Садржај

1 Квалификације (1. круг) 3

Задатак: Цифре се преклапају . . . 3
Задатак: Највећа предност домаћих . . . 4
Задатак: Промена школе . . . 5
Задатак: Топ . . . 6
Задатак: Набрајање . . . 7
Задатак: Мица штуцалица . . . 9
Задатак: Погрешна операција . . . 11
Задатак: Разврставање . . . 12
Задатак: Најдаљи најближи . . . 13
Задатак: Wifi снага . . . 15
Задатак: Мах НЗД . . . 19
Задатак: Недостајућа страница четвороугла . . . 20
Задатак: Ексел . . . 21
Задатак: Крађа дијаманта . . . 23
Задатак: Број сегмената са производом нула . . . 26
Задатак: Један-два-три-четири . . . 28

2 Квалификације (2. круг) 31

Задатак: Закуцавање . . . 31
Задатак: Непознати број . . . 32
Задатак: Ски пас . . . 33
Задатак: Гориво . . . 35
Задатак: Прва и последња цифра . . . 38
Задатак: Најгушћи подскуп . . . 39
Задатак: Месец рођења . . . 40
Задатак: Врабац и мачка . . . 42
Задатак: Неравнотежа . . . 44
Задатак: Дорћол опен . . . 46
Задатак: Непарни делиоци . . . 49
Задатак: Шибице . . . 51
Задатак: Биоскоп . . . 53
Задатак: Најбрже до циља . . . 54
Задатак: Најмање промена . . . 55
Задатак: Амљез . . . 58

3 Општинско такмичење 65

Задатак: Дани у недељи . . . 65
Задатак: Топлотни појас . . . 66
Задатак: Робот достављач . . . 67
Задатак: Минимах . . . 69
Задатак: Обим паралелограма . . . 72
Задатак: Адресе . . . 73
Задатак: Слепо куцање речи . . . 75
Задатак: Одбојкашки резултат . . . 76
Задатак: Бејзбол хитац . . . 79

Задатак: Годишње доба 80
Задатак: Пар-непар у круг 82

4 Окружно такмичење **85**
Задатак: Обим или површина 85
Задатак: Заједничка другарица 85
Задатак: Поправка степеништа 87
Задатак: Замени највећу цифру 89
Задатак: Путовање 89
Задатак: Производ простих 90
Задатак: Реченице у заградама 92
Задатак: Такмичари 93
Задатак: Троугао највећег обима 95
Задатак: Турнир 95
Задатак: Маказице 97
Задатак: Конопци 99

5 Државно такмичење **103**
Задатак: Мотивација 103
Задатак: Харсхад бројеви 104
Задатак: Ђаволски квадрат 105
Задатак: Патикарница 108
Задатак: Тешке торбе 112
Задатак: Квантна заврзлама 114
Задатак: Ротирање речи 116
Задатак: Бициклисти 120
Задатак: Горе-доле-лево-десно 122
Задатак: Збир простих 125
Задатак: Домине 128

Глава 1

Квалификације (1. круг)

Задатак: Цифре се преклапају

Аутор: Душан Пођадић

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

Опис улаза

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

Опис излаза

Исписати тражени петоцифрен број.

Пример 1

Улаз	*Излаз*
145 563	14563

Објашњење

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

Пример 2

Улаз

```
788
851
```

Излаз

```
78851
```

Решење

Решење задатка

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

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

```cpp
#include <iostream>

using namespace std;

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

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

```cpp
#include <iostream>

using namespace std;

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

Задатак: Највећа предност домаћих

Аутор: Милан Вујделија

Дат је резулатат кошаркашке утакмице након сваке четвртине. Колика је највећа могућа предност домаћих?

Опис улаза

У сваком од четири реда стандардног улаза налазе се по два цела ненегативна броја раздвојена размаком. Сваки ред улаза одговара резултату након једне четвртине редом. Први број у реду је број поена које је постигла домаћа екипа, а други број је број поена које је постигла гостујућа екипа. Ниједан од бројева на улазу није већи од 200.

Опис излаза

На стандардни излаз исписати само један цео број, највећу могућу предност домаће екипе. Ако домаћа екипа ни у једном тренутку није могла да има предност, исписати 0.

Пример 1

Улаз	*Излаз*
21 18 39 40 63 54 78 77	24

Објашњење

Резултат је у једном тренутку могао да буде $78 : 54$, што је 24 поена предности.

Пример 2

Улаз

```
0 1
1 3
2 5
4 9
```

Излаз

```
0
```

Објашњење

Домаћа екипа никада није имала предност. Најповољнији резулатат за њу је био на почетку утакцмице ($0 : 0$).

Решење

Нека су резултати по четвртинама редом $a_1 : b_1$, $a_2 : b_2$, $a_3 : b_3$ и $a_4 : b_4$. Најповољнији резултат за домаћу екипу током прве четвртине је могао да буде $a_1 : 0$, док су најповољнији могући резултати током друге, треће и четврте четвртине редом $a_2 : b_1$, $a_3 : b_2$, $a_4 : b_3$. Ови резултати настају ако током сваке четвртине прво домаћа екипа постигне све своје поене, а затим гостујућа све своје.

Према томе, највећа вођства или најмањи заостаци домаћих у поенима по четвртинама су a_1, $a_2 - b_1$, $a_3 - b_2$, $a_4 - b_3$. Тражени резултат је највећи од ова четири броја. Приметимо да највећи од ових бројева не може да буде негативан, јер је $a_1 \geq 0$.

```cpp
#include <iostream>

using namespace std;

int main() {
    int a1, b1, a2, b2, a3, b3, a4, b4;
    cin >> a1 >> b1 >> a2 >> b2 >> a3 >> b3 >> a4 >> b4;
    int prednost = max(max(a1, a2-b1), max(a3-b2, a4-b3));
    cout << prednost << endl;
    return 0;
}
```

Задатак: Промена школе

Аутор: Душан Пођадић, Јелена Петровић

На крају првог разреда средње школе Ивана је одлучила да се пребаци из медицинске школе у гимназију. У гимназији су одлучили да ће јој као просек у првом разреду рачунати само одређене предмете - оне који постоје у обе школе. Закључено је да се k Иваниних оцена неће рачунати за просек у новој школи. Одредити нови Иванин просек оцена ако је познат просек из медицинске школе, укупан број предмета које је слушала у медицинској (n) и k оцена које јој се неће рачунати. Претпоставити да је Ивана завршила разред, тј. да није имала ниједну закључену јединицу.

Просек оцена $a_1, a_2, ..., a_n$ рачуна се на следећи начин:

$$\frac{a_1 + a_2 + ... + a_n}{n}.$$

Опис улаза

У првом реду стандардног улаза налази се природан број n, а у другом реду број k ($1 \leq k < n \leq 50$). У наредних k редова је дато k Иваниних оцена (природни бројеви од 2 до 5) које се не рачунају. У последњем реду се налази број p који представља Иванин просек из медицинске школе који је заокружен на две децимале.

Опис излаза

Исписати нови Иванин просек, заокружен на 2 децимале.

Пример 1

Улаз	*Излаз*
3 1 4 4.33	4.50

Пример 2

Улаз	*Излаз*
12 3 4 5 4 4.58	4.67

Решење

Нека су $a_1, a_2, ..., a_n$ Иванине оцене на крају првог разреда средње медицинске школе, где су са $a_1, ..., a_k$ означене оцене које се неће рачунати. Просек свих оцена, означен са p, рачуна се на следећи начин:

$p = \frac{a_1+a_2+...+a_n}{n}$.

Са $\hat{p}$ ћемо означити број p заокружен на две децимале. Како су $\hat{p}$ и $a_1, ..., a_k$ познати, збир преосталих оцена можемо добити користећи претходну формулу, водећи притом рачуна о заокруживању јер су оцене природни бројеви:

$a_{k+1} + ... + a_n = p \cdot n - (a_1 + ... + a_k) = \text{round}(\hat{p} \cdot n) - (a_1 + ... + a_k)$.

Пошто преосталих оцена има $n-k$, њихов просек се може израчунати дељењем израчунатог збира преосталих оцена са $n - k$ (из услова задатка важи $n - k > 0$):

$\frac{a_{k+1}+...+a_n}{n-k} = \frac{\text{round}(\hat{p}\cdot n)-(a_1+...+a_k)}{n-k}$.

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

using namespace std;

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

    int zbirOcena = 0;
    for (int i = 0; i < k; ++i)
    {
        int ocena;
        cin >> ocena;
        zbirOcena += ocena;
    }

    double prosek;
    cin >> prosek;

    double noviProsek = ((int)round(prosek*n) - zbirOcena) / (n-k);
    cout << fixed << showpoint << setprecision(2) <<  noviProsek << endl;

    return 0;
}
```

Задатак: Топ

Аутор: Иван Дрецун

Мали Душан је тек почео да учи како се игра шах. Учитељ му је на шаховску таблу ставио једног топа. Топ је фигура која може да нападне било које поље у врсти и колони у којој се налази. Врсте на шаховској табли означене су бројевима од 1 до 8, а колоне словима од `A` до `H` енглеске абецеде. Малог Душана занима да ли ће топ моћи да нападне његову фигуру ако је стави на неко поље. Напиши програм који даје Малом Душану одговор на то питање за сва поља за која га то занима.

Опис улаза

Са стандардног улаза се уноси позиција топа у стандардном шаховском запису. У наредном реду се уноси број n који представља број питања која Мали Душан поставља. Након тога се у наредних n редова уносе позиције поља за које Мали Душан жели да добије одговор.

Опис излаза

На стандардни излаз за свако питање исписати `DA` уколико топ напада то поље, односно `NE` у супротном. Одговоре исписивати у засебним редовима.

Пример

Улаз	*Излаз*
A4	NE
3	DA
B7	NE
D4	
H2	

Решење

Опис главног решења

У овом блоку се описује главно решење задатка.

```
#include <iostream>

using namespace std;

int main() {
  string top;
  cin >> top;

  int n;
  cin >> n;

  for (int i = 0; i < n; i++) {
    string polje;
    cin >> polje;

    if (top[0] == polje[0] || top[1] == polje[1])
      cout << "DA" << endl;
    else
      cout << "NE" << endl;
  }

  return 0;
}
```

Задатак: Набрајање

Аутор: Милан Вуīделија

Написати програм који учитава речи док не учита реч `gotovo`, а исписује реченицу којом се набрајају све претходне речи.

Опис улаза

У сваком реду стандардног улаза налази се по једна реч, написана малим словима енглеске абецеде. Речи нису дуже од 30 слова. Последња реч је реч “gotovo”. Укупан број речи је најмање 2, а највише 30.

Опис излаза

На стандардни излаз исписати све речи које претходе речи `gotovo`, тако да између узастопних речи стоји запета и размак, осим између последње две, где стоји слово `i` са размацима око њега.

Пример 1

Улаз	*Излаз*
perica	perica
gotovo	

Пример 2

Улаз	*Излаз*
lala	lala i sosa
sosa	
gotovo	

Пример 3

Улаз	*Излаз*

`vuk`
`lija`
`meda`
`zeka`
`magarac`
`gotovo`
	`vuk, lija, meda, zeka i magarac`

Решење

Пошто прва реч није реч `gotovo` (укупан број речи је бар 2), прву учитану реч можемо одмах да испишемо. У случају да је друга реч `gotovo`, поступак је већ завршен. Ако друга реч није `gotovo`, ипак не можемо да је испишемо пре него што прочитамо и следећу реч, јер тек тада знамо да ли пре друге речи треба да напишемо зарез или слово `i`.

Закључујемо да је потребно да програм у сваком тренутку рада памти бар две последње учитане речи, да би могао да испише прву од те две речи. Назовимо зато последње две учитане речи *претходна* и *следећа* реч. Сада цео поступак можемо да формулишемо овако:

- учитај и испиши прву реч
- учитај следећу реч
- све док та следећа реч није реч `gotovo`:
  - нека претходна реч преузме вредност од следеће
  - учитај нову следећу реч
  - ако је та следећа реч `gotovo`, испиши слово `i` и претходну реч, иначе испиши зарез и претходну реч
- заврши исписивање текућег реда

```
#include <iostream>

using namespace std;

int main() {
    string prethodna, sledeca;
    cin >> prethodna >> sledeca;
    cout << prethodna;
    while (sledeca != "gotovo")
    {
        prethodna = sledeca;
        cin >> sledeca;
        if (sledeca == "gotovo")
            cout << " i " << prethodna;
        else
            cout << ", " << prethodna;
    }
    cout << endl;
    return 0;
}
```

Претходно решење може да се измени тако да се резултат накупља у једној текстуалној променљивој и да се оспише тек на крају. Такво решење је прихватљиво само у случају да укупан број речи није велики (нпр. мањи је од хиљаду).

```
#include <iostream>

using namespace std;

int main() {
    string prethodna, sledeca;
    cin >> prethodna >> sledeca;
    string odgovor = prethodna;
```

```
    while (sledeca != "gotovo")
    {
        prethodna = sledeca;
        cin >> sledeca;
        if (sledeca == "gotovo")
            odgovor = odgovor + " i " + prethodna;
        else
            odgovor = odgovor + ", " + prethodna;
    }
    cout << odgovor << endl;
    return 0;
}
```

Задатак: Мица штуцалица

У соби има n сијалица, од којих свака има свој прекидач. На почетку су неке сијалице укључене, а неке искључене. Паја је таман толико порастао да може да притиска прекидаче, што га веома забавља. Паја је m пута притиснуо неки од прекидача. Сваки пут када је у соби био потпуни мрак, мала Милица је штуцнула.

Написати програм, који за дато почетно стање сијалица и редослед притискања прекидача одређује колико пута је Милица штуцнула.

Опис улаза

У првом реду стандардног улаза је природан број n, $n \leq 50000$.

У другом реду је n бројева раздвојених по једним размаком, од којих је сваки једнак 0 или 1. Нуле означавају искључене сијалице, а јединице укључене. Ови бројеви нису сви једнаки нули.

У трећем реду је природан број m, $m \leq 50000$.

У четвртом реду је m бројева раздвојених размаком, који представљају редом индексе (тј. редне бројеве, бројећи од 0) сијалица чије прекидаче је редом Паја притискао.

Опис излаза

На стандардни излаз исписати један неозначен цео број, који означава колико пута је мала Милица штуцнула.

Пример 1

Улаз	*Излаз*
5 1 1 0 1 0 9 0 1 2 3 2 1 0 1 0	2

Објашњење

Стања сијалица после притикања прекидача су редом

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

Према томе, Милица је штуцнула после петог и после деветог притиска прекидача, што је укупно два штуцања.

Пример 2

Улаз

```
3
1 1 1
12
0 1 0 1 0 1 0 1 2 1 2 1
```

Излаз

```
0
```

Решење

Свакако нам је потребна једна целобројна променљива помоћу које бројимо колико пута су све сијалице биле искључене, тј. колико пута је мала Мица штуцнула. Ову променљиву ћемо звати бројач штуцања. Поред тога, да бисмо могли да утврдимо да ли су све сијалице искључене, треба да имамо неки низ у коме памтимо стање свих сијалица (нуле за искључене, јединице за укључене). Нека је то низ a.

Када се притисне прекидач са индексом p, елемент a_p треба да промени вредност из 0 у 1 или обрнуто, тј. треба да добије вредност $1-a_p$. Након сваке промене, тј. притиска на неки од прекидача, треба да установимо да ли су све сијалице искључене.

Једна могућност је да после сваке промене пребројимо укључене сијалице, па ако укључених сијалица има 0, да повећамо бројач штуцања. Број операција у овом решењу је сразмеран производу броја сијалица и броја притисака на прекидаче. То може да буде до $50\,000 \cdot 50\,000 = 2\,500\,000\,000$ операција, а тај број операција не може да се изврши у предвиђеном времену.

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

using namespace std;

int main() {
    int m, n, p, stuc = 0;
    cin >> n;
    vector<int> a(n);
    for(int i = 0; i < n; i++) {
        cin >> a[i];
    }

    cin >> m;
    for(int i = 0; i < m; i++) {
        cin >> p;
        a[p] = 1 - a[p];
        bool sveIsklj = true;
        for(int i = 0; i < n; i++)
            sveIsklj = sveIsklj && (a[i] == 0);

        if (sveIsklj)
            stuc++;
    }
    cout << stuc << endl;
    return 0;
}
```

Нешто боље решење је да после сваке промене тражимо само прву укључену сијалицу, па ако је не нађемо, да повећамо бројач штуцања. Нажалост, и даље постоји врло неповољан случај, а то је када је прва укључена сијалица при крају низа.

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

using namespace std;

int main() {
```

```
    int m, n, p, stuc = 0;
    cin >> n;
    vector<int> a(n);
    for(int i = 0; i < n; i++) {
        cin >> a[i];
    }

    cin >> m;
    for(int i = 0; i < m; i++) {
        cin >> p;
        a[p] = 1 - a[p];
        bool sveIsklj = true;
        for(int i = 0; i < n && sveIsklj; i++)
            sveIsklj = sveIsklj && (a[i] == 0);

        if (sveIsklj)
            stuc++;
    }
    cout << stuc << endl;
    return 0;
}
```

Бољи начин да после промене установимо да ли су све сијалице искључене је да одржавамо број укључених сијалица. То значи да када Паја укључи неку сијалицу, треба да повећамо број укључених сијалица, а када искључи неку сијалицу да тај број смањимо. После ажурирања броја укључених сијалица треба још да проверимо да ли је тај број нула и ако јесте, да повећамо бројач штуцања. На тај начин долазимо до ефикаснијег решења, јер је у њему укупан број операција након учитавања сразмеран са бројем притисака на прекидаче.

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

using namespace std;

int main() {
    int m, n, p, brUklj = 0, stuc = 0;
    cin >> n;
    vector<int> a(n);
    for(int i = 0; i < n; i++) {
        cin >> a[i];
        brUklj += a[i];
    }

    cin >> m;
    for(int i = 0; i < m; i++) {
        cin >> p;
        if (a[p] == 0) { a[p] = 1; brUklj++; }
        else { a[p] = 0; brUklj--; }
        if (brUklj == 0)
            stuc++;
    }
    cout << stuc << endl;
    return 0;
}
```

Задатак: Погрешна операција

Аутор: Небојша Варница

За дати природан број $n > 1$ одредити разломак који сабран са n даје исти резултат као и када се тај разломак

помножи са n. Разломак треба да буде потпуно скраћен (бројилац и именилац треба да буду узајамно прости).

Опис улаза

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

Опис излаза

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

Пример

Улаз	*Излаз*
`3`	`3/2`

Објашњење

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

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

Решење

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

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

Одатле следи да је:

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

тј.

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

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

```
#include <iostream>

using namespace std;

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

Задатак: Разврставање

Аутор: Душан Пођадић

Филип учествује на школском такмичењу у малом фудбалу на ком се ђаци такмиче по екипама, којих укупно има k. Пре развртавања су сви поређани у редове, а у сваком реду има по m ученика. Разврставање креће од почетка првог реда и ђаци се редом развтставају у екипе 1, 2, … k, 1, 2 … Када се разврстају ђаци из првог реда, прелази се на првог ђака у наредном реду и тако до краја. Филип се налази у i-том реду и пребројао је да је он -ти ђак у том реду (сва бројања крећу од 1). Одредити редни број срећне екипе у коју ће он запасти.

Опис улаза

Уносе се редом природни бројеви k, m, i и j ($j \leq m$), сваки у посебном реду. Сви бројеви су природни и мањи од 500.

Опис излаза

Исписати један цео број – редни број екипе у којој ће играти Филип.

Пример

Улаз	*Излаз*
12 8 4 3	3

Објашњење

Са слике се види да ће Филип играти у екипи 3. Са X су означени ученици који стоје у редовима, а у загради пише у којој ће екипи играти. Филип је означен плавом бојом.

Решење

Да бисмо одредили екипу у којој ће Филип наступати, прво је потребно одредити који по реду ће Филип бити разврстан (односно колико има ученика испред њега). Број ученика испред Филипа одређујемо тако што прво видимо колико има редова испред Филипа и за сваки ред који се налази испред њега на број ученика додајемо m, што је број ученика у реду. Дакле, ако се Филип налази у првом реду тај број ће бити 0, ако се налази у другом реду биће m, ако се налази у трећем биће $2m$ итд. Одатле закључујемо да ако се Филип налази у i -том реду, број ученика у редовима испред њега ће бити $(i-1)*m$.

На овај број треба додати све ученике који се налазе у истом реду као и Филип, али су испред њега. Поново, ако је Филип први у свом реду то ће бити 0, ако је други биће 1 итд. Дакле, пошто је Филип на j-том месту у свом реду, треба додати број $j-1$. Коначно долазимо до тога да се испред Филипа налази $(i-1)*m+j-1$ ученика.

Ако сваког ученика "обележимо" бројем ученика испред њега, тада 0. ученик иде у групу 1, 1. у групу 2 … $k-1$-и ученик иде у групу k, а затим k-ти поново у групу 1. Пошто видимо да се на сваких k бројач група ресетује, закључујемо да можемо посматрати групу као остатак при дељењу броја којим смо обележили ученика са k. Наравно, пошто групе бројимо од 1, а не од 0, на крају треба додати +1 на број групе.

```cpp
#include <iostream>

using namespace std;

int main() {
    int k, m, i, j;
    cin >> k >> m >> i >> j;

    int rBr =  (i - 1) * m + j - 1;
    int grupa = rBr % k + 1;
    cout << grupa << endl;

    return 0;
}
```

Задатак: Најдаљи најближи

Аутор: Милан Вуīделија

Написати програм који од 4 учитана цела броја исписује онај, коме је растојање до најближег од осталих бројева максимално. Ако има више таквих бројева, исписати их све, редом по величини од најмањег од највећег.

Опис улаза

На стандардном улазу се налазе четири цела броја из интервала $[-1000000, 1000000]$, сваки посебном реду.

Опис излаза

На стандардни излаз исписати оне од учитаних бројева који испуњавају описани услов, сваки у посебном реду.

Пример 1

Улаз	*Излаз*
7	7
2	
4	
1	

Пример 2

Улаз	*Излаз*
4	2
2	4
7	7
9	9

Пример 3

Улаз	*Излаз*
4	1
1	9
6	
9	

Решење

Задатак се лакше решава ако најпре уредимо 4 учитана броја a, b, c, d по величини. Након што постигнемо да важи $a \leq b \leq c \leq d$, можемо лако за сваки од њих да одредимо даљину до најближег од преостала три:

- $a_m = b - a$
- $b_m = \min(b - a, c - b)$
- $c_m = \min(c - b, d - c)$
- $d_m = d - c$

Нека је $m = max(a_m, b_m, c_m, d_m)$. Потребно је још исписати сваки од бројева за који је растојање од њега до најближег од осталих једнако m.

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

using namespace std;

int main() {
    int a, b, c, d;
    cin >> a >> b >> c >> d;
    if (a>b) swap(a, b);
    if (b>c) swap(b, c);
    if (c>d) swap(c, d);

    if (a>b) swap(a, b);
    if (b>c) swap(b, c);
    if (a>b) swap(a, b);
    int doA = b - a;
    int doB = min(b - a, c - b);
    int doC = min(c - b, d - c);
    int doD = d - c;
    int dMax = max(max(doA, doB), max(doC, doD));
    if (dMax == doA) cout << a << endl;
    if (dMax == doB) cout << b << endl;
    if (dMax == doC) cout << c << endl;
    if (dMax == doD) cout << d << endl;
    return 0;
}
```

Сортирање се још једноставније спроводи ако су бројеви учитани у низ.

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

using namespace std;

int main() {
    int a[4];
    cin >> a[0] >> a[1] >> a[2] >> a[3];
```

```
    sort(a, a+4);
    int doA = a[1] - a[0];
    int doB = min(a[1] - a[0], a[2] - a[1]);
    int doC = min(a[2] - a[1], a[3] - a[2]);
    int doD = a[3] - a[2];
    int dMax = max({doA, doB, doC, doD});
    if (dMax == doA) cout << a[0] << endl;
    if (dMax == doB) cout << a[1] << endl;
    if (dMax == doC) cout << a[2] << endl;
    if (dMax == doD) cout << a[3] << endl;
    return 0;
}
```

Задатак: Wifi снага

Аутор: Филип Марић

Познате су локације кућа дуж једне улице (њихове целобројне x-координате) и локације WiFi предајника који су постављени дуж те улице (поново њихове целобројне x-координате). Одредити минималну снагу сигнала коју је потребно подесити свим предајницима (свима се мора подесити иста снага) да би свака кућа имала сигнал. Ако је снага предајника на позицији x једнака d, тада се њиме обухватају куће на позицијама из интервала $[x - d, x + d]$.

Опис улаза

Са стандардног улаза се учитава број кућа m ($1 \leq m \leq 10^5$), а затим локације кућа (бројеви између 0 и 10^6). Затим се учитава број предајника n ($1 \leq n \leq 10^5$), а затим локације предајника (бројеви између 0 и 10^6).

Опис излаза

На стандардни излаз исписати минималну снагу потребну да се све куће покрију.

Пример 1

Улаз	*Излаз*
4 1 2 3 4 2 1 5	2

Објашњење

Објашњење

Ако је снага предајника 2, тада први обухвата интервал $[-1, 3]$, а други покрива интервал $[3, 7]$, што покрива све куће. Ако би снага била 1, тада би први обухватао интервал $[0, 2]$, а други $[4, 6]$, па кућа на позицији 3 не би била покривена.

Пример 2

Улаз

```
10
13 4 18 9 16 38 25 42 7 19
5
2 16 33 26 10
```

Излаз

```
9
```

Решење

Груба сила

Решење грубом силом подразумева да мало по мало повећавамо снагу све док све куће не буду покривене.

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

using namespace std;

bool sveKucePokrivene(const vector<int>& kuce,
                      const vector<int>& predajnici,
                      int snaga) {
  for (int k : kuce) {
    bool pokrivena = false;
    for (int p : predajnici) {
      if (p-snaga <= k && k <= p+snaga) {
        pokrivena = true;
        break;
      }
    }
    if (!pokrivena)
      return false;
  }
  return true;
}

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

  int snaga = 0;
  while(!sveKucePokrivene(kuce, predajnici, snaga))
    snaga++;

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

Сортирање и техника два показивача

Можемо одредити најмању потребну снагу за сваку кућу и одредити максимум тих вредности. За сваку кућу одређујемо најближи предајник (анализирајући најближи предајник који је испред ње и најближи предајник који је иза ње) и најмања потребна снага за ту кућу је растојање до њој најближег предајника.

Ако сортирамо низ кућа и низ предајника до решења можемо доћи техником два показивача.

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

using namespace std;

int main() {
  int m;
  cin >> m;
```

```
  vector<int> kuce(m);
  for (int i = 0; i < m; i++)
    cin >> kuce[i];
  int n;
  cin >> n;
  vector<int> predajnici(n);
  for (int i = 0; i < n; i++)
    cin >> predajnici[i];

  sort(begin(kuce), end(kuce));
  sort(begin(predajnici), end(predajnici));

  int snaga = 0;
  int p = 0;
  for (int k = 0; k < m; k++) {
    while (p < n && predajnici[p] < kuce[k])
      p++;
    int potrebno;
    if (p == 0)
      potrebno = predajnici[p] - kuce[k];
    else if (p == n)
      potrebno = kuce[k] - predajnici[p-1];
    else
      potrebno = min(predajnici[p] - kuce[k], kuce[k] - predajnici[p-1]);
    snaga = max(snaga, potrebno);
  }

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

Сортирање и бинарна претрага

Можемо одредити најмању потребну снагу за сваку кућу и одредити максимум тих вредности. За сваку кућу одређујемо најближи предајник (анализирајући најближи предајник који је испред ње и најближи предајник који је иза ње) и најмања потребна снага за ту кућу је растојање до њој најближег предајника.

Ако сортирамо само низ предајника, најближи предајник иза куће можемо наћи бинарном претрагом (тражимо најмањи број у низу који је већи или једнак од дате вредности, што се може урадити библиотечком функцијом).

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

using namespace std;

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

  int snaga = 0;
```

```
  sort(begin(predajnici), end(predajnici));
  for (int kuca : kuce) {
    int p = distance(begin(predajnici),
                     lower_bound(begin(predajnici), end(predajnici), kuca));
    int potrebno;
    if (p == n)
      potrebno = kuca - predajnici[n-1];
    else if (p == 0)
      potrebno = predajnici[0] - kuca;
    else
      potrebno = min(kuca - predajnici[p-1], predajnici[p] - kuca);

    snaga = max(snaga, potrebno);
  }

  cout << snaga << endl;

  return 0;
}
```

Бинарна претрага по решењу

До оптималног решења се може доћи и процесом бинарне преграге по решењу. Знамо да оно лежи у интервалу $[0, R]$, где је R распон између најближе и најдаље куће. Половимо тај интервал, уз коришћење помоћне функције којом се проверава да ли је текућа снага довољна да све куће буду покривене.

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

using namespace std;

bool sveKucePokrivene(const vector<int>& kuce,
                      const vector<int>& predajnici,
                      int snaga) {
  for (int k : kuce) {
    auto it = lower_bound(begin(predajnici), end(predajnici), k - snaga);
    if (it == end(predajnici) || *it > k + snaga)
      return false;
  }
  return true;
}

int main() {
  int m;
  cin >> m;
  vector<int> kuce(m);
  for (int i = 0; i < m; i++)
    cin >> kuce[i];
  int n;
  cin >> n;
  vector<int> predajnici(n);
  for (int i = 0; i < n; i++)
    cin >> predajnici[i];
  sort(begin(predajnici), end(predajnici));

  int l = 0;
  int d = abs(*min_element(begin(kuce), end(kuce)) -
              *max_element(begin(kuce), end(kuce)));
```

```
  while (l <= d) {
    int snaga = l + (d - l) / 2;
    if(sveKucePokrivene(kuce, predajnici, snaga))
      d = snaga - 1;
    else
      l = snaga + 1;
  }

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

Задатак: Max НЗД

Аутор: Филип Марић

Напиши програм који на основу познатог производа два позитивна природна броја a и b одређује највећу могућу вредност њиховог највећег заједничког делиоца.

Опис улаза

Са стандардног улаза се учитава број $p = a \cdot b$ ($1 \leq p \leq 10^{18}$).

Опис излаза

На стандардни излаз исписати максималну могућу вредност за НЗД.

Пример 1

Улаз	*Излаз*
600	10

Објашњење

Највећи НЗД се добија када се број 600 представи као производ бројева 20 и 30.

Пример 2

Улаз

```
123456
```

Излаз

```
8
```

Решење

Груба сила

Решење грубом силом подразумева да број n на све начине разложимо на производ два броја $a \cdot b$, израчунамо њихов НЗД и одредимо максимум свих тако добијених бројева. Без губитка на општости можемо претпоставити да је $a \leq b$, па је довољно испитивати све вредности од 1 до $\sqrt{n}$.

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

using namespace std;

typedef unsigned long long ull;

ull nzd(ull a, ull b) {
  while (b != 0) {
    ull mod = a % b;
    a = b;
```

```
    b = mod;
  }
  return a;
}

int main() {
  ull p;
  cin >> p;
  ull f = 2;
  ull maxNzd = 1;
  while (f * f <= p) {
    if (p % f == 0)
      maxNzd = max(maxNzd, nzd(f, p / f));
    f++;
  }
  cout << maxNzd << endl;
  return 0;
}
```

Растављање на просте чиниоце

Ефикасније решење се добија ако се број растави на просте чиниоце: $p_1^{k_1} p_2^{k_2} \dots p_m^{k_m}$. Највећи НЗД се добије ако се сваки чинилац p_i равномерно раздели у два броја a и b. Сваки од бројева ће садржати фактор $p_i^{\lfloor \frac{k_i}{2} \rfloor}$ (заиста, ако је k_i паран, оба ће имати тачно тај фактор, а ако је непаран, онда ће један од бројева садржати тај фактор, а други број ће садржати фактор $p_i^{\lfloor \frac{k_i}{2} \rfloor + 1}$).

На пример, важи $600 = 2^3 \cdot 3 \cdot 5^2$. Највећи НЗД се може добити ако бројеви a и b приме по једну двојку и по једну петицу (ирелевантно је како ће се распоредити преостала двојка и тројка).

```
#include <iostream>

using namespace std;

int main() {
  unsigned long long p;
  cin >> p;
  unsigned long long max = 1;
  unsigned long long f = 2;
  while (f * f <= p) {
    int k = 0;
    while (p % f == 0) {
      p /= f;
      k++;
      if (k % 2 == 0)
        max *= f;
    }
    f++;
  }
  cout << max << endl;
  return 0;
}
```

Задатак: Недостајућа страница четвороугла

Аутор: Милан Вујделија

Темена четвороугла су означена великим словима енглеске абецеде. Ознаке страница се добијају читањем темена у круг, увек у истом смеру. На пример, ако су темена редом (идући по контури) X, Y, Z, W, онда су

ознаке страница XY, YZ, ZW и WX. Дате су ознаке неке три странице у произвољном редоследу страница, а треба одредити ознаку четврте.

Опис улаза

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

Опис излаза

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

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

Решење

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

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

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

```
#include <iostream>

using namespace std;

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

    return 0;
}
```

Задатак: Ексел

Аутор: Душан Пејадић

Мајкрософт Ексел програм садржи табеле чији су редови означени бројевима (1, 2, 3 …), а колоне словима (A, B, C,… Y, Z). Када се у табелу дода превише колона (више од 26 колико слова има енглеска абецеда), колоне почињу да се означавају са два слова (AA, AB, AC …, AZ, BA, BB, BC … ZZ), затим са три (AAA, AAB, AAC … AAZ, ABA, ABB, … ZZZ) и тако даље. Ако знамо да је у табели последња колона означена карактером (или низом караткера) s, одредити колико колона има у табели.

Опис улаза

У првој линији стандардног улаза се уноси ниска карактера s. Сви карактери су велика слова енглеске абецеде и има их највише 6.

Опис излаза

У једини ред стандардног излаза исписати цео број који представља укупан број колона у табели.

Пример 1

Улаз	*Излаз*
F	6

Објашњење

Колоне редом иду: A, B, C, D, E i F - дакле 6 колона.

Пример 2

Улаз

```
AC
```

Излаз

```
29
```

Пример 3

Улаз

```
EXCEL
```

Излаз

```
2708874
```

Решење

Овај задатак се практично своди на пребацивање из бројног система са основом 26 у бројни систем са основом 10. То се ради тако што се тражени број иницијализује на 0 и онда се иде редом кроз цифре. Досада одређени број се множи са 26 (основом система из кога се пребацује) и додаје се вредност цифре. Треба обратити пажњу да колоне у екселу нису класичан систем у основи 26 јер не постоји 0 већ се креће од 1 (‘A’). Једина модификација која је потребна у алгоритму да би се ово испоштовало је то да се при додавању нове цифре на број још додатно дода 1.

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

using namespace std;

int main() {

   string kol;

   cin >> kol;

   int brk = 0;

   for(int i = 0; i < kol.length(); i++){
      brk *= 26;
      brk += kol[i] - 'A' + 1;
```

```
    }

    cout << brk;
}
```

Задатак: Крађа дијаманта

Озлоглашени пљачкаши Тоша и Моша испланирали су да украду драгоцени дијамант. Оно што овај подухват чини тежим него раније јесте нови ласерски безбедносни систем постављен у просторијама где се дијамант налази.

Моша је успео да сазна да су ласери постављени веома специфично; постоји тачно n вертикалних ласера, где је i-ти ласер постављен на позицију v_i и m хоризонталних ласера, постављени на позицију h_i (ласере можемо посматрати као праве представљене једначином $x = v_i$ за вертикалне, односно $y = h_i$ за хоризонталне ласере).

У међувремену, Тоша је сазнао да један прозор на врху зграде може да се отвори, па ако се спусте низ конопац кроз њега, наћи ће се на координатама (x_1, y_1) у соби, а да је тачна локација дијаманта на координатама (x_2, y_2).

Све што је преостало је да се ласери онеспособе. Зато, Тошу и Мошу интересује колико минимално ласера морају да искључе да би могли несметано да дођу до дијаманта, не прелазећи ни преко једног ласера. Да ли можете да им помогнете?

Опис улаза

У првом реду стандардног улаза налазе се целобројне вредности x_1 и y_1 раздвојене размаком, координате места у соби на коме се Тоша и Моша налазе на почетку.

У другом реду налазе се целобројне вредности x_2 и y_2 раздвојене размаком, координате дијаманта у просторији.

У трећем реду налази се природан број n, број вертикалних ласера.

У четвртом реду налази се n целобројних вредности раздвојених размаком, где i-та вредност v_i представља x координату вертикалног ласера.

У петом реду налази се природан број m, број хоризонталних ласера.

У шестом реду налази се m целобројних вредности раздвојених размаком, где i-та вредност h_i представља y координату хоризонталног ласера.

Опис излаза

Природан број који представља минималан број ласера које Тоша и Моша морају да искључе.

Ограничења

$$-10^9 \leq x_1, y_1, x_2, y_2 \leq 10^9$$

$$1 \leq n, m \leq 10^5$$

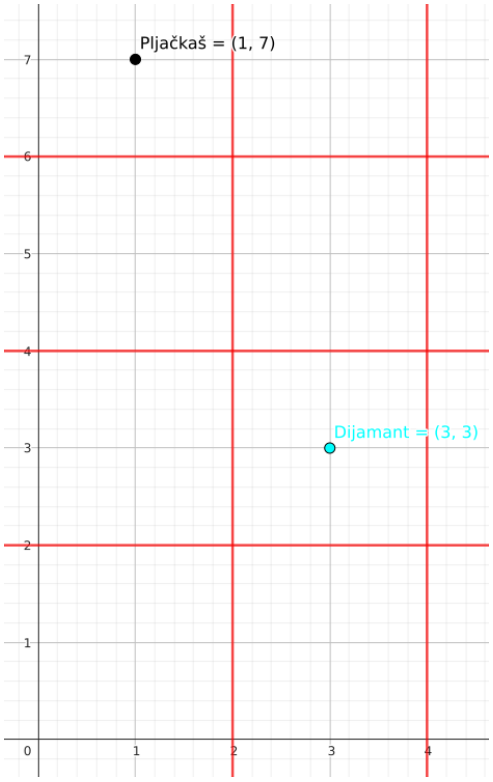
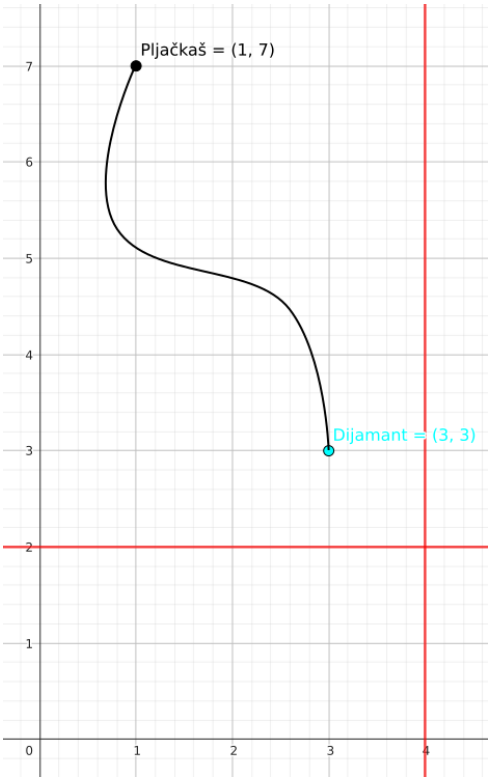
$$-10^9 \leq v_i, h_i \leq 10^9$$

Пример 1

Улаз	*Излаз*
1 7 3 3 2 2 4 3 2 4 6	3

Објашњење

Минималан број ласера које морамо да искључимо је 3, на пример хоризонталне ласере на позицијама $y = 6$ и $y = 4$ и вертикалан ласер на позицији $x = 2$.

Почетно стање	После искључивања
	

Пример 2

Улаз

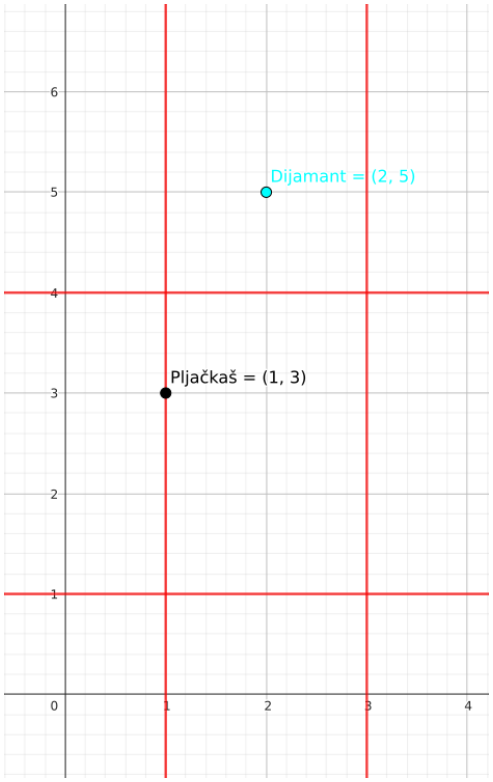
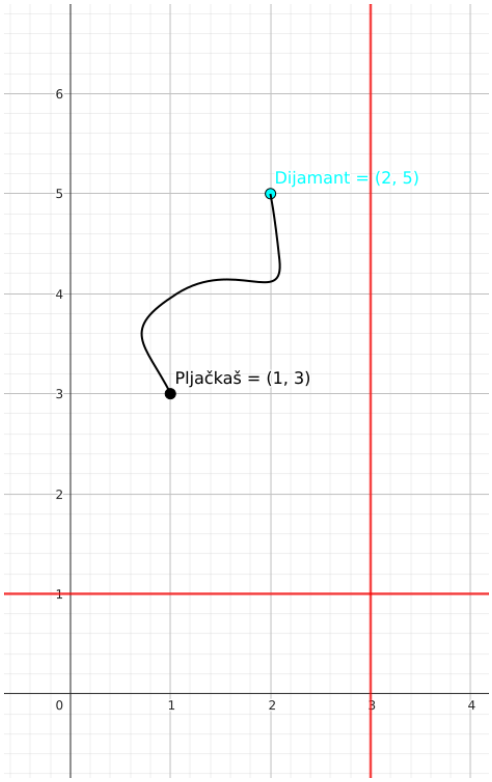
```
1 3
2 5
2
1 3
2
1 4
```

Излаз

```
2
```

Објашњење

Минималан број ласера које морамо да искључимо је 2, на пример хоризонтални ласер на позицији $y = 4$ и вертикалан ласер на позицији $x = 1$.

Почетно стање	После искључивања
	

Решење

Опис главног решења

У овом блоку се описује главно решење задатка.

```
#include <iostream>

using namespace std;

bool je_izmedju(int x1, int x2, int x)
{
   return abs(x1 - x) + abs(x2 - x) == abs(x1 - x2);
}

int izbroj_izmedju(int c1, int c2, int n)
{
   int br = 0;
   for (int i = 0; i < n; i++)
   {
      int c;
      cin >> c;
      if (je_izmedju(c1, c2, c))
         br++;
   }
   return br;
}

int main()
{
   int x1, y1, x2, y2;
```

```
    cin >> x1 >> y1 >> x2 >> y2;
    int br = 0;
    int n;
    cin >> n;
    br += izbroj_izmedju(x1, x2, n);
    int m;
    cin >> m;
    br += izbroj_izmedju(y1, y2, m);
    cout << br << "\n";
    return 0;
}
```

Задатак: Број сегмената са производом нула

Аутор: Милан Вуīелија

Написати програм који за дати низ целих бројева одређује број сегмената (поднизова са узастопним елементима) датог низа, којима је производ елемената једнак нули.

Опис улаза

У првом реду стандардног улаза је број n $(1 \leq n \leq 100000)$, број елемената низа. У другом реду је n целих бројева из интервала $[-100, 100]$ раздвојених по једним размаком, елементи низа.

Опис излаза

На стандардни излаз исписати један цео број, тражени број сегмената.

Пример 1

Улаз	*Излаз*
`5` `1 2 3 0 5`	`8`

Објашњење

То су сегменти `1 2 3 0`, `1 2 3 0 5`, `2 3 0`, `2 3 0 5`, `3 0`, `3 0 5`, `0`, `0 5`.

Пример 2

Улаз

```
3
0 1 0
```

Излаз

```
5
```

Објашњење

То су сегменти `0` (прва нула), `0 1`, `0 1 0`, `1 0`, `0` (друга нула).

Решење

Производ сегмента је нула ако и само ако је бар један од бројева у том сегменту једнак нули. Према томе, потребно је одредити број сегмената који садрже бар једну нулу.

Сваки сегмент је одређен индексом свог почетка и краја. Зато задатак можемо да решимо помоћу двоструке `for` петље, где промењиву спољне петље користимо као индекс почетка сегмента, а промењиву унутрашње петље као индекс краја. Логичка промењива imaNula означава да ли у текућем сегменту низа постоји елемент једнак нули.

```
#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];

    long long br = 0;
    for (int i = 0; i < n; i++) {
        bool imaNula = false;
        for (int j = i; j < n; j++) {
            if (a[j] == 0) imaNula = true;
            if (imaNula) br++;
        }
    }
    cout << br << endl;
    return 0;
}
```

Мана претходног решења је да се две `if` наредбе извршавају за сваки сегмент, а сегмената има нешто више од $n^2/2$, где је n дужина низа. То може да буде преко $50\,000 \cdot 50\,000/2 = 1\,250\,000\,000$ операција, што је превише за расположиво време.

Претходно решење можемо да побољшамо тако што од сваког почетка сегмента (позиција i у низу) тражимо први елемент низа једнак нули. Нека је та прва нула на позицији j. Тада од свих сегмената који почињу на позицији i, нулу садрже сегменти $[i..j], [i..j+1], [i..j+2], \ldots, [i..n-1]$. Таквих сегмената има $n-j$, па резултат (бројач тражених сегмената) можемо да увећамо за $n-j$ и тиме смо обрадили све сегменте са почетком i. У случају да почевши од позиције i нема нула у низу, променљива j ће достићи вредност n, па додавање $n-j=0$ на резултат не мења вредност резултата.

```
#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];

    long long br = 0;
    for (int i = 0; i < n; i++) {
        int j = i;
        while (j < n && a[j] != 0)
            j++;

        br += n-j;
    }
    cout << br << endl;
    return 0;
}
```

Неповољан случај за претходно решење је да у низу има врло мало нула, или их нема уопште. У том случају за сваки почетак i сегмента, унутрашња петља која тражи нулу иде до краја низа, па је број операција поново једнак укупном броју сегмената.

Да бисмо добили још боље решење, треба да приметимо да приликом тражења нула у низу, променљива j не мора за сваки нови почетак i да почне од тог i, јер смо већ утврдили да између позиција i и j нема нула у низу. Захваљујући томе, када је $j > i$ можемо да уштедимо време не враћајући вредност j на почетак интервала.

Пошто се у овом решењу j никад не смањује, укупан број операција је сразмеран са дужином низа.

```
#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 j = 0;
    long long br = 0;
    for (int i = 0; i < n; i++) {
        if (j < i) j = i;
        while (j < n && a[j] != 0)
            j++;

        br += n-j;
    }
    cout << br << endl;
    return 0;
}
```

Задатак: Један-два-три-четири

Аутор: Милан Вуīелија

Написати програм, који за дате природне бројеве N и P редом исписује P лексикографски најмањих уређених N-торки природних бројева, таквих да важи:

- сваки елемент N-торке је један од бројева 1, 2, 3, 4
- у свака 4 узастопна елемента N-торке јављају се бар три различите вредности

Нека је B_N број свих N-торки које испуњавају ове услове. Ако је $B_N < P$, исписати свих тих B_N N-торки.

Опис улаза

У првом реду стандардног улаза број N, $4 \leq N \leq 50$. У другом реду стандардног улаза број P, $1 \leq P \leq 300$.

Опис излаза

У сваком од $min(B_N, P)$ редова исписати по једну N-торку бројева 1, 2, 3, 4. Елементе N-торке исписати без размака између њих.

Пример 1

Улаз	*Излаз*
5	11231
17	11232
	11233
	11234
	11241
	11242
	11243
	11244
	11321
	11322
	11323
	11324
	11341
	11342
	11343
	11344
	11421

Објашњење

Дати улаз значи да је потребно да се испише првих 17 уређених енторки за $n = 5$, тј. првих 17 уређених петорки.

Уређене петорке бројева $1, 2, 3, 4$ лексикографским редом су:

11111, 11112, 11113, 11114, 11121, 11122, 11123, 11124, 11131, 11132, 11133, 11134, 11141, 11142, 11143, 11144,

11211, 11212, 11213, 11214, 11221, 11222, 11223, 11224, 11231, 11232, 11233, 11234, 11241, 11242, 11243, 11244,

11311, 11312, 11313, 11314, 11321, 11322, 11323, 11324, 11331, 11332, 11333, 11334, 11341, 11342, 11343, 11344,

11411, 11412, 11413, 11414, 11421, 11422, 11423, 11424, 11431, 11432, 11433, 11434, 11441, 11442, 11443, 11444…

Многе од ових петорки не испуњавају услов да међу свака четири узастопна броја у петорци буду бар три различите вредности. Такве су, на пример, све оне петорке које у прва четири броја имају бар три јединице, или две јединице и две двојке, или две јединице и две тројке. Зато је лексикографски прва петорка која испуњава поменути услов 11231. Следеће петорке које испуњавају услов су (лексикографским редом) 11232, 11233 и даље како је наведено у излазу.

Пример 2

Улаз

```
14
10
```

Излаз

```
11231123112311
11231123112312
11231123112313
11231123112314
11231123112321
11231123112324
11231123112331
11231123112334
11231123112341
11231123112342
```

Решење

Тражене N-торке је најлакше генерисати помоћу рекурзивне функције. Довољно је да као параметар функцији проследимо број генерисаних елемената N-торке.

Одмах на уласку у функцију проверавамо да ли је исписан тражени број N-торки. Ако јесте, нема потреба да се даље генеришу N-торке и функција може да заврши са радом.

Ако је број генерисаних елемената текуће N-торке једнак N, функција исписује елементе N-торке и завршава са радом. У противном, за сваки од наставака 1, 2, 3, 4 проверавамо да ли је одговарајући, а за оне који су одговарајући рекурзивно настављамо са генерисањем.

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

using namespace std;

vector<int> a;
int n, ispisati;

bool DobarSufiks(int i) {
    if (i < 2)
        return true;
    if (i == 2)
        return a[i] != a[i - 1] || a[i] != a[i - 2];

    vector<int> br(5);
    for (int k = i - 3; k <= i; k++)
        br[a[k]] = 1;
    return br[1] + br[2] + br[3] + br[4] > 2;
}

void ProduziNiz(int i) {
    if (ispisati == 0)
        return;

    if (i == n) {
        for (int k = 0; k < n; k++)
            cout << a[k];
        cout << endl;
        ispisati--;
        return;
    }

    for (int c = 1; c <= 4; c++) {
        a[i] = c;
        if (DobarSufiks(i))
            ProduziNiz(i + 1);
    }
}


int main() {
    cin >> n >> ispisati;
    a.resize(n);
    ProduziNiz(0);
    return 0;
}
```

Глава 2

Квалификације (2. круг)

Задатак: Закуцавање

Аутор: Милан Вујделија

Кошаркашки обруч за децу је на висини 240 центиметара, а пречник кошаркашке лопте је 24 центиметара. Виктор са пода може да дохвати висину h центиметара. Колико Виктор треба да скочи да би могао да закуца лопту у кош?

Претпоставља се да је за закуцавање довољно досегнути висину која је за пречник лопте изнад висине обруча.

Опис улаза

На стандардном улазу је цео број h $(100 \leq h \leq 280)$, висина у центиметрима коју Виктор дохвата са пода.

Опис излаза

На стандардни излаз исписати један цео број, потребну висину скока.

Пример 1		**Пример 2**	
Улаз	*Излаз*	*Улаз*	*Излаз*
`220`	`44`	`270`	`0`

Решење

На основу текста задатка закључујемо да Виктор треба да досегне висину од $240 + 24 = 264$ центиметара да би могао да закуца. Нека је h висина коју Виктор дохвата са пода.

Потребна висина скока је $264 - h$, осим када је овај број негативан. У том случају потребан скок је нула (не може се скакати на висину нижу од пода). Према томе, одговор је већи од борјева $264 - h$ и 0.

```
#include <iostream>

using namespace std;

int main() {
  int h;
  cin >> h;
  int potrebno = max(264 - h, 0);
  cout << potrebno << endl;
  return 0;
}
```

Ако висину h или већу висину Виктор може да досегне са пода, потребан скок је 0, а у противном је потребан скок $264 - h$. Ова два случаја можемо да разликујемо у програму помоћу једне `if` наредбе.

```
#include <iostream>

using namespace std;
```

```
int main() {
    int h;
    cin >> h;
    if (h > 264)
        cout << 0 << endl;
    else
        cout << 264 - h << endl;
    return 0;
}
```

Задатак: Непознати број

Аутор: Филип Марић

Миша је замислио један природан број. Након тога га је помножио са 8, добијени производ је сабрао са 10, добијени збир је поделио са 2 и од добијеног количника је одузео 8. Помози Цеци, Мишиној другарици, да на основу резултата који је Миша добио погоди број који је Миша замислио.

Опис улаза

Са стандардног улаза се уноси резултат који је Миша добио (природан број n између 1 и 1000).

Опис излаза

На стандардни излаз исписати број који је Миша замислио, или текст `greska`, ако је негде погрешио у рачуну и не постоји начин да се од неког природног броја добије резултат који је он добио.

Пример 1

Улаз	*Излаз*
17	5

Објашњење

Миша је замислио број 5. Затим га је помножио са 8 и добио 40. На 40 је додао 10 и добио 50. Затим је 50 поделио са 2 и добио 25. На крају је од 25 одузео 8 и добио је 17.

Пример 2

Улаз

```
28
```

Излаз

```
greska
```

Пример 3

Улаз

```
1
```

Излаз

```
1
```

Решење

Да бисмо решили задатак потребно је да прво учитамо природни број r. Број r настаје тако што је Миша применио описане рачунске операције над оригиналним природним бројем x. Из текста задатка видимо да је редослед операција следећи:

$$r = (8x + 10)/2 - 8.$$

Дакле, знајући број r и редослед примењених операција, потребно је да одредимо број x. Задатак се практично своди на примењивање свих рачунаских операција из текста задатка, али обрнутим редоследом. Корак по корак, креирамо решење:

$$\begin{aligned} r+8 &= (8x+10)/2 \\ (r+8)\cdot 2 &= 8x+10 \\ (r+8)\cdot 2-10 &= 8x. \end{aligned}$$

На овом месту је кључни део задатка. Последњи израз који нам треба је:

$$x = ((r+8)\cdot 2-10)/8.$$

Резултат x мора бити природни број. Међутим, то неће бити могуће ако десна страна израза није потпун количник: Прецизније разликујемо два случаја:

1. Израз $(r+8)\cdot 2-10$ је дељив са 8. У том случају имамо решење које је облика: $x = ((r+8)\cdot 2-10)/8$.
2. Израз $(r+8)\cdot 2-10$ није дељив са 8. У том случају, знамо да решење не постоји и да је Миша погрешио у рачуну, па као резултат треба да испишемо `greska`.

Решење је у наставку.

```
#include <iostream>

using namespace std;

int main() {
  int r;
  cin >> r;
  r = (r + 8) * 2 - 10;
  if (r % 8 == 0)
    cout << r / 8 << endl;
  else
    cout << "greska" << endl;
  return 0;
}
```

Приликом решавања једначине изрази се могу упрошћавати. Важи:

$$x = ((r+8)\cdot 2-10)/8 = (2r+16-10)/8 = (2r+6)/8 = (r+3)/4.$$

```
#include <iostream>

using namespace std;

int main() {
  int r;
  cin >> r;
  if ((r + 3) % 4 == 0)
    cout << (r + 3) / 4 << endl;
  else
    cout << "greska" << endl;
  return 0;
}
```

Задатак: Ски пас

Аутор: Душан Пејадић

На једном скијалишту у Србији цена дана скијања зависи од узраста. Деца млађа од 5 година не плаћају ски карту, деца од 5 до 12 година плаћају 2800 динара дан, људи од 13 до 64 године плаћају 3500 динара дан, а сениори (65+ година) плаћају 3200 динара дан. Такође, за сваки дан након 10. остварује попуст и плаћа се половина цене дана. Уколико је познато да Надежда има n година и жели да уплати k дана скијања, одредити колико укупно динара треба да плати.

Опис улаза

У првом реду улаза се налази природан број $n(1 \leq n \leq 140)$. У другом реду се налази приородан број $k(1 \leq k \leq 90)$.

Опис излаза

У једином реду излаза исписати колико новца треба Надежда да плати своје скијање.

Пример

Улаз	*Излаз*
85 12	35200

Објашњење

Пошто је Надежда сениор, њена цена је 3200 динара по дану. За првих 10 дана треба да плати $10 \cdot 3200 = 32000$, а за наредна 2 дана $2 \cdot (3200/2) = 3200$.

Решење

Да бисмо решили задатак потребно је да прво учитамо природне бројеве n и k који редом означавају број година скијаша и број дана за које потребна ски карта.

Решење задатка можемо да поделимо у две целине. Прва целина је одређивање цене ски карте по дану на основу броја година скијаша. Нека се та променљива назива $cena$. Можемо претпоставити да је основна цена заправо дечија, па променљиву $cena$ можемо иницијализовати на 0. Да бисмо до краја одредили цену ски карте по дану, треба нам једноставно гранање у којем ћемо испитати припадност вредности n датим интервалима и доделити променљивој $cena$ одговарајућу вредност. Укратко, вршимо следеће провере:

1. Ако је број година између 5 и 12, цена по дану је 2800.
2. Ако је број година између 13 и 64, цена по дану је 3500.
3. Ако је број година већи или једнак 65, цена по дану је 3200.

Друга целина у задатку је одређивање укупне цене ски карте. Да бисмо то урадили, потребно је да испитамо број дана за које се купује ски карта.

1. Ако је број дана мањи или једнак 10, укупна цена ће бити $k \cdot cena$.
2. Ако је број дана већи од 10, првих десет дана се рачуна по пуној цени, док за сваки следећи дан треба урачунати попуст од 50%. Због тога, укупну цену рачунамо по формули $10 * cena + (k - 10) \cdot cena/2$.

```
#include <iostream>

using namespace std;

int main() {

    int n, k;
    cin >> n >> k;

    int cena = 0;
    if (n >= 5 && n <= 12)
        cena = 2800;
    else if (n>12 && n <= 64)
        cena = 3500;
    else if (n >= 65)
        cena = 3200;

    if (k <= 10)
```

```
        cout << k*cena << endl;
    else
        cout << (10*cena + (k-10)*cena/2) << endl;

    return 0;
}
```

Алтернативно елегантније решење може се добити издвајањем логике којом се рачуна укупна цена у посебну функцију. Поред тога, могуће је и користити уграђене функције `max` и `min` приликом одређивање укупне цене.

```
#include <iostream>

using namespace std;

int cena(int c, int k){
    return min(k, 10) * c + max(0, k - 10) * c / 2;
}

int main() {

    int g, k;
    cin >> g >> k;

    if(g < 5)
      cout << 0;
    else if(g <= 12)
      cout << cena(2800, k);
    else if(g < 65)
      cout << cena(3500, k);
    else
      cout << cena(3200, k);

    return 0;
}
```

Задатак: Гориво

Аутор: Владимир Кузмановић

Пера, Жика и Мика су одлучили да иду на море колима и да се смењују за воланом током пута. Први је за воланом био Пера и превезао је p километара, затим је Жика превезао z километара и на крају је возио Мика који је превезао m километара. Познато је да на пут крећу са пуним резервоаром горива и да аутомобил са пуним резервоаром може да пређе k километара. Написати програм који одређује ко је био за воланом на сваком стајању за допуну горива. Подразумева се да је приликом сваког стајања за воланом био возач који довози аутомобил на пумпу. У случају да Пера, Жика и Мика треба да се зауставе на пумпи на крају путовања, подразумевати да ће прво сипати гориво, па тек онда ићи у хотел и да се у том случају занемарује растојање до хотела.

Опис улаза

У сваком од 4 реда стандардног улаза налази се по један природан број. Бројеви који се учитавају редом представљају вредности p, z, m, k. Бројеви p, z, m, k су између 1 и 100000.

Опис излаза

На стандардни излаз исписати имена возача који су били за воланом у тренутку сваког доласка на пумпу. Свако име исписати у посебном реду. Имена возача треба исписати малим словима енглеског алфабета.

Пример 1

Улаз	*Излаз*
700	pera
900	zika
800	mika
600	mika

Објашњење

Прву паузу за сипање горива треба да направе након 600км, када је за воланом Пера. Следећу паузу праве након следећих 600км, када је за воланом Жика. Наредну паузу праве након још 600км када је за воланом Мика. Последњу паузу праве након још 600км, тј. на самом крају пута, када је опет за воланом Мика.

Пример 2

Улаз

```
100
100
100
100
```

Излаз

```
pera
zika
mika
```

Објашњење

Сваки возач је превезао тачно 100км и аутомобил има домет од 100км са пуним резервоаром. Прво стајање ће бити након 100км, што значи да на пумпу Пера довози аутомобил и завршава својих 100км вожње. Затим, Жика седа за волан и вози својих 100км, чиме ће потрошити цео резервоар горива, па је он следећи који довози аутомобил на пумпу. Након сипања горива, Мика седа за волан и вози својих 100км. Мика ће потрошити цео резервоар горива и он је следећи који довози аутомобил на пумпу. Након што сипају гориво, стигли су на своје одредиште.

Пример 3

Улаз

```
300
500
750
400
```

Излаз

```
zika
zika
mika
```

Решење

Да бисмо решили задатак потребно је прво да учитамо све неопходне податке са стандардног улаза. Потребно је да користимо целе бројеве да бисмо учитане вредности p, z, m и k представили на исправан начин у програму.

Затим, потребно је да уочимо да нам треба укупан пут који су Пера, Мика и Жика прешли на свом путовању. Дакле, рачунамо збир:

$$ukupno = p + z + m.$$

Након тога, потребно је да уочимо да на сваких k пређених километара Пера, Жика и Мика морају да направе паузу да би допунили резервоар. Прецизније, треба да откријемо ко је био за воланом на сваких k километара када су правили паузу.

Да бисмо то постигли, потребно је бројач у петљи иницијализујемо са вредности k, да га након сваке итерације увећавамо за k све док не пређемо вредност $ukupno$ и да у свакој итерацији петље проверавамо ко је био за воланом у датом тренутку. Провере вршимо тако што испитамо да ли вредност бројача припада одговарајућем интервалу:

1. Ако је вредност бројача мања или једнака p, онда је за воланом био Пера.
2. Ако је вредност бројача већа од p и мања или једнака $p + z$, онда је за воланом био Жика.
3. Ако је вредност бројача већа од $p + z$, онда је за воланом био Мика.

Одговарајуће решење је у наставку.

```
#include <iostream>

using namespace std;

int main() {
   int p, z, m, k;
   cin >> p >> z >> m >> k;

   int ukupno = p + z + m;
   for (int i = k; i <= ukupno; i += k) {
      if (i <= p) {
         cout << "pera" << endl;
      } else if (p < i && i <= (p + z)) {
         cout << "zika" << endl;
      } else {
         cout << "mika" << endl;
      }
   }

   return 0;
}
```

Алтернативно решење прати исту логику као и главно решење, само је уместо петље `for` искоришћена петља `while`.

```
#include <iostream>

using namespace std;

int main() {

   int p,z,m,k;
   cin >> p;
   cin >> z;
   cin >> m;
   cin >> k;

   int ukupno = p + z + m;
   int kilometraza = k;
   while ( kilometraza <= ukupno) {
      if (kilometraza <= p) {
         cout << "pera"<<std::endl;
      }
      else if (p < kilometraza && kilometraza <= (p + z)) {
         cout << "zika" << std::endl;
      }
      else {
         cout << "mika" << std::endl;
      }
      kilometraza += k;
```

```
  }

  return 0;
}
```

Задатак: Прва и последња цифра

Аутор: Филип Марић

На капији древног града налази се загонетка у којој се крије лозинка за улазак у тај град. Загонетка се састоји од слова и бројева поређаних у више редова. Међутим, пуно симбола је наведено само да би збунило оне који желе да уђу у град. Наиме, за одређивање лозинке битне су само прва и последња цифара у сваком реду (а у сваком реду постоје бар две цифре). Од њих се у сваком реду добија по један двоцифрени број, а збир свих тих двоцифрених бројева нам даје лозинку за улаз.

Опис улаза

Са стандардног улаза се уноси број n ($1 \leq n \leq 100$), а затим n редова који садрже речи које се састоје од највише 50 слова и цифара (у сваком реду сигурно постоје бар две цифре).

Опис излаза

На стандардни излаз исписати лозинку.

Пример 1

Улаз	*Излаз*

`3`
`12345`
`a1b2c3`
`abc121abc`
	`39`

Објашњење

- У првом реду је прва цифра `1`, а последња `5`, па је двоцифрен број 15.
- У другом реду је прва цифра `1`, а последња `3`, па је двоцифрен број 13.
- У трећем реду је прва цифра `1`, а последња `1`, па је двоцифрен број 11.

Збир ова три двоцифрена броја је 15+13+11=39.

Пример 2

Улаз

```
1
7a1
```

Излаз

```
71
```

Решење

У петљи учитавамо и обрађујемо једну по једну реч са улаза. Потребно је да за сваку реч одредимо прву и последњу цифру. За то ћемо употребити две променљиве `prva_cifra` и `poslednja_cifra` и анализираћемо један по један карактер текуће речи. Променљиву `poslednja_cifra` ћемо ажурирати сваки пут када наиђемо на цифру, тако да ће њена завршна вредност заиста бити последња цифра речи. Променљивој `prva_cifra` ћемо иницијално доделити вредност -1 и ажурираћемо је само ако јој је вредност -1. Након тога она ће добити вредност прве цифре и надаље неће бити ажурирана. Од издвојених цифара лако формирамо број и додајемо га на укупан збир.

```
#include <iostream>

using namespace std;

int main() {
  int n;
```

```
  cin >> n;
  int lozinka = 0;
  for (int i = 0; i < n; i++) {
    string rec;
    cin >> rec;
    int prva_cifra = -1;
    int poslednja_cifra = -1;
    for (char c : rec) {
      if (isdigit(c)) {
        if (prva_cifra == -1)
          prva_cifra = c - '0';
        poslednja_cifra = c - '0';
      }
    }
    lozinka += 10*prva_cifra + poslednja_cifra;
  }
  cout << lozinka << endl;
  return 0;
}
```

Задатак: Најгушћи подскуп

Аутор: Иван Дрецун

Дат је скуп природних бројева. Густина његовог подскупа рачуна се као број елемената подскупа подељен разликом између његовог највећег и најмањег елемента. Напиши програм који одређује подскуп са највећом густином чији је број елемената тачно k. Уколико постоји више подскупова који испуњавају овај услов, исписати онај чији су елементи најмањи.

Опис улаза

Са стандардног улаза се уносе величина датог скупа n ($1 \leq n \leq 2 \cdot 10^5$) и величина траженог подскупа k ($1 \leq k \leq n$). У наредном реду се уносе елементи скупа a_i ($1 \leq a_i \leq 10^9$) одвојени размаком.

Опис излаза

На стандардни излаз исписати елементе траженог подскупа у растућем поретку. Уколико постоји више подскупова који испуњавају услов задатка, исписати онај чији су елементи најмањи.

Пример 1

Улаз	*Излаз*
`6 3` `8 3 10 1 13 6`	`6 8 10`

Пример 2

Улаз	*Излаз*
`5 2` `8 5 1 7 4`	`4 5`

Решење

Како се у задатку тражи подскуп фиксне величине, довољно је пронаћи такав подскуп да му се највећи и најмањи елемент разликују што мање. Ако имамо скуп $\{1, 3, 4, 6, 8\}$ и тражимо подскуп величине 3, нема потребе да разматрамо подскуп $\{1, 3, 6\}$, када знамо да подскуп $\{1, 3, 4\}$ сигурно има већу густину. Дакле, довољно је да сортирамо дати скуп и посматрамо само поднизове узастопних елемената тражене дужине у добијеном низу. Тражимо такав подниз да му се последњи и први елемент разликују што мање.

Пошто број елемената у скупу може бити велик, потребно је користити неки од ефикасних алгоритама за сортирање (најједноставније је да употребимо библиотечку функцију за сортирање).

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

using namespace std;

int main() {
        int n, k;
```

```
        cin >> n >> k;

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

        sort(begin(a), end(a));

        int mind = a[n-1] - a[0];
        int mini = 0;
        for(int i = k - 1; i < n; i++)
                if(a[i] - a[i - k + 1] < mind) {
                        mind = a[i] - a[i - k + 1];
                        mini = i - k + 1;
                }

        for(int i = mini; i < mini + k; i++)
                cout << a[i] << ' ';
        cout << '\n';

        return 0;
}
```

Задатак: Месец рођења

Аутор: Милан Вуīделија

У матичном броју грађана трећа и четврта цифра означавају месец рођења.

Написати програм који за дати матични број исписује прва три слова месеца рођења.

Опис улаза

На стандардном улазу налази се низ од 13 цифара без размака.

Опис излаза

На стандардни излаз исписати један од текстова `jan`, `feb`, `mar`, `apr`, `maj`, `jun`, `jul`, `avg`, `sep`, `okt`, `nov`, `dec`.

Пример 1

Улаз	*Излаз*
`0312998714216`	`dec`

Објашњење

Трећа и четврта цифара датог броја (записане заједно) су `12`, па је месец рођења децембар.

Пример 2

Улаз

```
2103009718495
```

Излаз

```
mar
```

Решење

Најпре треба одредити редни број месеца на основу матичног броја. Поступак одређивања овог броја види се из програма.

Када имамо редни број месеца, можемо да га искористимо као индекс у низу свих могућих одговора, којих има 12. Пошто могући бројеви месеца почињу од 1, а индекси у низу од 0, можемо да додамо један празан стринг на почетак низа. На тај начин одговор `jan` се налази на позицији са индексом 1, а слично важи и за остале месеце, тако да је индекс сваког могућег одговора управо редни борј месеца.

```
#include <iostream>

using namespace std;

int main() {
    string naziviMeseci[] = {
    "", "jan", "feb", "mar", "apr", "maj", "jun",
    "jul", "avg", "sep", "okt", "nov", "dec"};

    string jmbg;
    cin >> jmbg;
    int mesec = (jmbg[2] - '0')*10 + (jmbg[3] - '0');
    cout << naziviMeseci[mesec] << endl;
    return 0;
}
```

Друго решење:

Најпре треба да издвојимо из матичног броја део који садржи цифре на позицијама 3 и 4. За вредност тог дела има 12 могућности, па случајеве можемо да разликујемо помоћу 12 надовезаних наредби `if`. Остаје да у сваком од случајева испишемо правилан одговор.

```
#include <iostream>

using namespace std;

int main() {
    string jmbg;
    cin >> jmbg;
    string mesec =  jmbg.substr(2, 2);
    if (mesec == "01")
        cout << "jan" << endl;
    else if (mesec == "02")
        cout << "feb" << endl;
    else if (mesec == "03")
        cout << "mar" << endl;
    else if (mesec == "04")
        cout << "apr" << endl;
    else if (mesec == "05")
        cout << "maj" << endl;
    else if (mesec == "06")
        cout << "jun" << endl;
    else if (mesec == "07")
        cout << "jul" << endl;
    else if (mesec == "08")
        cout << "avg" << endl;
    else if (mesec == "09")
        cout << "sep" << endl;
    else if (mesec == "10")
        cout << "okt" << endl;
    else if (mesec == "11")
        cout << "nov" << endl;
    else if (mesec == "12")
        cout << "dec" << endl;
    return 0;
}
```

Задатак: Врабац и мачка

Аутор: Милан Вуџелија

Врабац лети унутар интервала $[a, b]$ на бројевној правој $(a < b)$, а мачка шета унутар интервала $[c, d](c < d)$. Интервали садрже своје крајње тачке. Врабац зна да, када слети, мачка ће му прићи најближе што може. Зато врабац жели да слети на такво место да након што му мачка приђе, он буде што даље од ње. Које је највеће растојање на коме врабац може да буде након приласка мачке?

Опис улаза

На стандардном улазу су цели бројеви a, b, c, d, сваки у посебном реду. Сви бројеви су позитивни и нису већи од 2000000000 (две милијарде).

Опис излаза

На стандардни излаз исписати један цео број, највеће растојање које врабац може да обезбеди.

Пример 1

Улаз	*Излаз*
2 20 4 14	6

Објашњење

```
0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
.  .  в  в  в  в  в  в  в  в  в  в  в  в  в  в  в  в  в  в  .  .  .  .
.  .  .  .  м  м  м  м  м  м  м  м  м  м  .  .  .  .  .  .  .  .  .  .
```

Врабац може да се креће по словима **в** (и да лети изнад њих), а мачка да се креће по словима **м**. За врабца је најбоље да слети у тачку 20, а мачка ће након тога да му приђе до тачке 14. Тада је растојање између њих једнако 6 и то је тражено растојање.

Пример 2

Улаз

```
5
12
4
15
```

Излаз

```
0
```

Пример 3

Улаз

```
100
200
120
300
```

Излаз

```
20
```

Пример 4

Улаз

```
400
700
100
200
```

Излаз

```
500
```

Решење

Ако макча може да стане у сваку тачку доступну врапцу, онда је врапцу свеједно где ће да стане, јер ће га мачка свакако ухватити, а тражени резултат је у том случају 0.

Ако постоји тачка која је врапцу доступна а мачки није, онда је та тачка или лево или десно од целог мачкиног интервала. Од свих таквих тачака лево од мачкиног интервала најбоља је крајња лева тачка интервала врапца, а од свих таквих тачака десно од мачкиног интервала најбоља је крајња десна тачка интервала врапца.

Према томе, постоје само две тачке које су кандидати за најбољу тачку врапца, а то су крајеви његовог интервала. Једна или обе ове тачке могу да буду врло лоше за врапца, али сигурни смо да никаква трећа тачка не може да буде боља од обе ове тачке и зато нема потребе да размишљамо о било којим додатним тачкама осим ове две.

Остаје још само да се одреди колико највише врабац може да буде удаљен од мачке у свакој од својих крајњих тачака, када му мачка приђе најближе што може. Начин на који можемо да израчунамо поменута растојања и њихов максимум може да се разуме читањем програма.

```
#include <iostream>

using namespace std;

int main() {
    int vpoc, vkraj, mpoc, mkraj;
    cin >> vpoc >> vkraj >> mpoc >> mkraj;
    int dLevo = (vpoc < mpoc) ? mpoc - vpoc : 0;
    int dDesno = (vkraj > mkraj) ? vkraj - mkraj : 0;
    int d = max(dLevo, dDesno);
    cout << d << endl;
   return 0;
}
```

Други начин да напишемо програм на основу истог размишљања је овај. Нека су $[v_{poc}, v_{kraj}]$ и $[m_{poc}, m_{kraj}]$ интервали врапца и мачке редом.

- ако је интервал врапца у потпуности садржан у интевалу мачке, тј. ако је $m_{poc} \leq v_{poc} < v_{kraj} \leq m_{kraj}$, резултат је 0
- у противном, бар једна од разлика $m_{poc} - v_{poc}$, $v_{kraj} - m_{kraj}$ је позитивна, па је резултат једнак већој од њих (није битно што друга разлика може да буде негативна).

```
#include <iostream>

using namespace std;

int main() {
    int vpoc, vkraj, mpoc, mkraj;
    cin >> vpoc >> vkraj >> mpoc >> mkraj;
    if (vpoc > mpoc && vkraj < mkraj)
        cout << 0 << endl;
    else
        cout << max(mpoc - vpoc, vkraj - mkraj) << endl;
   return 0;
}
```

Лоше решење би било да се за сваку тачку доступну врапцу одређује најближи положај мачке, а затим најдаљи од свих тих најближих положаја. Резултат који се добија јесте тачан, али због много рачунања програм је врло спор.

```
// Lose resenje
#include <iostream>
```

```
using namespace std;

int vPoc, vKraj, mPoc, mKraj;
int d(int x){
    if (x < mPoc)
        return mPoc - x;
    else if (x < mKraj)
        return 0;
    else
        return x - mKraj;
}

int main() {
    cin >> vPoc >> vKraj >> mPoc >> mKraj;
    int rez = 0;
    for (int x  = vPoc; x <= vKraj; x++)
        rez = max(rez, d(x));

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

Задатак: Неравнотежа

Аутор: Милан Вуīделија

Четворо ђака првака при повратку из школе пролазе кроз парк и увек сврате на клацкалицу. Осим што воле да се клацкају, они радо покушавају да претегну једни друге, јер је клацкалица нова и добро подмазана. При томе се ђаци труде да подела буде што равноправнија, да би игра била занимљивија.

Наши ђаци прво седну мирно по двоје на сваку страну клацкалице без својих школских торби, а затим, ако је клацкалица у неравнотежи, ђаци са лакше стране узму неколико школских торби. Након тога почиње игра претезања.

Написати програм који за дате масе четворо првака одговара на питање да ли они могу да постигну да збир маса ђака и могућих торби буде исти на обе стране и колико **најмање** торби је потребно за то у случају потврдног одговора. Свака торба је тешка два килограма.

Опис улаза

На стандардном улазу су четири цела позитивна броја, мања од 100, сваки у посебном реду. Ови бројеви представљају масе ђака.

Опис излаза

У први ред стандардног излаза исписати `da` или `ne`. Уколико је у први ред исписана реч `da`, у други ред исписати **најмањи могућ број** потребних торби, а то мора да буде `0`, `1` , `2` , `3` или `4`.

Пример 1

Улаз	*Излаз*
`40`	`da`
`35`	`1`
`37`	
`44`	

Објашњење

Када на једну страну седну деца од 35 и 44 килограма, а на другу од 37 и 40, неравнотежа је $(35+44)-(37+40)=79-77=2$, па равнотежа може да се постигне тако што лакши пар узме једну школску торбу.

Пример 2

Улаз

```
44
41
30
43
```

Излаз

```
ne
```

Објашњење

Како год да се ђаци распореде без торби, збир њихових маса са једне стране је за више од 8 килограма већи од збира са друге стране, па равнотежу није могуће постићи додавањем торби.

Пример 3

Улаз

```
35
35
35
35
```

Излаз

```
da
0
```

Решење

Нека је дат било који распоред деце на клацкалици, по два детета са сваке стране. Разлику збирова маса на левој и десној страни зваћемо неравнотежом.

Можемо да приметимо да у случају да је неравнотежа при једном распореду непарна, онда је неравнотежа при сваком распореду непарна, па додавањем торби од по два килограма ниједна неравнотежа не може да постане парна, а тиме ни нула. То значи да у случају једне непарне неравнотеже нема решења.

У случају да је најмања неравнотежа већа од укупне масе свих торби, тј. од 8 килограма, поново нема решења. Преостаје случај када је најмања неравнотежа парна и мања од 10. Тада равнотежа може да се постигне, а тражени број торби добијамо дељењем најмање неравнотеже са 2.

Остаје да одредимо најмању неравнотежу.

Могући распореди су:

- да ђаци маса a и b седе наспрам ђака маса c и d
- да ђаци маса a и c седе наспрам ђака маса b и d
- да ђаци маса a и d седе наспрам ђака маса b и c

Најмању неравнотежу можемо да израчунамо тако што израчунамо све три могуће неравнотеже и нађемо њихов минимум.

```
#include <iostream>

using namespace std;

int main() {
    int a, b, c, d;
    cin >> a >> b >> c >> d;

    int neravnoteza = min(abs(a+b-c-d), min(abs(a+c-b-d), abs(a+d-b-c)));
    if (neravnoteza <= 8 && neravnoteza % 2 == 0)
        cout << "da" << endl << neravnoteza / 2 << endl;
    else
        cout << "ne" << endl;
```

```
    return 0;
}
```

Најмања неравнотежа се добија када дете најмање масе и дете највеће масе седе на једној страни клацкалице, а остала два детета на другој страни. Докажимо ово тврђење.

Доказ: означимо масе ђака словима a, b, c, d, тако да важи $a \leq b \leq c \leq d$ (тако бирамо ознаке). Користећи ове ознаке, тврђење је да је неравнотежа најмања ако ђаци маса a и d седе наспрам ђака маса b и c.

Као што већ знамо, осим наведеног распореда седења постоје још само два: да a и c седе наспрам b и d, или да a и b седе наспрам c и d. Доказаћемо да неравнотежа при овим распоредима не може да буде мања од неравнотеже при распореду a и d наспрам b и c.

Разликоваћемо два случаја: $a + d \geq b + c$ и $b + c > a + d$.

1. Нека је $a + d \geq b + c$. Тада:

$$N_{ab} = |(c + d) - (a + b)| = (d - b) + (c - a) \geq (d - b) - (c - a) = (d + a) - (b + c) = |(d + a) - (b + c)| = N_{ad}$$
$$N_{ac} = |(b + d) - (a + c)| = (d - c) + (b - a) \geq (d - c) - (b - a) = (d + a) - (b + c) = |(d + a) - (b + c)| = N_{ad}$$

1. Нека је $b + c > a + d$. Тада:

$$N_{ab} = |(c + d) - (a + b)| = (c - a) + (d - b) \geq (c - a) - (d - b) = (b + c) - (a + d) = |(b + c) - (a + d)| = N_{ad}$$
$$N_{ac} = |(b + d) - (a + c)| = (b - a) + (d - c) \geq (b - a) - (d - c) = (b + c) - (a + d) = |(b + c) - (a + d)| = N_{ad}$$

Овим је доказ завршен. Следи да најмања неравнотежа може да се добије и као $|(b + c) - (a + d)|$, где су a и d најмања и највећа маса детета.

```
#include <iostream>

using namespace std;

int main() {
    int a, b, c, d;
    cin >> a >> b >> c >> d;

    int levo = max(max(a,b), max(c,d)) + min(min(a,b), min(c,d));
    int desno = a+b+c+d-levo;
    int neravnoteza = abs(levo-desno);
    if (neravnoteza % 2 == 0 && neravnoteza < 10)
        cout << "da" << endl << neravnoteza / 2 << endl;
    else
        cout << "ne" << endl;
    return 0;
}
```

Задатак: Дорћол опен

Аутор: Душан Пођадић

Два другара Маре и Вук организују тениски турнир Дорћол опен се игра по систему “на испадање” (губитник меча испада, победник пролази у наредну рунду) и има k кола и на њему учествује 2^k такмичара. Договорили су се да Маре буде главни судија, а Вук директор турнира. Маре није превише систематично бележио резултате и Вуку их је донео у следећој форми: за сваки меч пише ко је у њему победио и ко је изгубио, али не пише у ком је колу одигран. Помозите Вуку да на основу ових података направи поредак играча на турниру.

Опис улаза

У првом реду се налази природни број k - број кола које турнир има ($1 \leq k \leq 15$). У сваком наредном реду се налази резултат једног од мечева на турниру у формату идентификациони број победника и идентификациони број губитника, раздвојени размаком. Сваки такмичар има јединствен идентификациони број у распону од 0 до $2^k - 1$.

Опис излаза

У првом реду исписати победника турнира. У другом реду исписати другопласираног. У сваком наредном реду исписати идентификационе бројеве играча који су испали у колу пре (у трећем реду полуфиналe, у четвртом четвртфинале, у петом осмина финала итд. колико год да их има). У сваком реду је распоред идентификационих бројева произвољан.

Пример 1

Улаз	*Излаз*
2	3
3 0	0
3 1	1 2
0 2	

Објашњење

У првом колу је играч 3 победио играча 1, а играч 0 играча 2. У другом колу је играч 3 победио играча 0. Дакле победник турнира је играч 3, играч 0 је други, а играчи 1 и 2 су испали у полуфиналу.

Пример 2

Улаз

```
3
7 2
6 7
6 5
1 4
6 1
3 0
1 3
```

Излаз

```
6
1
3 7
0 4 2 5
```

Решење

Решење бројањем победа

Прво је потребно схватити да је број мечева за један мањи од броја играча. То се може лако закључити јер за сваког играча осим за победника важи да је тачно једном изгубио меч, а пошто у сваком мечу неко мора да изгуби закључујемо да је број мечева једнак броју играча минус 1.

Направићемо низ парова `pobede` у ком један члан пара представља ид играча, а други члан број његових победа. Овај низ попуњавамо током учитавања мечева тако што сваки пут повећамо број победу пару чији је ид учитан као победник меча. Након учитавања, сортирамо овај низ нерастуће по броју победа. Након сортирања први члан низа је победник турнира (има највише победа), други је другопласирани, трећи и четврти су полуфиналисти итд. У сваком реду исписујемо све ид-еве играча који имају исти број победа како се и тражило у задатку.

Сложеност решења

Сложеност овог решења је $(2^k * \log 2^k) = O(k \cdot 2^k)$ јер је потребно сортирати све играче којих има 2^k.

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

using namespace std;

bool compare(pair<int, int> p1, pair<int, int> p2){
  return p2.second < p1.second;
}

// Niz parova koji predstavljaju pobede igraca. Prvi clan para je id igraca, a drugi je broj njegovih pobeda
vector<pair<int, int>> pobede;

int main() {

  int k;
  cin >> k;
  int brojIgraca = (int)pow(2, k);
  int brojMeceva = brojIgraca - 1;

  for(int i = 0; i < brojIgraca; i++){
    pair<int, int> x(i, 0);
    pobede.push_back(x);
  }

  for(int i = 0; i < brojMeceva; i++){
    int pobednik, gubitnik;
    cin >> pobednik >> gubitnik;
    pobede[pobednik].second++; // uvecavamo broj pobeda pobednika za 1
  }

  sort(pobede.begin(), pobede.begin() + brojIgraca, compare);
  int brPobeda = pobede[0].second;
  for(int i = 0; i < brojIgraca; i++){
    if(pobede[i].second == brPobeda) // svi igraci u trenutnom kolu
      cout << pobede[i].first << " ";
    else{ // prelazimo u naredno kolo
      cout << endl << pobede[i].first << " ";
      brPobeda = pobede[i].second;
    }
  }

  return 0;
}
```

Решење са више пролазака кроз низ бројача победа

Ово решење је слично претходном, с тим што не сортирамо низ победа већ по њему тражимо прво играче који имају k победа (победник), затим $k-1$ победа (другопласирани), $k-2$ победа (полуфиналисти) итд. и редом их исписујемо.

Сложеност решења

Ово решење је сложености $O((2^k)^2) = O(4^k)$.

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

```cpp
#include <cmath>

using namespace std;

int main() {
    int k, pobednik, gubitnik;
    cin >> k;
    int brojIgraca = (int)pow(2, k);
    int brojMeceva = brojIgraca - 1;
    vector <int> pobede(brojIgraca, 0);

    for(int i = 0; i < brojMeceva; i++) {
        cin >> pobednik >> gubitnik;
        pobede[pobednik]++;
    }

    for(int brojPobeda = k; brojPobeda >= 0; brojPobeda--) {
        for(int i = 0; i < brojIgraca; i++) {
            if(pobede[i] == brojPobeda)
                cout << i << " ";

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

Решење са више пролазака кроз низ бројача победа

Ово решење је практично исто као и претходно, само другачије записано.

Задатак: Непарни делиоци

Аутор: Оћњен Тешић

Напиши програм који за унети природни број n одређује најмањи природан број m такав да $n \cdot m$ има непаран број делилаца у скупу природних бројева (укључујући 1 и $n \cdot m$).

Опис улаза

У једином реду стандардног улаза се налази један природан број n ($n \leq 10^{12}$).

Опис излаза

У једином реду стандардног излаза исписати један природан број m, који представља најмањи природан број такав да $n \cdot m$ има непаран број делилаца у скупу природних бројева.

Тест примери су подељени у четири независне (дисјунктне) групе: - У тест примерима вредним 10 поена: $n \leq 10^2$. - У тест примерима вредним 10 поена: $n \leq 10^3$. - У тест примерима вредним 30 поена: $n \leq 10^6$. - У тест примерима вредним 50 поена: $n \leq 10^{12}$.

Пример 1

Улаз	*Излаз*
3	3

Објашњење

Делиоци броја $3 \cdot 3 = 9$ су $1, 3, 9$. Лако се види да бројеви $3 \cdot 1 = 3$ и $3 \cdot 2 = 6$ имају 2, односно 4 делиоца у скупу природних бројева.

Пример 2

Улаз

```
12
```

Излаз

```
3
```

Објашњење

Делиоци броја $12 \cdot 3 = 36$ су $1, 2, 3, 4, 6, 9, 12, 18, 36$. Лако се види да бројеви $12 \cdot 1 = 12$ и $12 \cdot 2 = 24$ имају 6, односно 8 делиоца у скупу природних бројева.

Решење

Решење када $n \leq 10^2$

У овом случају је довољно испитати све бројеве $1, 2, 3, \ldots, n$ као кандидате за m, проверити петљом од 1 до $n \cdot m$ колико бројева у интервалу $[1, n \cdot m]$ дели број $n \cdot m$ и одредити најмањи m такав да непаран број бројева у интервалу $[1, n \cdot m]$ дели број $n \cdot m$. Ово решење ради у сложености $\mathcal{O}(n^3)$.

Решење када $n \leq 10^3$

У овом случају треба урадити малу модификацију првог случаја. Делиоце ћемо избројати петљом од 1 до $\sqrt{n \cdot m}$. Ово решење ради у сложености $\mathcal{O}(n^2)$.

Да би се решиле и остале групе тест примера, потребно је приметити следеће: ***Број има непаран број делилаца ако и само ако је квадрат природног броја.***

Ово се може закључити на два начина.

Први начин. Ако природан број d дели n, тада и $\frac{n}{d}$ дели n. Из тога можемо закључити да, уколико су за свако d, бројеви d и $\frac{n}{d}$ различити, сваки број d има свог пара, па би то резултирало парним бројем делилаца. Бројеви су једнаки ако важи $n = d^2$, тј. ако је n квадрат природног броја (тада и даље сви делиоци осим $\sqrt{n}$ имају свог пара).

Други начин. Посматрајмо канонску факторизацију броја $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$. Сваки делилац броја n има облик $d = p_1^{s_1} p_2^{s_2} \cdots p_k^{s_k}$, где је $0 \leqslant s_i \leqslant \alpha_i$ за свако $i = 1, 2, \ldots, k$. Сваки од експонената s_i може се изабрати на $\alpha_i + 1$ начина. То нам даје укупно $(\alpha_1 + 1)(\alpha_2 + 1) \cdots (\alpha_k + 1)$ могућности за делилац d. То је уједно и број делилаца броја n. Приметимо да је број делилаца непаран ако и само ако је $\alpha_i + 1$ непаран, односно α_i паран за свако $i = 1, 2, \ldots, k$. То управо значи да је n потпун квадрат.

Према томе, задатак смо свели на еквивалентан задатак - *да одредимо најмање m такво да је $n \cdot m$ потпун квадрат*.

Решење када $n \leq 10^6$

У овом случају је довољно испитати петљом све бројеве $1, 2, 3, \ldots, n$ као кандидате за m и проверити да ли је тренутни број помножен са n потпун квадрат. Ово решење ради у сложености $\mathcal{O}(n)$.

Комплетно решење

Да бисмо задатак решили за 100 поена, потребно је да посматрамо канонску факторизацију броја $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$. Циљ нам је да број $n \cdot m$ буде потпун квадрат тј. да сви експоненти буду парни бројеви. Стога, број m ћемо потражити у облику $m = p_1^{\beta_1} p_2^{\beta_2} \cdots p_k^{\beta_k}$ (тада $\alpha_i + \beta_i$ треба да буде паран број за свако $i = 1, 2, \ldots, k$). Уколико је експонент (простог фактора p_i) α_i паран, тада ће m бити најмањи за $\beta_i = 0$, а уколико је експонент (простог фактора p_i) α_i непаран, тада ће m бити најмањи за $\beta_i = 1$.

Дакле, m се добија као производ простих бројева са непарним експонентом у факторизацији броја n. Ово решење ради у сложености факторизације броја тј. $\mathcal{O}(\sqrt{n})$.

```
#include <iostream>
```

```
using namespace std;

long long OstD(long long& x, long long d) {
    long long dd = d*d;
    while (x % dd == 0)
        x = x / dd;

    if (x % d == 0) {
        x = x / d;
        return d;
    } else {
        return 1;
    }
}

long long KvOst(long long x) {
    long long ost2 = OstD(x, 2);
    long long ost3 = OstD(x, 3);
    long long ost = ost2 * ost3;
    long long d = 5;
    long long d1 = 2;
    while (d*d <= x) {
        ost *= OstD(x, d);
        d += d1;
        d1 = 6-d1;
    }

    return ost*x;
}

int main() {
    long long n;
    cin >> n;
    cout << KvOst(n) << endl;
    return 0;
}
```

Задатак: Шибице

Аутор: Владан Ковачевић

Дато је n шибица (палидрваца) које не морају бити исте дужине. Испитати да ли се коришћењем свих датих шибица може формирати квадрат. Шибице не смеју да се ломе на мање делове, већ само да се надовезују.

Опис улаза

Са стандардног улаза уноси се број n ($1 \leq n \leq 16$), затим n бројева који представљају дужине шибица (дужине су природни бројеви мањи или једнаки 500).

Опис излаза

На стандардни излаз исписати 1 ако је могуће формирати квадрат, у супротном исписати 0.

Пример 1

Улаз	*Излаз*
6 3 2 4 1 2 4	1

Објашњење

Можемо формирати квадрат чија је дужина једне ивице 4, тако што ће прве две ивице да буду шибице дужине

4, трећа ивица ће да буде сачињена од 2 шибице дужине 2, док ће четврта ивица да буде сачињена од шибица дужина 1 и 3.

Пример 2

Улаз

```
6
3 2 4 2 2 4
```

Излаз

```
0
```

Пример 3

Улаз

```
6
2 4 6 8 10 12
```

Излаз

```
0
```

Решење

Ако укупна дужина свих шибица није дељива са 4, од њих се не може направити квадрат. Ако јесте дељив са 4, задатак решавамо рекурзивном функцијом којом набрајамо све могуће распореде шибица на 4 странице троугла. Жељена дужина сваке странице је четвртина укупне дужине свих страница. Функција прима текућу шибицу и тренутне дужине страница квадрата (добијене распоређивањем свих претходних шибица). Покушавамо да текућу шибицу придружимо свакој страници квадрата (то је могуће ако се додавањем текуће шибице и даље добија дужина странице која је мања или једнака од текуће дужине). Након придруживања текуће шибице некој страници, рекурзивно покушавамо да распоредимо све остале шибице. Ако нема више шибица које треба да се распореде, успешно смо распоредили све шибице.

Приметимо да се програм може мало убрзати тако што прву шибицу увек доделимо само првој страници квадрата.

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

using namespace std;

bool moze_kvadrat(const vector<int>& sibice, int i,
                  vector<int>& tekuce_duzine_stranica,
                  int zeljena_duzina_stranice) {
  // ispitali smo sve sibice
  if (i == sibice.size())
    return true;

  // pokusavamo da tekucu sibicu dodelimo jednoj po jednoj stranici
  // kvadrata
  for (int k = 0; k < 4; k++) {
    if (sibice[i] + tekuce_duzine_stranica[k] <= zeljena_duzina_stranice) {
      tekuce_duzine_stranica[k] += sibice[i];
      if (moze_kvadrat(sibice, i+1,
                       tekuce_duzine_stranica, zeljena_duzina_stranice))
        return true;
      tekuce_duzine_stranica[k] -= sibice[i];
    }
  }
  return false;
}
```

```
bool moze_kvadrat(const vector<int>& sibice) {
  // racunamo ukupnu duzinu svih sibica
  int ukupna_duzina = 0;
  for (int sibica: sibice)
    ukupna_duzina += sibica;
  // ako ona nije deljiva sa 4, tada se sigurno ne moze napraviti kvadrat
  if (ukupna_duzina % 4 != 0)
    return false;
  // pokusavamo da svaku od sibica dodelimo
  vector<int> duzine_stranica{sibice[0], 0, 0, 0};
  return moze_kvadrat(sibice, 1, duzine_stranica, ukupna_duzina / 4);
}

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

  cout << moze_kvadrat(sibice) << endl;
  return 0;
}
```

Задатак: Биоскоп

Аутор: Оћьен Тешић

У једном чудном биоскопу се налази n редова у којима **не мора** да се налази исти број седишта. У првом реду се налази a_1 седишта, у другом реду се налази a_2 седишта, …, у n-том реду се налази a_n седишта. Парови долазе на пројекцију филма, двоје људи који су пар треба да седе једно поред другог (у истом реду). Колико највише парова може да буде у биоскопу у исто време?

Опис улаза

Прва линија стандардног улаза садржи природан број n који представља број редова у биоскопу (у биоскопу је највише 1000 редова). Наредних n линија стандардног улаза садрже по један природан број – i-ти члан низа a, који представља број седишта у i-том реду биоскопа.

Опис излаза

На стандардни излаз исписати тражени број парова.

Пример

Улаз	*Излаз*
5 7 8 4 5 6	14

Решење

Ако у реду са индексом i има паран број седишта, тада у тај ред може да седне $\frac{a_i}{2}$ парова. Слично, ако у реду са индексом i има непаран број седишта, тада у тај ред може да седне $\frac{a_i - 1}{2}$ парова. Приметимо да је у оба случаја тај број једнак количнику целобројно подељених бројева a_i и 2. Стога, одговор ће бити $\left\lfloor \frac{a_1}{2} \right\rfloor + \left\lfloor \frac{a_2}{2} \right\rfloor + \left\lfloor \frac{a_3}{2} \right\rfloor + \cdots + \left\lfloor \frac{a_n}{2} \right\rfloor$.

```
#include <iostream>
```

```cpp
using namespace std;

int main()
{
    int n;
    cin >> n;
    int broj_sedista;
    int rezultat = 0;
    for(int i = 0; i < n; i++){
        cin >> broj_sedista;
        rezultat += broj_sedista / 2;
    }
    cout << rezultat << endl;
    return 0;
}
```

Задатак: Најбрже до циља

Аутор: Милан Вугделија

Тачке у матрици представљају празна поља, а слова T представљају капије система телепорта. Након стајања на капију, кретање може да се настави из исте или било које друге капије. За употребу телепорта не троши се никакво додатно време.

У матрици још постоји једно слово `S` (старт) и једно слово `C` (циљ). Дозвољени су кораци у сваком од 4 смера, тј. по једно поље горе, доле, лево или десно, ако се тиме не излази из матрице.

Написати програм који за дату матрицу одређује најмањи могућ број корака од старта до циља.

Опис улаза

У првом реду стандардног улаза налазе се два цела позитивна броја R и K, мања или једнака 200, раздвојена једним размаком. Број R је број редова, а K број колона матрице.

У сваком од наредних R редова налази се по K знакова раздвојених по једним размаком. Знакови могу да буду `.`, `T`, `S` или `C`, с тим да се појављује укупно тачно једно `S` и тачно једно `C`.

Опис излаза

На стандардни излаз исписати један неозначен цео број, најмањи могућ број корака од старта до циља.

Пример 1

Улаз	*Излаз*

`4 7`
`. . . . T . .`
`. S . . . . .`
`. . . . . . C`
`T . . T . . T`
	`4`

Објашњење

Један низ од 4 корака који доводе од старта до циља су: доле, доле, лево, горе.

Пример 2

Улаз

```
3 8
. . . . . T T .
. S . . . . . .
T . . C . . . T
```

Излаз

```
3
```

Објашњење

Један низ од 3 корака који доводе од старта до циља су: десно, десно, доле.

Решење

Након (или током) учитавања матрице, могу у једном пролазу кроз њу да се одреде координате старта S_i, S_j и циља C_i, C_j.

У другом пролазу кроз матрицу, за свако слово `T` може да се одреди растојање (број потребних корака) од старта до тог слова, као и од тог слова до циља. Успут, у истом пролазу, може да се израчуна минимум свих растојања од старта до неког слова `T`, означен са ST_{min}, као и минимум свих растојања од неког слова `T` до циља , означен са TC_{min}.

Дужина најкраћег пута од старта до циља користећи систем телепорта је $ST_{min} + TC_{min}$. Дужина најкраћег пута уопште је или ова, или број корака од старта до циља, па се коначан одговор добија као мања од ове две вредности, тј. као $\min(ST_{min} + TC_{min}, |S_i - C_i| + |S_j - C_j|)$

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

int main()
{
        int r, k, iStart = 0, jStart = 0, iCilj = 0, jCilj = 0;
        cin >> r >> k;
        vector <vector <char>> a(r, vector <char>(k, '.'));
        for (int i = 0; i < r; i++) {
                for (int j = 0; j < k; j++) {
                        cin >> a[i][j];
                        if (a[i][j] == 'S') {
                                iStart = i;
                                jStart = j;
                        }
                        else if (a[i][j] == 'C') {
                                iCilj = i;
                                jCilj = j;
                        }
                }
        }
        int minST = r + k, minTC = r + k;
        for (int i = 0; i < r; i++) {
                for (int j = 0; j < k; j++) {
                        if (a[i][j] == 'T') {
                                minST = min(minST, abs(iStart - i) + abs(jStart - j));
                                minTC = min(minTC, abs(iCilj - i) + abs(jCilj - j));
                        }
                }
        }
    int distSC = abs(iStart - iCilj) + abs(jStart - jCilj);
        cout << min(distSC, minST + minTC) << endl;
}
```

Задатак: Најмање промена

Аутор: Иван Дрецун

Најпознатији анонимни хакер Душан се запрепастио када је прочитао рачун за Интернет. Душан је својим невероватним хакерским способностима (комшија Мића му ради у Интернету) сазнао да ако своју потрошњу интернета представи као низ нула и јединица, Интернет ће му наплатити по један динар за сваку промену са

0 на 1 или обрнуто, читајући низ са лева на десно. На пример, ако му је потршња интернета `000110`, рачун за тај месец ће бити 2 динара.

Одлучио је да следећег месеца хакује самог себе у нади да ће му стићи мањи рачун. Пошто хаковање није лако, он може да одабере само један број k и затим промени (са 0 на 1 или обрнуто) сваки k-ти елемент низа. На пример, ако је низ `000110` и он одабере број 2, након измене низ постаје `010011`. Напиши програм који за Душанову потрошњу следећег месеца одређује колики је најмањи могући рачун који ће му стићи након хаковања.

Опис улаза

Са стандардног улаза се уноси један стринг нула и јединица који представља потрошњу за наредни месец. Дужина стринга је највише 100000.

Опис излаза

На стандардни излаз исписати два природна броја одвојена размаком. Први број представља најбоље k (између 1 и n) које Душан може да одабере, а други број представља рачун за такво k. Ако постоји више таквих вредности за k, исписати најмању.

Пример 1

Улаз	*Излаз*
`0010111`	`4 1`

Објашњење

Рачуни за $k = 1$, $k = 2$, ... редом су `1101000`, `0111101`, `0000101`, `0011111`, `0010011`, `0010101` и `0010110`.

Пример 2

Улаз

```
00101
```

Излаз

```
2 1
```

Објашњење

Рачуни за $k = 1$, $k = 2$, ... редом су `11010`, `01111`, `00001`, `00111` и `00100`.

Решење

Претрага по k

Задатак је могуће решити испробавањем сваке дозвољене вредности броја k и одређивањем износа рачуна. Приликом претраге памтимо најбоље решење.

Сложеност оваквог решења је $O(n^2)$ зато што за сваку вредност броја k морамо одредити вредност рачуна проласком кроз цео низ.

```
#include <iostream>

using namespace std;

void flip(char& c) {
        if(c == '0')
                c = '1';
        else
                c = '0';
}

int diff(string& s) {
        int d = 0;
        for(int i = 1; i < s.size(); i++)
                d += s[i] != s[i - 1];
```

```
        return d;
}

int main() {
        string s;
        cin >> s;

        int n = s.size();

        int mind = diff(s);
        int mink = 1;
        for(int k = 1; k <= n; k++) {
                for(int i = k - 1; i < n; i += k)
                        flip(s[i]);

                int d = diff(s);
                if(d < mind) {
                        mind = d;
                        mink = k;
                }

                for(int i = k - 1; i < n; i += k)
                        flip(s[i]);
        }

        cout << mink << ' ' << mind << '\n';
        return 0;
}
```

Ефикасно решење

Ефикасније решење је могуће постићи ако запазимо да приликом одређивања вредности рачуна није потребно пролазити кроз цео низ. Довољно је једном, пре почетка претраге, израчунати укупан рачун. Затим, када желимо да одредимо вредност рачуна за неко k, довољно је да израчунамо за колико ће се број разлика повећати или смањити приликом обртања свих k-тих елемената. Што је вредност броја k већа, то ће бити све мање елемена на којима се прави измена. Приметимо да већ на половини претраге, када је $k > \frac{n}{2}$, обрћемо само по један елемент низа.

Прецизније, сложеност овог алгоритма је $O(n \log n)$. У првом кораку обрћемо свих n елемената, у другом кораку $\frac{n}{2}$ елемената, у трећем $\frac{n}{3}$, итд. Укупан број корака ће бити мањи од $n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n-1} + \frac{n}{n} = n(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{n}{n}) < n(1 + \frac{1}{2} + \frac{1}{3} + \ldots) = O(n \log n)$. Образложење последње једнакости може се наћи на Википедији (потражите чланак `Harmonic_series_(mathematics)`).

```
#include <iostream>

using namespace std;

int main() {
        string s;
        cin >> s;

        int n = s.size();

        int diff = 0;
        for(int i = 1; i < n; i++)
                diff += s[i] != s[i - 1];

        int diff_min = diff;
        int diff_k = 1;
```

```
        for(int k = 2; k <= n; k++) {

                int flip_diff = diff;
                for(int i = k - 1; i < n; i += k) {
                        if(s[i] == s[i - 1])
                                flip_diff++;
                        else
                                flip_diff--;

                        if(i < n - 1) {
                                if(s[i + 1] == s[i])
                                        flip_diff++;
                                else
                                        flip_diff--;
                        }
                }

                if(flip_diff < diff_min) {
                        diff_min = flip_diff;
                        diff_k = k;
                }
        }

        cout << diff_k << ' ' << diff_min << '\n';
        return 0;
}
```

Задатак: Амљез

Аутор: Душан Попадић

На чаробној планети Аљмез објекти могу да имају позитивну и негативну масу. На тој планети се налази велики координатни систем на који је Анави поставила лагане лопте дуж у-осе. Све лопте које је поставила имају апсолутну вредност масе између 0,1 и 0,15 гк-а (гк је мера за масу на Аљмезу), с тим што је Анави водила рачуна да буде приближно једнак број лопти са позитивном и негативном масом (ти бројеви се разликују за највише 1). Анавин млађи брат Пилиф тренира лабдуф (спорт у коме је дозвољено само трчање уназад и додиривање лопте ногама) и жели да вежба свој шут. Он прилази свакој лопти и шутира је истом јачином од 126 А (А је мерна јединица за јачину шута лопте) у правцу одређеном х-осом (дакле не мења се у-координата лопте). Лопта лети и пада на место чија се х-координата добија када се јачина шута (126) подели масом лопте. На пример ако имамо лопту чија је у-координата 20,3 и маса -0,15, она ће слетети на место са координатама (-840, 20,3) јер је 126/-0,15 = -840. Када је Анави видела шта је Пилиф урадио, прво се јако наљутила, а онда му рекла да мора да покупи све лопте. Пошто Пилифа наравно мрзи да то уради, замолио те је да му помогнеш тако што ћеш одредити којим редом да купи лопте тако да укупна дистанца коју пређе буде најкраћа могућа. Занемари дистанцу коју Пилиф треба да пређе од своје тренутне позиције до прве лопте са твог списка. У случају више тачних решења исписати било које.

Помоћ: Ако имамо две лопте на координатама (x_1, y_1) и (x_2, y_2), растојање између њих се рачуна као $\sqrt{(x_1 - x_2) \cdot (x_1 - x_2) + (y_1 - y_2) \cdot (y_1 - y_2)}$.

Опис улаза

У првом реду стандардног улаза се налази n $(3 \leq n \leq 12)$, број лопти које је Анави поставила. У наредних n редова се налазе по два броја раздвојена размаком: y-координата лопте $(-50 \leq y_i \leq 50)$ и маса лопте $(0.1 \leq |m| \leq 0.15)$.

Ограничења:

- У тест примерима вредним 20 поена важи $y_i = 0$.
- У тест примерима вредним 50 поена додатно важи $n \leq 8$.

Опис излаза

У једином реду стандардног излаза исписати n различитих бројева одвоједних размаком – тражени редослед лопти. На пример, ако Пилиф треба да оде прво по 3. лопту која је унета на улазу, па по прву, па по другу исписати 3 1 2.

Пример

Улаз	*Излаз*

`6`
`5 -0.1`
`5 -0.15`
`0 -0.15`
`0 0.1`
`-5 0.1`
`0 0.15`
	`1 2 3 6 4 5`

Објашњење

Са слике се види да је тражени распоред оптималан. Алтернативно решење је `5 4 6 3 2 1`.

Решење

Проблем пред нама је добро познат проблем путујућег трговца у ком је потребно одредити најкраће растојање при обиласку свих тачака у равни.

Решење сортирањем по x оси

Уколико наивно приступимо решавању овог проблема можемо помислити да је довољно да обилазимо тачке тако што кренемо од најлевље и онда идемо редом по x оси. Ово решење, наравно, није тачно у општем случају и доноси само 10 поена.

Сложеност решења

Сложеност овог решења је једнака сложености сортирања низа, односно $n \cdot \log n$

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

using namespace std;


struct lopta{
  double x;
  double y;
  int ind;
};

// Funkcija za poredjenje lopti po x osi
bool compare(lopta& a, lopta& b){
  return a.x <= b.x;
}

int main() {

  int n;
  cin >> n;

  vector<lopta> lopte(n);

  for(int i = 0; i < n; i++){
    lopta x;
```

```
    double m;
    double y;
    cin >> y >> m;
    x.x = 126 / m;
    x.y = y;
    x.ind = i + 1;
    lopte[i] = x;
  }

  sort(lopte.begin(), lopte.end(), compare);

  for(int i = 0; i < n; i++){
    cout << lopte[i].ind << " ";
  }

  return 0;
}
```

Решење генерисањем свих распореда лопти

Класично решење за овај проблем је генерисање свих распореда (пермутација) тачака, одређивање укупног растојања за сваки од тих распореда и онда одређивање распореда који има најмање растојање. Пермутације генеришемо користећи рекурзивну методу $permutacije$. Растојање између две тачке у равни се одређује коришћењем Питагорине теореме. Ово решење доноси 50 поена.

Сложеност решења

Генерисање свих распореда има сложеност која је једнака броју распореда, дакле $O(n!)$. Пошто је за сваки распоред додатно потребно проћи кроз њега и одредити укупно растојање које се пређе ходајући од лопте до лопте, то додаје додатни фактор на сложеност и укупна сложеност је $O(n \cdot n!)$. Ово решење је могуће додатно побољшати тако што ће се при генерисању пермутација успут рачунати и укупно растојање и онда би укупна сложеност била $(n!)$.

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

using namespace std;


vector<double> ys;
vector<double> xs;


// Funkcija za odredjivanje rastojanja izmedju dve tacke u ravni Pitagorinom teoremom
double rastojanje(double x1, double y1, double x2, double y2){
  return sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
}

// najbolja permutacija do sada
vector<int> najPerm;
// najmanje rastojanje do sada
double najR = 1000000000;

void permutacije(int i, int n, vector<int>& perm){
  if(i == n){
    // Ako smo stigli do kraja odredjujemo ukupno rastojanje za generisanu permutaciju
    // i pamtimo je ako je najmanja do sada.
```

```
    vector<int> p(n);
    double r = 0;

    for(int k = 0; k < n; k++){
      p[k] = perm[k];
      if(k > 0)
        r += rastojanje(xs[p[k]], ys[p[k]], xs[p[k-1]], ys[p[k-1]]);
    }
    if(r <= najR){
      najR = r;
      najPerm = p;
    }
  }
  else{
    for(int j = i; j < n; j++){
      int p = perm[i];
      perm[i] = perm[j];
      perm[j] = p;
      permutacije(i + 1, n, perm);
      p = perm[i];
      perm[i] = perm[j];
      perm[j] = p;
    }
  }
}


int main() {

  int n;
  cin >> n;

  for(int i = 0; i < n; i++){
    double m;
    double y;
    cin >> y >> m;
    xs.push_back(126 / m);
    ys.push_back(y);
  }

  // Niz perm predstavlja jednu permutaciju inicijalno ucitanih lopti. U njemu se ne nalaze lopte vec indeksi iz originalr
  vector<int> perm(n);
  for(int i = 0; i < n; i++)
    perm[i] = i;

  permutacije(0, n, perm);

  for(int i = 0; i < n; i++){
    cout << najPerm[i] + 1 << " ";
  }

  return 0;
}
```

Решење раздвајањем на две групе

Можемо приметити да се све лопте групишу у два правоугаоника: један који је лево од y осе и један који је десно. Та два правоугаоника су међусобно доста удаљена: Ако имамо најтеже могуће лопте различитих

знакова (-0,15 и 0,15) које имају исту y координату, оне ће обе пасти на удаљеност од $126/0.15 = 840$ од y осе, односно биће на међусобном растојању од тачно $840+840 = 1680$. Са друге стране, две лопте које су са исте стране y осе ће бити међусобно најудаљеније ако падну у два темена квадрата која су по дијагонали, то растојање биће $\sqrt{(420^2 + 100^2)} = 408$ што је значајно мање од до било које лопте која је са супротне стране y осе. Одавде је интуитивно јасно да се оптимална путања добија тако што се прво обиђу тачке са једне, а затим са друге стране y осе, тј. Пилиф само једном треба да пређе преко y осе. Ово се може и лако доказати тако што се сваки наредни (осим првог) прелазак преко y осе може заменити увећењем додатне (краће) везе између две лопте са исте стране и променом почетне лопте. Дакле решење је одредити све распореде са једне и са друге стране y осе којих има по $7! = 5040$ и онда пробати комбинацију сваке леве са сваком десном и тако одредити укупно најкраћи пут.

Сложеност решења

Генерисање свих распореда има сложеност која је једнака броју распореда, дакле $O((n/2)!)$ са сваке стране. Затим је потребно пробати сваку леву са сваком десном, па се добија коначна сложеност од $((n/2)! \cdot (n/2)!)$. Поређења ради за $n = 14$, $n!\ 87 \cdot 10^9$, а $(n/2)! \cdot (n/2)!\ 25 \cdot 10^6$. Ово решење доноси максималан број поена.

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

using namespace std;


vector<double> ysl;
vector<double> xsl;
vector<int> il;
int nl = 0;

vector<double> ysd;
vector<double> xsd;
vector<int> id;
int nd = 0;

vector<vector<int>> perm1;
vector<double> rast1;
int n1 = 0;

vector<vector<int>> perm2;
vector<double> rast2;
int n2 = 0;


double rastojanje(double x1, double y1, double x2, double y2){
  return sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
}

double rastojanje1(int i, int j){
  return rastojanje(xsl[i], ysl[i], xsd[j], ysd[j]);
}


void printVektor(vector<int>& v, int n){
  for(int i = 0; i < n; i++){
    cout << v[i] << " ";
  }
  cout << endl;
}
```

```
void printVektorD(vector<double>& v, int n){
  for(int i = 0; i < n; i++){
    cout << v[i] << " ";
  }
  cout << endl;
}

void permutacije(int i, int n, vector<int>& perm, int m){
  if(i == n){
    vector<int> p(n);
    double r = 0;
    if(m < 0){
        for(int k = 0; k < n; k++){
          p[k] = perm[k];
          if(k > 0)
            r += rastojanje(xsl[p[k]], ysl[p[k]], xsl[p[k-1]], ysl[p[k-1]]);
        }
        perm1.push_back(p);
        rast1.push_back(r);
        n1++;
    }
    else{
      for(int k = 0; k < n; k++){
      p[k] = perm[k];
      if(k > 0)
        r += rastojanje(xsd[p[k]], ysd[p[k]], xsd[p[k-1]], ysd[p[k-1]]);
      }
      perm2.push_back(p);
      rast2.push_back(r);
      n2++;
    }
  }
  else{
    for(int j = i; j < n; j++){
      int p = perm[i];
      perm[i] = perm[j];
      perm[j] = p;
      permutacije(i + 1, n, perm, m);
      p = perm[i];
      perm[i] = perm[j];
      perm[j] = p;
    }
  }
}


int main() {

  int n;
  cin >> n;

  for(int i = 0; i < n; i++){
    double m;
    double y;
    cin >> y >> m;
    if(m < 0){
      xsl.push_back(126 / m);
      ysl.push_back(y);
```

```
      il.push_back(i+1);
      nl++;
    }
    else{
      xsd.push_back(126 / m);
      ysd.push_back(y);
      id.push_back(i+1);
      nd++;
    }
  }

  vector<int> perm(max(nl, nd));
  for(int i = 0; i < nl; i++)
    perm[i] = i;
  permutacije(0, nl, perm, -1);
  for(int i = 0; i < nd; i++)
    perm[i] = i;
  permutacije(0, nd, perm, 1);

  double min = 100000000;
  int im, jm;

  for(int i = 0; i < n1; i++){
    for(int j = 0; j < n2; j++){
      double r = rast1[i] + rast2[j] + rastojanje1(perm1[i][nl-1], perm2[j][0]);
      if(r < min){
        min = r;
        im = i;
        jm = j;
      }
    }
  }

  for(int i = 0; i < nl; i++){
    cout << il[perm1[im][i]] << " ";
  }
  for(int i = 0; i < nd; i++){
    cout << id[perm2[jm][i]] << " ";
  }

  return 0;
}
```

Максималан број поена је могуће добити и коришћењем принципа динамичког програмирања за решавање порблема путујућег трговца, али познавање тог решења се не очекује од ученика на овом нивоу такмичења.

Глава 3

Општинско такмичење

Задатак: Дани у недељи

Аутор: Оџњен Тешић

На једној далекој планети, једна недеља траје n дана и дани су нумерисани бројевима $1, 2, 3, \ldots, n$. Дане бројимо од првог дана прве недеље. Познато је да је данас дан са укупним редним бројем a.

Напиши програм који одређује који ће дан у недељи по реду бити за x дана (та недеља не мора бити прва недеља).

Опис улаза

Прве три линије стандардног улаза садрже по један природан број. То су редом бројеви n, a и x, такви да је $1 \leq n \leq 1000$, $1 \leq a \leq 100000$, $1 \leq x \leq 100000$.

Опис излаза

На стандардни излаз исписати тражени редни број дана.

Пример 1

Улаз	*Излаз*
15 2 18	5

Објашњење

Данас је 2. дан, а за 18 дана ће бити 20. дан. Дани у другој недељи су 16, 17, 18, 19, 20, 21, …, 29. и 30. Према томе, то ће бити пети дан те недеље.

Пример 2

Улаз

```
7
3
20
```

Излаз

```
2
```

Објашњење

Данас је 3. дан, а за 20 дана ће бити 23. дан. Дани у четвртој недељи су 22, 23, 24, 25, 26, 27. и 28. Према томе, то ће бити други дан те недеље.

Решење

Приметимо, дани прве недеље су нумерисани бројевима од 1 до n, дани друге недеље бројевима од $n+1$ до $2n$, дани треће недеље бројевима од $2n+1$ до $3n$ итд. Једно решење је да израчунамо којој недељи припада дан са редним бројем $a+x$, па да на основу тога одредимо који је то дан у тој недељи. Други начин је да приметимо да је ово управо остатак при дељењу броја $a+x$ са n. Једини случај када треба бити пажљив је када је $a+x$ дељив са n. Тада је остатак при дељењу једнак 0, али је одговор n.

```
#include <iostream>

using namespace std;

int main() {
    int n, a, x;
    cin >> n >> a >> x;
    int redniBrojDana = (a + x) % n;
    if(redniBrojDana == 0)
        cout << n;
    else
        cout << redniBrojDana;
    return 0;
}
```

Задатак: Топлотни појас

Аутор: Филип Марић

Планета Земља се дели на следећих 5 топлотних појасева:

- *жарки појас* се простире између северног и јужног повратника
- *северни умерени појас* се простире између северног повратника и северног поларника
- *северни хладни појас* се простире изнад северног поларника
- *јужни умерени појас* се простире између јужног повратника и јужног поларника
- *јужни хладни појас* се простире испод јужног поларника

Ако се зна да се повратници налазе на $23°30'$ северне односно јужне географске ширине, а поларници на $66°30'$ северне односно јужне географске ширине, напиши програм који на основу унете географске ширине неког града одређује топлотни појас коме припада.

Опис улаза

Са стандардног улаза се уноси географска ширина. Уноси се прво број степени, затим број минута и онда слово `s` односно `j` које означава да ли је у питању северна или јужна географска ширина. Претпоставити да унети град није ни на једном повратнику ни на једном поларнику.

Опис излаза

На стандардни излаз исписати ознаку `z` (жарки), `su` (северни умерени), `sh` (северни хладни), `ju` (јужни умерени) или `jh` (јужни хладни), у зависности од појаса коме унета географска ширина припада.

Пример 1

Улаз	*Излаз*
15 28 j	z

Објашњење

Пошто се град налази на јужној полулопти, али изнад јужног повратника, у питању је жарки појас.

Пример 2

Улаз

```
40
15
```

```
s
```

Излаз

```
su
```

Објашњење

Град се налази на северној полулопти, између северног повратника и северног поларника, па се налази у северном умереном појасу.

Пример 3

Улаз

```
66
42
j
```

Излаз

```
jh
```

Објашњење

Град се налази на јужној полулопти, испод јужног поларника, па се налази у јужном хладном појасу.

Решење

Најједноставнији начин да поредимо углове задате у степенима и минутима је да их пре поређења претворимо у минуте. Угао који има s степени и m минута има укупно $60s + m$ минута. Након тога се до решења долази анализом свих могућих случајева.

```
#include <iostream>

using namespace std;

int uMinute(int s, int m) {
  return s * 60 + m;
}

int main() {
  int stepeni, minuti;
  char sj;

  cin >> stepeni >> minuti >> sj;

  if (uMinute(stepeni, minuti) < uMinute(23, 30))
    cout << "z" << endl;
  else if (uMinute(stepeni, minuti) < uMinute(66, 30))
    cout << sj << "u" << endl;
  else
    cout << sj << "h" << endl;
  return 0;
}
```

Задатак: Робот достављач

Аутор: Милан Вујделија

У једној фирми која се бави онлајн продајом, за доставу купљене робе користе се роботи. Купци које ће Робот Миле сутра услуживати станују сви на истој страни исте улице. Миле за сваку адресу зна њену координату, тј. растојање од почетка улице до те зграде. Миле је програмиран тако да адресе обилази редом којим су му задате.

Напиши програм који за дати број достава и дате координате зграда (растојања од зграда од почетка улице) израчунава и исписује укупно растојање које ће робот Миле прећи од прве до последње доставе.

Опис улаза

У првом реду стандардног улаза налази се природан број n, не већи од 30. У сваком од следећих n редова налази се по један неозначен цео број, растојање од почетка улице до следеће зграде (тј. координата зграде) у коју треба доставити наруџбину. Ови бројеви нису већи од 10000.

Опис излаза

На стандардни излаз исписати само један цео број, укупну дужину пута који пређе робот Миле.

Пример 1

Улаз	*Излаз*
5 2 7 7 1 3	13

Објашњење

Позиције зграда у редоследу достављања су 2, 7, 7, 1, 3. Миле прво иде од позиције 2 до позиције 7, затим од 7 до 7 (достава у истој згради), па од 7 до 1 и на крају од 1 до 3. Растојања пређена од доставе до доставе су редом 5, 0, 6, 2, а збир тих растојања је 13.

Пример 2

Улаз

```
1
7
```

Излаз

```
0
```

Објашњење

Прва и последња достава су иста достава. То значи да се Миле уопште не креће између прве и последње доставе и да је пређено растојање од прве до последње доставе једнако 0.

Решење

Зграде можемо да посматрамо као тачке на бројевној полуправој. Координате зграда су дати цели бројеви. Растојање између две зграде једнако је апсолутној вредности разлике њихових координата.

За датих n бројева треба да се израчуна збир растојања између узастопних зграда, а то је збир апсолутних вредности разлика суседних бројева.

Да би решење било једноставније, након читања броја n може и прва од датих n координата да се учита пре петље, а да се кроз петљу прође само $n - 1$ пута (а не n пута).

У петљи се учитава координата следеће зграде, растојање од претходне зграде до ње се додаје на збир и на крају се нова зграда запамти као претходна да би се рачунање исправно наставило у следећем пролазу кроз петљу.

На крају само треба да се испише израчунати збир.

```
#include <iostream>

using namespace std;

int main() {
    int n, gdeSamBio, gdeSamSad, predjeniPut;
    cin >> n;
    cin >> gdeSamBio;
```

```
    predjeniPut = 0;
    for (int i = 1; i < n; i++)
    {
        cin >> gdeSamSad;
        predjeniPut += abs(gdeSamSad - gdeSamBio);
        gdeSamBio = gdeSamSad;
    }
    cout << predjeniPut << endl;
    return 0;
}
```

Задатак: Минимах

Аутор: Владимир Кузмановић

Наставник је Перици задао домаћи задатак у којем на основу 4 задате декадне цифре треба да одреди разлику највећег и најмањег четвороцифреног броја који се могу формирати помоћу тих цифара. Написати програм који ће на основу 4 унете декадне цифре помоћи Перици да провери да ли је тачно урадио домаћи.

Опис улаза

У сваком од 4 реда стандардног улаза налази се по једна декадна цифра, од којих бар једна није нула.

Опис излаза

На стандардни излаз исписати само један природан број који представља разлику највећег и најмањег четвороцифреног броја који се помоћу датих цифара могу формирати.

Пример 1

Улаз	*Излаз*
3 7 5 9	6174

Објашњење

Највећи четвороцифрени број који се може формирати помоћу датих цифара је 9753, а најмањи је 3579. На стандардни излаз треба исписати њихову разлику $9753 - 3579 = 6174$.

Пример 2

Улаз

```
2
0
5
6
```

Излаз

```
4464
```

Објашњење

Највећи четвороцифрени број који се може формирати помоћу датих цифара је 6520, а најмањи је 2056. На стандардни излаз треба исписати њихову разлику $6520 - 2056 = 4464$.

Напомена: У задатку се тражи најмањи четвороцифрени број, па треба водити рачуна да број не сме почињати нулом. Управо због тога, најмањи четвороцифрени број је 2056.

Пример 3

Улаз

```
0
0
```

```
5
0
```

Излаз

```
0
```

Објашњење

Највећи четвороцифрени број који се може формирати помоћу датих цифара је 5000, а најмањи је такође 5000. На стандардни излаз треба исписати њихову разлику $5000 - 5000 = 0$.

Напомена: У задатку се тражи најмањи четвороцифрени број, па треба водити рачуна да број не сме почињати нулом. Управо због тога, најмањи четвороцифрени број је 5000, што ће уједно бити и највећи четвороцифрени број.

Решење

Цифре можемо сортирати употребом библиотечке функције. Због водећих нула потребно је обратити пажњу на формирање најмањег броја.

```
#include <iostream>

using namespace std;

int main() {

    int a, b, c, d;
    cin >> a >> b >> c >> d;

    // sortiranje
    if(a > b) swap(a, b);
    if(b > c) swap(b, c);
    if(c > d) swap(c, d);

    if(a > b) swap(a, b);
    if(b > c) swap(b, c);

    if(a > b) swap(a, b);


    // odredjivanje najveceg i najmanjeg broja
    int maxNum = d*1000 + c*100 + b*10 + a;
    int minNum = 0;
    if (a != 0) {
        minNum = a*1000 + b*100 + c*10 + d;
    }
    else if (b != 0) {
        minNum = b*1000 + c*10 + d;
    }
    else if (c != 0) {
        minNum = c*1000 + d;
    }
    else {
        minNum = d*1000;
    }

    cout << (maxNum - minNum) << endl;
    return 0;
}
```

Исти ефекат се може постићи и на мало другачији начин.

```cpp
#include <iostream>

using namespace std;

int main() {
    int a, b, c, d;
    cin >> a >> b >> c >> d;

    // sortiranje
    if(a > b) swap(a, b);
    if(b > c) swap(b, c);
    if(c > d) swap(c, d);

    if(a > b) swap(a, b);
    if(b > c) swap(b, c);

    if(a > b) swap(a, b);

    // odredjivanje najveceg i najmanjeg broja
    int maxNum = d*1000 + c*100 + b*10 + a;

    if (a == 0) swap(a, b);
    if (a == 0) swap(a, c);
    if (a == 0) swap(a, d);
    int minNum = a*1000 + b*100 + c*10 + d;

    cout << (maxNum - minNum) << endl;
    return 0;
}
```

Уместо библиотечке функције, могуће је извршити сортирање и ручно.

```cpp
#include <iostream>

using namespace std;

int main() {

   int m1, m2, m3, m4;
   int x;

   // prva cifra
   cin >> x;
   m4 = x;

   // druga cifra
   cin >> x;
   if (x > m4) {
      m3 = m4;
      m4 = x;
   }
   else {
      m3 = x;
   }

   // treca cifra
   cin >> x;
   if (x > m4) {
      m2 = m3;
```

```
      m3 = m4;
      m4 = x;
   }
   else if (x > m3) {
      m2 = m3;
      m3 = x;
   }
   else {
      m2 = x;
   }

   // cetvrta cifra
   cin >> x;
   if (x > m4) {
      m1 = m2;
      m2 = m3;
      m3 = m4;
      m4 = x;
   }
   else if (x > m3) {
      m1 = m2;
      m2 = m3;
      m3 = x;
   }
   else if (x > m2){
      m1 = m2;
      m2 = x;
   }
   else {
      m1 = x;
   }

   int maxNum = m4*1000 + m3*100 + m2*10 + m1;
   int minNum = 0;
   if (m1 != 0) {
      minNum = m1*1000 + m2*100 + m3*10 + m4;
   }
   else if (m2 != 0) {
      minNum = m2*1000 + m3*10 + m4;
   }
   else if (m3 != 0) {
      minNum = m3*1000 + m4;
   }
   else {
      minNum = m4*1000;
   }

   cout << (maxNum - minNum) << endl;

   return 0;
}
```

Задатак: Обим паралелограма

Аутор: Филип Марић

Ако је познат обим паралелограма O и то да је једна страница за d дужа од друге, напиши програм који одређује дужине страница овог паралелограма.

Опис улаза

Са стандардног улаза се уноси цео број O (између 1 и 200), а затим цео број d (између 1 и 100). Бројеви O и d су у сваком тесту такви да су тражене дужине страница цели бројеви.

Опис излаза

На стандардни излаз исписати у истом реду два цела броја раздвојена једним размаком. Ти бројеви треба да буду редом целобројне дужине страница a и b, тако да је $a \leq b$.

Пример 1

Улаз	*Излаз*
46 3	10 13

Објашњење

Страница $b = 13$ је заиста за 3 дужа од странице $a = 10$, а обим је $2(a + b) = 2(10 + 13) = 2 \cdot 23 = 46$.

Пример 2

Улаз

```
102
1
```

Излаз

```
25 26
```

Решење

Ако је краћа страница a, дужа $b = a + d$, важи да је $O = 2(a + b) = 2(a + a + d) = 4a + 2d$. Зато је $a = (O - 2d)/4$.

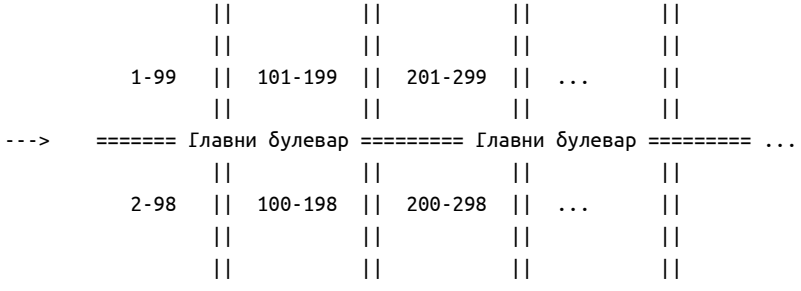
```cpp
#include <iostream>

using namespace std;

int main() {
  int O, d;
  cin >> O >> d;
  int a = (O - 2*d) / 4;
  int b = a + d;
  cout << a << " " << b << endl;
  return 0;
}
```

Задатак: Адресе

Са обе стране Главног булевара налазе се блокови зграда. Сваки блок је са једне стране булевара. Зграде са парним бројевима налазе се са десне, а зграде са непарним бројевима са леве стране булевара. Бројеви зграда које се налазе у истом блоку су исте парности и почињу истом цифром стотина (претходне цифре су им такође једнаке, ако постоје). Бројеви зграда у узастопним блоковима припадају узастопним стотинама (изузетак су бројеви дељиви са 100, види интервале у објашњењима).



Ако је потребно стићи са једне дате адресе на другу адресу у Главном булевару, одредити да ли се циљна адреса у односу на полазну налази у истој згради, у истом блоку, блоку преко пута, претходном блоку, следећем блоку, претходном блоку преко пута, следећем блоку преко пута, или неком другом блоку.

Опис улаза

У сваком од два реда стандардног улаза налази се по један један природан број, не већи од 100000. Први број је број зграде из које се полази, а други је број циљне зграде.

Опис излаза

На стандардни излаз исписати један од следећих текстова: `ista zgrada`, `isti blok`, `sledeci blok`, `prethodni blok`, `preko puta`, `sledeci blok preko puta`, `prethodni blok preko puta` или `daleko`.

Савет: да би се избегле грешке у куцању, најбоље је да се ови текстови копирају у програм.

Пример 1

Улаз	*Излаз*
`123`	`isti blok`
`147`	

Објашњење

Оба броја су непарна, па су зграде са исте стране булевара. При томе оба броја припадају интервалу $[100, 199]$, па се зграде налазе у истом блоку.

Пример 2

Улаз

```
158
249
```

Излаз

```
sledeci blok preko puta
```

Објашњење

Бројеви су различите парности, па се зграде налазе са различитих страна булевара, тј. циљна зграда је у неком блоку преко пута. Пошто број полазне зграде припада интервалу $[100, 199]$, а број циљне зграде интервалу $[200, 299]$, циљна зграда је у следећем блоку преко пута.

Решење

Циљна зграда може да буде "далеко" у односу на претходну било да је на истој или супротној страни булевара, неколико блокова пре или после блока у коме је полазна зграда. Да не бисмо морали да о сваком од тих случајева посебно водимо рачуна, најбоље је да одговор `daleko` буде подразумевани одговор, односно почетна вредност променљиве у којој држимо одговор. То је вредност која ће остати у променљивој ако не буде испуњен ниједан од услова под којима бисмо циљну зграду описали другачије.

Услови које треба да испитамо могу да се организују хијерархијски:

- ако су бројеви a и b једнаки, то је иста зграда
- иначе, ако су a и b исте парности (ако су зграде са исте стране булевара)
  - ако им је исти количник при дељењу са 100, зграда b је у истом блоку
  - иначе, ако је количник зграде b већи за један, зграда b је у следећем блоку
  - иначе, ако је количник зграде b мањи за један, зграда b је у претходном блоку
- иначе (бројеви различите парнсти, зграде са различитих страна булевара)
  - ако им је исти количник при дељењу са 100, зграда b је преко пута
  - иначе, ако је количник зграде b већи за један, зграда b је у следећем блоку преко пута
  - иначе, ако је количник зграде b мањи за један, зграда b је у претходном блоку преко пута

У свим осталим случајевима, зграда је далеко.

```
#include <iostream>

using namespace std;
```

```
int main() {
    int a, b;
    cin >> a >> b;
    bool istaStrana = (a%2 == b%2);
    int aStotina = a/100;
    int bStotina = b/100;
    string gde = "daleko";

    if (a==b)
        gde = "ista zgrada";
    else if (istaStrana) {
        if (aStotina == bStotina)
            gde = "isti blok";
        else if (bStotina == aStotina + 1)
            gde = "sledeci blok";
        else if (bStotina == aStotina - 1)
            gde = "prethodni blok";
    }
    else {
        if (aStotina == bStotina)
            gde = "preko puta";
        else if (bStotina == aStotina + 1)
            gde = "sledeci blok preko puta";
        else if (bStotina == aStotina - 1)
            gde = "prethodni blok preko puta";
    }

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

Задатак: Слепо куцање речи

Аутор: Филип Марић

Наставница информатике учи децу слепо куцање. За зада су обрадили само слова у основном реду (`a`, `s`, `d`, `f`, `g`, `h`, `j`, `k`, `l`). Напиши програм који помаже наставници да са списка речи одабере оне речи, које може да уврсти у вежбање тј. да одабере речи које се састоје само од слова из основног реда.

Опис улаза

Са стандардног улаза се учитава број n ($1 \leq n \leq 100$), а затим n речи, од којих свака садржи највише 10 малих слова енглеске абецеде (свака реч је у посебном реду).

Опис излаза

На стандардни излаз у једном реду исписати укупан број речи које се састоје од слова из основог реда, као и њихов укупан број слова (то је број слова које ђаци треба да откуцају у целом вежбању). Бројеве раздвојити једним размаком.

Пример 1

Улаз	*Излаз*

`4`
`alfa`
`lisa`
`kajak`
`beta`
	`2 9`

Објашњење

Речи које се састоје само од слова из основног реда су `alfa` и `kajak` и њихов укупан број слова је 4+5=9.

Пример 2

Улаз

```
3
grad
gama
delta
```

Излаз

```
0 0
```

Објашњење

Ниједна од ових речи не може да се уврсти у вежбање, па је и број речи и број слова у њима једнак 0.

Решење

Проверу да ли су сва слова речи из основног реда можемо извршити класичним алгоритмом линеарне претраге и имплементирати је у помоћној функцији. Анализирамо један по један карактер дате речи и ако нађемо на карактер који није у првом реду враћамо вредност нетачно. Ако прођемо све карактере речи и при том не наиђемо карактер ван основног реда, враћамо тачно.

У главном програму анализирамо једну по једну реч. За све оне за које помоћном функцијом уврдимо да се састоје искључиво из карактера основог реда увећавамо бројач речи за један и бројач слова за дужину те речи.

```
#include <iostream>

using namespace std;

bool OK_rec(const string& rec) {
  for (char c : rec)
    if (c != 'a' && c != 's' && c != 'd' && c != 'f' && c != 'g' && c != 'h' && c != 'j' && c != 'k' && c != 'l')
      return false;
  return true;
}

int main() {
  int n;
  cin >> n;
  int broj_reci = 0;
  int broj_slova = 0;
  for (int i = 0; i < n; i++) {
    string rec;
    cin >> rec;
    if (OK_rec(rec)) {
      broj_reci++;
      broj_slova += rec.size();
    }
  }

  cout << broj_reci << " " << broj_slova << endl;
  return 0;
}
```

Задатак: Одбојкашки резултат

Аутор: Душан Попадић

Напиши програм који на основу списка добијених поена од почетка одбојкашког меча одређује тренутни резултат.

Одбојкашки меч се игра док један тим не освоји три сета. Сет осваја први тим који освоји 25 или више поена, са разликом најмање 2. Дакле, тим може да освоји сет резултатом $25 : 0$, $25 : 1$, … или резултатом $25 : 23$. Резултат $25 : 24$ не означава крај сета јер разлика у поенима није барем 2. У том случају се игра наставља док један од тимова не направи разлику у поенима од најмање 2 (дакле $26 : 24$, $27 : 25$, $28 : 26$ …). Уколико дође до $2 : 2$ у сетовима, последњи сет се не игра до 25 него до 15 уз остала правила која остају иста (мора да се направи најмање 2 разлике). Када један тим освоји сет, од следећег поена се почиње нови сет у коме оба тима имају 0 поена на почетку.

Како се може доћи до резултата $25 : 24$**?** Ово се може десити уколико у једном тренутку оба тима имају по 24 поена, тада ће наредни освојен поен довести до $25 : 24$ или $24 : 25$, али то неће завршити сет.

Опис улаза

Са стандардног улаза се учитава ниска карактера `A` или `B` који редом означавају да ли је поен добио тим `A` или тим `B`. Гарантовано је да су улазни подаци могући, односно да ако неко слово означава поен којим се завршава меч, онда иза њега нема више слова на улазу. Улазна ниска садржи најмање један, а највише 500 карактера.

Опис излаза

У првом реду стандардног улаза исписати резултат у сетовима, у другом резултат у поенима у текућем сету (у сваком реду прво исписати резултат тима `A`, затим двотачку и на крају резултат тима `B`). Уколико је **меч завршен** исписати само резултат у сетовима, без резултата у поенима.

Пример 1

Улаз	*Излаз*
`AABABAABBBAAAABABBAABABAAAABBAAABABABABABABBABBA`	`1:0` `2:4`

Објашњење

Објашњење

Тим `A` осваја први сет резултатом $25 : 17$ (сет се завршио у тренутку када је тим `A` освојио свој 25. поен), а у другом сету води тим `B` резултатом $2 : 4$.

Пример 2

Улаз

```
BABABABABABABABBBABABABAAABABABABABABABABABABABAABBAB
ABBAAAAABBABBBAABBABABAAABABABABBBAABAAABBBBBBBAAABABBAA
```

Излаз

```
0:1
27:26
```

Објашњење

Објашњење

Први сет осваја тим `B` резултатом $29 : 31$. Сет се није завршио раније јер ниједан тим није успео да направи разлику од најмање 2 поена, а да је притом освојио бар 25 поена. У другом сету је тренутно резултат $27 : 26$ и сет није готов јер још нико није направио разлику од 2 поена, а да притом има бар 25 поена (иако су тимови током самог сета водили за више од 2, нису тада имали барем 25 поена).

Пример 3

Улаз

```
BABABABABABABABBBABABABAAABABABABABABABABABABABAABBABABBAAAAA
BBABBBAABBABABAAABABABABBAABAAABBBBBBBAAABABBAAAAAAAAAAAAAAAAAAA
AAAAAAAAABBBBBBBBBBBBBBBBBBBBBBABABABABABABABABABABBBBABB
```

Излаз

```
2:3
```

Објашњење

Објашњење

Први сет осваја тим `B` резултатом $29 : 31$. Други сет осваја тим `A` резултатом $28 : 26$. Трећи сет осваја тим `A` резултатом $25 : 0$, а четврти тим `B` резултатом $0 : 25$. Пети сет осваја тим `B` резултатом $11 : 15$. Пошто је меч готов, не исписујемо поене већ само сетове.

Решење

Главни део решења овог задатка је имплементација функције `dodajPoenTimu` у којој се додаје поен тиму који се проследи као први параметар. Функцији се као други параметар прослеђује противнички тим. У функцији се повећа број поена првом тиму, а онда се проверава да ли то доводи до краја сета. Уколико је број поена тог тима сада већи од 25 (или 15 ако је 2:2 у сетовима) и разлика у односу на други тим је бар 2, тим A је освојио сет, а бројеви поена оба тима се враћају на 0 због почетка наредног сета.

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

using namespace std;

struct tim {
  int setova;
  int poena;
};

void dodajPoenTimu(tim& A, tim& B){
  A.poena++;
  bool taj_brejk = A.setova == 2 && B.setova == 2;
  int cilj = taj_brejk ? 15 : 25;
  if (A.poena >= cilj && A.poena - B.poena >= 2) {
    A.setova++;
    A.poena = B.poena = 0;
  }
}

void init(tim& X){
  X.setova = 0;
  X.poena = 0;
}

int main() {
  tim A, B;
  init(A);
  init(B);
  string poeni;
  cin >> poeni;
  for (char poen : poeni) {
    if (poen == 'A') {
      dodajPoenTimu(A, B);
    } else if (poen == 'B') {
      dodajPoenTimu(B, A);
    }
  }

  cout << A.setova << ":" << B.setova << endl;
  if (A.setova < 3 && B.setova < 3) {
    cout << A.poena << ":" << B.poena << endl;
  }

  return 0;
```

```
}
```

Задатак: Бејзбол хитац

Аутор: Филип Марић

Бејзбол је веома популаран спорт у САД. Џек је три пута ударио лоптицу и измерено је колико је она одлетала и то у мерама које се користе у САД (јарди, стопе, инчи). Нас занима колики је најдаљи хитац од та три, али у мерама које ми разумемо (метри и центиметри). Напиши програм који нам помаже да то одредимо.

Напомена: Један инч има 2,54 центиметра. Једна стопа има 12 инча, а један јард има 3 стопе.

Опис улаза

Са стандардног улаза се уносе дужине три хица. За сваки хитац је дат цео број јарди, стопа и инча. Бројеви су дати у једном реду, раздвојени размаком, а ниједна дужина не прелази 50 јарди.

Опис излаза

На стандардни излаз исписати два цела броја раздвојена једним размаком, број метара и центиметара најдаљег хица, при чему је број центиметара заокругљен на најближи цео број (који ће у свим примерима бити једнозначан).

Пример 1

Улаз	*Излаз*

`18 2 7`
`18 1 7`
`14 0 9`
	`17 25`

Објашњење

Најдужи хитац је онај од 18 јарди, 2 стопе и 7 инча, што је 1724,66 центиметара. Заокругљивањем на најближи цео број добија се 1725 центиметара, што је 17 метара и 25 центиметара.

Пример 2

Улаз

```
21 1 7
18 0 3
24 2 11
```

Излаз

```
22 83
```

Објашњење

Најдужи хитац је онај од 24 јарда, 2 стопе и 11 инча, што је 2283,46 центиметара. Заокругљивањем на најближи цео број добија се 2283 центиметра, што је 22 метра и 83 центиметра.

Решење

У j јарди, f стопа и i инча има $12(3j + f) + i$ инча тј. $(12(3j + f) + i) \cdot 2,54$ центиметара. Дужине сва три хица претварамо у центиметре, одређујемо њихов максимум и заокружујемо добијени број центиметара на најближи цео број. Након тога број метара добијамо као целобрјни количник са 100, а број центиметара као остатак при дељењу са 100.

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

using namespace std;

int main() {
  int h1, h2, h3;
  int j, s, i;
```

```
  cin >> j >> s >> i;
  h1 = (j*3 + s)*12 + i;
  cin >> j >> s >> i;
  h2 = (j*3 + s)*12 + i;
  cin >> j >> s >> i;
  h3 = (j*3 + s)*12 + i;
  int cm = round(max({h1, h2, h3}) * 2.54);
  cout << cm / 100 << " " << cm % 100 << endl;
  return 0;
}
```

Задатак: Годишње доба

Аутор: Филип Марић

Рећи ћемо да пролеће почиње 20. марта (укључујући и тај дан) и траје до 21. јуна (без тог дана). Лето почиње 21. јуна (укључујући тај дан) и траје до 23. септембра (без тог дана). Јесен почиње 23. септембра (укључујући и тај дан) и траје до 21. децембра (без тог дана). Осталих дана је зима. Напиши програм који на основу унетог датума одређује годишње доба.

Опис улаза

Са стандардног улаза се уноси дан, а затим и месец (сваки број у посебном реду). Број 1 означава јануар, 2 фебруар, 3 март итд. Сматрати да је унети датум исправан.

Опис излаза

На стандардни излаз исписати слово `p` (пролеће), `l` (лето), `j` (јесен) или `z` (зима).

Пример 1

Улаз	*Излаз*
`1` `2`	`z`

Објашњење

Унет је први фебруар и он је током зиме.

Пример 2

Улаз

```
20
3
```

Излаз

```
p
```

Објашњење

Унет је 20. март и сматрамо да је он први дан пролећа.

Пример 3

Улаз

```
19
3
```

Излаз

```
z
```

Пример 4

Улаз

```
21
6
```

Излаз

```
l
```

Решење

Задатак је решити угнежђеним гранањем. Прво испитујемо месец. Ако откријемо да се ради о неком месецу који се простире кроз два годишња доба, посебно испитујемо дан.

```
#include <iostream>

using namespace std;

int main() {
  int dan, mesec;
  cin >> dan >> mesec;
  if (mesec < 3)
    cout << 'z' << endl;
  else if (mesec == 3) {
    if (dan < 20)
      cout << 'z' << endl;
    else
      cout << 'p' << endl;
  } else if (mesec < 6)
    cout << 'p' << endl;
  else if (mesec == 6) {
    if (dan < 21)
      cout << 'p' << endl;
    else
      cout << 'l' << endl;
  } else if (mesec < 9)
    cout << 'l' << endl;
  else if (mesec == 9) {
    if (dan < 23)
      cout << 'l' << endl;
    else
      cout << 'j' << endl;
  } else if (mesec < 12)
    cout << 'j' << endl;
  else if (mesec == 12) {
    if (dan < 21)
      cout << 'j' << endl;
    else
      cout << 'z' << endl;
  }
  return 0;
}
```

Задатак можемо решити и тако што датум представимо у облику четвороцифреног броја (који добијамо тако што месец помножмо са 100 и додамо дан).

```
#include <iostream>

using namespace std;

int main() {
    int dan, mesec;
    cin >> dan >> mesec;
    int mes_dan = 100 * mesec + dan;
```

```
    if (mes_dan < 320)
        cout << 'z' << endl;
    else if (mes_dan < 621)
        cout << 'p' << endl;
    else if (mes_dan < 923)
        cout << 'l' << endl;
    else if (mes_dan < 1221)
      cout << 'j' << endl;
    else
      cout << 'z' << endl;

    return 0;
}
```

Задатак: Пар-непар у круг

Аутор: Филип Марић

За дати природан број n исписати све низове бројева дужине $2n$ који испуњавају следећа три услова:

- садрже сваки број од 1 до $2n$ тачно једном;
- у низу се прво налазе непарни, а затим парни бројеви;
- део са непарним бројевима се добија ротирањем низа $1, 3, \ldots, 2n-1$, а део са парним бројевима се добија ротирањем низа $2, 4, \ldots, 2n$. Ротирање низа подразумева пребацивање првог елемента на крај низа (и то може бити поновљено 0 или више пута).

На пример, за $n = 3$ један такав низ је $3, 5, 1, 2, 4, 6$. Заиста, у њему се сваки број од 1 до $2n = 6$ јавља тачно једном, прво иду непарни бројеви $3, 5, 1$, а затим парни $2, 4, 6$, део $3, 5, 1$ се добија ротирањем низа $1, 3, 5$ за једну позицију (први елемент низа $1, 3, 5$ је пребачен на крај), а део $2, 4, 6$ ротирањем низа $2, 4, 6$ за нула позиција.

Опис улаза

Са стадардног улаза се учитава број n ($1 \leq n \leq 50$).

Опис излаза

На стандардни излаз исписати све тражене низове. Низови треба да буду поређани лексикографски (како је приказано у примерима): када се упореде било која два исписана низа, вредност првог елемента који им је различит мора бити мања у оном низу који је раније исписан.

Пример 1

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

Пример 2

Улаз	*Излаз*
3	1 3 5 2 4 6
	1 3 5 4 6 2
	1 3 5 6 2 4
	3 5 1 2 4 6
	3 5 1 4 6 2
	3 5 1 6 2 4
	5 1 3 2 4 6
	5 1 3 4 6 2
	5 1 3 6 2 4

Решење

Задатак решавамо коришћењем угнежђених петљи. Две спољне петље набрајају за колико места се ротирају непарни и за колико места се ротирају парни бројеви. Када су одређени ти бројеви (назовимо их i и j) исписујемо низове непарних тј. парних бројева. Сви непарни бројеви су облика $2k + 1$, а парни облика $2k + 2$. Ефекат ротације постижемо тако што k увећамо за i тј. j, а затим пронађемо остатак при дељењу са n. На пример, ако је $n = 4$ и $i = 2$ врши се следећи испис низа непарних бројева:

```
k:              0   1   2   3
2k+1:           1   3   5   7
```

```
(k+i)%n         2   3   0   1
2((k+i)%n)+1    5   7   1   3
```

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

using namespace std;

int main() {
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      for (int k = 0; k < n; k++)
        cout << 2*((k+i)%n) + 1 << " ";
      for (int k = 0; k < n; k++) {
        cout << 2*((k+j)%n) + 2;
        if (k < n)
          cout << " ";
      }
      cout << endl;
    }
  }
  return 0;
}
```

Глава 4

Окружно такмичење

Задатак: Обим или површина

Аутор: Филип Марић

Напиши програм који израчунава обим или површину правоугаоника.

Опис улаза

У прва два реда стандардног улаза се налазе два природна броја од 1 до 100 које представљају дужине страница правоугаоника. У трећем реду се налази или слово `O` или слово `P` (велико слово енглеске абецеде).

Опис излаза

Ако је учитано слово `O` на стандардни излаз исписати обим, а ако је унето слово `P`, исписати површину датог правоугаоника.

Пример 1

Улаз	*Излаз*
7	30
8	
O	

Пример 2

Улаз	*Излаз*
3	12
4	
P	

Решење

Након што се учитају подаци, гранањем се одређује да ли се тражи обим или површина. Обим се израчунава формулом $O = 2(a + b)$, а површина $P = a \cdot b$.

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

int main() {
  int a, b;
  cin >> a >> b;
  char c;
  cin >> c;
  if (c == 'O')
    cout << 2*(a+b) << endl;
  else if (c == 'P')
    cout << a*b << endl;
  return 0;
}
```

Задатак: Заједничка другарица

Аутор: Милан Вуїелија

Једна девојчица је позвала три своје другарице, које се међусобно не познају, на партију њихове омиљене игре учетворо. Ако је дато ко кога познаје, одредити име заједничке другарице. Уколико познанства не одговарају опису из задатка, исписати реч `nemoguce`.

Опис улаза

У сваком од три реда стандардног улаза налазе се по два различита имена раздвојена размаком, а то су имена једног пара девојчица које се познају. Имена се састоје од најмање једног, а највише 15 слова енглеске абецеде. Ни у која два реда се не појављује исти пар имена (ни у истом, ни у обрнутом редоследу).

Опис излаза

На стандардни излаз исписати име заједничке другарице, или реч `nemoguce`.

Пример 1

Улаз	*Излаз*

`Ana Milica`
`Milica Jovana`
`Tamara Milica`
	`Milica`

Објашњење

Име `Milica` се појављује у сва три реда, што значи да се Милица познаје са три друге девојчице, а оне се не познају међусобно. Према томе, заједничка другарица је Милица.

Пример 2

Улаз

```
Ana Branka
Branka Maja
Maja Ana
```

Излаз

```
nemoguce
```

Објашњење

Објашњење

Не постоји име које се појављује у сва три реда.

Пример 3

Улаз

```
Radmila Vasilija
Vasilija Svetlana
Svetlana Ljubica
```

Излаз

```
nemoguce
```

Објашњење

Објашњење

Не постоји име које се појављује у сва три реда.

Решење

Пошто подаци не морају да буду у складу са описом, добар почетак је да уведемо логичку променљиву која говори да ли подаци јесу или нису у складу са описом. Нека је то променљива `moguce`. Почетну вредност ове променљиве постављамо на "тачно", па ако установимо да подаци нису у складу са описом, променићемо јој вредност на "нетачно".

Означимо имена из првог реда a_1 и b_1, из другог a_2 и b_2, а из трећег са aa_3 и b_3. Ако се име a_1 понови у другом реду, тј. ако $a_1 = a_2 \vee a_1 = b_2$, тада је a_1 кандидаткиња за заједничку другарицу и подаци у прва два реда су у складу са опсиом. Слично, ако се име b_1 понови у другом реду, тј. ако $b_1 = a_2 \vee b_1 = b_2$, тада је

b_1 кандидаткиња за заједничку другарицу и подаци у прва два реда су и тада у складу са описом. Преостали случај је да се ниједно име из првог реда не појављује у другом реду, а у том случају подаци нису у складу са описом (променљивој `moguce` додељујемо вредност “нетачно”).

Ако су подаци у прва два реда у складу са описом (променљива `moguce` још увек има вредност “тачно”), треба још да се провери да ли се име кандидаткиње за заједничку другарицу појављује и у трећем реду. Ако се не појављује, онда укупни подаци ипак нису у складу са описом, па ћемо променљивој `moguce` и у том случају да доделимо вредност “нетачно”.

Након свих наведених провера, на основу вредности променљиве `moguce` можемо да закључимо шта треба да буде исписано. Ако променљива `moguce` има вредност “тачно”, исписујемо име кандидаткиње, јер се њено име појавило у сва тру реда и она је заједничка другарица. У противном треба да испишемо реч `nemoguce`.

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

int main() {
    string a1, b1, a2, b2, a3, b3, kandidatkinja = "";
    cin >> a1 >> b1 >> a2 >> b2 >> a3 >> b3;
    bool moguce = true;
    if (a1 == a2 || a1 == b2)
        kandidatkinja = a1;
    else if (b1 == a2 or b1 == b2)
        kandidatkinja = b1;
    else
        moguce = false;

    if (kandidatkinja != a3 && kandidatkinja != b3)
        moguce = false;

    if (moguce)
        cout << kandidatkinja << endl;
    else
        cout << "nemoguce" << endl;
    return 0;
}
```

Алтернативна имплементација може бити следећа.

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

int main()
{
    string a, b, c, d, e, f;
    cin >> a >> b >> c >> d >> e >> f;
    if ((a == c || a == d) && (a == e || a == f))
        cout << a;
    else if ((b == c || b == d) && (b == e || b == f))
        cout << b;
    else
        cout << "nemoguce";
    return 0;
}
```

Задатак: Поправка степеништа

Аутор: Милан Вуīелија

Поједини степеници једног старог каменог степеништа су неравномерно утонули у земљиште, тако да се при кретању од почетка до краја степеништа можда не иде све време нагоре, као што је било док је степениште било ново, већ може да има и суседних степеника исте висине, па чак и делова степеништа на којима се иде надоле. Степениште може да се поправи тако што се на неке степенике постави по једна или више нових плоча. Висина сваке плоче је 1.

Написати програм који за дате целобројне висине степеника одређује коллико најмање плоча је потребно, да би сваки степеник (осим првог) био виши од претходног.

Опис улаза

У првом реду стандардног улаза се налази један природан број n ($1 \leq n \leq 50$) који представља број степеника. У другом реду је n природних бројева раздвојених по једним размаком, висине степеника редом.

Опис излаза

На стандардни излаз исписати један цео број, најмањи број плоча потребних за постизање циља.

Пример 1

Улаз	*Излаз*
1 4	0

Објашњење

Постоји само један степеник (висине 4), па нема ни суседних степеника исте висине, ни делова који воде надоле. Зато ово степениште не треба поправљати, тј. број плоча потребних за поправку је 0.

Пример 2

Улаз

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

Излаз

```
9
```

Објашњење

Ако се на трећи степеник ставе три плоче, на четврти пет, а на пети једна, сваки степеник ће бити виши од претходног (висине ће бити $3, 4, 2+3=5, 1+5=6, 6+1=7$). При томе је употребљено $3+5+1=9$ плоча, а са мање плоча није могуће да се оствари циљ.

Решење

У задатку се очекује да пребројимо све плоче које треба додати на степенике, да би то степениште било поправљено. За то ћемо да употребимо једну целобројну променљиву као бројач. Почетна вредност бројача је, наравно, нула. Да бисмо одредили најмањи могућ број плоча које треба додати, довољно је да сваки степеник (осим почетног) који је једнак или нижи од претходног повисимо до висине за један веће од висине претходног степеника. При томе број додатих плоча додајемо и на бројач, који исписујемо након обраде свих степеника.

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

int main() {
  int n, prethodni, tekuci;
  cin >> n >> prethodni;
  int brPloca = 0;
  for (int i = 1; i < n; i++) {
    cin >> tekuci;
    if (tekuci <= prethodni) {
      brPloca += prethodni - tekuci + 1;
      tekuci = prethodni + 1;
    }
```

```
    prethodni = tekuci;
  }
  cout << brPloca << endl;
  return 0;
}
```

Задатак: Замени највећу цифру

Аутор: Оћњен Тешић

Написати програм који учитава природни број n ($n \leq 10^9$) и цифру c и исписује број који се добија када се у броју n сва појављивања његове највеће цифре замене цифром c. Гарантује се да примери неће бити такви да ће новодобијени број имати прву цифру 0 (тј. прва цифра новог броја неће бити 0).

Опис улаза

У првом реду стандардног улаза се уноси природан број n и размаком одвојена цифра c.

Опис излаза

На стандардни излаз исписати новодобијени број.

Пример

Улаз	*Излаз*
`48283417 0`	`40203417`

Објашњење

Највећа цифра броја 48283417 је 8. Када заменимо сва појављивања цифре 8 цифром 0, добијамо број 40203417.

Решење

Задатак може да се реши коришћењем ниски.

```
#include <iostream>

using namespace std;

int main() {
  string s;
  cin >> s;

  int c;
  cin >> c;

  char najveca = -1;
  for (char c : s)
    najveca = max(c, najveca);

  for (int i = 0; i < s.length(); i++)
    if (s[i] == najveca)
      s[i] = c + '0';

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

Задатак: Путовање

Аутор: Оћњен Тешић

Путујући од Аграда до Београда Маја пролази поред четири путоказа (није обавезно тим редом којим су приказани на Улазу). Путоказ са редним бројем i је описан са два броја: a_i и b_i - ово значи да је тај путоказ удаљен a_i километара од Аграда и b_i километара од Београда. Међутим, један путоказ је нетачан. Одредити који.

Опис улаза

У првој линији стандардног улаза се налазе два броја a_1 и b_1.

У другој линији стандардног улаза се налазе два броја a_2 и b_2.

У трећој линији стандардног улаза се налазе два броја a_3 и b_3.

У четвртој линији стандардног улаза се налазе два броја a_4 и b_4.

Маја не воли дуго да путује, па су сви бројеви на улазу мањи од 10^9.

Опис излаза

На стандардни излаз исписати тачно један број – редни број путоказа који је нетачан. Путокази су нумерисани редним бројевима - први - редним бројем 1, други - редним бројем 2, трећи - редним бројем 3 и четврти - редним бројем 4.

Пример

Улаз	*Излаз*
2 9 8 3 9 4 5 6	3

Решење

Потребно је исписати редни број реда на улазу где је збир $a_i + b_i$ другачији од збира у преостала три реда. Ово се може урадити користећи сортирање, али је довољно неколико угнежђених гранања као у коду испод.

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

int main() {
  int a1, b1, a2, b2, a3, b3, a4, b4;
  cin >> a1 >> b1 >> a2 >> b2 >> a3 >> b3 >> a4 >> b4;
  int zbir1 = a1 + b1, zbir2 = a2 + b2, zbir3 = a3 + b3, zbir4 = a4 + b4;
  if (zbir1 == zbir2) {
    if (zbir1 == zbir3)
      cout << 4;
    else
      cout << 3;
  } else {
    if (zbir1 == zbir3)
      cout << 2;
    else
      cout << 1;
  }
  return 0;
}
```

Задатак: Производ простих

Аутор: Оџьен Тешић

Дат је природан број n ($n \leq 10^9$). Одредити производ простих цифара у том броју. За природан број кажемо да је *прост* ако има тачно два различита делиоца у скупу природних бројева. Бројеви 0 и 1 нису прости.

Опис улаза

На стандардном улазу се уноси тачно један природан број n.

Опис излаза

На стандардни излаз треба исписати један природан број - производ простих цифара у том броју.

Уколико број нема простих цифара, исписати `-1` на стандардни излаз.

Пример 1

Улаз	*Излаз*
`52483579`	`1050`

Објашњење

Производ простих цифара је $5 \cdot 2 \cdot 3 \cdot 5 \cdot 7 = 1050$.

Пример 2

Улаз

```
186
```

Излаз

```
-1
```

Објашњење

Бројеви 1, 8 и 6 нису прости.

Пример 3

Улаз

```
1026
```

Излаз

```
2
```

Решење

Просте цифре су само 2, 3, 5 и 7. Можемо издвајати једну по једну цифру броја (одређивањем остатка и количника при дељењу са 10), проверавати да ли је проста и ако јесте, множити производ (иницијализован на 1) њоме. Уз то морамо помоћу посебне логичке променљиве пратити да ли се јавила нека проста цифра (ако није, треба исписати -1).

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n;
  bool postojiProst = false;
  int proizvodProstih = 1;
  while (n > 0) {
    int poslednjaCifra = n % 10;
    if (poslednjaCifra == 2 || poslednjaCifra == 3 ||
        poslednjaCifra == 5 || poslednjaCifra == 7) {
      postojiProst = true;
      proizvodProstih *= poslednjaCifra;
    }
    n /= 10;
  }
  if (!postojiProst)
    cout << -1;
  else
    cout << proizvodProstih;
```

```
  return 0;
}
```

Задатак: Реченице у заградама

Аутор: Филип Марић

Дата је ниска који садржи појединачне речи и реченице раздвојене размацима, при чему су реченице заграђене угластим заградама. Напиши програм који одређује све записане речи и реченице.

Опис улаза

Са стандардног улаза се учитава ниска дужине највише 10000 карактера састављена од малих слова енглеске абецеде, размака и угластих заграда. Ниска је исправно записана и садржи речи и реченице (заграде нису угнежђене тј. унутар једних не налазе се друге заграде).

Опис излаза

На стандардни излаз исписати сваку реч, односно реченицу у посебном реду.

Пример 1

Улаз

```
[zdravo svima] dobar dan [kako ste] pozdrav
```

Излаз

```
zdravo svima
dobar
dan
kako ste
pozdrav
```

Пример 2

Улаз

```
pozdrav svima
```

Излаз

```
pozdrav
svima
```

Пример 3

Улаз

```
Danas je [takmicenje] [iz programiranja]
```

Излаз

```
Danas
je
takmicenje
iz programiranja
```

Решење

Можемо преписивати један по један карактер са улаза на излаз, при чему заграде не преписујемо, а размаке мењамо преласком у нови ред (ако нису унутар заграда) или их преписујемо неизмењене (ако су унутар заграда).

Можемо и да градимо ниске које представљају речи и реченице. Пролазимо кроз карактере дате ниске. Помоћу логичке променљиве пратимо да ли се текући карактер налази унутар заграда. Када се наиђе на размак, ако се налази у заградама, дописујемо га на крај текуће ниске, а ако се не налази, исписујемо тренутну ниску и започињемо нову. Тренутну ниску исписујемо и када дођемо до краја улазне ниске.

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

using namespace std;

int main() {
  string tekst;
  getline(cin, tekst);
  bool u_zagradama = false;
  string trenutno = "";
  for (char c : tekst) {
    if (c == '[')
      u_zagradama = true;
```

```
    else if (c == ']')
      u_zagradama = false;
    else if (c == ' ') {
      if (u_zagradama)
        trenutno += c;
      else {
        cout << trenutno << endl;
        trenutno = "";
      }
    } else
      trenutno += c;
  }
  cout << trenutno << endl;
  return 0;
}
```

Задатак: Такмичари

Аутор: Оћьен Тешић

За пар такмичара кажемо да је *добар* ако је разлика у поенима два такмичара највише k (може бити и тачно k). Одредити број добрих парова на том такмичењу.

Опис улаза

У првом реду стандардног улаза се налази број такмичара n ($n \leq 10^5$).

У другом реду стандардног улаза се налази n природних бројева мањих од 10^5 – поени такмичара.

У трећем реду се налази највећа допуштена разлика поена k ($1 \leq k \leq 10^5$).

У 50% примера ће важити $n \leq 10^3$.

Опис излаза

На стандардни излаз треба исписати тачно један цео број - број добрих парова на том такмичењу.

Пример 1

Улаз	*Излаз*
6 1 12 2 9 5 8 4	8

Објашњење

Такмичари са следећим поенима чине добар пар: $(1, 2)$, $(1, 5)$, $(12, 9)$, $(12, 8)$, $(2, 5)$, $(9, 5)$, $(9, 8)$, $(5, 8)$.

Пример 2

Улаз

```
10
15 3 24 17 9 16 3 22 19 8
5
```

Излаз

```
14
```

Решење

Решење грубом силом

У решењу грубом силом се пореде свака два такмичара и увећава се број парова ако је њихова разлика у поенима не већа од k. Како је број парова једнак $\frac{n \cdot (n-1)}{2}$, то је сложеност овог решења $O(n^2)$, што је довољно само за мале вредности n.

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

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
  int k;
  cin >> k;
  int broj_dobrih = 0;
  for (int j = 1; j < n; j++)
    for (int i = 0; i < j; i++)
      if (abs(a[j] - a[i]) <= k)
        broj_dobrih++;
  cout << broj_dobrih << endl;
  return 0;
}
```

Решење техником два показивача

Задатак се ефикасно може решити на више начина ако се низ прво сортира (на пример, након сортирања може се применити бинарна претрага). Приказаћемо решење техником два показивача.

На почетку низ соритрамо растуће. Леви показивач i иницијализујемо на почетак низа, а десни на други елемент. Показивач j померамо све док је $a_j - a_i \leq k$. Када прођемо кроз све такве парове, повећамо број парова за $j - i$. Тиме су пронађени сви парови у којима учествује такмичар са скором a_i, па померимо показивач i за једно место удесно и понављамо процес (померање j) за нову вредност i. Поступак се понавља тако даље све док показивач десни показивач не дође до краја низа.

Коначно, како је најсложенија операција у претходно описаном решењу заправо алгоритам сортирања, то $O(n \log n)$ представља укупну временску сложеност решења овог задатка која је квазилинеарна по дужини улазног низа.

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

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; i++)
    cin >> a[i];
  int k;
  cin >> k;
  sort(begin(a), end(a));
  long long broj_dobrih = 0;
  int i = 0;
  for (int j = 1; j < n; j++) {
    while (i < j && a[j] - a[i] > k)
      i++;
```

```
    broj_dobrih += j - i;
  }
  cout << broj_dobrih << endl;
  return 0;
}
```

Задатак: Троугао највећег обима

Аутор: Филип Марић

Ако су познате дужине страница неколико троуглова, одредити вредност највећег обима тих троуглова.

Опис улаза

Са стандардног улаза се уноси број n ($1 \leq n \leq 100$), а затим у наредних n редова по три броја који представљају дужине страница троугла (све тројке бројева задовољавају одговарајуће неједнакости трогула).

Опис излаза

На стандардни излаз исписати највећи обим троугла.

Пример

Улаз	*Излаз*
3 3 4 5 10 3 11 5 7 8	24

Решење

Учитавамо тројке бројева (странице троугла), рачунамо њихов збир (обим троугла) и израчунавамо максимум тако добијених збирова.

```
#include <iostream>

using namespace std;

int main() {
   int n;
   cin >> n;
   int m = 0;
   for(int i = 0; i < n; i++) {
      int a, b, c;
      cin >> a >> b >>c;
      m = max(a+b+c, m);
   }
   cout << m << endl;
   return 0;
}
```

Задатак: Турнир

Аутор: Оџењен Тешић

Иван је записао резултате 4 четвртфинална меча, 2 полуфинална меча и финалног меча нокаут фазе турнира (укупно 7 мечева). Резултати који су дати на улазу нису сортирани (на пример, последњи ред на улазу не значи нужно да је тај меч последњи одигран). Одредити победника и другопласираног на турниру.

Опис улаза

Седам линија стандардног улаза садрже по 2 имена раздвојена једном размаком. Прво име у реду представља победника, а друго пораженог у том мечу. Сва имена се састоје од најмање једног, а највише 10 слова енглеске абецеде.

Опис излаза

У првом реду стандардног излаза исписати победника турнира. У другом реду стандардног излаза исписати другопласираног на турниру.

Пример

Улаз	*Излаз*

`Bojan Aleksa`
`Vanja Goran`
`Emil Zivadin`
`Emil Vanja`
`Vanja Bojan`
`Emil Dusan`
`Dusan Djordje`
	`Emil`
	`Vanja`

Објашњење

У четврфиналу су резултати били следећи:

- Емил је победио Живадина,
- Душан је победио Ђорђа,
- Бојан је победио Алексу,
- Вања је победио Горана.

У полуфиналу је Емил победио Душана, а Вања је победио Бојана.

Коначно, у финалу је Емил победио Вању.

Решење

Приметимо да једино победник турнира има три победе и зато се једино његово име појављује три пута као прво од два имена по редовима улаза. Слично томе, једино учесник који је изгубио у финалу има две победе и његово име се појављује два пута као прво у реду. Према томе, довољно је да се анализирају прва имена по редовима улаза и да се пронађу имена која се као прва у реду појављују три, односно два пута.

Један начин да пронађемо тражена имена је да помоћу мапе пребројимо појављивања сваког имена као првог у реду (кључ ће бити име, а вредност број појављивања). Након пребројавања, проласком кроз имена свих учесника који имају бар једну победу лако проналазимо име које се појављује као име победника три пута (освајач турнира), односно два пута (учесник који је поражен у финалу).

```
#include <iostream>
#include <map>

using namespace std;

int main() {
  // racunamo broj pobeda svakog igraca
  map<string, int> brPobeda;
  for (int i = 0; i < 7; ++i) {
    string ime1, ime2;
    cin >> ime1 >> ime2;
    brPobeda[ime1]++;
  }

  // pobednik je onaj ko je pobedi0 3 puta
  for (auto kv : brPobeda)
    if (kv.second == 3)
      cout << kv.first << endl;

  // finalista je onaj ko je pobedio 2 puta
  for (auto kv : brPobeda)
    if (kv.second == 2)
      cout << kv.first << endl;
```

```
  return 0;
}
```

Тражена имена можемо да нађемо и тако што у низу од 7 имена учесника који су побеђивали, за једно по једно име проверавамо да ли се у низу појављује потребан број пута. Име које се појави 3 пута је име освајача, а име које се појави 2 пута је име учесника који је поражен у финалу.

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

// pronalazi ucesnika koji se u nizu pobednik javlja k puta
string ucesnikSaKPobeda(int k, string pobednik[]) {
  for (int i = 0; i < 7; ++i) {
    string kandidat = pobednik[i];
    int nadjeniBrojPobeda = 0;
    for (int j = 0; j < 7; ++j)
      if (kandidat == pobednik[j])
        nadjeniBrojPobeda++;
    if (nadjeniBrojPobeda == k)
      return kandidat;
  }
  return "";
}

int main() {
  // ucitavamo podatke o pobednicima i porazenima
  string pobednik[7];
  string porazeni[7];
  for (int i = 0; i < 7; ++i)
    cin >> pobednik[i] >> porazeni[i];
  // pobednik turnira je onaj takmicar koji ima 3 pobede
  cout << ucesnikSaKPobeda(3, pobednik) << endl;
  // finalista turnira je onaj takmicar koji ima 2 pobede
  cout << ucesnikSaKPobeda(2, pobednik) << endl;
  return 0;
}
```

Задатак: Маказице

Аутор: Милан Вуīделија

У једној дечијој игри, свако дете стоји поред свог дрвета, осим једног, којег ћемо звати кројач. Кројач нема дрво крај којег би стајао, већ иде од детета до детета и пита: “Код кога су маказице?”. Упитано дете на то каже једно од имена остале деце. Док се кројач креће ка поменутом детету, сва остала деца покушавају да размене места, а кројач покушава да заузме неко дрво, тако што стигне до дрвета пре другог детета, или бар истовремено са њим. Ако у томе успе, његову улогу преузима дете које је остало без дрвета.

У овом задатку се претпоставља да се сва деца крећу истом брзином и да кројач одмах примећује сваку започету размену места и креће ка ближем од два привремено слободна дрвета. Прво дете покушава да процени са колико друге деце би оно могло да размени место, а да их кројач не угрожава. Можете ли да му помогнете?

Опис улаза

У првом реду стандардног улаза су два цела броја раздвојена размаком, координате кројача. У другом реду је један цео број n, број дрвећа, тј. број деце која нису кројач, не већи од 50. У сваком од наредних n редова су по два цела броја раздвојена размаком, координате једног детета. Све координате су из интервала $[0, 30000]$.

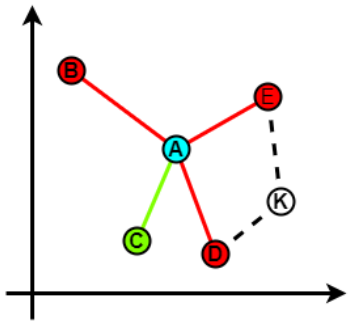
Опис излаза

На стандардни излаз исписати један цео број, број деце са којима прво дете може безбедно да размени место.

Пример 1

Улаз	*Излаз*
19 7 5 11 11 3 17 8 4 14 3 18 15	1

Објашњење



Пример 1

Крojач је означен тачком $K(19, 7)$, прво дете тачком $A(11, 11)$, а остала деца тачкама $B(3, 17)$, $C(8, 4)$, $D(14, 3)$ и $E(18, 15)$.

- Растојање између A и B је $\sqrt{(11-3)^2+(11-17)^2} = \sqrt{64+36} = 10$, а растојање између K и A је $\sqrt{(19-11)^2+(7-11)^2} = \sqrt{64+16} = \sqrt{80} < 10$, што значи да крojач стиже до тачке A пре него што A и B размене места, па та размена није безбедна;
- Растојање између A и C је $\sqrt{(11-8)^2+(11-4)^2} = \sqrt{9+49} = \sqrt{58}$, док су растојања $KA = \sqrt{80}$ и $KC = \sqrt{(19-8)^2+(7-4)^2} = \sqrt{130}$ оба већа од AC, па A и C могу безбедно да размене места;
- Растојање између A и D је $\sqrt{(11-14)^2+(11-3)^2} = \sqrt{9+64} = \sqrt{73}$, а растојање $KD = \sqrt{(19-14)^2+(7-3)^2} = \sqrt{25+16} = \sqrt{41} < AD$, па крojач стиже до тачке D пре него што A и D размене места;
- Растојање између A и E је $\sqrt{(11-18)^2+(11-15)^2} = \sqrt{49+16} = \sqrt{65}$, а растојање $KE = \sqrt{(19-18)^2+(7-15)^2} = \sqrt{1+64} = \sqrt{65} = AE$, па крojач стиже до тачке E истовремено са A и размена места за A и E није безбедна.

Закључујемо да је од 4 разматране размене само једна безбедна (то је тазмена између A и C), па на излаз треба исписати `1`.

Пример 2

Улаз

```
100 100
4
50 50
150 50
50 150
150 150
```

Излаз

```
0
```

Решење

Нека је K тачка у којој се налази крojач, A тачка у којој се налази прво дете, а P тачка у којој се налази још неко дете, које није ни крojач ни прво дете. Тада A и P могу безбедно да размене места ако и само ако је

дужина дужи AP мања и од дужине дужи KA и од дужине дужи KP.

Дужине дужи могу да се израчунају помоћу Питагорине теореме. На пример, дужина дужи AK је $\sqrt{(K_x - A_x)^2 + (K_y - A_y)^2}$. Међутим, неједнакости које важе за дужине дужи, важиће и за квадрате тих дужина. То значи да можемо да избегнемо рачунање квадратних корена и да поредимо квадрате дужина дужи. Тиме програм постаје бржи и прецизнији (поређење целих бројева је увек поптпуно прецизно, док поређење реалних бројева понекад доводи до грешке).

Из ове анализе следи да се задатак своди на пребројавање оних тачака P за које је $KA^2 > AP^2 \wedge KP^2 > AP^2$, односно $(K_x - A_x)^2 + (K_y - A_y)^2 > (A_x - P_x)^2 + (A_y - P_y)^2$ и $(K_x - P_x)^2 + (K_y - P_y)^2 > (A_x - P_x)^2 + (A_y - P_y)^2$.

```
#include <iostream>

using namespace std;

int main() {
    long long kx, ky, x0, y0, xi, yi;
    int n;
    cin >> kx >> ky >> n >> x0 >> y0;
    long long d0k = (x0-kx)*(x0-kx) + (y0-ky)*(y0-ky);
    long long br = 0;
    for(int i = 1; i < n; i++)
    {
        cin >> xi >> yi;
        long long d0i = (x0-xi)*(x0-xi) + (y0-yi)*(y0-yi);
        long long dki = (kx-xi)*(kx-xi) + (ky-yi)*(ky-yi);
        if (d0i < min(d0k, dki))
            br++;
    }
    cout << br << endl;
    return 0;
}
```

Задатак: Конопци

Аутор: Оџьен Тешић

Мика је дао Пери N ($N \leq 2 \cdot 10^5$) канапа целобројних дужина (сви канапи су краћи од 10^{18}). Пера избацује **тачно један** канап из скупа канапа који му је дат. Након што Пера избаци тај канап, он жели да преостале канапе исече на комаде истих дужина (сви комади морају бити једнаких дужина, **без обзира од ког канапа су потекли**). Пера жели да избаци канап након ког би преосталих $N - 1$ канапа могао да исече на комаде **што веће дужине**. Колико износи та дужина (дужина сваког комада)? *Имајте на уму да Ваше решење мора да буде довољно ефикасно - да се извршава за мање од 1 секунде.*

Опис улаза

У првом реду стандардног улаза се налази природан број N - број канапа ($N \leq 2 \cdot 10^5$).

У другом реду стандардног улаза се налази тачно N природних бројева - дужине канапа (сви канапи су краћи од 10^{18}).

Додатна ограничења

Тест примери су подељени у три дисјунктне (независне) групе:

- У тест примерима вредним 40 поена: $N \leq 10^3$ и $1 \leq A_i \leq 10^4$ за свако $i = 1, 2, 3, \ldots, N$;
- У тест примерима вредним 20 поена: $N \leq 4 \cdot 10^4$ и $1 \leq A_i \leq 10^4$ за свако $i = 1, 2, 3, \ldots, N$;
- У тест примерима вредним 40 поена: $N \leq 2 \cdot 10^5$ и $1 \leq A_i \leq 10^{18}$ за свако $i = 1, 2, 3, \ldots, N$.

Опис излаза

На стандардни излаз треба исписати тачно један природан број - максималну дужину на коју можемо исећи канапе тако да комади који остану буду сви исте дужине.

Пример 1

Улаз	*Излаз*
3 8 39 12	4

Објашњење

Пера може да избаци канап дужине 39. Тада може да исече преостала два канапа на комаде дужине 4. Уколико би Пера избацио канап дужине 8, максимална дужина на коју би могао да исече преостала два канапа је 3. Уколико би Пера избацио канап дужине 12, максимална дужина на коју би могао да исече преостала два канапа је 1. Према томе, максимална дужина је 4.

Пример 2

Улаз

```
4
10 13 12 8
```

Излаз

```
2
```

Решење

Јасно, НЗД низа је највећи природан број који дели све елементе низа. Приметимо да се задатак заправо своди на *Одредити максимални НЗД низа бројева из кога избацујемо тачно један елемент*. У свим решењима ћемо за рачунање НЗД два броја користити познати Еуклидов алгоритам.

Решење када $N \leq 10^3$ и $1 \leq A_i \leq 10^4$ за свако $i = 1, 2, 3, \ldots, N$

У овом подзадатку је довољно да грубом силом израчунамо НЗД низа без једног елемента. Имплементација овог решења је једноставна - прво ћемо петљом одабрати који елемент избацујемо из низа, а потом ћемо угњежђеном петљом да одредимо НЗД низа. Ово решење је сложености $\mathcal{O}(N^2 \log M)$, где је M дужина најдужег канапа.

Решење када $N \leq 4 \cdot 10^4$ и $1 \leq A_i \leq 10^4$ за свако $i = 1, 2, 3, \ldots, N$

У овом подзадатку је довољно приметити следеће: да би неки број делио све осим евентуално једног члана низа, он мора да дели први или други елемент низа на улазу. Прво ћемо одредити све делиоце првог елемента низа, затим избројати за сваки од њих колико елемената низа дели. Исти поступак понављамо за други елемент низа. Решење је највећи од ових делилаца који дели бар $N - 1$ чланова низа. Сложеност овог решења је $\sqrt{A_0} + D_0 \cdot N + \sqrt{A_1} + D_1 \cdot N$, где је D_0 број делилаца од A_0, а D_1 број делилаца од A_1 (лако се види да је $D_0, D_1 \leqslant 64$ - ово се нпр. може израчунати на локалном рачунару проласком кроз све бројеве од 1 до 10^4 и рачунањем броја делилаца свих тих бројева).

Комплетно решење

Нека је `pref_nzd[i]` НЗД низа од 0. до i-тог елемента, а `suf_nzd[i]` НЗД низа од $(N-1)$-ог до i-тог елемента. Ове вредности лако рачунамо, по аналогији са префиксним и суфиксним сумама. Тада је одговор $\max(\text{NZD}(pref[i-1], suf[i+1]))$ кад i пролази од 1 до $N-2$. Посебно треба обрадити случајеве када избацујемо први или последњи елемент низа (пошто тада не постоји префикс пре њега, односно суфикс низа након њега). Како је у описаном решењу рачунање префиксног и суфиксног НЗДа доминантније у односу на један пролазак кроз низ користећи те вредности, то је укупна временска сложеност овог решења $\mathcal{O}(N \log M)$, где је M дужина најдужег канапа.

```
#include <iostream>

using namespace std;

#define MAXN 200010
```

```
long long duzine[MAXN];
long long pref_nzd[MAXN];
long long suf_nzd[MAXN];

long long nzd(long long a, long long b) {
  while (b != 0) {
    long long ost = a % b;
    a = b;
    b = ost;
  }
  return a;
}

int main() {
  int n;
  cin >> n;
  for (int i = 0; i < n; i++)
    cin >> duzine[i];
  // pref_nzd[i] je NZD niza na pozicijama pozicije iz intevala [0, i]
  // suf_nzd[i] je NZD niza na pozicijama iz intervala [i, n-1]
  pref_nzd[0] = duzine[0];
  suf_nzd[n-1] = duzine[n-1];
  for(int i = 1; i < n; i++)
    pref_nzd[i] = nzd(duzine[i], pref_nzd[i-1]);
  for(int i = n - 2; i >= 0; i--)
    suf_nzd[i] = nzd(duzine[i], suf_nzd[i+1]);
  // ako izbacujemo prvi ili poslednji element
  long long rezultat = max(suf_nzd[1], pref_nzd[n-2]);
  // ako izbacujemo neki element u sredini
  for(int i = 1; i < n - 1; i++)
    rezultat = max(rezultat, nzd(pref_nzd[i-1], suf_nzd[i+1]));
  cout << rezultat;
  return 0;
}
```

Глава 5

Државно такмичење

Задатак: Мотивација

Аутор: Владимир Кузмановић

Наставник је одлучио да мотивише своје ученике тако што ће мало снизити критеријум за закључивање оцена. Одлучио се за следећи критеријум:

1. Просек оцена током полугодишта је већи или једнак 4.4, закључна оцена је 5.
2. Просек оцена током полугодишта је мањи од 4.4 и већи или једнак 3.4, закључна оцена је 4.
3. Просек оцена током полугодишта је мањи од 3.4 и већи или једнак 2.4, закључна оцена је 3.
4. Просек оцена током полугодишта је мањи од 2.4 и већи или једнак 1.4, закључна оцена је 2.
5. Просек оцена током полугодишта је мањи од 1.4 закључна оцена је 1.

Током полугодишта ученик је оцењен 5 пута уобичајеним бројчаним оценама од 1 до 5. Написати програм који ће на основу ученикових оцена током полугодишта одредити његову закључну оцену.

Опис улаза

У сваком од 5 редова стандардног улаза налази се по једна оцена ученика која може имати вредности 1,2,3,4 или 5.

Опис излаза

На стандардни излаз исписати један природан број који представља закључну оцену ученика.

Пример 1

Улаз	*Излаз*
4 5 5 4 5	5

Објашњење

Просек оцена ученика је 4.6, па ће према утврђеном критеријуму закључна оцена бити 5.

Пример 2

Улаз

```
2
3
5
4
3
```

Излаз

```
4
```

Објашњење

Просек оцена ученика је 3.4, па ће према утврђеном критеријуму закључна оцена бити 4.

Решење

Решење задатка се своди на израчунавање просека унетих бројева и затим испитивање којем интервалу тај просек припада. Програмска реализација решења је у наставку.

```
#include <iostream>

using namespace std;

int main() {
   int o1, o2, o3, o4, o5;
   cin >> o1 >> o2 >> o3 >> o4 >> o5;

   double p = (o1 + o2 + o3 + o4 + o5) / 5.0;

   if (p >= 4.4) {
      cout << 5;
   } else if (p < 4.4 && p >= 3.4) {
      cout << 4;
   } else if (p < 3.4 && p >= 2.4) {
      cout << 3;
   } else if (p < 2.4 && p >= 1.4) {
      cout << 2;
   } else {
      cout << 1;
   }

   return 0;
}
```

Задатак: Харсхад бројеви

Аутор: Душан Пошић

Харсхад бројеви су они бројеви који су дељиви својим збиром цифара. Име долази из санскрита и на српски бисмо могли да преведемо као “бројеви који доносе радост” (harṣa (радост) + da (давати)). За задата два цела броја a и b одредити колико има харсхад бројева у интервалу $[a, b] = \{a, a + 1, \ldots, b - 1, b\}$.

Опис улаза

У једином реду стандардног улаза се налазе два цела броја a и b ($-10^5 \leq a \leq b \leq 10^5$).

Опис излаза

У једином реду стандардног излаза исписати тражени број харсхад бројева.

Пример 1

Улаз	*Излаз*
15 20	2

Објашњење

Харсхад бројеви у овом опсегу су 18 (18 је дељиво са 1+8=9) и 20 (20 је дељиво са 2+0=2).

Пример 2

Улаз

```
-5 5
```

Излаз

```
10
```

Објашњење

Сви једноцифрени бројеви осим 0 су харсхад бројеви. Од -5 до 5 има 5 негативних и 5 позитивних једноцифрених бројева, дакле одговор је 10.

Решење

У овом задатку је потребно да се прође кроз све бројеве из интервала и за сваки одреди збир цифара и онда провери да ли је сам број дељив тим збиром. Цифре броја се одређују уназад тако што се прво одреди последња цифра као остатак при дељењу са 10, а онда се та цифра "одсече" дељењем броја са 10. Овај поступак се наставља док број не дође до нуле.

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

using namespace std;

// funkcija racuna zbir cifara broja b
int zbirCifara(int b) {
  int zbir = 0;
  while (b > 0) {
    zbir += b % 10;  // na zbir cifara se dodaje poslednja cifra
    b /= 10;  // odsecamo poslednju cifru sa broja
  }
  return zbir;
}

// funkcija koja proverava da li je broj deljiv zbirom svojih cifara
bool harshad(int x) {
  if (x == 0) return false;
  x = abs(x); // na rezultat ne utice da li je broj pozitivan ili negativan
  // proveravamo da li je broj deljiv zbirom svojih cifara
  return x % zbirCifara(x) == 0;
}

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

  int broj = 0;
  for (int i = a; i <= b; i++)
    if (harshad(i))
      broj++;

  cout << broj << endl;

  return 0;
}
```

Задатак: Ђаволски квадрат

Аутор: Оћьен Тешић

Маџични квадраџ реда n је квадрат димензија $n \times n$ попуњен различитим бројевима код кога је збир бројева у свакој врсти (реду), свакој колони (ступцу) и свакој дијагонали једнак. Вредност свих тих збирова, S, назива се *карактеристични збир маџичног квадраџа*.

Ђаволски квадрат реда n (или *савршени*) је магичан квадрат димензија $n \times n$ попуњен различитим бројевима код кога је збир бројева и на свакој „изломљеној" дијагонали једнак карактеристичном збиру S. Под изломљеном дијагоналом подразумевамо дијагоналу квадрата која се може добити од полазног, ако би се квадрат поделио на 2 правоугаоника, па затим да ти правоугаоници замене места.

Један пример ђаволског квадрата је на слици испод.

1	14	4	15
8	11	5	10
13	2	16	3
12	7	9	6

Пример ђаволског квадрата

Дати квадрат је магични пошто је збир у свакој врсти, колони и дијагонали једнак 34. Збирови у изломљеним дијагоналама су $14 + 5 + 3 + 12 = 34$, $4 + 10 + 13 + 7 = 34$, $15 + 8 + 2 + 9 = 34$, $4 + 11 + 13 + 6 = 34$, $14 + 8 + 3 + 9 = 34$ и $1 + 10 + 16 + 7 = 34$, па је унети квадрат ђаволски.

Опис улаза

У првом реду стандардног улаза се уноси природан број n ($3 \leqslant n \leqslant 1000$).

У наредних n редова се уноси n природних бројева. Сви елементи квадрата су природни бројеви мањи или једнаки од 10^6.

Опис излаза

На излаз треба исписати једну од порука `nije ni magican ni djavolski`, `jeste magican ali nije djavolski` или `djavolski`.

Пример 1

Улаз
```
4
1 14 4 15
8 11 5 10
13 2 16 3
12 7 9 6
```
Излаз
```
djavolski
```

Пример 2

Улаз
```
6
35 1 6 26 19 24
3 32 7 21 23 25
31 9 2 22 27 20
8 28 33 17 10 15
30 5 34 12 14 16
4 36 29 13 18 11
```
Излаз
```
jeste magican ali nije djavolski
```

Пример 3

Улаз
```
3
1 1 1
1 1 1
1 1 2
```
Излаз
```
nije ni magican ni djavolski
```

Решење

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

// maksimalna dimenzija matrice je 1000
const int MAXN = 1000;
```

```
bool proveriMagican(int kvadrat[MAXN][MAXN], int n, long long karakteristicniZbir) {
  // proveravamo svaku vrstu i
  for (int i = 0; i < n; i++) {
    // zbir elemenata vrste i
    long long zbir = 0;
    for (int j = 0; j < n; j++)
      zbir += kvadrat[i][j];

    if (zbir != karakteristicniZbir)
      return false;
  }

  // proveravamo svaku kolonu j
  for (int j = 0; j < n; j++) {
    // zbir kolone j
    long long zbir = 0;
    for (int i = 0; i < n; i++)
      zbir += kvadrat[i][j];

    if (zbir != karakteristicniZbir)
      return false;
  }

  // proveravamo glavnu dijagonalu

  // zbir glavne dijagonale
  long long zbir = 0;
  for (int i = 0; i < n; i++)
    zbir += kvadrat[i][i];
  if (zbir != karakteristicniZbir)
    return false;

  // proveravamo sporednu dijagonalu

  // zbir sporedne dijagonale
  zbir = 0;
  for (int i = 0; i < n; i++)
    zbir += kvadrat[i][n-i-1];

  if (zbir != karakteristicniZbir)
    return false;

  return true;
}

bool proveriDjavolski(int kvadrat[MAXN][MAXN], int n, int karakteristicniZbir) {
  // provaravamo glavne izlomljene dijagonale
  for (int rbr = 1; rbr < n; rbr++) {
    // zbir izlomljene dijagonale
    long long zbir = 0;
    int kolona = rbr;
    for (int vrsta = 0; vrsta < n; vrsta++){
      zbir += kvadrat[vrsta][kolona % n];
      kolona++;
    }
    if (zbir != karakteristicniZbir)
      return false;
```

```
  }

  // proveravamo sporedne izlomljene dijagonale
  for (int rbr = 0; rbr < n - 1; rbr++) {
    // zbir izlomljene dijagonale
    long long zbir = 0;
    int vrsta = rbr + n;
    for (int kolona = 0; kolona < n; kolona++) {
      zbir += kvadrat[vrsta % n][kolona];
      --vrsta;
    }
    if (zbir != karakteristicniZbir)
      return false;
  }

  return true;
}

int main() {
  // ucitavamo matricu kvadrata
  int kvadrat[MAXN][MAXN];
  int n;
  cin >> n;
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      cin >> kvadrat[i][j];

  // karakteristican zbir je zbir prve kolone
  long long karakteristicniZbir = 0;
  for (int i = 0; i < n; i++)
    karakteristicniZbir += kvadrat[i][0];

  if (!proveriMagican(kvadrat, n, karakteristicniZbir))
    cout << "nije ni magican ni djavolski" << endl;
  else if (!proveriDjavolski(kvadrat, n, karakteristicniZbir))
    cout << "jeste magican ali nije djavolski" << endl;
  else
    cout << "djavolski" << endl;
  return 0;
}
```

Задатак: Патикарница

Аутор: Владимир Кузмановић

Популарна продавница патика, *Патикарница*, одлучила је да својим купцима понуди нову услугу. На улазу у *Патикарницу* биће постављен екран осетљив на додир који ће купцима предлагати најскупље моделе патика које се уклапају у буџет који су планирали да издвоје за нове патике. *Патикарница* је купила сву неопходну опрему да би корисницима пружила нову услугу, али је замолила нас да напишемо програм који ће на основу доступних патика у продавници и новца који је купац спреман да издвоји, приказати списак најскупљих модела патика које купац може да приушти.

Патикарница на стању има n различитих модела патика описаних својим именом и ценом. Током дана у *Патикарницу* улази k купаца. Сваки купац на улазу уноси количину новца x_i коју жели да издвоји за нове патике и систем му приказује списак најскупљих патика које може да приушти.

Опис улаза

У првом реду учитава се број n, $(1 \leq n \leq 10^5)$, различитих модела патика у понуди *Патикарнице*, затим се у n наредних редова учитава назив модела патика (ниска максималне дужине 20 карактера) и његова цена

(природан број мањи од 50000). Након тога, учитава се број k, $(1 \leq k \leq 10^6)$, купаца који су ушли у продавницу, а затим у k следећих редова буџети x_i $(1 \leq i \leq k)$ појединачних купаца (природни бројеви мањи од 50000).

Опис излаза

На стандардни излаз за сваког од k купаца и њихове буџете x_i исписати лексикографски сортиран списак најскупљих патика које могу да приуште. У случају да не постоје патике које купац може да приушти исписати ниску `nema`.

Пример

Улаз	*Излаз*

`10`
`NikeAirMaxIVO 16799`
`AsicsMetaspeedSky+ 29999`
`NikeVaporFly3 32899`
`AsicsSkzElite 17199`
`NikeAirMax90 20699`
`NikeJordanMaxAura5 17199`
`NikeStreakFly 13999`
`Adidas4DFWD3 25999`
`AdidasUltraBoostLight 16799`
`AdidasXPLRBoost 17199`
`2`
`18000`
`12000`
	`AdidasXPLRBoost 17199`
	`AsicsSkzElite 17199`
	`NikeJordanMaxAura5 17199`
	`nema`

Објашњење

Први купац је као свој буџет унео износ од 18000 динара и најскупље патике које може да приушти су `AdidasXPLRBoost`, `AsicsSkzElite` и `NikeJordanMaxAura5` по цени од `17199` динара.

Други купац је као свој буџет унео износ од 12000 динара и у понуди `Патикарнице` нема патика које купац може да приушти, па се као резултат упита штампа реч `nema`.

Решење

Линеарна претрага

На почетку самог задатка прво ћемо учитати број патика у понуди n, а затим и све моделе патика. Моделе патика ћемо чувати као низ уређених парова у којима је први елемент модел патика, а други елемент њихова цена.

Из текста задатка се лако уочава да нам треба велики број претраживања постојећег низа, одакле се лако закључује да би било згодно да низ прво сортирамо, тако да постигнемо да су сви модели патика исте цене један уз други и уз то да су модели патика сортирани лексикографски. Зато низ сортирамо по цени патика неопадајуће, при чему патике исте цене сортирамо лексикографски неопадајуће. На пример,

```
13999 NikeStreakFly
16799 AdidasUltraBoostLight
16799 NikeAirMaxIVO
17199 AsicsSkzElite
17199 AdidasXPLRBoost
17199 NikeJordanMaxAura5
20699 NikeAirMax90
25999 Adidas4DFWD3
29999 AsicsMetaspeedSky+
32899 NikeVaporFly3
```

Најлакши начин да се то постигне је да се направи низ уређених парова, где иде прво цена, па затим модел. Парови се подразумевано пореде тако што се прво пореди прва компонента (цена), а ако је прва компонента иста, онда друга (модел). То је тачно поредак који нам треба и не морамо да програмирамо никакаву посебну функцију поређења.

Након сортирања обрађујемо једног по једног купца. Када учитамо буџет купца, треба да одредимо најскупљи пар патика који он може да приушти. То можемо урадити линеарном претрагом, од краја низа. Умањујемо бројач све док су цене строго веће од буџета (водећи рачуна да не испаднемо ван граница низа).

Ако су све патике скупље од буџета, бројач ће доћи до -1 и тада исписујемо `nema`. У супротном треба да нађемо позицију првог пара патика које имају исту цену као ове најскупље које купујемо. То можемо да радимо још једном линеарном претрагом. Настављамо да смањујемо бројач све док не дођемо до првог јефтинијег пара (или не испаднемо из низа). Прве патике које купујемо се налазе на следећој позицији и тада само исписујемо податке о свим патикама које ћемо купити.

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

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0);

  // ucitavamo podatke o patikama
  int n;
  cin >> n;
  vector<pair<int, string>> patike(n);
  for (int i = 0; i < n; i++)
    cin >> patike[i].second >> patike[i].first;

  // sortiramo patike neopadajuce po ceni, a one iste cene
  // neopadajuce leksikografski po nazivu
  sort(patike.begin(), patike.end());

  // obradjujemo k kupaca
  int k;
  cin >> k;
  for (int i = 0; i < k; i++) {
    // budzet trenutnog kupca
    int budzet;
    cin >> budzet;

    // odredjujemo poziciju poslednjeg para patika koji moze da se kupi
    int poslednjiPriustiv = n-1;
    while (poslednjiPriustiv >= 0 && patike[poslednjiPriustiv].first > budzet)
      poslednjiPriustiv--;

    if (poslednjiPriustiv < 0) {
      // ni jedan par patika ne može da se priušti (sve cene su veće od budžeta)
      cout << "nema" << '\n';
    } else {
      // odredjujemo poziciju prvog para patika koji je iste cene kao ovaj
      // par koji kupujemo
      int cena = patike[poslednjiPriustiv].first;
      int prviTeCene = poslednjiPriustiv;
      while (prviTeCene >= 0 && patike[prviTeCene].first == cena)
        prviTeCene--;
      // stigli smo do pocetka niza ili prvog para koji je nize cene
      // prvi priustiv je na narednoj poziciji
      prviTeCene++;

      // ispisujemo sve parove koji su iste cene kao onaj koji kupujemo
      for (int j = prviTeCene; j <= poslednjiPriustiv; j++)
```

```
        cout << patike[j].second << " " << patike[j].first << '\n';
    }
  }
}
```

Бинарна претрага

Пошто је низ сортиран по ценама, претраге можемо да вршимо ефикасније уз помоћ бинарне претраге.

Можемо да дефинишемо функцију која бинарном претрагом одређује позицију последњег пара патика чија је цена мања или једнака од дате вредности. На тај начин можемо да одредимо позицију последњих патика које можемо да приуштимо. Ако је та позиција -1, онда су све патике скупље од буџета и исписујемо `nema`. Ако није, онда треба да нађемо позицију првих патика које имају ову цену `cena`. Грешка би била да то радимо линеарном претрагом, јер низ патика исте цене може такође бити дугачак. Зато новом бинарном претрагом можемо да пронађемо позицију последњих патика чија је цена строго мања од цене патика које купујемо (она је заправо последња позиција патика чија је цена мања или једнака од `cena-1`). Прве патике које ћемо купити се налазе сигурно на наредној позицији.

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

using namespace std;

// pozicija poslednjem elementa niza koji je manji ili jednak x
int poslednjiManjiIliJednak(vector<pair<int, string>>& patike, int x) {
  // koristimo algoritam binarne pretrage
  int l = 0;
  int d = patike.size() - 1;
  while (l <= d) {
    int m = l + (d-l) / 2;
    if (patike[m].first > x)
      d = m - 1;
    else
      l = m + 1;
  }
  return d;
}


int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0);

  // ucitavamo podatke o patikama
  int n;
  cin >> n;
  vector<pair<int, string>> patike(n);
  for (int i = 0; i < n; i++)
    cin >> patike[i].second >> patike[i].first;

  // sortiramo patike neopadajuce po ceni, a one iste cene
  // neopadajuce leksikografski po nazivu
  sort(patike.begin(), patike.end());

  // obradjujemo k kupaca
  int k;
  cin >> k;
  for (int i = 0; i < k; i++) {
```

```
    // budzet trenutnog kupca
    int budzet;
    cin >> budzet;

    // pronalazimo poziciju poslednjeg para patika koji kupac moze da priusti
    int poslednjiPriustiv = poslednjiManjiIliJednak(patike, budzet);
    if (poslednjiPriustiv == -1) {
      // ni jedan par patika ne može da se priušti (sve cene su veće od budžeta)
      cout << "nema" << '\n';
    } else {
      // cena patika koje kupujemo je jednaka ceni
      // poslednjeg priustivog para patika
      int cena = patike[poslednjiPriustiv].first;
      // trazimo prvi par patika te cene
      // odredicemo njegovog prethodnika (to je poslednji cija je cena strogo
      // manja od cene, tj. manja ili jednaka od cena-1)
      int prviTeCene = poslednjiManjiIliJednak(patike, cena-1) + 1;
      for (int j = prviTeCene; j <= poslednjiPriustiv; j++)
        cout << patike[j].second << " " << patike[j].first << '\n';
    }
  }

  return 0;
}
```

Задатак: Тешке торбе

Аутор: Милан Вуīделија

Два мала путника имају укупно четири тешке торбе. Сваки од путника може да понесе највише две торбе, по једну у свакој руци. Све торбе су заједничке и било који путник може да носи било коју торбу. У случају да су им торбе претешке, наши путници желе да користе што мање туђе помоћи.

Позната је тежина сваке торбе, као и највећи терет који може да понесе сваки од путника. Потребно је одредити најмању укупну тежину торби које они не могу да понесу.

Опис улаза

У првом реду стандардног улаза су четири природна броја раздвојена по једним размаком, тежине торби. У другом реду су два природна броја раздвојена једним размаком, тежине које могу да понесу мали путници. Сви бројеви су мањи од 1000.

Опис излаза

На стандардни излаз исписати један неозначен цео број, најмању укупну тежину торби за које је путницима потребна помоћ.

Пример 1

Улаз *Излаз*

```
1 2 3 4      0
5 6
```

Објашњење

Један путник може да понесе торбе тежина 1 и 4, а други торбе тежина 2 и 3, тако да им није потребна помоћ.

Пример 2

Улаз

```
2 3 4 5
5 6
```

Излаз

```
3
```

Објашњење

Један путник може да понесе торбу тежине 5, а други торбе тежина 2 и 4, па им је потребна помоћ за торбу тежине 3.

Пример 3

Улаз

```
12 20 15 17
17 14
```

Излаз

```
35
```

Решење

Означимо торбе са A, B, C, D. Путници могу да покушају да носе торбе на шест различитих начина:

- први путник торбе A и B, а други торбе C и D;
- први путник торбе A и C, а други торбе B и D;
- први путник торбе A и D, а други торбе B и C;
- први путник торбе B и C, а други торбе A и D;
- први путник торбе B и D, а други торбе A и C;
- први путник торбе C и D, а други торбе A и B.

Довољно је да у сваком од ових случајева израчунамо збир тежина торби које путници могу да понесу. Максимум тих збирова је највећа тежина коју путници заједно могу да понесу. Преостала тежина, за коју им је потребна помоћ, биће једнака допуни тог максималног збира до укупне тежине свих торби.

Да бисмо једноставније обрадили сваки од шест набројаних случајева, корисно је да напишемо функцију која за дате тежине две торбе и тежину коју један путник може да подигне израчунава тежину коју би тај путник заиста понео. Он ће редом покушати да понесе обе торбе, затим тежу, па лакшу торбу и понеће прво шта од тога буде могао да понесе. Ако не може да понесе ни лакшу торбу, путник неће понети ништа, па функција треба да врати нулу.

```
#include <iostream>

using namespace std;

int f(int t1, int t2, int nosivost)
{
    int laksa = min(t1, t2);
    int teza = max(t1, t2);
    if (t1 + t2 <= nosivost) return t1 + t2;
    else if (teza <= nosivost) return teza;
    else if (laksa <= nosivost) return laksa;
    return 0;
}

int main() {
    int a, b, c, d, p1, p2;
    cin >> a >> b >> c >> d >> p1 >> p2;

    int m = f(a, b, p1) + f(c, d, p2);
    m = max(m, f(a, c, p1) + f(b, d, p2));
    m = max(m, f(a, d, p1) + f(b, c, p2));
    m = max(m, f(b, c, p1) + f(a, d, p2));
    m = max(m, f(b, d, p1) + f(a, c, p2));
    m = max(m, f(c, d, p1) + f(a, b, p2));
    cout << a + b + c + d - m << endl;
```

```
    return 0;
}
```

Задатак: Квантна заврзлама

Аутор: Душан Пођадић

Ђоле је квантни физичар који је осмислио експеримент у оквиру својих докторских студија. Ђоле покушава да одређену честицу доведе у стабилан положај на собној температури. Да би проверио да ли је успео, он мери брзину те честице током времена. Да би се сматрало да је честица у стабилном стању, њена брзина мора да током неких k узастопних тренутака осцилује, тј. да се током тих k тренутака брзина наизменично повећава и смањује. Ђоле је сјајан физичар, али није тако добар програмер и он је замолио вас да на основу низа мерења одредите да ли је експеримент успео.

Опис улаза

У првом реду стандардног улаза се налазе два природна броја n и k ($2 \leq k \leq n \leq 10^6$). У другом реду се налази n позитивних реаланих бројева који су раздвојени размацима. Ови бројеви представљају брзине честице у сваком од n тренутака колико траје експеримент.

Опис излаза

У првом реду стандардног улаза исписати `da` или `ne` у зависности од тога да ли је експеримент успео. У другом реду исписати дужину најдужег подниза тренутака у коме је брзина честице осциловала (која може бити и већа од k).

Пример 1

Улаз	*Излаз*
`12 7` `2.7 3.7 6.9 7.8 4.3 17.11 2.6 5.8 5.9 4.1 7.9 6.2`	`ne` `6`

Објашњење

У низу брзина који се учитава не постоји подниз од 6 узастопних елемената у коме важи да брзина осцилује. Најдужи подниз има 6 елемената: 6.9, 7.8, 4.3, 17.11, 2.6, 5.8

Пример 2

Улаз

```
10 4
6.8 5.3 4.1 9.17 3.7 2.1 1.0 9.9 17.64 2.1
```

Излаз

```
da
4
```

Објашњење

Подниз узастопних тренутака 5.3, 4.1, 9.17, 3.7 испуњава тражени услов.

Пример 3

Улаз

```
7 4
2.2 7.5 3.5 3.5 5.7 9.9 10.0
```

Излаз

```
ne
3
```

Објашњење

У низу брзина који се учитава не постоји подниз од 4 узастопна елемента у коме важи да брзина осцилује. Обратите пажњу да подниз 2.2, 7.5, 3.5, 3.5 не испуњава услов јер су последња два елемента једнака.

Решење

На почетку треба приметити да је за одређивање испуњености услова о осциловању брзина увек потребно посматрати два суседна елемента.

Због тога имамо две промењљиве `prethodni` и `trenutni` у које памтимо тренутно посматрану вредност брзине и вредност која је била у претходном тренутку. Такође, треба да приметимо да за свака два различита узастопна броја у низу забележених брзина важи да чине подниз осцилаторних брзина дужине два. Из тог разлога ћемо сваки пут када почињемо нови подниз осцилаторних брзина (када се претходни прекине) тренутну његову дужину постављати на 2. Изузетак је када прекинемо низ јер су два елемента иста, тада постављамо тренутну дужину на 1. Користимо и промењљиву смер у којој памтимо да ли тренутни елемент низа треба да буде мањи или већи од претходног да би се подниз осцилаторних брзина наставио. Вредност промењљиве је 0 која означава да је смер неодређен. Ово се дешава када имамо два узастопна иста елемента, па нам је онда тренутни низ дужине 1 и следећи елемент може бити и мањи и већи од њега.

Пролазимо кроз све унете брзине и гледамо да ли однос тренутне и претходное одговара траженом смеру. Уколико не одговара прекидамо тренутни низ и започињемо нови, иначе повећавамо тренутни низ за 1 и мењамо очекивани смер. При томе сваки пут проверавамо да ли је тренутно највећи низ већи од максималног и ако јесте памтимо га. На крају је само потребно упоредити најдужи остварен низ са траженим k.

```
#include <iostream>

using namespace std;

int main() {
  // duzina niza i potreban broj oscilacija
  int n, k;
  cin >> n >> k;

  // duzina trenutnog niza oscilacija
  int t = 1;
  // duzina najduzieg niza oscilacija
  int maxT = 1;

  // odzavamo prva dva clana niza
  double prethodni, trenutni;
  cin >> trenutni;

  // promenljiva smer nam govori da li sledeci broj treba da bude
  // manji ili veci od onog ispred sebe da bi se niz nastavio:
  // -1 znaci da ocekujemo manji, 1 da ocekujemo veci,
  // a 0 znaci da je svejedno
  int smer = 0;

  // vrsimo dodatnih n-2 koraka
  for (int i = 1; i < n; i++) {
    // prelazimo na sledeci korak
    prethodni = trenutni;
    cin >> trenutni;

    // ako je trenutni veci od prethodnog, a ocekivali smo manji ili
    // ako je trenutni manji a ocekivali smo veci ili ako su jednaki
    // onda prekidamo niz oscilacija
    if ((trenutni > prethodni && smer == -1) ||
        (trenutni < prethodni && smer == 1) ||
        trenutni == prethodni) {
      // zapocinjemo novi niz
      if (trenutni > prethodni) {
        smer = -1;
        t = 2;
```

```
      } else if (trenutni < prethodni) {
         smer = 1;
         t = 2;
      } else {
         t = 1;
         smer = 0;
      }
    } else {
      // povecavamo niz za 1 i okrecemo smer
      t++;
      smer = trenutni > prethodni ? -1 : 1;
    }

    // azuriramo duzinu maksimalnog niza oscilacija
    if (t > maxT)
      maxT = t;
  }

  // provaravamo da li je bilo dovoljno oscilacija
  if (maxT >= k)
    cout << "da" << endl;
  else
    cout << "ne" << endl;
  // ispisujemo duzinu najduzeg niza
  cout << maxT << endl;
  return 0;
}
```

Задатак: Ротирање речи

Аутор: Филип Марић

Када се притисне тастер на тастатури, прво слово сваке речи у тексту се премести на последње место те речи. Колико је пута потребно притиснути тастер да би се реченица вратила у почетно стање?

Опис улаза

Са стандардног улаза се учитава једна линија текста која је састављена од малих слова енглеске абецеде и размака и има највише 10^6 карактера. Сви примери су такви да се реченица враћа у првобитно стање након највише 10^{18} притисака тастера.

Опис излаза

На стандардни излаз исписати најмањи број притисака тастера (већи од 0) потребних да се реченица врати у првобитно стање.

Пример 1

Улаз	*Излаз*
`mala sirena zivi na dnu mora`	`12`

Објашњење

Притисцима тастера добијају се наредне реченице

```
mala sirena zivi na dnu mora
alam irenas iviz an nud oram
lama renasi vizi na udn ramo
amal enasir iziv an dnu amor
mala nasire zivi na nud mora
alam asiren iviz an udn oram
lama sirena vizi na dnu ramo
amal irenas iziv an nud amor
```

```
mala renasi zivi na udn mora
alam enasir iviz an dnu oram
lama nasire vizi na nud ramo
amal asiren iziv an udn amor
mala sirena zivi na dnu mora
```

Пример 2

Улаз

```
sestaci sedmaci i osmaci vole informatiku
```

Излаз

```
924
```

Пример 3

Улаз

```
dada tata i mama
```

Излаз

```
2
```

Решење

Аутор: Филип Марић

Основно решење

Ако реч s дужине $n = |s|$ није периодична, потребно је n притисака тастера да се она врати у почетно стање. Она се враћа у то стање и након $2n$, $3n$, $4n$ итд. притисака тастера. На пример, реч `abca` се враћа у почетни положај након 4 притиска тастера. Она се враћа у тај положај и након 8, 12, 16 итд. притисака тастера. Дакле, да би се непериодична реч вратила у почетни положај број притисака тастера је њен садржалац.

Пошто захтевамо да се свака реч врати у почетни положај, тражимо најмањи број који је садржалац дужина свих речи. Ако имамо k речи $(s_1, \ldots, s_k)$, најмањи број притисака тастера је $NZS(n_1, \ldots, n_k)$, где је $n_i = |s_i|$ дужина речи s_i. На пример, дужине речи у реченици `mala sirena zivi na dnu mora` су 4, 6, 4, 2, 3, 4 и њихов НЗС је 12. Дакле, да би програм исправно радио за све примере у којима су речи непериодичне, довољно је одредити НЗС дужина свих учитаних речи (што се може лако урадити Еуклидовим алгоритмом).

Ако је реч s дужине n периодична (на пример, `abcabcabc`), тада је број притисака тастера потребних да се реч врати у почетни положај једнак дужини најкраћег периода речи (период речи је реч t таква да је $s = t^k = \underbrace{tt\ldots t}_{k}$). Јасно је да дужина периода $p = |t|$ мора да дели дужину речи n. У основном решењу можемо пробати све могуће дужине периода (од 1 па до $n/2$) и испитати да ли постоји период те дужине (ако реч није периодична, њен најкраћи период је дужине n). Да би реч t дужине p била период речи s довољно је да је сваки карактер на позицији i која се креће од позиције p до позиције $n - 1$ једнак карактеру на позицији $i \bmod p$, међутим, још брже се испитује да ли је једнак карактеру на позицији $i - p$, а за њега је раније установљено да је једнак карактеру на позицији $(i - p) \bmod p = i \bmod p$.

Дакле, претходним алгоритмом је потребно за сваку реч одредити дужину најкраћег периода и наћи НЗС свих тако добијених бројева.

Ово решење на овом нивоу такмичења доноси максимални број поена.

```cpp
#include <iostream>

using namespace std;

// provera da li je prefiks duzine p reci s period reci s
bool proveriPeriod(const string& s, int p) {
  // duzina reci mora biti deljiva duzinom perioda
  if (s.length() % p != 0)
    return false;
```

```
  // ako je period duzine p svaki karakter mora biti jednak
  // karakteru koji se nalazi p pozicija levo od njega
  for (int i = p; i < s.length(); i++)
    if (s[i] != s[i-p])
      return false;
  return true;
}

// pronalazi duzinu najkraceg perioda reci s
int period(const string& s) {
  // proveravamo sve potencijalne periode
  for (int p = 1; p <= s.length() / 2; p++)
    if (proveriPeriod(s, p))
      return p;
  // rec nije periodicna
  return s.length();
}

// najveci zajednicki delilac
long long nzd(long long a, long long b) {
  // Euklidov algoritam
  if (b == 0) return a;
  return nzd(b, a % b);
}

// najmanji zajednicki sadrzalac
long long nzs(long long a, long long b) {
  return (a / nzd(a, b)) * b;
}

int main() {
  ios_base::sync_with_stdio(false);
  // ucitavamo reci i trazimo nzs svih njihovih perioda
  string rec;
  long long n = 1;
  while (cin >> rec)
    n = nzs(n, period(rec));
  cout << n << endl;
  return 0;
}
```

Унапређено решење

Решење се може унапредити тако што ће оптимизовати испитивање периодичности речи s. Пошто период дели дужину речи, пожељно је одредити делиоце броја $n = |s|$ и испитивати само оне периоде који су у скупу тих делилаца.

Ако постоји период дужине d и важи да је kd такође делилац броја $n = |s|$, тада је префикс дужине kd такође период речи. На пример, у речи `abcabcabcabc` дужине 12, најкраћи је период `abc` дужине 3, међутим, пошто $2 \cdot 3 = 6$ такође дели дужину 12, и реч `abcabc` је такође период. Зато, ако установимо да префикс дужине 6 није период, онда можемо одмах установити ни да префикс дужине 3 не може бити период (а ни дужине 2). Са друге стране, ако установимо да реч дужине kd јесте период, онда у циљу провере мањих периода не морамо проверавати целу ниску, већ само њен префикс дужине kd. На пример, када установимо да је `abcabc` период, да бисмо проверили да ли је `abc` период не морамо гледати целу ниску од 12 карактера, већ је довољно само да проверимо да ли је `abc` период ниске `abcabc` дужине 6 (другу половину не морамо да проверавамо јер је иста првој, што је установљено када је утврђено да је `abcabc` период оригиналне ниске).

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

```cpp
#include <algorithm>
#include <cmath>
#include <map>

using namespace std;

// prolanazi sve delioce broja n, uređene od najmanjeg do najveceg
vector<int> pronadjiDelioce(int n) {
  vector<int> delioci;
  long long d;
  for (d = 1; d*d < n; d++)
    if (n % d == 0) {
      delioci.push_back(d);
      delioci.push_back(n/d);
    }
  if (d*d == n)
    delioci.push_back(d);
  sort(begin(delioci), end(delioci));
  return delioci;
}

// pod pretpostavkom da je p delilac dužine reči n, proveravamo da li je
// je prefiks reči s dužine p period reči s
bool proveriPeriod(const string& s, int n, int p) {
  for (int i = p; i < n; i++)
    if (s[i] != s[i-p])
      return false;
  return true;
}

int period(const string& s) {
  // potencijalni periodi moraju biti delioci dužine reči
  auto delioci = pronadjiDelioce(s.length());

  // da li je broj period reči s
  map<int, bool> jePeriod;
  // na početku samo znamo da je dužina reči s dužina jednog njenog perioda
  for (int d : delioci)
    jePeriod[d] = false;
  jePeriod[s.length()] = true;

  // najkraći pronađen period
  int rezultat = s.length();
  // obilazimo delioce unazad (dužinu reči preskačemo)
  for (int i = delioci.size() - 2; i >= 0; i--) {
    int p = delioci[i];
    // tražimo da li je neki delilac pp umnožak delioca p
    // dužina reči s svakako jeste
    int pp;
    for (int j = i+1; j < delioci.size(); j++)
      if (delioci[j] % p == 0)
        pp = delioci[j];
    // ako pp nije dužina perioda, onda ni p nije dužina perioda
    if (!jePeriod[pp])
      jePeriod[p] = false;
    else {
      // ako pp jeste dužina perioda, onda proveravamo da li je p dužina perioda
      // međutim, gledamo samo prefiks niske s dužine pp
```

```
      jePeriod[p] = proveriPeriod(s, pp, p);
      // ako je pronađen period, on je najkraći, pa ažuriramo rezultat
      if (jePeriod[p])
        rezultat = p;
    }
  }
  return rezultat;
}

// najveci zajednicki delilac
long long nzd(long long a, long long b) {
  // Euklidov algoritam
  if (b == 0) return a;
  return nzd(b, a % b);
}

// najmanji zajednicki sadrzalac
long long nzs(long long a, long long b) {
  return (a / nzd(a, b)) * b;
}

int main() {
  ios_base::sync_with_stdio(false);
  // ucitavamo reci i trazimo nzs svih njihovih perioda
  string rec;
  long long n = 1;
  while (cin >> rec)
    n = nzs(n, period(rec));
  cout << n << endl;
  return 0;
}
```

Најефикасније решење је засновано на примени алгоритама текста (на пример, алгоритма КМП и оно превазилази такмичарски програм за ОШ, па га стога ни не приказујемо).

Задатак: Бициклисти

Аутор: Милан Вујделија

У бициклистичкој трци учествује неколико четворочланих екипа. Са старта сви крећу истовремено. Као време екипе рачуна се време трећег члана по редоследу пролажења кроз циљ. Дата су времена N такмичара, при чему знамо да је N дељиво са 4. Прва 4 времена су времена такмичара домаће екипе, остала времена су измешана. Који је најбољи и најлошији пласман домаће екипе на основу датих времена?

Опис улаза

У првом реду стандардног улаза је цео број N, укупан број такмичара ($1 \leq N \leq 50000$, N је дељиво са 4).

У другом реду је N целих позитивних бројева, времена такмичара. Сви ови бројеви су међусобно различити и мањи од милион.

У трећем реду је реч `max` или реч `min`.

Опис излаза

На стандардни излаз исписати један број. Ако је у последњем реду улаза била реч `max`, тај број треба да буде најбољи могућ пласман домаће екипе, а у противном (ако је последњем реду улаза била реч `min`) број треба да буде најслабији могућ пласман.

Пример 1

Улаз	*Излаз*

`16`
`9 12 10 11 13 1 14 3 5 15 2 7 16 8 6 4`
`max`
	`2`

Објашњење

Укупно има 16 такмичара, односно 4 екипе (укључујући и домаћу). Времена такмичара домаће екипе су 9, 10, 11, 12, тако да је време целе екипе 11.

Домаћа екипа може да буде друга, нпр. ако су времена такмичара који припадају најбољој екипи 1, 2, 3, 4 (време те екипе је 3), времена такмичара из треће екипе 5, 6, 13, 14 (време екипе је 13), а времена четврте екипе 7, 8, 15, 16 (време екипе је 15). Испоставља се да не постоји распоред гостујућих такмичара по екипама, при коме би домаћа екипа била прва. Према томе, најбољи могућ пласман домаће екипе је друго место, па треба исписати `2`.

Пример 2

Улаз

```
16
9 12 10 11 13 1 14 3 5 15 2 7 16 8 6 4
min
```

Излаз

```
3
```

Објашњење

Број такмичара и њихова времена су иста као у првом примеру, тако да је време домаће екипе поново 11.

Домаћа екипа може да буде трећа, нпр. ако су времена такмичара који припадају најбољој екипи 1, 2, 3, 13, времена такмичара из друге екипе 4, 5, 6, 14, а времена четврте екипе 7 8 15 16. Гостујући такмичари (њих 12) не могу да се распореде у три гостујуће екипе, тако да све те три екипе буду боље од домаће. Према томе, најслабији могућ пласман домаће екипе је треће место, па треба исписати `3`.

Пример 3

Улаз

```
20
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
min
```

Излаз

```
1
```

Објашњење

Укупно има пет екипа. Време домаће екипе је 3. Како год били распоређени такмичари остале четири екипе, домаћа екипа је свакако најбоље пласирана (њен најслабији могућ пласман је прво место), па треба исписати `1`.

Решење

Најпре треба одредити време домаће екипе, а то је треће по величини од прва четири времена. Нека је од преостала $n - 4$ времена њих m мањих од времена домаће екипе, што значи да их је још $v = n - 4 - m$ већих од времена домаће екипе. Означимо број преосталих екипа (не рачунајући домаћу) са $e = \frac{n}{4} - 1$.

Гостујућа екипа је боља од домаће ако су у њој бар три такмичара са временом бољим од времена домаће екипе. Таквих екипа не може бити више од $\frac{m}{3}$, а ни више од e. Зато је највећи могућ број екипа бољих од домаће $B = \min(\frac{m}{3}, e)$, а најслабији могућ пласман домаће екипе је $B + 1$ (боље екипе заузимају највише првих B места, а домаћа екипа следеће, $B + 1$-во место).

Слично претходном, екипа је слабија од домаће ако у њој има бар два такмичара са временом слабијим од времена домаће екипе. Таквих екипа не може бити више од $\frac{v}{2}$, а ни више од e. Зато је највећи могућ број екипа слабијих од домаће $S = \min(\frac{v}{2}, e)$, а најбољи могућ пласман домаће екипе је $e - S$ (слабије екипе заузимају највише последњих S места, а домаћа екипа место које им претходи, а то је место са редним бројем $e - S$).

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

using namespace std;

int main() {
    int n;
    cin >> n;
    int brEkipa = n / 4;

    vector<int> domaci(4);
    for (int i = 0; i < 4; i++)
        cin >> domaci[i];

    sort(domaci.begin(), domaci.end());
    int tDomace = domaci[2];

    int brBoljihTakmicara = 0;
    for (int i = 4; i < n; i++)
    {
        int x;
        cin >> x;
        if (x < tDomace)
            brBoljihTakmicara++;
    }
    string staSeTrazi;
    cin >> staSeTrazi;

    int maxBrBoljihEkipa = min(brEkipa - 1, brBoljihTakmicara / 3);
    int najgoriPlasman = maxBrBoljihEkipa + 1;

    int brLosijihTakmicara = n - 4 - brBoljihTakmicara;
    int maxBrLosijihEkipa = min(brEkipa - 1, brLosijihTakmicara / 2);
    int najboljiPlasman = brEkipa - maxBrLosijihEkipa;

    if (staSeTrazi == "max")
        cout << najboljiPlasman << endl;
    else
        cout << najgoriPlasman << endl;

    return 0;
}
```

Задатак: Горе-доле-лево-десно

Аутор: Милан Вуīелија

Полазећи од неке тачке, цртају се 4 дужи, од којих је свака водоравна или усправна. Одредити да ли се крај четврте дужи поклапа са почетком прве. У случају потврдног одговора одредити и површину површи ограничене добијеном изломљеном линијом.

Опис улаза

У сваком од 4 реда стандардног улаза налази се једна од речи `gore`, `dole`, `levo`, или `desno`, иза које је један природан број, одвојен од речи једним размаком. Број није већи од 1000.

Опис излаза

У случају да се изломљена линија не завршава тамо где је почела, на стандардни излаз исписати реч `ne`.

У случају да се изломљена линија завршава тамо где је почела, на стандардни излаз исписати реч `da` и један неозначен цео број одвојен од речи `da` једнм размаком. Исписани број треба да буде једнак траженој површини.

Пример 1

Улаз	*Излаз*

`gore 3`
`levo 5`
`dole 3`
`desno 5`
	`da 15`

Пример 2

Улаз	*Излаз*

`gore 3`
`dole 3`
`levo 5`
`desno 5`
	`da 0`

Пример 3

Улаз	*Излаз*

`gore 3`
`levo 5`
`dole 3`
`desno 4`
	`ne`

Решење

За сваку дуж можемо да израчунамо вектор помераја, тј. водораван и усправан померај приликом прелажења датог растојања у датом смеру. Сабирањем ових помераја добијамо укупан водораван и усправан померај. Ако неки од ова два укупна помераја након обраде све четири дужи не буде нула, треба исписати `ne`. Размотримо даље случај када су оба ова помераја нуле.

Да би кретањем по овим дужима била оивичена нека површ, потребно је и довољно да се водоравни и усправни помераји појављују наизменично. Да бисмо ово проверили, можемо да памтимо правац претходног помераја и поредимо га са правцем текућег помераја. Уколико се неки правац (не смер) појави два пута узастопно, то ћемо запамтити помоћу логичке променљиве која говори да ли постоји површина. Успут, при водоравном померају ажурирамо ширину, а при усправном висину површи, која у овом задатку мора да буде правоугаона, ако постоји.

Након обраде све четири дужи, разликујемо следеће случајеве:

- Ако су водоравни и усправни померај нису оба једнака нули, исписујемо `ne`.
- Ако су водоравни и усправни померај једнаки нули и узастопни смерови ортогонални, исписујемо `da` и производ висине и ширине.
- Ако су водоравни и усправни померај једнаки нули, а узастопни смерови нису сви ортогонални, исписујемо `da` и нулу.

```
#include <iostream>

using namespace std;

int main() {
    string prethodniPravac = "";
    bool imaPovrsinu = true;
    int sirina = 0, vodoravniPomeraj = 0;
    int visina = 0, uspravniPomeraj = 0;

    for (int i = 0; i < 4; i++) {
        string smer;
        int rastojanje;
        cin >> smer >> rastojanje;
        if (smer == "desno" || smer == "levo") {
            if (smer == "desno")
                vodoravniPomeraj += rastojanje;
            else
                vodoravniPomeraj -= rastojanje;
            sirina = rastojanje;

            if (prethodniPravac == "vodoravno")
                imaPovrsinu = false;
            prethodniPravac = "vodoravno";
        }
        else { // smer == "gore" || smer == "dole"
            if (smer == "gore")
                uspravniPomeraj += rastojanje;
```

```
            else
                uspravniPomeraj -= rastojanje;
            visina = rastojanje;

            if (prethodniPravac == "uspravno")
                imaPovrsinu = false;
            prethodniPravac = "uspravno";
        }
    }
    if (vodoravniPomeraj != 0 || uspravniPomeraj != 0)
        cout << "ne" << endl;
    else if (imaPovrsinu)
        cout << "da" << " " << sirina * visina << endl;
    else
        cout << "da" << " " << 0 << endl;

    return 0;
}
```

Друго решење је варијација претходног. Коришћењем речника (мапе) `predznak` можемо да смањимо број гранања при обради дужи и ажурирању водоравног и усправног помераја, односно висине и ширине правоугаоне површи, ако се испостави да она постоји.

Након обраде све четири дужи проверавамо да ли је крај четврте дужи једнак почетку прве, тј. да ли су и укупан водоравни померај и укупан усправни померај једнаки нули. Ако нису, треба исписати `ne`. Ако су оба укупна помеаја нула, онда треба још проверити да ли је кретањем оивичена нека површ. То ће бити случај ако и само ако су испуњена следећа два услова:

- Током кретања забележен је и водораван и усправан помак.
- Смер првог помака је супротан смеру трећег, а смер другог помака супротан је смеру четвртог.

Први од ова два услова не мора да се проверава, јер у случају да није било обе врсте помака (водоравног и усправног), једна од променљивих `sirina` и `visina` задржава почетну вредност `0`, па ће и производ висине и ширине бити `0`, што и треба исписати у том случају.

Према томе, када су оба укупна помераја нуле, подслучајеве можемо да разликујемо на основу другог поменутог услова. Конкретно, ако је први смер супротан трећем а други четвртом, треба исписати `da` и производ висине и ширине, а у противном исписујемо `da` и нулу.

Провера да ли су нека два смера међусобно супротна најлакше се обавља помоћу још једног речника, а то је речник супротних смерова, дефинисан на почетку програма.

```
#include <iostream>
#include <map>

using namespace std;

map<string, int> predznak = { {"levo", -1}, {"desno", 1}, {"dole", -1}, {"gore", 1} };
map<string, string> suprotanSmer = { {"levo", "desno"}, {"desno", "levo"}, {"dole", "gore"}, {"gore", "dole"}};

int main() {
    int sirina = 0, visina = 0, rastojanje = 0;
    int vodoravniPomeraj = 0, uspravniPomeraj = 0;
    string smer[4] = { "", "", "", "" };
    for (int i = 0; i < 4; i++)
    {
        cin >> smer[i] >> rastojanje;
        if (smer[i] == "levo" || smer[i] == "desno")
        {
            sirina = rastojanje;
            vodoravniPomeraj += predznak[smer[i]] * rastojanje;
        }
```

```
        else // smer[i] == "gore" || smer[i] == "dole"
        {
            visina = rastojanje;
            uspravniPomeraj += predznak[smer[i]] * rastojanje;
        }
    }
    if (vodoravniPomeraj != 0 || uspravniPomeraj != 0)
        cout << "ne" << endl;
    else if (suprotanSmer[smer[0]] == smer[2] && suprotanSmer[smer[1]] == smer[3])
        cout << "da" << " " << sirina * visina << endl;
    else
        cout << "da" << " " << 0 << endl;

    return 0;
}
```

У трећем решењу, на основу смерова и растојања израчунавамо координате тачака изломљене линије. То можемо да урадимо помоћу неколико наредби гранања, али у овом решењу смо као елегантније решење употребили речнике (мапе) који пресликавају назив смера у јединични вектор тог смера. На пример, `kx["dole"]` је `0`, а `ky["dole"]` је `-1`, па је јединични вектор за смер `dole` једнак $(0, -1)$.

Као и у претходним решењима, након обраде свих дужи проверавамо да ли је крај четврте дужи једнак почетку прве. У овом решењу то се своди на проверу да ли је последња тачка једнака полазној. Ако није, треба исписати `ne`. У случају да су те тачке једнаке, кретањем је оивичена нека површ ако и само ако се прва тачка по обе координате разликује од треће, а друга тачка се по обе координате разликује од четврте.

Када је овај услов испуњен, површину можемо да израчунамо и као `p1 = abs((x[0]-x[2])*(y[0]-y[2]))` и као `p2 = abs((x[1]-x[3])*(y[1]-y[3]))`. У супротном ће бар један од ова два израза имати вредност 0. Према томе, уз одговор `da` можемо просто да испишемо вредност мањег од ова израза.

```
#include <iostream>
#include <map>

using namespace std;

int x[5], y[5];
map<string, int> kx = { {"levo", -1}, {"desno", 1}, {"dole", 0}, {"gore", 0} };
map<string, int> ky = { {"levo", 0}, {"desno", 0}, {"dole", -1}, {"gore", 1} };
int main() {
    for (int i = 0; i < 4; i++) {
        string smer;
        int rastojanje;
        cin >> smer >> rastojanje;
        x[i + 1] = x[i] + kx[smer] * rastojanje;
        y[i + 1] = y[i] + ky[smer] * rastojanje;
    }
    if (x[4] != 0 || y[4] != 0)
        cout << "ne" << endl;
    else
    {
        int p1 = abs((x[0]-x[2])*(y[0]-y[2]));
        int p2 = abs((x[1]-x[3])*(y[1]-y[3]));
        cout << "da" << " " << min(p1, p2) << endl;
    }
    return 0;
}
```

Задатак: Збир простих

Аутор: Оћьен Тешић

Дато је Q упита облика a, b ($a \leqslant b$). Одговор на сваки упит је збир свих простих бројева између a-тог и b-тог простог броја (укључујући a-ти и b-ти). За природан број кажемо да је *прост* ако има тачно два различита делиоца у скупу природних бројева. Бројеви 0 и 1 нису прости.

Опис улаза

У првом реду стандардног улаза се уноси природан број Q ($Q \leqslant 10^5$).

У наредних Q редова се уносе по два природна броја - оба су мања од $5 \cdot 10^5$.

Опис излаза

У Q редова исписати тачно један број - збир свих простих бројева између a-тог и b-тог простог броја (укључујући a-ти и b-ти).

Пример

Улаз	*Излаз*
4	26
2 5	7
4 4	53
3 7	41
5 7	

Објашњење

Објашњење. Прости бројеви су 2, 3, 5, 7, 11, 13, 17, 19, 23, ...

Збир простих бројева од другог до петог простог броја је $3 + 5 + 7 + 11 = 26$.

Четврти прост број је 7.

Збир простих бројева од трећег до седмог простог броја је $5 + 7 + 11 + 13 + 17 = 53$.

Збир простих бројева од петог до седмог простог броја је $11 + 13 + 17 = 41$.

Решење

Решење са динимаички одређеним границама

Основни корак у решењу јесте да се одреди низ простих бројева. Претпоставимо да желимо да одредимо првих k простих бројева. Они се могу ефикасно одредити Ератостеновим ситом. Ератостеновим ситом можемо за све бројеве из интервала $[0, n]$ одредити који број јесте, а који није прост. Након тога можемо пролазити кроз тај низ и сваки прост број убацивати у низ простих (све док у низ простих не пребацимо k простих бројева).

Поставља се природно питање колики треба да буде број n да бисмо гарантовали да ће k-ти прост број бити већи или једнак n. Из теорије бројева је позната граница $p_k < k(\log k + \log(\log k)$, која важи за $k \geq 6$. За $k < 6$ можемо n поставити на вредност 11 (јер је $p_5 = 11$).

Број k зависи од упита тј. од интервала $[a_i, b_i]$. За вредност k можемо узети највећу вредност бројева b_i.

```
#include <bits/stdc++.h>

using namespace std;

// za sve brojeve iz intervala [0, n] odredjujemo da li su prosti
vector<bool> Eratosten(int n) {
  vector<bool> prost(n+1, true);
  prost[0] = prost[1] = false;
  for (int p = 2; p * p <= n; p++)
    if (prost[p])
      for (int i = p * p; i <= n; i += p)
        prost[i] = false;
  return prost;
}
```

```
// vektor koji sadrzi prvih k prostih brojeva
vector<int> k_prostih(int k) {
  // rezultujuci vektor prostih brojeva
  vector<int> prosti;
  // na kraju ce vektor sadrzati k elemenata, pa odvajamo potrebnu
  // kolicinu memorije
  prosti.reserve(k);
  // k-ti prost broj je sigurno manji ili jednak n
  int n = max(11, (int)(k * (log(k) + log(log(k)))));
  // Eratostenovim sitom za sve brojeve [0, n] odredjujemo da li su prosti
  vector<bool> prost = Eratosten(n);
  // proste brojeve ubacujemo u niz prosti, sve dok ne ubacimo k brojeva
  for (int i = 2; prosti.size() < k; i++)
    if (prost[i])
      prosti.push_back(i);
  return prosti;
}

int main() {
  // ucitavamo sve upite [a, b] i odredjujemo najvecu gornju granicu b
  int q;
  cin >> q;
  vector<pair<int, int>> ab(q);
  int maxb = 0;
  for (int i = 0; i < q; i++) {
    int a, b;
    cin >> a >> b;
    ab[i] = {a, b};
    maxb = max(maxb, b);
  }

  // odredjujemo k brostih brojeva za k = maxb
  vector<int> prosti = k_prostih(maxb);
  // izracunavamo prefiksne sume ovog niza
  vector<long long> prefix(maxb+1);
  prefix[0] = 0;
  for(int i = 1; i <= maxb; i++)
    prefix[i] = prefix[i-1] + prosti[i-1];

  // odgovaramo na upite
  for (const auto [a, b] : ab)
    cout << prefix[b] - prefix[a-1] << endl;

  return 0;
}
```

Решење са статичким ограничењима

Мало једноставнији (али спорији) програм можемо добити ако број простих бројева које одређујемо не одредимо на основу онога што нам је заиста потребно у задатку (унетих интервала $[a_i, b_i]$), већ на основу граница датих у тексту задатка. Максимална вредност броја b_i је $500\,000$, што можемо узети за вредност k (тј. број простих бројева које одређујемо). $500\,000$-ти прост број је сигурно мањи од вредности $k(\log k + \log(\log k)$, што је мало мање од 8 милиона, па ту вредност можемо узети за вредност n (границу Ератостеновог сита).

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 8000000;
const int MAXB = 500001;
```

```
bool prost[MAXN];
int ntiprost[MAXB];
long long prefix[MAXB];

// za svaki broj iz intervala [0, MAXN] odredjuje da li je prost
void Eratosten() {
  memset(prost, true, sizeof(prost));
  prost[0] = prost[1] = false;
  for (int p = 2; p * p <= MAXN; p++) {
    if (prost[p] == true) {
      for (int i = p * p; i <= MAXN; i += p)
        prost[i] = false;
    }
  }
}

// popunjava niz ntiprost tako da se na poziciji k nalazi k-ti prost broj
// (ntiprost[1] = 2, ntiprost[2] = 3, ntiprost[3] = 5, ...)
// niz se popunjava seve do pozocije MAXB-1
void Prosti() {
  Eratosten();
  int brojac = 1;
  for(int i = 2; brojac < MAXB; i++)
    if (prost[i])
      ntiprost[brojac++] = i;
}

int main() {
  Prosti();

  // na poziciji k se nalazi zbir prvih k prostih brojeva
  prefix[0] = 0;
  for(int i = 1; i < MAXB; i++)
    prefix[i] = prefix[i - 1] + ntiprost[i];

  // ucitavamo upite i odgovaramo na njih
  int q;
  cin >> q;
  for (int i = 0; i < q; i++) {
    int a, b;
    cin >> a >> b;
    cout << prefix[b] - prefix[a - 1] << endl;
  }
  return 0;
}
```

Задатак: Домине

Аутор: Иван Дрецун

Дато је n домина поређаних у низ и сваке суседне две домине су на међусобном растојању 1. Домине могу имати различите висине. Могуће је уклонити неке домине из низа (осим прве и последње) и затим оборити прву домину. Прва домина ће оборити *само* прву следећу домину, и то уколико јој је она на удаљености мањој или једнакој од висине прве. Тај поступак се наставља докле год је могуће оборити наредну домину. Написати програм који одређује на колико начина је могуће уклонити неке домине тако да се обарањем прве домине обара и последња. Број уклоњених домина може бити и 0.

Опис улаза

Са стандардног улаза се уноси број n ($1 \leq n \leq 10^6$). Затим се уноси n елемената a_i ($1 \leq a_i \leq 10^6$) који представљају висине домина.

Опис излаза

На стандардни излаз исписати број тражених поднизова. Како тај број може бити јако велик, све операције вршити по модулу $10^9 + 7$.

Пример

Улаз	*Излаз*
`5` `1 3 1 2 1`	`3`

Објашњење

Поднизови `1 3 . . 1`, `1 3 . 2 1` и `1 3 1 2 1` су такви да се обарањем прве домине обара и последња, док нпр. подниз `1 3 1 . 1` није такав. То је зато што се обарањем прве домине обара друга, обарањем друге трећа, али обарањем треће домине (чија је висина 1) није могуће оборити последњу домину (која јој је на удаљености 2).

Решење

Динамичко програмирање

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

const int M = 1000000007;

using namespace std;

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

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

  vector<int> dp(n);
  dp[n - 1] = 1;
  for(int i = n - 2; i >= 0; i--)
    for(int j = i + 1; j <= min(n - 1, i + h[i]); j++)
      dp[i] = (dp[i] + dp[j]) % M;

  cout << dp[0] << endl;

  return 0;
}
```

Оптимизација префиксним сумама

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

const int M = 1000000007;

using namespace std;

int main() {
  int n;
```

```
  cin >> n;

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

  vector<int> dp(n), sum(n + 1);
  dp[n - 1] = sum[n - 1] = 1;
  for(int i = n - 2; i >= 0; i--) {
    int l = i + 1, d = min(n, i + h[i] + 1);
    dp[i] = (M + sum[l] - sum[d]) % M;
    sum[i] = (sum[i + 1] + dp[i]) % M;
  }

  cout << dp[0] << endl;

  return 0;
}
```