
      if (c >= 10)
        c = (c / 10) + (c % 10);
    }
    zbir += c;
  }
  cout << (10 - zbir % 10) % 10 << endl;
  return 0;
}
```

Задатак: Мешалица

Аутор: Иван Дрецун

На располагању нам је мешалица која може да меша редослед елемената низа. Мешалица елементе увек меша на исти начин. Нека је дат низ који се добија мешањем низа `0 1 2 ... n-1`. Напиши програм који одређује низ који се добија мешањем низа `0 1 2 ... n-1` m пута.

Опис улаза

Са стандардног улаза се уносе природни бројеви n ($1 \leq n \leq 50000$) и m ($0 \leq m \leq 10^{18}$) одвојени размаком. У наредном реду се уноси n различитих бројева између 0 и $n-1$ одвојених размаком, који представљају елементе низа `0 1 2 ... n-1` након једног мешања.

У тест примерима вредним 70 поена важе додатна ограничења $n, m \leq 100$.

Опис излаза

На стандардни излаз исписати n различитих природних бројева између 0 и $n-1$ одвојених размаком који представљају елементе низа `0 1 2 ... n-1` након m мешања.

Пример 1

Улаз	*Излаз*
`4 2` `0 2 3 1`	`0 3 1 2`

Објашњење

Почетни низ је `0 1 2 3`. Првим мешањем добија се `0 2 3 1`. Другим мешањем добија се `0 3 1 2`.

Пример 2

Улаз

```
5 3
1 2 0 4 3
```

Излаз

```
0 1 2 4 3
```

Објашњење

Почетни низ је `0 1 2 3 4`. Првим мешањем добија се `1 2 0 4 3`. Другим мешањем добија се `2 0 1 3 4`. Трећим мешањем добија се `0 1 2 4 3`.

Пример 3

Улаз

```
4 3
1 0 3 2
```

Излаз

```
1 0 3 2
```

Објашњење

Почетни низ је `0 1 2 3`. Првим мешањем добија се `1 0 3 2`. Другим мешањем добија се `0 1 2 3`. Трећим мешањем добија се `1 0 3 2`.

Решење

Симулација мешања

Мешањем (можемо рећи и пермутовањем) низа a мешалицом (можемо рећи и пермутацијом) p добија се низ који на свакој позицији i садржи елемент a_{p_i}. Решење грубом силом подразумева да се почетни низ који садржи елементе од 0 до $n-1$ промеша (можемо рећи и пермутује) m пута мешалицом p.

```cpp
#include <iostream>
#include <vector>
#include <numeric>

using namespace std;

// niz p permutujemo permutacijom q
vector<int> mesaj(int n, const vector<int>& p, const vector<int>& q) {
  vector<int> r(n);
  for(int i = 0; i < n; i++)
    r[i] = p[q[i]];
  return r;
}

int main() {
  int n;
  int64_t m;
  cin >> n >> m;

  // ucitavamo permutaciju p
  vector<int> p(n);
  for(int i = 0; i < n; i++)
    cin >> p[i];

  // popunjavamo niz r elementima 0, 1, ..., n-1
  vector<int> r(n);
  iota(begin(r), end(r), 0);

  // permutujemo r permutacijom p, m puta tj. permutacijom p^m
  for (int i = 0; i < m; i++)
    r = mesaj(n, p, r);

  // ispisujemo rezultat
  for(auto x : r)
    cout << x << ' ';
  cout << endl;

  return 0;
}
```

Брзо степеновање

Дејство мешалице p на низ a можемо означити са $p \cdot a$. Тада наш програм треба да израчуна резултат $p \cdot p \cdot p \cdot \ldots \cdot p \cdot a$, где је a почетни низ. Пошто су мешалице низови, мешалицама се може деловати и на мешалице (производ две мешалице је нова мешалица). Операција $\cdot$ је асоцијативна па се може дефинисати и степен мешалице и наш резултат тражи да се израчуна $p^m \cdot a$. Израчунавање степена за великом m грубом силом је споро, али се може много брже израчунати познатим алгоритмом брзог степеновања.

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

using namespace std;

// niz p permutujemo permutacijom q
vector<int> mesaj(int n, const vector<int>& p, const vector<int>& q) {
  vector<int> r(n);
  for(int i = 0; i < n; i++)
    r[i] = p[q[i]];
  return r;
}

int main() {
  int n;
  int64_t m;
  cin >> n >> m;

  // ucitavamo permutaciju p
  vector<int> p(n);
  for(int i = 0; i < n; i++)
    cin >> p[i];

  // upisujemo u niz r elemente 0, 1, ..., n-1
  vector<int> r(n);
  iota(begin(r), end(r), 0);

  // permutujemo niz r permutacijom p^m koju izracunavamo algoritmom
  // brzog stepenovanja
  while (m > 0) {
    if (m % 2 == 1)
      r = mesaj(n, r, p);
    p = mesaj(n, p, p);
    m /= 2;
  }

  // ispisujemo rezultat
  for(auto x : r)
    cout << x << ' ';
  cout << endl;

  return 0;
}
```

Задатак: Сеча стабала

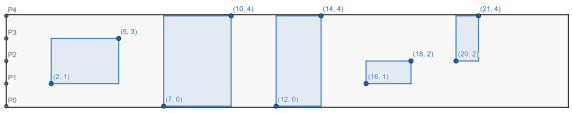
Аутор: Оћьен Нешковић

Ана и Биљана спремају дрво за огрев за зиму. Ана је насекла комаде дрва разних димензија и истоварила их на траку. На траци се налази тестера која сече комаде дрвета на два дела. Тестера се поставља на неку позицију на почетку, затим се укључује трака и дрва се крећу ка тестери. Када комад дрвета дође до тестере

он бива пресечен на два дела, један део пада са леве, други са десне стране траке. *У току сечења се тестера не може померати.*

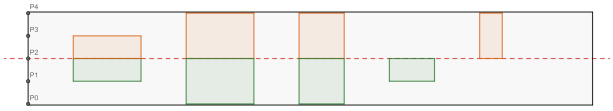
Биљана треба да одлучи где да постави тестеру тако да се сво дрво подели што боље. Ана и Биљана су одлучиле да након што се заврши сечење, *Ана ће узети све што је пало са леве стране, а Биљана све што је пало са десне стране. Биљана хоће да постави тестеру тако да она и Ана добију што сличнију количину дрвета.*

Комади дрвета су сви прилично правилног облика. Биљана има напредну машину за сечење која је сликала све комаде дрвета кад су били постављени и забележила облике приближно као правоугаонике.



Пример података са Биљанине машине

Биљана може поставити тестеру на позиције: $0, 1, 2, \ldots$. На пример, на претходној слици, уколико Биљана постави тестеру на позицију 2 дрво ће бити подељено на следећи начин:



Подељени комади дрвета

Ана ће у овом случају добити $3 + 6 + 4 + 2 = 15$ комада дрвета, а Биљана: $3 + 6 + 4 + 2 = 15$

Помозите Биљани и напишите програм који ће за дате облике дрвета пронаћи најбоље место за тестеру тако да је апсолутна разлика комада дрвета које Ана и Биљана добију што мања.

Опис улаза

У првом реду стандардног улаза уноси се природан број $n \leq 5 \cdot 10^5$ - број комада дрвета на траци.

У наредних n линија стандардног улаза уносе се по 4 броја раздвојена белинама $0 \leq x_0, y_0, x_1, y_1 \leq 10^9$.

Који представљају редом x и y координате доњег левог темена правоугаоника, x и y координату горњег десног темена правоугаоника.

У 10% тест примера ће важити $n \leq 20$ и $\max_y \leq 200$ где је $\max_y$ највећа унета y-координата.

У 25% тест примера ће важити $n \cdot \max_y \leq 5 \cdot 10^5$.

Опис излаза

У једом реду стандардног излаза исписати минималну апсолутну разлику која се може постићи ако се тестера позиционира оптимално.

Пример 1

Улаз	*Излаз*

`5`
`2 1 5 3`
`7 0 10 4`
`12 0 14 4`
`16 1 18 2`
`20 2 21 4`
	`0`

Објашњење

Ово је пример из текста задатка.

Пример 2

Улаз

```
5
2 3 5 5
5 0 7 2
8 1 10 5
11 2 13 7
14 0 18 1
```

Излаз

```
4
```

Објашњење

Најмања разлика (4) се постиже ако се тестера постави на позицију 3.

Решење

Задатак се ефикасно решава бинарном претрагом по решењу. Наиме померањем тестере разлика између левог и десног дела све време расте (креће од негативних вредности јер када је тестера на позицији 0 сво дрво пада на десну страну, а завршава се на позитивним вредностима јер када је тестера на максималној позицији сво дрво пада на леву страну). Циљ је пронаћи позицију тестере такву да је разлика што ближа нули.

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

struct tacka
{
  long long x, y;
};
struct pravougaonik
{
  tacka l, r;
};

inline long long povrsina(const pravougaonik& p)
{
  return (p.r.x - p.l.x) * (p.r.y - p.l.y);
}

// Racuna razliku izmedju dela koji ce dobiti Ana i Biljana ako se testera stavi na poz
long long razlika(const vector<pravougaonik>& pravougaonici, long long poz)
{
  long long a = 0, b = 0;
  for (int i = 0; i < pravougaonici.size(); i++) {
    // pravougaonik je skroz ispod testere
    if (poz >= pravougaonici[i].r.y)
      b += povrsina(pravougaonici[i]);

    // Pravougaonik je skroz iznad testere
    else if (poz <= pravougaonici[i].l.y)
      a += povrsina(pravougaonici[i]);

    // Pravougaonik ce biti presecen
    else {
      tacka presek_l = { pravougaonici[i].l.x,poz };
      tacka presek_d = { pravougaonici[i].r.x,poz };
      pravougaonik gornji = { presek_l,pravougaonici[i].r };
      pravougaonik donji = { pravougaonici[i].l,presek_d };
      a += povrsina(gornji);
      b += povrsina(donji);
    }
```

```
  }
  return a - b;
}

int main()
{
  ios::sync_with_stdio(false);
  cin.tie(0);
  int n; cin >> n;
  vector<pravougaonik> pravougaonici(n);
  long long maxy = 0;
  for (int i = 0; i < n; i++) {
    cin >> pravougaonici[i].l.x >> pravougaonici[i].l.y;
    cin >> pravougaonici[i].r.x >> pravougaonici[i].r.y;
    maxy = max(maxy, pravougaonici[i].r.y);
  }

  long long l = 0, r = maxy;
  long long pl = razlika(pravougaonici, l), pr = razlika(pravougaonici, r);
  while (r - l > 1) {
    long long m = l + (r - l) / 2;
    long long p = razlika(pravougaonici, m);
    if (p >= 0) {
      l = m;
      pl = p;
    } else {
      r = m;
      pr = p;
    }
  }

  cout << min(abs(pl), abs(pr)) << endl;
  return 0;
}
```