Садржај

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 Вуīелија

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 $a_1 \geq 0$.

```
#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 Пошадић, 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:

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

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 Вуīелија

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.

```
#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 Вугделија

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		**Пример 2**		**Пример 3**	
Улаз	*Излаз*	*Улаз*	*Излаз*	*Улаз*	*Излаз*
7	7	4	2	4	1
2		2	4	1	9
4		7	7	6	
1		9	9	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 $p_i^{\left\lfloor \frac{k_i}{2} \right\rfloor}$ (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 $p_i^{\left\lfloor \frac{k_i}{2} \right\rfloor + 1}$).

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 Вугделија

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 Поп̄адић

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

Опис главног решења

У овом блоку се описује главно решење задатка.

```
#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

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

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

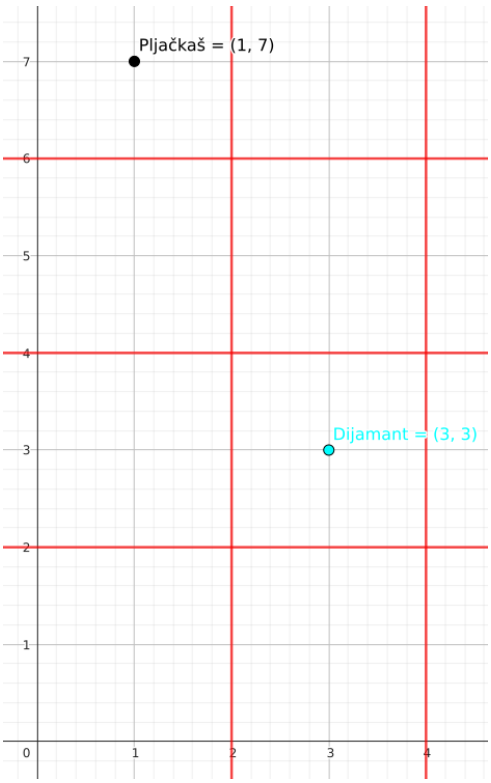
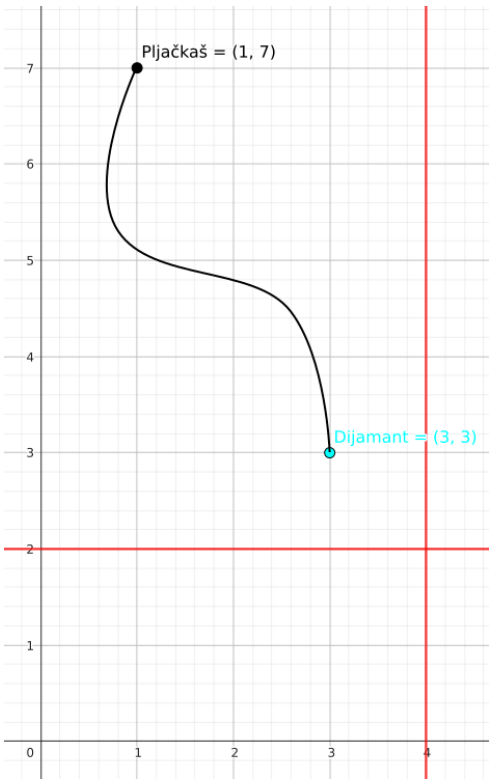
$$-10^9 \leq v_i, h_i \leq 10^9$$

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

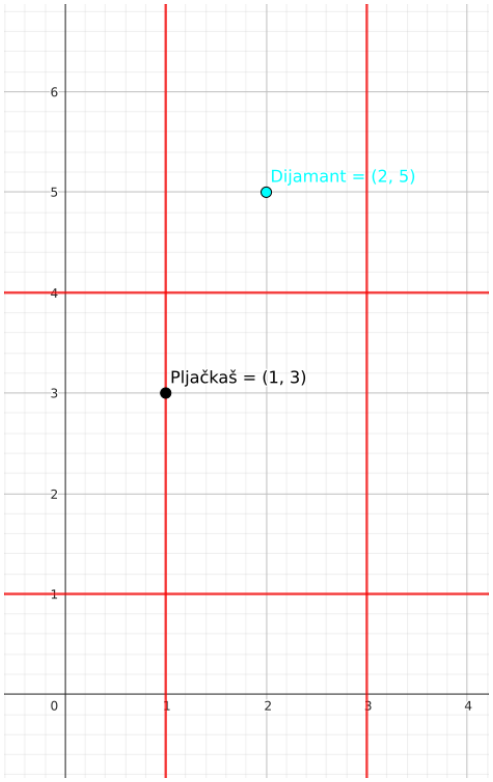
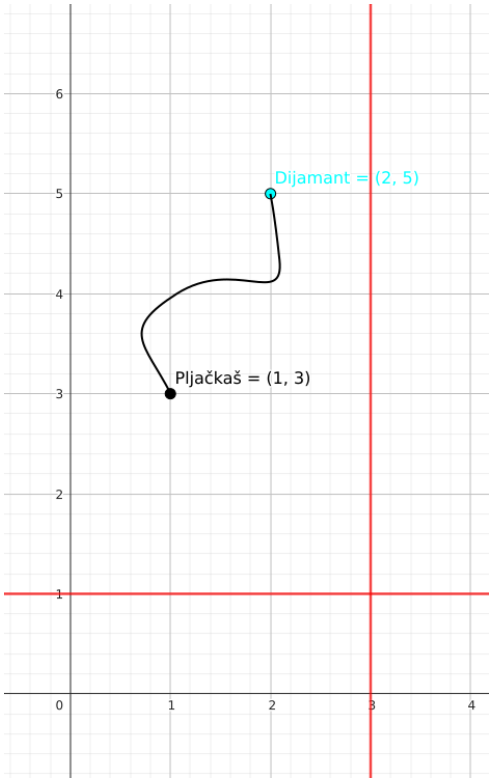
```
1 3
2 5
2
1 3
2
1 4
```

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 Вуīделија

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 променљиву спољне петље користимо као индекс почетка сегмента, а променљиву унутрашње петље као индекс краја. Логичка променљива 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 тртажених 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 Вуџелија

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;
}
```