Садржај

1 Сезона 2021/22. **1**

1.1 Квалификације – први круг . . . 1

Задатак: Рам . . . 1

Задатак: Повећај сваку цифру . . . 2

Задатак: Папир камен маказе . . . 2

Задатак: Ближе црти . . . 3

Задатак: Укупно време . . . 4

Задатак: Квадрати максималне површине . . . 5

Задатак: Мајице . . . 7

Задатак: Сличне речи . . . 8

Задатак: Зграде . . . 10

Задатак: Куповина . . . 12

Задатак: Највећи двоцифрени број . . . 13

Задатак: Такси растојање око правоугаоника . . . 14

Задатак: Клин . . . 16

Задатак: Тачке . . . 18

Задатак: Боје у кругу . . . 20

Задатак: Карте . . . 21

Задатак: Постављање домина . . . 23

Задатак: Шпанска серија . . . 25

1.2 Квалификације – други круг . . . 28

Задатак: Столњак . . . 28

Задатак: Паковање . . . 28

Задатак: Самогласници . . . 29

Задатак: Дежурство . . . 31

Задатак: Магацин . . . 32

Задатак: Коцке . . . 34

Задатак: Број посета кућици . . . 37

Задатак: Телеграф . . . 38

Задатак: Скочко . . . 40

Задатак: Подморница . . . 42

Задатак: Четвороугао . . . 45

Задатак: Играчка . . . 46

Задатак: Множење и кореновање . . . 48

Задатак: Сечење . . . 49

Задатак: Прозор и ограда . . . 51

Задатак: Чинилац мање . . . 52

Задатак: Експеримент . . . 54

Глава 1

Сезона 2021/22.

1.1 Квалификације – први круг

Задатак: Рам

Аутор: Нина Икодиновић

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

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

Излаз: На стандардни излаз исписати само обим унутрашњег дела рама.

Пример 1

Улаз

```
150
10
```

Излаз

```
70
```

Пример 2

Улаз

```
200
17
```

Излаз

```
64
```

Решење

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

```
#include <iostream>

using namespace std;

int main() {
   int n, m;
    cin >> n >> m;
    int obim = n - 8 * m;
```

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

Задатак: Повећај сваку цифру

Аутор: Небојша Варница

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

Улаз: У првом и једином реду стандардног улаза налази се двоцифрен број

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

Пример 1

Улаз

```
47
```

Излаз

```
58
```

Пример 2

Улаз

```
39
```

Излаз

```
410
```

Решење

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

```
#include <iostream>

using namespace std;

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

Задатак: Папир камен маказе

Аутор: Небојша Варница

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

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

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

Пример 1

Улаз

```
P
M
```

Излаз

```
2
```

Пример 2

Улаз

```
K
K
```

Излаз

```
0
```

Решење

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

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

using namespace std;

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

Задатак: Ближе црти

Аутор: Милан Вуīелија

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

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

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

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

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

Пример 1

Улаз

```
96
85
```

Излаз

```
Boba
```

Пример 2

Улаз

```
100
100
```

Излаз

```
Pera
```

Пример 3

Улаз

```
91
102
```

Излаз

```
Boba
```

Решење

Најједноставније решење се добија надовезаним наредбама гранања.

Једноставан случај у коме одмах знамо одговор је када је Боба бацио преко црте, а тада је победник Пера. У противном (ако Боба није бацио преко црте), следећи једноставан случај је када је Пера бацио преко црте, тада је Боба победио. Преостали су случајеви када ни један играч није пребацио црту. Под тим условом, Пера је победио ако је бацио исто или даље него Боба, а Боба је победио ако је бацио даље него Пера.

```
#include <iostream>

using namespace std;

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

Задатак: Укупно време

Аутор: Филип Марић

Петар жели да измери укупно време које проводи играјући своју омиљену видео-игру. Сваки пут када је покренуо игру он је записао време почетка и када је искључио игру он је записао време завршетка играња. Напиши програм који на основу тих времена израчунава укупно време играња.

Улаз: Са стандардног улаза се учитава број n ($1 \leq n \leq 100$) пута колико је Петар играо игру током недељу дана. Након тога се учитава n линија које садрже времена почетка и завршетка игре. Свако време се задаје у формату `h m` (сат и минут раздвојени размаком), при чему су време почетка и време завршетка игре увек у истом дану (Петар никад није остао будан до поноћи).

Излаз: На стандардни излаз исписати укупно време (број сати и минута раздвојене размаком).

Пример 1

Улаз

```
3
14 35 14 55
9 20 10 13
7 13 9 7
```

Излаз

```
3 7
```

Пример 2

Улаз

```
1
14 35 14 55
```

Излаз

```
0 20
```

Решење

Потребно је израчунати трајање сваке појединачне игре и затим сабрати тако добијена времена. Трајање појединачне игре можемо најлакше израчунати ако и време почетка и време завршетка претворимо у минуте (тако што сате помножимо са 60 и саберемо са минутима). Пошто на тај начин добијамо укупно време у минутима, по завршетку сабирања је потребно да израчунамо број сати (као целобројни количник при дељењу са 60) и број минута (као остатак при дељењу са 60).

```
#include <iostream>

using namespace std;

int main() {
  int n;
  cin >> n >> ws;
  int mukupno = 0;
  for (int i = 0; i < n; i++) {
    int spoc, mpoc, skraj, mkraj;
    cin >> spoc >> mpoc >> skraj >> mkraj;
    mukupno += (60*skraj + mkraj) - (60*spoc + mpoc);
  }
  cout << mukupno / 60 << " " << mukupno % 60 << endl;
  return 0;
}
```

Задатак: Квадрати максималне површине

Аутор: Нина Икодиновић

Васа је добио задатак да изреже лист папира димензије N пута M на квадрате максималне површине. Васа најпре исеца највећи могући квадрат тако да сече лист папира по најдужој страници (нпр. за лист димензије 3x7 највећи могући квадрат је 3x3). Потом Вас склони квадрат и над преосталим правоугаоником понови исецање квадрата највеће површине. Наставља исту операцију све док преостали правоугаоник не постане квадрат. Написати програм који ће израчунати број квадрата који ће Васа добити након исецања на описани начин.

Улаз: У првом реду стандардног улаза налази се природан број M, а у другом природан број N. Бројеви M и N нису већи од 2^{60}.

Излаз: На стандардни излаз исписати број квадрата који се могу добити наведеним исецањем.

Пример 1

Улаз

```
3
7
```

Излаз

```
5
```

Пример 2

Улаз

```
23
6
```

Излаз

```
9
```

Решење

Директно решење подразумева да се исеца један по један квадрат тј. да се мања страница правоугаоника одузима од веће и да се у сваком бројач квадрата увећава за 1. Поступак престаје када се добије квадрат тј. када се две димензије изједначе.

Иако коректан, овај поступак је неефикасан (нарочито у случајевима када је једна страна правоугаоника много мања од друге).

```
#include <iostream>

using namespace std;

int main() {
  long long m, n;
  cin >> m >> n;
  long long br = 1;
  while (n != m) {
    if (n < m)
      m = m - n;
    else
      n = n - m;
    br++;
  }
  cout << br << endl;
  return 0;
}
```

Нека су странице правоугаоника $M \times N$ и нека је $M < N$. Тада ће првих $\lfloor \frac{N}{M} \rfloor$ исечених квадрата бити димензије $M \times M$, а преостали правоугаоник ће бити димензије $N \bmod M$.

Уз мало додатне анализе, може се приметити да исто тврђење важи и када је $M = N$, па чак и када је $M > N$ (потврдите ово на примерима).

Одавде можемо да изведемо следећи поступак: док год је M позитивно, бројач исечених квадрата увећавамо за $\lfloor \frac{N}{M} \rfloor$, а бројеве M и N замењујемо редом са $N \bmod M$ и M.

По завршетку поступка M ће бити нула, што значи да је цео правоугаоник исечен на квадрате, а сви квадрати пребројани.

```
#include <iostream>

using namespace std;

int main() {
```

```
  long long m, n, r;
  cin >> m >> n;
  long long br = 0;
  while (m > 0) {
    br += n / m;
    r = n % m;
    n = m;
    m = r;
  }
  cout << br;
  return 0;
}
```

Задатак: Мајице

Аутор: Милан Вуīелија

Сваки учесник кампа треба да добије по једну мајицу. A учесника је рекло да им одговара мала или средња величина, а B учесника да им одговара средња или велика. На располагању већ имамо S малих, M средњих и L великих мајица. Одредити колико мајица је потребно додатно поручити, да би их било довољно за све учеснике кампа.

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

Излаз: Један цео број, број мајица које треба поручити.

Пример 1

Улаз

```
5
6
7
1
4
```

Излаз

```
1
```

Пример 2

Улаз

```
5
6
1
1
8
```

Излаз

```
3
```

Пример 3

Улаз

```
5
6
2
2
2
```

Излаз

```
5
```

Пример 4

Улаз

```
5
6
2
9
2
```

Излаз

```
0
```

Решење

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

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

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

using namespace std;

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

Задатак: Сличне речи

Аутор: Милан Вуђелија

Написати програм, који за низ од N датих речи једнаке дужине проверава да ли се сваке две узастопне речи у низу разликују на тачно једној позицији.

Улаз: У првом реду стандардног улаза је природан број N ($2 \leq N \leq 20$). У сваком од N наредних редова је по једна реч, која се састоји од великих слова енглеске абецеде. Дужина свих N речи је иста, и то најмање једно, а највише 30 слова.

Излаз: На стандардни излаз исписати само реч `da` или реч `ne`.

Пример 1

Улаз

```
4
KOBILA
POBILA
POBIJA
DOBIJA
```

Излаз

```
da
```

Пример 2

Улаз

```
3
KAKAV
KAKVA
KAKVO
```

Излаз

```
ne
```

Пример 3

Улаз

```
2
PERCE
PERCE
```

Излаз

```
ne
```

Решење

Користимо логичку променљиву `reciSuSlicne`, у којој ће се налазити одговор на питање да ли су сви прегледани парови узастопних речи слични, тачније: да ли се разликују на тачно једној позицији. На крају рада програма ћемо у овој променљивој имати коначан одговор. На почетку "сви испитани" парови испуњавају услов (јер сви елементи празног скупа испуњавају било који услов), па је почетна вредност променљиве `true`.

Пошто сваку нову реч поредимо са претходном, згодно је да прву реч учитамо пре петље. У петљи учитавамо нову реч и пребројимо позиције на којима се она разликује од претходне речи. Ако је број тих позиција различит од један, коначан одговор је познат (одречан је) и може се прекинути даље испитивање. Ако је број тих позиција једнак један, испитивање се наставља. У том случају потребно је још да ажурирамо претходну реч (у следећем проласку кроз петљу претходна реч ће бити ова која је у текућем проласку била нова).

```
#include <iostream>

using namespace std;

int main() {
    bool reciSuSlicne = true;
    string prethRec, sledRec;
    int n;
    cin >> n >> prethRec;
    for (int rec = 1; rec < n && reciSuSlicne; rec++) {
        cin >> sledRec;
        int brRazlSlova = 0;
        for (int slovo = 0; slovo < prethRec.size(); slovo++)
            if (prethRec[slovo] != sledRec[slovo])
                brRazlSlova++;

        if (brRazlSlova != 1)
            reciSuSlicne = false;

        prethRec = sledRec;
    }
    if (reciSuSlicne)
        cout << "da";
    else
        cout << "ne";
   return 0;
}
```

Задатак: Зграде

Аутор: Душан Пођадић

За потребе научног експеримента научницима је потребно да у Њујорку пронађу две зграде на чијим ће крововима обавити одређена мерења притиска. Да би резултати били прецизни потребно је да разлика у висинама између тих зграда буде велика, али због неких услова везаних за апаратуру коју користе, та разлика не сме бити ни превелика. Максимална дозвољена разлика у висинама зграда је утврђена и износи h. Пошто зграда у Њујорку има много, научницима је врло тешко да пронађу одговарајуће зграде и због тога су замолили тебе за помоћ. Дат је низ од n зграда разних висина у коме је потребно да пронађеш две зграде чија је разлика у висинама што већа, али мања или једнака h.

Улаз: У првом реду стандардног улаза налазе се два броја: максимална дозвољена разлика у висини h (h је природан број, $1 \leq h \leq 2 \cdot 10^9$) и број зграда n ($2 \leq n \leq 10^5$) одвојени размаком. У наредних n редова се налазе природни бројеви (мањи или једнаки од $2 \cdot 10^9$) који представљају висине зграда.

Напомена: Гарантује се да ће постојати барем две зграде чија је разлика у висинама мања или једнака h.

Ограничења

- У тест примерима вредним 50 поена важи $2 \leq n \leq 1000$.
- У осталим тест примерима нема додатних ограничења.

Излаз: У једином реду стандардног улаза исписати прво висину мање зграде, а затим висину веће зграде које ће бити коришћене за експеримент. Бројеве одвојити размаком. У случају да постоји више решења, исписати оно код ког су зграде ниже (ако је једно решење 2 и 6, а друго 4 и 8, треба исписати прво).

Пример 1

Улаз

```
5 6
12
4
20
16
22
17
```

Излаз

```
12 17
```

Пример 2

Улаз

```
8 5
13
4
20
11
57
```

Излаз

```
4 11
```

Решење

Решење грубом силом

Најједноставније решење је да се са две угнежђене петље прође кроз сваки пар зграда и провери се да ли им је разлика у висинама мања од h. Уколико јесте тада проверавамо да ли је разлика у висинама већа од до сада највеће запамћене разлике и уколико јесте памтимо је као нову највећу разлику и памтимо посматрани пар зграда. С обзиром да се у случају више решења тражи оно у коме се појављују мање зграде треба и о

томе водити рачуна, па ћемо памтити у случају да се појави пар зграда који постиже исту разлику као до сада најбољи пар, тај нови пар упамтити уколико су му зграде ниже, а у супротном задржавамо стари пар.

Потребно проћи кроз све парове зграда којих има $\frac{n \cdot (n-1)}{2}$. Како је максималан број зграда $n_{max} = 10^5$, максималан број парова је нешто мањи од $5 \cdot 10^9$ тако да кроз толики број парова не можемо проћи за $1s$ колико је временско ограничење. Како је у задатку речено да ће у тест примерима вредним 50п важити да је n до 1000, у том случају имамо око $5 \cdot 10^5$ парова и можемо их све обићи за $1s$. Дакле, ово решење нам може донети 50п на овом задатку.

Сортирање и техника два показивача

Претходно решење има такозвану квадратну сложеност ($\frac{n \cdot (n-1)}{2} = O(n^2)$), а нама је потребна сложеност која је боља од тога: линеарне $O(n)$ или сублинеарне $O(n \log n)$. Алгоритми сублинеарне сложености ће бити извршени за мање $1s$ када је $n\_max = 10^5$ јер је $n_{max} \log n_{max}$ реда величина једног милиона.

Пошто алгоритми сублинеарне сложености пролазе временско ограничење, то значи да можемо да приуштимо сортирање низа и треба размислити да ли нам сортирање нешто доноси. У овом задатку се испоставља да доноси.

Прво је потребно сортирати низ висина зграда растуће (уз могућност појаве више зграда исте висине). Сада ћемо поставити два показивача (два цела броја која представљају индексе у низу) на прва два елемента у низу (рецимо i на први и j на други елемент). Прво ћемо померати показивач j ка крају низа докле год је разлика у висинама зграда на коју показују i и j мања или једнака h. Све време памтимо и највећу до сада постигнуту разлику, као и пар зграда који ју је постигао. Када j дође довољно далеко да је разлика у виснама зграда премашила h, тада померамо i док разлика не буде поново у дозвољеном оквиру. Након тога понављамо поступак (померамо j док разлика не буде већа од h, па померамо i док се разлика не врати у дозвољени оквир) док j не стигне до краја низа.

Треба уочити да када померамо i ка крају низа, није потребно да поново проверавамо зграде које су на позицијама мање од j. Ово се може видети на следећем примеру:

Пример. Нека је низ висина зграда: 1, 3, 6, 7, 8, 9, 15 и нека је $h = 6$.

На почетку постављамо i да показује на нулту позицију у низу (на којој се налази висина 1), а j на прву (висина 3).

Сада померамо j удесно и памтимо сваки пут нову највећу разлику. Када j стигне до позиције 3 (висина 7), максимална разлика коју ћемо упамтити је 6.

Сада се j помера на позицију 4 (висина 8) и разлика је сада $8 - 1 = 7$ што премашује h. Сада ћемо померити i за једно место удесно. Показивач i сада показује на позицију 1 (висина 3).

Оно што је потребно уочити је да није потребно проверавати разлику у висинама на позицији 1 (висина 3) и на позицији 3 (висина 7) јер смо већ проверавали разлику на позицијама 0 и 3, а та разлика је сигурно већа (или једнака) разлици висина на позицијама 1 и 3 пошто је зграда на позицији 0 нижа од зграде на позицији 1 (на почетку смо сортирали низ). Слично нема потребе проверавати разлику у висинама између зграде на позицији 1 и било које зграде лево од позиције 3. Дакле нема потребе враћати j уназад након померања i.

Такође, треба напоменути да уколико наиђемо на пар који има исту разлику у висинама као до сада запамћени најбољи пар, нећемо памтити нови пар јер нам се тражи нижи пар. Пошто пролазимо кроз растуће сортиран низ, сигурно је претходно запамћени пар нижи од новог.

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

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0);
  int n;
```

```
  int h;
  cin >> h >> n;
  vector<int> zgrade(n);
  for(int i = 0; i < n; i++)
    cin >> zgrade[i];
  sort(begin(zgrade), end(zgrade));
  int i = 0, j = 1;
  int maxD = 0;
  int in = 0, iv = 0;
  while(j < n) {
    if(zgrade[j] - zgrade[i] > h)
      i++;
    else {
      if(zgrade[j] - zgrade[i] > maxD) {
        maxD = zgrade[j] - zgrade[i];
        in = i;
        iv = j;
      }
      j++;
    }
  }
  cout << zgrade[in] << " " << zgrade[iv];

  return 0;
}
```

Задатак: Куповина

Аутор: Небојша Варница

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

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

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

Пример 1

Улаз

```
30
40
9
300
```

Излаз

```
6
3
```

Пример 2

Улаз

```
55
75
10
1000
```

Излаз

```
0
0
```

Решење

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

```
#include <iostream>

using namespace std;

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

Задатак: Највећи двоцифрени број

Аутор: Милан Вуīделија

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

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

Излаз: На стандардни излаз исписати само један двоцифрен број.

Пример 1

Улаз

```
235
```

Излаз

```
35
```

Објашњење

Највећи од бројева 23, 25, 35.

Пример 2

Улаз

```
412
```

Излаз

```
42
```

Објашњење

Највећи од бројева 41, 42, 12.

Решење

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

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

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

using namespace std;

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

Задатак: Такси растојање око правоугаоника

Аутор: Милан Вуђелија

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

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

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

Излаз: На стандардни излаз исписати тражену дужину на четири децимале.

Пример

Улаз

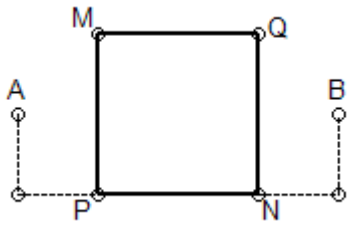
```
0.0 -2.00
0.0 2.00
-2.0 1.0
1.0 -1.0
```

Излаз

```
6.0000
```

Објашњење

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



Слика 1.1: Такси растојање

Решење

Упутство:

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

```
  7  |  8  |  9
     |     |
-----+-----+-----
     |     |
  4  |  5  |  6
     |     |
-----+-----+-----
     |     |
  1  |  2  |  3
```

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

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

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

using namespace std;

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

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

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

```
        + abs(ay-by);
  else
      d = abs(ax-bx) + abs(ay-by);

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

Задатак: Клин

Аутор: Милан Вуīделија

Написати програм који утврђује да ли низ датих речи представља формацију, која се у енигматици назива клин.

Низ речи чини клин ако свака следећа реч може да се добије од претходне избацивањем једног слова и по потреби мењањем редоследа осталих слова. Последња реч у формацији клина треба да има једно слово.

Улаз: У првом реду стандардног улаза се налази цео позитиван број N, број речи у низу. У сваком од следећих N редова је по једна реч састављена од великих слова енглеске абецеде. Број N и дужине речи нису већи од 20.

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

Пример 1

Улаз

```
4
PLAV
VAL
LA
A
```

Излаз

```
DA
```

Пример 2

Улаз

```
3
VLAGA
LAVA
VAL
```

Излаз

```
NE
```

Објашњење

Последња реч није дужине 1.

Пример 3

Улаз

```
3
VLAGA
LAV
L
```

Излаз

```
NE
```

Објашњење

Речи се не скраћују за по једно слово.

Пример 4

Улаз

```
4
HLAD
LAV
LA
L
```

Излаз

```
NE
```

Објашњење

Друга реч се не добија од прве по описаном правилу.

Решење

Из поставке задатка је јасно да дужине речи морају да буду редом n, $n-1$, … 1. Због тога за сваку учитану реч прво проверавамо да ли је одговарајуће дужине, па ако није - даље провере могу да се зауставе јер знамо да је коначан одговор одречан.

Осим дужина, за сваку реч осим прве треба да проверимо да ли може да се добије од претходне на начин описан у поставци задатка. Да бисмо то што једноставније проверили, згодно је да приликом учитавања сваке речи слова у њој сортирамо растуће. Након сортирања, пролазимо кроз претходну и текућу реч упоредо. Када су слова у речима иста, напредујемо у обе речи, а када су различита, напредујемо само у претходној (дужој) речи и бројимо разлику. Да би тражени услов био испуњен, по проласку кроз две речи, број разлика треба да буде 1. Ако је добијени број разлика мањи или већи, даље провере могу да се прекину, јер је и у овом случају коначан одговор одречан.

Уколико по проласку кроз све речи нисмо наишли на неправилности, коначан одговор је потврдан.

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

using namespace std;

int main() {
    int nPre;
    string sPre, sSled;
    cin >> nPre >> sPre;
    sort(sPre.begin(), sPre.end());
    bool nizJeKlin = true;
    if (sPre.size() != nPre)
        nizJeKlin = false;

    for (int nSled = nPre - 1; nSled > 0 && nizJeKlin; nSled--) {
        cin >> sSled;
        sort(sSled.begin(), sSled.end());
        if (sSled.size() != nSled)
            nizJeKlin = false;
        else {
            int iPre = 0, iSled = 0, brPreskocenih = 0;
            for (iPre = 0; iPre < nPre; iPre++) {
                if (iSled >= nSled || sPre[iPre] != sSled[iSled])
                    brPreskocenih++;
                else iSled++;
            }
            if (brPreskocenih != 1)
```

```
                nizJeKlin = false;
        }
        sPre = sSled;
        nPre = nSled;
    }

    if (nizJeKlin)
        cout << "DA" << endl;
    else
        cout << "NE" << endl;
   return 0;
}
```

Задатак: Тачке

Аутор: Душан Пођадић

Дат је низ затворених интервала на x-оси координатног система (интервал је ограничен почетном и крајњом тачком које се задају) и низ тачака. За сваки од интервала одредити колико му унетих тачака припада.

Напомена: Уколико тачка има x-координату која одговара самој граници интервала, сматра се да та тачка припада интервалу. Интервали се могу преклапати што може довести до тога да се једна тачка налази у више интервала.

Улаз: У првом реду стандардног улаза уноси се број тачака m ($1 \leq m \leq 10^5$), а у наредних m редова x-координата сваке од тачака (природни бројеви $1 \leq x_i \leq 2 \cdot 10^9$). У наредном реду се уноси број затворених интервала n ($1 \leq n \leq 10^5$), а у наредних n редова се уносе по два природна броја ($1 \leq l_i < d_i \leq 2 \cdot 10^9$) која означавају границе тих интервала (уноси се прво лева па десна граница).

Ограничења

- У тест примерима вредним 50 поена важи $1 \leq n, m \leq 1000$.
- У осталим тест примерима нема додатних ограничења.

Излаз: У n редова стандардног излаза исписати број тачака које припадају сваком од интервала (по редоследу уношења интервала).

Пример

Улаз

```
5
3
6
7
1
4
3
1 3
2 6
8 9
```

Излаз

```
2
3
0
```

Решење

Решење грубом силом

Најједноставније решење је да се за сваки интервал прође кроз низ тачака и провери за сваку тачку да ли припада интервалу. Уколико тачка припада интервалу треба увећати бројач и када се провере све тачке треба исписати вредност бројача.

Пошто је максималан број и интервала и тачака 10^5, угњежђене петље ће направити укупно $10^5 \cdot 10^5 = 10^10$ итерација што је превише да би се постигло за временско ограничење од $1s$. Како у тест примерима вредности 50 поена важи да је и број интервала и број тачака до 1000, у тим тест примерима број итерација ће бити највише $1000 \cdot 1000 = 10^6$, што је довољно мало да задовољи временско ограничење. Дакле, ово решење доноси 50 поена на задатку.

Решење бинарном претрагом

Да бисмо убрзали претрагу тачака које припадају интервалу можемо сортирати низ тачака. Сада пролазимо кроз сваки интервал и уместо да пролазимо кроз све тачке, довољно је да нађемо прву и последњу тачку у низу које припадају интервалу. Означимо границе интервала са A и B, а низ тачака са . Сада је потребно пронаћи најлевљу тачку T_i која се налази у интервалу, односно за коју важи $A \leq T_i$. Такође, потребно је наћи најдешњу тачку $\_j$ која припада интервалу, односно за коју важи $\_ \leq B$. Пошто је низ тачака сортиран све тачке између ће, такође, припадати посматраном интервалу. Тражене тачке се могу наћи бинарном претрагом која је модификована тако да не тражи конкретну вредност у низу него прву вредност која је већа или једнака односно мања или једнака од задате вредности. Након тога је лако да се, на основу добијених индекса тражених тачака, одреди колико тачака има између и самим тим колико тачака припада посматраном интервалу. За потребе бинарне претраге могу се на пример користити функције `upper_bound` и `lower_bound` из C++ стандардне библиотеке.

Сложеност сортирања низа тачака је $O(m \log m)$. Сложеност бинарне претраге је $(\log m)$, а пошто је потребно извршити бинарне претраге за сваки од интервала, сложеност тог дела решења је $O(n \log m)$. Укупна сложеност целог решења је $O(m \log m + n \log m) = O((m + n) \log m)$. Како ни n ни m сигурно нису већи од 10^5, ово решење је довољно ефикасно да задовољи временско ограничење од $1s$.

Потребно је обратити пажњу да није довољно да се само лева граница интервала тражи бинарном претрагом, а да се онда, почевши од ње, линеарно тражи десна зато што интервали могу бити веома велики тако да се ефективно за сваки интервал прође кроз скоро цео низ тачака, а то се своди на већ коментарисано лошије решење.

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

using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    int m;
    cin >> m;
    vector<int> tacke(m);
    for(int i = 0; i < m; i++)
        cin >> tacke[i];

    sort(tacke.begin(), tacke.end());

    int n;
    cin >> n;
    for(int i = 0; i < n; i++) {
        int lg, dg;
        cin >> lg >> dg;
        auto ub = upper_bound(tacke.begin(), tacke.end(), dg);
        auto lb = lower_bound(tacke.begin(), tacke.end(), lg);
        cout << distance(lb, ub) << "\n";
    }

   return 0;
}
```

Задатак: Боје у кругу

Аутор: Иван Дрецун

Перица и другари седе у кругу и смишљају коју ће игру следеће да играју. Договорили су се да ће се поделити у тачно три екипе (плаву, зелену и, наравно, црвену), али тако да деца која седе једна до другог морају бити у различитим екипама. Перицу и другаре занимају сви начини да направе поделу по екипама. Пошто још увек нису смислили коју ће игру следеће играти, замолили су тебе да им помогнеш и напишеш програм који ће исписати све поделе по екипама.

Улаз: Са стандардног улаза се уноси цео број n ($2 \leq n \leq 16$) који означава са колико другара се Перица игра.

Излаз: На стандардни излаз исписати онолико редова колико постоји могућих подела. Сваки ред треба да се састоји из тачно $n+1$ карактера који редом означавају Перичину екипу и затим екипу сваког следећег друга удесно. Плаву екипу представити малим словом `p`, зелену малим словом `z` и црвену малим словом `c`. Редове исписати уређене по абецедном редоследу.

Пример

Улаз

```
2
```

Излаз

```
cpz
czp
pcz
pzc
zcp
zpc
```

Решење

Приметимо да је потребно исписати сва бојења дужине $n+1$ која испуњавају услове.

Задатак је могуће решити формирањем свих распореда боја и исписивањем само оних који испуњавају услове задатка (појављују се све три боје, сваке две суседне се разликују). Распореди се могу формирати рекурзивном претрагом - прво формирањем свих распореда бојења који почињу словом `c`, затим оних који почињу словом `p` и коначно оних који почињу словом `z`. Сложеност оваквог решења је $O(n3^n)$ и не осваја максималан број поена.

Брже решење је могуће постићи уколико формирамо само она бојења која немају две суседне једнаке боје. Можемо модификовати рекурзивну претрагу претходног решења тако да уместо генерисања бојења која почињу сваком бојом генеришемо само бојења која почињу бојом различитом од претходне. На пример, уколико је у претходном кораку претраге фиксирана боја `p`, довољно је проверити сва бојења којима је следећа боја `c` или `z`. Овим се сложеност смањује на $O(n2^n)$ што је довољно за максималан број поена.

```
#include <iostream>

using namespace std;

void generisiBojenja(int i, string& boje, bool sveBoje) {
  if (i == boje.size()) {
    if (sveBoje && boje[i - 1] != boje[0])
      cout << boje << '\n';
    return;
  }

  for(auto boja : {'c', 'p', 'z'})
    if(boje[i - 1] != boja) {
      boje[i] = boja;
      generisiBojenja(i + 1, boje,
                      sveBoje || (boja != boje[0] && boja != boje[1]));
```

```
    }
}

void generisiBojenja(int n) {
  string boje(n, ' ');
  for(auto boja : {'c', 'p', 'z'}) {
    boje[0] = boja;
    generisiBojenja(1, boje, false);
  }
}

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

  generisiBojenja(n + 1);

  return 0;
}
```

Задатак: Карте

Аутор: Небојша Варница

Карте за игру имају ознаке A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2 - сложено од најјаче до најслабије карте. Осим ознаке свака карта има и "боју": пик, каро, херц и треф. Са картама се могу играти различите игре: "пасијанс", "таблић", "реми", "преферанс", "бриџ", "покер", "ајнц", "црна дама", …

У преферансу учествују три играча. На почетку игре се бира адутска боја. У сваком кругу игре играчи бацају по једну карту. Победник је играч који има најјачу карту у адутској боји. Ако нико не баци такву карту онда је победник играч који баци најјачу карту у боји коју је бацио први играч.

Примери: ако је адутска боја херц а играчи су бацили:

- K херц, 9 херц, A треф - победник је први играч јер је бацио најјачу карту у адутској боји
- A пик, 10 херц, K каро - победник је други играч јер је једини бацио адутску боју
- J треф, K каро, Q треф - победник је трећи играч јер нико није бацио адутску боју а трећи играч је бацио најјачу карту у боји коју је бацио први играч

Напишите програм који за унету адутску боју и три унете различите карте израчунава који је играч победник круга.

Улаз: Са стандардног улаза у првом реду се задаје адутска боја (почетно латинично слово назива боје - P, K, H или T) а затим у три следећа реда три карте: прво боја (почетно латинично слово назива боје - P, K, H или T) а затим ознака карте (A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3 или 2) - без размака

Излаз: На стандардном излазу приказати редни број играча који је победио у том кругу

Пример 1

Улаз

```
H
HK
H9
TA
```

Излаз

```
1
```

Пример 2

Улаз

```
H
PA
H10
KK
```

Излаз

```
2
```

Пример 3

Улаз

```
H
TJ
KK
TQ
```

Излаз

```
3
```

Решење

Опис главног решења

Упоређујемо прве две карте па се јача од њих упоређује са трећом.

Приликом упоређивања прво обрадимо случај када су обе адутске, па затим када је једна адутска а друга није и на крају када ниједна није адутска.

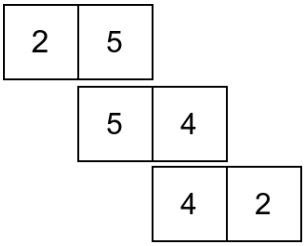
```
#include <iostream>

using namespace std;

int vrednost(char ozn)
{
  if (ozn >= '2' && ozn <= '9')
    return ozn-'0';
  switch(ozn) {
  case '1' : return 10;
  case 'J' : return 12;
  case 'Q' : return 13;
  case 'K' : return 14;
  case 'A' : return 15;
  }
  return 0;
}

int main()
{
  char adut, boja[3], oznaka[3];
  int res = 0;
  cin >> adut;
  for(int i=0; i<3; i++) {
    string x;
    cin >> x;
    boja[i] = x[0];
    oznaka[i] = x[1];
  }
  for(int i=1; i<3; i++) {
    if (boja[res] == adut && boja[i] == adut) {
      if (vrednost(oznaka[res]) < vrednost(oznaka[i]))
        res = i;
```

```
    } else if(boja[res]!=adut && boja[i]==adut)
      res=i;
    else if (boja[res]==boja[i]) {
      if (vrednost(oznaka[res]) < vrednost(oznaka[i]))
        res=i;
    }
  }
  cout << (res+1) << endl;
  return 0;
}
```

Задатак: Постављање домина

Аутор: Милан Вуīделија

У игри домина прва домина се поставља на празан сто, а свака следећа се надовезује на један или други крај низа раније постављених домина. Да би домина могла да се надовеже на неки крај, један од њена два броја мора да буде једнак броју на том крају. Приликом надовезивања, једнаки бројеви се стављају један до другог и остају у унутрашњости низа, а преостали број на домини постаје нови крај низа. На пример, ако је на почетку игре на сто стављена домина са бројевима 5 и 4, на њу је могла да буде надовезана домина са бројевима 5 и 2, а затим и она са бројевима 4 и 2, као што је приказано. После оваквог надовезивања, на крајевима низа се налазе бројеви 2 и 2.



Слика 1.2: Постављање домина

Написати програм, који за дати низ домина исписује да ли су оне могле да буду одигране и надовезане на низ редом у коме су наведене.

Улаз: У првом реду стандардног улаза је цео број n ($1 \leq n \leq 100$), број домина. У сваком од следећих n редова су по два једноцифрена природна броја, који представљају једну домину. Редослед два броја који представљају једну домину није битан (свака домина може да се окрене, ако је то потребно због надовезивања).

Излаз: На стандардни излаз исписати само реч `da` или реч `ne`.

Пример 1

Улаз

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

Излаз

```
da
```

Пример 2

Улаз

```
4
2 1
```

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

Излаз

```
ne
```

Решење

Прву домину увек можемо да поставимо. Током учитавања следећих домина, треба памтити два броја, који се налазе на крајевима претходно формираног низа и на које може да се надовеже нова домина.

Код сваке нове домине, потребно је да се за сваки крај низа и сваку страну домине провери да ли се уклапају, што је укупно 4 начина да се домина постави. Ако ниједан од та 4 начина није могућ, коначан одговор је NE. Ако домина може да се стави само на један крај низа, тај крај низа добија вредност другог краја домине и поступак се наставља.

Остаје да се размотри ситуација када домина може да се стави на било који крај низа. На пример, на крајевима низа су бројеви 2 и 5, а и домина саджи управо бројеве 2 и 5. Домину можемо да ставимо тако, да после стављања на крајевима буду бројеви 2 и 2, или бројеви 5 и 5. Размислимо шта све може да се догоди са појављивањем следеће домине.

- Ако следећа домина нема ниједан од бројева 2 и 5, коначан одговор је NE.
- Ако следећа домина има само један од бројева 2 и 5, нпр. 5 и 4, претходну домину је требало ставити тако да крајеви низа буду 5 и 5, а нову тако да крајеви постану 5 и 4.
- Ако следећа домина има управо бројеве 2 и 5, све једно је како смо ставили претходну домину, јер у оба случаја после стављања нове домине на крајевима низа ће бити бројеви 2 и 5.

У сва три случаја нејднозначност из претходног потеза се одмах разрешава. Према томе, довољно је да употребимо једну логичку променљиву (у програму је то променљива dupli), која означава да је наступила ова привремена нејднозначност. То значи да се од два броја које памтимо као крајеве низа, један налази на оба краја, а други ни на једном. Који је који од та два, одлучујемо већ након читања следеће домине и неједнозначност престаје.

```
#include <iostream>

using namespace std;

int main() {
    int n, x, y, a, b;
    cin >> n >> x >> y;
    bool dupli = false, moze = true;
    for (int i = 1; i < n; i++) {
        cin >> a >> b;
        if (a == x && b == y)
            dupli = !dupli;
        else if (a == y && b == x)
            dupli = !dupli;
        else
        {
            if (a == x)
                if (dupli) y = b; else x = b;
            else if (a == y)
                if (dupli) x = b; else y = b;
            else if (b == x)
                if (dupli) y = a; else x = a;
            else if (b == y)
                if (dupli) x = a; else y = a;
            else
                moze = false;
```

```
            dupli = false;
        }
    }
    if (moze)
        cout << "da" << endl;
    else
        cout << "ne" << endl;
    return 0;
}
```

Задатак: Шпанска серија

Аутор: Иван Дрецун

Писци шпанске серије "Игра пресолца" одлучили су да у последњој епизоди направе велико окупљање на које ће бити позвани неки од n ликова из серије. Наравно, како то обично бива у шпанским серијама, постоји списак тројки ликова који се међусобно не подносе. Задатак за писце је да одаберу које од n ликова треба позвати тако да из сваке тројке са списка **тачно једна** особа буде позвана. Напиши програм који рачуна колико највише људи може бити позвано поштујући задате услове и исписује један такав списак званица.

Улаз: У првом реду стандардног улаза се уносе цели бројеви n ($3 \leq n \leq 14$) и k ($1 \leq k \leq 140$) који редом представљају број ликова у серији и број тројки ликова који се међусобно неподносе. У наредних k редова уносе се по три различита броја који означавају три лика од којих тачно један треба бити позван. Ликови су означени редним бројевима између 1 и n.

Излаз: У првом реду стандардног излаза исписати број позваних људи. У другом реду исписати редне бројеве позваних људи одвојене размаком. Уколико постоји више решења, исписати било које. Уколико не постоји списак званица који задовољава све услове, у првом реду стандардног излаза исписати 0.

Пример 1

Улаз

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

Излаз

```
3
2 4 6
```

Пример 2

Улаз

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

Излаз

```
0
```

Решење

Потребно је испробати све подскупове ликова и за сваки подскуп проверити да ли испуњава задати низ услова, односно да ли се из сваке тројке појављује тачно једна особа у подскупу. Приликом претраге подскупова потребно је памтити највећи до сада пронађен подскуп који испуњава услове - по завршетку испитивања свих подскупова тај ће бити решење.

Подскупове је могуће генерисати рекурзивним поступком: испитивањем прво свих подскупова који не садрже једног лика, а затим испитивањем свих подскупова који га садрже. Подскупове је могуће генерисати и употребом алгоритма за одређивање наредног подскупа.

Сложеност поскупка је $O(k2^n)$, што је у реду за дата ограничења променљивих.

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

using namespace std;

struct trojka { int a, b, c; };

int max_velicina = 0;
vector<bool> max_podskup;

bool ispunjavaUslove(vector<bool>& podskup, vector<trojka>& uslovi) {
  for(auto t : uslovi) {
    int broj_ljudi = 0;
    broj_ljudi += podskup[t.a - 1];
    broj_ljudi += podskup[t.b - 1];
    broj_ljudi += podskup[t.c - 1];
    if(broj_ljudi != 1)
      return false;
  }
  return true;
}

void sviPodskupovi(int i, vector<bool>& podskup, int velicina, vector<trojka>& uslovi) {
  if (i == podskup.size()) {
    if (velicina > max_velicina && ispunjavaUslove(podskup, uslovi)) {
      max_velicina = velicina;
      max_podskup = podskup;
    }
    return;
  }

  podskup[i] = false;
  sviPodskupovi(i + 1, podskup, velicina, uslovi);

  podskup[i] = true;
  sviPodskupovi(i + 1, podskup, velicina + 1, uslovi);
}

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

  vector<trojka> trojke(k);
  for(int i = 0; i < k; i++)
    cin >> trojke[i].a >> trojke[i].b >> trojke[i].c;

  vector<bool> podskup(n);
  sviPodskupovi(0, podskup, 0, trojke);

  cout << max_velicina << '\n';
  if (max_velicina > 0)
    for (int i = 0; i < n; i++)
      if (max_podskup[i])
        cout << (i + 1) << ' ';
  cout << '\n';
  return 0;
}
```

Подскупове можемо представити и помоћу бинарног записа неозначених целих бројева.

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

using namespace std;

struct trojka { int a, b, c; };

int brojPozvanih(unsigned pozvani) {
  return __builtin_popcount(pozvani);
}

bool pozvan(unsigned pozvani, int o) {
  return pozvani & (1u << o);
}

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

  vector<trojka> trojke(k);
  for(int i = 0; i < k; i++)
    cin >> trojke[i].a >> trojke[i].b >> trojke[i].c;

  int max_podskup = 0;
  int max_pozvanih = 0;
  for(int podskup = 0; podskup < (1 << n); podskup++) {
    if (brojPozvanih(podskup) <= max_pozvanih)
      continue;
    bool sat = true;
    for (auto t : trojke) {
      int count = 0;
      count += (int)pozvan(podskup, t.a - 1);
      count += (int)pozvan(podskup, t.b - 1);
      count += (int)pozvan(podskup, t.c - 1);
      if (count != 1) {
        sat = false;
        break;
      }
    }
    if (sat) {
      max_podskup = podskup;
      max_pozvanih = brojPozvanih(podskup);
    }
  }

  cout << max_pozvanih << '\n';
  if (max_pozvanih > 0)
    for (int o = 1; o <= n; o++)
      if (pozvan(max_podskup, o-1))
        cout << o << ' ';

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

1.2 Квалификације – други круг

Задатак: Столњак

Аутор: Нина Икодиновић

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

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

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

Пример1

Улаз

```
400
350
120
```

Излаз

```
6
```

Пример2

Улаз

```
5000
3700
200
```

Излаз

```
450
```

Решење

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

```
#include <iostream>

using namespace std;

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

Задатак: Паковање

Аутор: Филип Марић

Ближи се Нова година и деца припремају поклоне за своје другаре. Поклони су у кутијама облика квадра и приликом паковања се облепљују украсним папиром. Ако је квадар димензија $a \times b \times c$, његове стране су два

правоугаоника површине $a \times b$, два правоугаоника површине $b \times c$ и два правоугаоника површине $a \times c$. Укупна површина је збир свих 6 површина правоугаоника. Ипак, да би се поклон могао лепо запаковати, потребно је да се одвоји мало вишка папира – најекономичније је да тај вишак буде површине која је једнака најмањој од три површине правоугаоника. Напиши програм који одређује колико је папира потребно да се запакује поклон.

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

Излаз: На стандардни излаз исписати тражену најмању површину папира за увијање.

Пример 1

Улаз

```
2
4
3
```

Излаз

```
58
```

Пример 2

Улаз

```
12
10
8
```

Излаз

```
672
```

Решење

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

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

using namespace std;

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

Задатак: Самогласници

Аутор: Филип Марић

Матеја воли речи које почињу и завршавају се самогласницима (на пример, **ауто**, **игла**, **окно**) - њима би дала оцену 5.

Мало су јој мање речи које само почињу или се само завршавају самогласником (на пример, **песма**, **авион**) - њима би дала оцену 3.

Најмање су јој драге речи које почињу и завршавају се сугласницима (на пример, **телевизор**, **рам**, **курсор**) - њима би дала оцену 1.

Улаз: Са стандардног улаза се учитавају три речи записане малим словима енглеске абецеде, свака у посебном реду. Речи нису дуже од 20 слова.

Излаз: На стандардни излаз исписати укупну оцену коју би Матеја дала за све три речи.

Пример 1

Улаз

```
auto
avion
televizor
```

Излаз

```
9
```

Пример 2

Улаз

```
oboa
klarinet
klavir
```

Излаз

```
7
```

Решење

Проверу да ли је дати карактер самогласник можемо извршити поређењем са свих пет самогласника. Пошто ову проверу треба вршити више пута, згодно је издвојити је у посебну функцију. Када се учита реч, потребно је проверити јој прво и последње слово. Првом слову ниске `s` можемо приступити са `s[0]` или `s.front()`, док последњем слову можемо приступити са `s[s.size()-1]` или `s.back()`. Гранањем прво проверавамо да ли су оба слова самогласници (треба да буде самогласник и једно и друго слово). Ако јесу, оцена речи је 5. Ако нису, проверамо ли је бар једно слово самогласник (треба да буде самогласник или једно или друго слово). Ако јесте, оцена речи је 3. Ако није, онда су оба слова сугласници и оцена речи је 1.

Пошто је потребно оценити три речи, згодно је оцену дефинисати у склопу посебне функције.

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

using namespace std;

bool samoglasnik(char c) {
  return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';
}

int ocena(const string& rec) {
  if (samoglasnik(rec.front()) && samoglasnik(rec.back()))
    return 5;
  if (samoglasnik(rec.front()) || samoglasnik(rec.back()))
    return 3;
  return 1;
}

int main() {
  int ukupno = 0;
  string rec;
  getline(cin, rec);
  ukupno += ocena(rec);
  getline(cin, rec);
  ukupno += ocena(rec);
  getline(cin, rec);
  ukupno += ocena(rec);
  cout << ukupno << endl;
```

```
  return 0;
}
```

Задатак: Дежурство

Аутор: Милан Вуīделија

У једном великом и сложеном систему се повремено дешавају инциденти, а тада је потребно да дежурни службеник брзо интервенише и отклони или премости проблем. Имамо податке о времену дешавања инцидената током N дана, као и податке о томе када је Илија био дежуран током тог периода. Потребно је одредити укупан број инцидената, као и број инцидената који су се догодили за време Илијиног дежурства.

Улаз: У првом реду стандардног улаза је цео број N ($1 \leq N \leq 20$), број дана током којих су праћени инциденти. У свакој од наредних N група по три реда улаза налазе се подаци за један дан, и то:

У првом реду у оквиру групе је низ од 24 слова. Свако од слова је `S` или `D`, а означава да ли је Илија био слободан или дежуран у одговарајућем сату тог дана. У другом реду у оквиру групе је један цео број, број инцидената B ($1 \leq B \leq 9$), који су се догодили тог дана. У трећем реду у оквиру групе је B целих бројева, сваки од 0 до 23, укључујући границе. Ти бројеви представљају сате у којима су се тог дана догодили инциденти.

Излаз: На стандардни излаз исписати два цела броја, раздвојена размаком. Први број је укупан број инцидената, а други је број инцидената који су се догодили за време Илијиног дежурства.

Пример

Улаз

```
3
DSSSDDSSDSSDSSSDSSSSDSSS
2
7 9
DSSDDSSDSSDSSSDSSSSDSSSS
5
3 3 4 8 8
DSSSDDSSDSSDSSDSSSSDSSSS
4
0 14 14 14
```

Излаз

```
11 7
```

Објашњење

Бројећи за сваки дан слова у распореду Илијиног дежурства од нуле, видимо да је Илија био дежуран за време 7 од укупно 11 инцидената (у трећем и четвртом сату другог дана и нултом и четрнаестом сату трећег дана).

Решење

Укупан број инцидената можемо да одредимо сабирајући бројеве инцидената по данима. Да бисмо пребројали инциденте током којих је Илија био дежуран, потребно је да за сваки дан прочитамо времена инцидената из тог дана и да бројимо оне инциденте којима одговара слово `D` у Илијином распореду дежурства.

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

using namespace std;

int main() {
    int nDana, nIncidenata, vreme;
    int brSvihInc = 0;
    int brDezInc = 0;
    string raspored;
    cin >> nDana;
    for (int d = 0; d < nDana; d++) {
```

```
        cin >> raspored >> nIncidenata;
        brSvihInc += nIncidenata;
        for (int i = 0; i < nIncidenata; i++) {
            cin >> vreme;
            if (raspored[vreme] == 'D')
                brDezInc++;
        }
    }
    cout << brSvihInc << " " << brDezInc << endl;
    return 0;
}
```

Задатак: Магацин

Аутор: Иван Дрецун

У магацину једне продавнице постоји n гомила кутија поређаних у низ. У оквиру сваке гомиле су кутије наслагане једна на другу. Перица је добио посао да испремешта кутије тако да разлика у висинама највише и најниже гомиле буде што мања. Пошто су кутије велике, не може да носи више од једне истовремено. Перица те је замолио да напишеш програм који одређује колико је најмање кутија потребно преместити како би завршио посао.

Улаз: Са стандардног улаза се уноси број n ($1 \leq n \leq 100$). Након тога се уноси n бројева h_i ($1 \leq h_i \leq 10^9$) који представљају висину сваке гомиле.

Излаз: На стандардни излаз исписати најмањи потребан број премештања кутија.

Пример 1

Улаз

```
4
1 5 2 1
```

Излаз

```
2
```

Објашњење: Могуће је преместити једну кутију са друге гомиле на прву и још једну кутију са друге гомиле на последњу. Након премештања висине гомила су `2 3 2 2`.

Пример 2

Улаз

```
6
5 5 5 3 4 4
```

Излаз

```
1
```

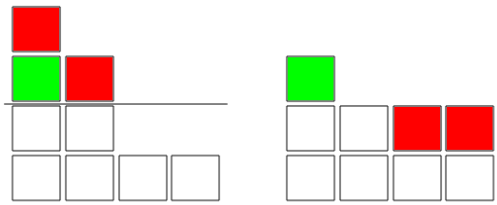
Објашњење: Могуће је преместити једну кутију са прве гомиле на четврту. Након премештања висине гомила су `4 5 5 4 4 4`.

Решење

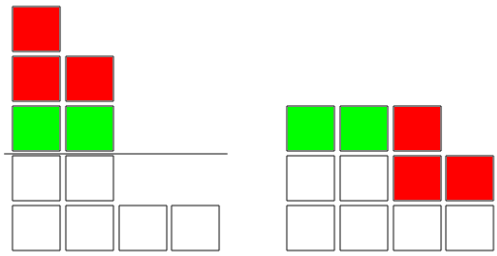
Приметимо да је увек могуће распоредити кутије тако да је разлика између највеће и најмање гомиле највише 1. Означимо са b укупан број кутија. Покушајмо да распоредимо кутије по гомилама тако да све гомиле буду исте величине p. Ако је то могуће, онда мора бити $p = \frac{b}{n}$. Ако није могуће, то значи да b није дељиво са n, и да при њиховом дељењу добијамо остатак $0 < r < n$, па можемо да одаберемо неких r гомила и на сваку од њих да ставимо по још једну кутију. На пример, уколико имамо укупно 13 кутија и 5 гомила, један могући распоред са минималном разликом је `3 3 3 2 2`. Дакле, након премештања свих кутија треба да имамо тачно r гомила висине $p + 1$, и тачно $n - r$ гомила висине p, где је $p = \lfloor \frac{b}{n} \rfloor$.

Одредимо сада минималан број премештања. Почетну вредност за број премештања можемо да израчунамо као укупан број кутија које се налазе на висини већој од p. На пример, ако је почетни распоред `4 3 1 1`, тада је $p = \lfloor \frac{4+3+1+1}{4} \rfloor = \lfloor 2,25 \rfloor = 2$, а број кутија изнад висине 2 је 3. Приметимо да од те 3 кутије једну

није потребно премештати зато што је остатак $r = 1$, па ће након премештања свакако остати једна кутија на висини $p + 1$.



Погледајмо још један пример. Ако је почетни распоред `5 4 1 1`, важи $p = 2$ и $r = 3$. Укупан број кутија изнад висине p је 5. Како имамо две гомиле чија је висина већа од p, а на крају ћемо имати три такве гомиле, од тих 5 кутија не морамо да премештамо две.



Нека је g број гомила на којима се налази строго више од p кутија, k број кутија које су постављене строго изнад висине p.

- Ако је $g \leq r$ тада ћемо на тих g гомила оставити по $p + 1$ кутија, док ћемо $k - g$ кутија прерасподелити тако да направимо $n - r$ гомила висине p и још $r - g$ кутија висине $p + 1$.
- Ако је $g > r$, тада ћемо преместити $k - r$ кутија тако да на r полазних гомила остане по $p + 1$ кутија, а да се све остале кутије преместе тако да остане још $n - r$ кутија висине p.

Дакле, број кутија које треба преместити је $k - \min(r, g)$.

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

using namespace std;

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

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

  int zbir = 0;
  for(int i = 0; i < n; i++)
    zbir += gomile[i];
  int prosek = zbir / n, ostatak = zbir % n;
  int brGomilaIznad = 0, brKutijaIznad = 0;
  for(int gomila : gomile) {
    if (gomila > prosek) {
      brGomilaIznad++;
      brKutijaIznad += gomila - prosek;
    }
  }

  int brPremestanja = brKutijaIznad - min(brGomilaIznad, ostatak);
  cout << brPremestanja << '\n';
```

```
  return 0;
}
```

Задатак: Коцке

Аутор: Душан Пођадић

Маша је, док је била мала, добила колекцију од 100 коцака које су обележене бројевима од 1 до 100 и обојене тако да су коцке са непарним бројевима розе, а са парним плаве. Током одрастања она је изгубила део колекције и сада јој је остало n коцака. Маша је, сређујући неке старе ствари, пронашла те коцке и поређала их у низ у растућем поретку. Након тога су дошли њени млађи сестра и брат Мина и Андреј и одлучили да јој измешају коцке тако што ће једно од њих изабрати две коцке у низу и обрнути редослед свих коцака између њих (рачунајући и две изабране коцке). Пошто Мина више воли розе боју, а Андреј плаву, Мина ће обртати само розе коцке игноришући плаве, а Андреј само плаве, игноришући розе. Свако од њих је овај поступак поновио више пута током игре. Након што су они отишли, вратила се Маша и изненађено приметила да су јој коцке измешане. Пошто више нема растући низ, одлучила је да замоли тебе да за њу пребројиш колико у овом измешаном низу коцака има узастопних растућих поднизова, као и да јој кажеш колико је дугачак најдужи од њих.

Улаз: У првом реду стандардног улаза се налази број n ($1 < n \leq 100$). У наредном реду се налази n бројева сортираних растуће који означавају бројеве на коцкама које је Маша на почетку поређала. У наредном реду се налази цео број q ($0 \leq q \leq 50$) који представља број премештања које су урадили Мина и Андреј. У наредних q редова се налази један карактер (`'m'` или `'a'`) који означава да ли је коцке померала Мина или Андреј, праћен са два броја l и d ($0 \leq l \leq d < n$) који означавају позицију леве и десне коцке у низу између којих се врши обртање (низ се индексира од 0).

Излаз: У првих q редова стандардног излаза исписати излгед низа након сваког премештања коцака. У наредном реду стандардног излаза исписати два броја, прво број узастопних растућих поднизова у промешаном низу, а затим дужину најдужег од њих.

Пример 1

Улаз

```
7
1 2 5 6 8 9 10
3
a 1 4
m 3 6
m 0 6
```

Излаз

```
1 8 5 6 2 9 10
1 8 5 6 2 9 10
9 8 5 6 2 1 10
5 2
```

Пример 2

Улаз

```
10
2 3 5 6 7 9 11 12 13 14
4
m 0 7
a 5 9
a 1 7
m 0 3
```

Излаз

```
2 11 9 6 7 5 3 12 13 14
2 11 9 6 7 5 3 14 13 12
2 11 9 14 7 5 3 6 13 12
```

```
2 9 11 14 7 5 3 6 13 12
5 4
```

Објашњење примера 2

индекс	0	1	2	3	4	5	6	7	8	9
низ	2	3	5	6	7	9	11	12	13	14

У првом премештању Мина обрће све непарне бројеве који су на индексима између 0 7, дакле бројеве 3, 5, 7, 9 и 11 и добија следећи низ:

индекс	0	1	2	3	4	5	6	7	8	9
низ	2	11	9	6	7	5	3	12	13	14

Затим Андреј обрће парне бројеве између индекса 5 и 9 и добија следећи низ:

индекс	0	1	2	3	4	5	6	7	8	9
низ	2	11	9	6	7	5	3	14	13	12

Затим Андреј обрће парне бројеве између индекса 1 и 7 и добија следећи низ:

индекс	0	1	2	3	4	5	6	7	8	9
низ	2	11	9	14	7	5	3	6	13	12

На крају Мина обрће непарне бројеве између индекса 0 и 3 и добија следећи низ:

индекс	0	1	2	3	4	5	6	7	8	9
низ	2	9	11	14	7	5	3	6	13	12

У добијеном низу постоје следећи узастопни растући поднизови: 2 9 11 14 | 7 | 5 | 3 6 13 | 12. Постоји 5 растућих узастопних поднизова од којих најдужи има дужину 4.

Решење

Овај задатак има два релативно независна дела која можемо посматрати одвојено. Први део је мешање парних или непарних делова низа, а други је касније пребројавање растућих поднизова узастопних бројева.

Обртање парних или непарних поднизова

Мешање коцака реализујемо у засебној функцији која поред низа коцака и граница дела низа који се обрће прима и цео број који одређује да ли је потребно обртати коцке са парним или непарним бројевима (тај број представља остатак при дељењу, па 0 означава парне, а 1 непарне). Када учитамо команду прво на основу првог карактера (слово `a` или `m`) одређујемо да ли радимо са парним или непарним бројевима.

Поставимо два показивача (два цела броја који су заправо индекси у низу) на задате граничне коцке. Након тога покрећемо петљу која се извршава док се показивачи не сретну. У свакој итерацији проверавамо да ли показивачи показују на бројеве задате парности, тј. да ли је остатак при дељењу броја на који показује сваки од показивача бројем 2 једнак p. Ако остатак није једнак жељеном тај показивач померамо ка другом показивачу и тако прескачемо (игноришемо) број погрешне парности. Када се деси случај да оба показивача показују на бројеве исправне парности, тада заменимо места тим бројевима у низу и настављамо даље тако што оба показивача приближимо један другом. Петља се завршава када се показивачи сретну или мимоиђу. На овај начин смо обезбедили да прескочимо све бројеве погрешне парности као и да мењамо најлевљи број исправне парности са најдешњим, други најлевљи са другим најдешњим и тако даље.

Пребројавање растућих поднизова

У овом делу задатка користимо бројач `tr` који памти колико до сада имамо чланова у тренутном растућем поднизу. Пролазимо кроз низ редом и докле год је тренутни елемент већи од претходног увећавамо `tr` за 1. Када тренутни елемент више није већи од претходног тиме се отпочиње нови низ. Проверавамо да ли је тренутна вредност `tr` већа од до сада највеће запамћене и ако јесте памтимо је. Такође, број растућих поднизова увећавамо за 1 и `tr` враћамо на 1 (јер тај нови подниз има једног члана - овог који тренутно посматрамо). Треба водити рачуна да уколико стигнемо до краја низа, морамо такође да проверимо да ли је тренутни (последњи) подниз већи од највећег до сада и ако јесте да га запамтимо. На крају само исписујемо колико има растућих поднизова узастопних елемената и дужину најдужег од њих.

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

using namespace std;

void ispis(int kocke[], int n) {
  for(int i = 0; i < n; i++)
    cout << kocke[i] << " ";
  cout << endl;
}

void izmesaj(int ost, int kocke[], int l, int d) {
  int i = l, j = d;
  while (i < j) {
    if (kocke[i] % 2 == ost)
      i++;
    else if (kocke[j] % 2 == ost)
      j--;
    else {
      swap(kocke[i], kocke[j]);
      i++, j--;
    }
  }
}

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

  int kocke[100];
  for(int i = 0; i < n; i++)
    cin >> kocke[i];

  int q;
  cin >> q;

  for (int i = 0; i < q; i++) {
    char c;
    cin >> c;
    int l, d;
    cin >> l >> d;
    izmesaj(c == 'a', kocke, l, d);
    ispis(kocke, n);
  }

  int tr = 1; // duzina tekuce rastuce serije
  int mx = 1; // duzina najduze rastuce serije
  int br = 1; // broj rastucih serija
```

```
  for (int i = 1; i < n; i++) {
    if (kocke[i] > kocke[i-1]) {
      tr++;
      if (tr > mx)
        mx = tr;
    } else {
      tr = 1;
      br++;
    }
  }

  cout << br << " " << mx << endl;

  return 0;
}
```

Задатак: Број посета кућици

Аутор: Филип Марић

Лик се у игрици креће по неограниченој мрежи квадратних поља. Свако поље је одређено својим координатама (координата у расте на горе) а играч креће са поља $(0, 0)$, које се сматра његовом "кућицом". Лик се премешта на суседно поље када год играч притисне неку од четири стрелице (означимо их са `<` за лево, `>` за десно, `^` за горе и `v` за доле). Написати програм који одређује колико се пута током игре играч налази у својој кућици.

Улаз: Са стандардног улаза се учитава ниска састављена од карактера `<`, `>`, `^` и `v` (њих највише 10^6).

Излаз: На стандардни излаз исписати тражени број посета пољу $(0, 0)$.

Пример 1

Улаз

```
<>^v><v^>^
```

Излаз

```
5
```

Објашњење

Играч редом обилази поља $(0, 0)$, $(-1, 0)$, $(0, 0)$, $(0, 1)$, $(0, 0)$, $(1, 0)$, $(0, 0)$, $(0, -1)$, $(0, 0)$, $(1, 0)$, $(1, 1)$, што значи да се на пољу $(0, 0)$ налази 5 пута.

Пример 2

Улаз

```
>>>^^^<<<vvv>>>^^^<<<vvv
```

Излаз

```
3
```

Решење

Симулираћемо кретање лика тако што ћемо у сваком тренутку пратити његове координате на табли. Обрађиваћемо један по један карактер и у складу са тим који смер кретања карактер описује, мењаћемо тренутну позицију (за карактер `<` смањујемо координату x за 1, за карактер `>` повећавамо координату x за 1, за карактер `^` повећавамо координату y за 1, а за карактер `v` смањујемо координату y за 1). Када одредимо нове координате проверавамо да ли су обе једнаке нули и ако јесу, тада увећавамо бројач посета кућици (пошто се на почетку лик налази на пољу $(0, 0)$ тј. у кућици, тај бројач треба иницијлизовати на 1).

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

using namespace std;
```

```
int main() {
  int x = 0, y = 0;
  int brojPoseta = 1;
  char c;
  while ((cin >> c) && c != '\n') {
    switch(c) {
    case '<':
      x--;
      break;
    case '>':
      x++;
      break;
    case '^':
      y++;
      break;
    case 'v':
      y--;
      break;
    }
    if (x == 0 && y == 0)
      brojPoseta++;
  }
  cout << brojPoseta << endl;
  return 0;
}
```

Задатак: Телеграф

Аутори: Душан Пођадић, Милица Мићић

У доба пре интернета и телефона, за брзи пренос порука често се користио телеграф. Код телеграфа је специфично што се сваки знак (слово, цифра…) шаље посебно. Маре и Паја су старомодни и за своју комуникацију још увек користе телеграф, али да би били сигурни да их нико не прислушкује развили су шифру за међусобно препознавање. Препознавање се одвија у 3 корака:

1. Паја замисли један природан број (A) и пошаље га Марету цифру по цифру.
2. Маре одреди просек цифара броја A и направи нови број који се добија тако што се из Пајиног броја избаце цифре које су строго мање од израчунатог просека. Након тога Маре помножи добијени број унапред договореним природним бројем k и резултат (број B) шаље Паји.
3. Паја формира нови број (C) тако што пролази кроз цифре броја B и наизменично их сабира и одузима, почевши од последње цифре (од последње цифре одузме претпоследњу, затим на њу дода ону испред претпоследње, па ону испред те опет одузима и тако наизменично сабира и одузима док не потроши све цифре). Тако добијен број поново шаље Марету и након тога настављају разговор нормално.

Улаз: У првом реду налазе се два броја n ($1 \leq n \leq 8$), број цифара броја A, и k ($1 \leq k \leq 20$), број којим Маре множи свој број. У наредних n редова се налазе цифре броја A.

Излаз: Исписати број B који је Маре послао Паји, а након тога у следећем реду број C који је Паја на крају послао Марету.

Пример 1

Улаз

```
6 3
2
4
3
7
8
```

```
1
```

Излаз

```
234
3
```

Објашњење

Пајин број има 6 цифара и то је 243781. Просек цифара тог броја је (2+4+3+7+8+1)/6 = 4,17. Након избацивања свих цифара које су мање од просека добија се 78, који Маре множи са k и добија $78 \cdot 3 = 234$. Паја на основу броја 234 добија свој нови број 4-3+2=3. Дакле, исписује се прво 234, а затим 3.

Пример 2

Улаз

```
3 6
1
6
5
```

Излаз

```
390
-6
```

Решење

Учитавамо цифре полазног броја A и смештамо их у низ цифара (почевши од цифре највеће тежине). Просек цифара је једнак количнику збира свих цифара са бројем цифара, па зато израчунавамо збир свих цифара. Након израчунавања просека филтрирамо цифре задржавајући само оне које су веће или једнаке од просека и од њих Хорнеровом схемом формирамо број који множимо са k и тако добијамо вредност B коју исписујемо. Формирање броја Хорнеровом схемом се врши тако што при сваком уносу цифре досадашњи број помножимо са 10 и додамо нову цифру као последњу, при чему се цифре обрађују слева надесно (од цифре највеће тежине).

Да бисмо добили број C који Паја враћа Марету потребно је да уклањамо једну по једну цифру са десног краја броја B, док нам не остане 0. Издвајамо једну по једну цифру добијеног броја, кренувши од цифре најмање тежине и од тих цифара формирамо тражен збир, водећи рачуна о томе да цифре на непарним позицијама одузимамо, а на парним додајемо на збир.

```
#include <iostream>

using namespace std;

int main() {
  // broj cifara i koeficijent kojim se množi
  int n, k;
  cin >> n >> k;

  // ucitavamo cifre broja A
  int cifreA[8];
  for (int i = 0; i < n; i++)
    cin >> cifreA[i];

  // izracunavamo prosek cifara broja A
  int zbir = 0;
  for (int i = 0; i < n; i++)
    zbir += cifreA[i];
  double prosek = (double)zbir / (double)n;

  // formiramo broj B od cifara koje su veće ili jednake od proseka
  long long B = 0;
  for (int i = 0; i < n; i++)
```

```
    if (cifreA[i] >= prosek)
      B = B * 10 + cifreA[i];
  // množimo ga sa k i ispisujemo dobijeni rezultat
  B *= k;
  cout << B << endl;

  // izdvajamo cifre broja B zdesna na levo i dodajemo ih ili
  // oduzimamo ih od rezultata u zavisnosti od parnosti tekuće
  // pozicije cifre
  bool parnaPozicija = true;
  int C = 0;
  while (B > 0) {
    int cifra = B % 10;
    if (parnaPozicija)
      C += cifra;
    else
      C -= cifra;
    B /= 10;
    parnaPozicija = !parnaPozicija;
  }
  cout << C << endl;

  return 0;
}
```

Задатак: Скочко

Аутор: Филип Марић

У игри "Скочко" играч погађа непознату комбинацију S, која се састоји од n симбола (једноставности ради, претпоставимо да су то цифре од 1 до 9). Након што играч напише свој низ P од n симбола, рачунар треба да му каже колико постоји симбола у непознатој комбинацији S, који се поклапају са симболима у низу P. При том се посебно броје симболи који су на истој позицији у оба низа, а посебно симболи из непознате комбинације који се налазе у низу P, али нису на истој позицији.

Улаз: Са стандардног улаза се уносе два низа симбола дужине n ($1 \leq n \leq 15$). Први је непозната комбинација S, а други је низ P, који је унео играч покушавајући да погоди непознату комбинацију.

Излаз: На стандардни излаз исписати број погођених симбола који су на одговарајућим позицијама и број погођених симбола који нису на одговарајућим позицијама (два броја раздвојена размаком).

Пример 1

Улаз

```
12345
54321
```

Излаз

```
1 4
```

Пример 2

Улаз

```
112233
112233
```

Излаз

```
6 0
```

Пример 3

Улаз

```
112233
132123
```

Излаз

```
3 3
```

Пример 4

Улаз

```
111111
122222
```

Излаз

```
1 0
```

Пример 5

Улаз

```
111122
222233
```

Излаз

```
0 2
```

Пример 6

Улаз

```
1222
2333
```

Излаз

```
0 1
```

Решење

Решење заснивамо на бројању појављивања сваке цифре. Учитавамо две ниске карактера, и затим бројимо колико пута се свака од 10 цифара јавила у првој и у другој ниски. Након тога одређујемо колико се цифара јавља на одговарајућим позицијама, тако што обрађујемо редом једну по једну позицију у нискама и поредимо карактере на тој позицији. Пошто те цифре желимо да изузмемо приликом одређивања цифара који се поклапају, али нису на одговарајућим позицијама, умањиваћемо број појављивања пронађених поклапајућих карактера. Број цифара које се поклапају, али нису на правом месту ћемо одредити тако што ћемо за сваку цифру пронаћи минимум броја појављивања у обе ниске (након уклањања појављивања на одговарајућим позицијама) и сабраћемо тако добијене минимуме за све цифре.

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

using namespace std;

int main() {
  string resenje, pogodak;
  cin >> resenje >> pogodak;

  // koliko puta se svaka cifra javlja u resenju i u pogotku
  int brojResenje[10] = {0};
  for (char c : resenje)
    brojResenje[c - '0']++;

  int brojPogodak[10] = {0};
  for (char c : pogodak)
    brojPogodak[c - '0']++;
```

```
  // koliko je cifara na pravom mestu (njih izuzimamo iz brojanja
  // pogodjenih cifara koje su van pravog mesta)
  int naPravomMestu = 0;
  for (int i = 0; i < resenje.size(); i++)
    if (resenje[i] == pogodak[i]) {
      naPravomMestu++;
      brojPogodak[resenje[i] - '0']--;
      brojResenje[resenje[i] - '0']--;
    }

  // racunamo broj cifara van pravog mesta (cifre koje su na pravom
  // mestu su vec izuzete)
  int vanPravogMesta = 0;
  for (int i = 0; i < 10; i++)
    vanPravogMesta += min(brojResenje[i], brojPogodak[i]);

  cout << naPravomMestu << " " << vanPravogMesta << endl;
  return 0;
}
```

Задатак: Подморница

Аутори: Иван Дрецун, Душан Пођадић

Подморница се састоји из n сегмената исте ширине и различитих висина, при чему је дно подморнице равно. Притисак извршен на један сегмент се рачуна као удаљеност између врха тог сегмента и површине воде, ако је врх тог сегмента испод површине, а иначе је нула. Укупан притисак на подморницу се рачуна као збир притиска извршеног на сваки сегмент. Познато је да помдорница не може да издржи притисак већи од p. Написати програм који одређује максималну дубину до које подморница може да иде. Дубина подморнице се рачуна као удаљеност између дна подморнице и површине воде.

Улаз: У првом реду стандардног улаза налазе се два цела броја, n ($1 \leq n \leq 100000$) и p ($0 \leq p \leq 10^9$). У другом реду је n целих бројева h_i ($1 \leq h_i \leq 10^9$) који представљају висине сегмената подморнице.

Излаз: На стандардни излаз исписати један цео број који представља максималну дубину до које подморница може да иде.

Пример

Улаз

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

Излаз

```
3
```

Објашњење:

На слици је приказана подморница на дубини 3. Испод сваког сегмента написан је притисак који је на њега извршен.

```
____x_
..xxx.
x.xxx.
xxxxxx
120002
```

Решење

Инкрементално потапање једног по једног сегмента

У задатку је потребно одредити максималну дубину коју подморница може да достигне, а да притисак на њу не буде преко дозвољене границе. Некада је решење такво да је цела подморница потопљена, а некада да је потопљен само један њен део. У сваком случају је пожељно да сортирамо низ висина сегмената и да онда извршимо анализу. Идеја је да пређемо кроз низ сегмената и покушамо да потопимо подморницу на дубину тако да је врх посматраног семгента тачно на површини и успут рачунамо притисак који је извршен на тако потопљену подморницу.

Обрађујемо један по један сегмент покушавајући да га у потпуности потопимо. Одржавамо текућу дубину подморнице d и текући притисак p на тој дубини. Ако је подморница након потапања i сегмената била на дубини d, тада се би се потпуним потапањем сегмента на позицији i висине h_i дубина повећала за $\Delta d = h_i - d$, док се притисак повећао за $\Delta p = i \cdot \Delta d$ и добија би се нови притисак $p' = p + \Delta p = p + i \cdot \Delta d$ (приметимо да нови притисак израчунавамо инкрементално на основу старог, без потребе за анализом појединачних сегмената).

Ако добијени притисак p' не прелази границу p_{max}, текућу дубину ажурирамо на h_i, текући притисак ажурирамо на p' и прелазимо на наредни сегмент.

Ако добијени притисак прелази границу p_{max}, тада није могуће у потпуности потопити сегмент висине h_i. Зато одређујемо највеће могуће повећање дубине Δd такво да је нови притисак $p' = p + i \cdot \Delta d$ и даље одржив тј. да је $p' \leq p_{max}$. Зато тражимо максимално Δd такво да је $\Delta d \leq \frac{p'-p}{i}$, а то је $\Delta d = \left\lfloor \frac{p'-p}{i} \right\rfloor$. Висину увећавамо за тако одређену разлику Δd, а притисак за $\Delta p = i \cdot \Delta d$. Пошто се повећањем дубине макар за 1 прелази граница притиска, овим је одређена највећа могућа дубина, па се петља која потапа један по један сегмент може прекинути.

Анализа сложености. Сложеношћу решења доминира иницијално сортирање $O(n \log n)$, док се касније потапање једног по једног сегмента врши у сложености $O(n)$. Ако би се притисак сваки пут рачунао изнова, уместо инкрементално, добило би се неефикасно решење сложености $O(n^2)$.

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

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0);

  int n;
  int64_t maxPritisak;
  cin >> n >> maxPritisak;
  vector<int64_t> h(n);
  for(int i = 0; i < n; i++)
    cin >> h[i];

  sort(begin(h), end(h));

  // tekuca dubina i pritisak na toj dubini
  int64_t dubina = 0, pritisak = 0;
  for (int i = 0; i < n; i++) {
    // ako segment visine h[i] jos nije ceo potopljen
    if (h[i] > dubina) {
      // pritisak koji bi nastalo potpunim potapanjem segmenta visine h[i]
      int64_t noviPritisak = pritisak + i * (h[i] - dubina);
      if (noviPritisak > maxPritisak) {
        // segment nije moguce potpuno potopiti pa racunamo koliko se
        // najvise moze potopiti
        int64_t deltaDubina = (maxPritisak - pritisak) / i;
        dubina += deltaDubina;
```

```
        pritisak += deltaDubina * i;
        break;
      }
      // potapamo ceo segment visine h[i]
      dubina = h[i];
      pritisak = noviPritisak;
    }
  }

  // ako je cela podmornica potopljena tako da joj je maksimalni
  // segment tik ispod povrsine, racunamo koliko se jos moze potopiti
  // dok se ne dostigne maksimalni pritisak
  dubina += (maxPritisak - pritisak) / n;

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

Бинарна претрага

За дату дубину d можемо једноставно израчунати притисак и видети да ли он превазилази задату границу. Повећањем дубине се повећава и притисак, па је за одређивање максималне дубине могуће применити бинарну претрагу (тзв. бинарну претрагу по решењу). Тражимо највећу вредност дубине за коју је притисак мањи од дате границе. Ако је дубина 0, притисак је 0, па је услов сигурно испуњен и вредност 0 можемо узети за иницијалну леву границу бинарне претраге. Означимо са h_{max} максималну висину сегмента подморнице. Ако је подморница на дубини d њен притисак је сигурно већи или једнак од $n \cdot (d - h_{max})$. Зато је притисак сигурно изнад границе ако је $(d - h_{max}) \cdot n > p_{max}$ тј. ако је $d > h_{max} + \frac{p_{max}}{n}$. Стога иницијалну вредност десне границе у бинарној претрази можемо поставити на вредност $h_{max} + \lfloor \frac{p_{max}}{n} \rfloor + 1$.

Бинарном претрагом тражимо највећу дозвољену дубину на следећи начин: проверићемо дубину која је на средини између леве и десне границе. Уколико је могуће спустити подморницу на ту дубину, померамо леву границу претраге на нађену дубину + 1. Уколико није могуће спустити подморницу на ту дубину, онда померамо десну границу на дубина - 1. Након овога понављамо поступак док се лева и десна граница не мимоиђу.

Остаје нам још само да за задату дубину проверимо да ли је могуће подморницу спустити дотле. То можемо урадити тако што прођемо кроз низ висина свих сегмената и рачунамо притисак на сваки сегмент појединачно и на крају одредимо укупан притисак као збир. Уколико је укупан притисак мањи или једнак највећем дозвољеном притиску, онда је могуће спустити подморницу на ту дубину, иначе није могуће.

Анализа сложености. Пошто се за фиксирану вредност дубине провера притиска може извршити у времену $O(n)$, укупна сложеност је $O(n \log (\frac{p_{max}}{n} + h_{max}))$. Како ни p_{max} ни h_{max} не могу бити већи од 10^9, тај логаритам је највише око 30, а величина низа је највише 10^5, па ово решење успешно завршава посао у оквиру задатог временског ограничења.

```
#include <iostream>
#include <vector>
#include <limits>
#include <cmath>
#include <algorithm>
using namespace std;

// provera da li se podmornica moze potopiti do date dubine tako da izdrzi dati
// maksimalni pritisak
bool pritisakOK(const vector<int>& h, int dubina, int pMax) {
  long long pritisak = 0;
  for (int hh : h){
    pritisak += max(dubina - hh, 0);
    if (pritisak > pMax)
      return false;
  }
```

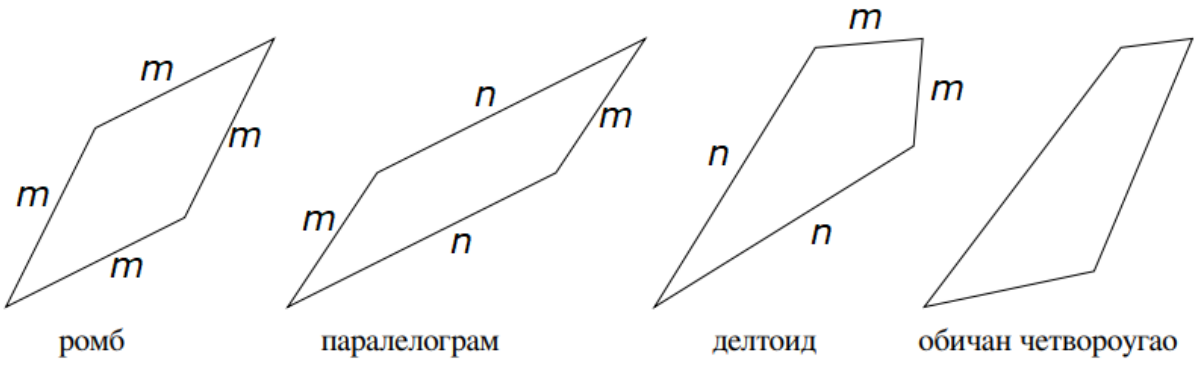
```
  return true;
}

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0);

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

  // binarna pretraga optimalnog resenja
  int lg = 0;
  int dg = *max_element(h.begin(), h.end()) + pMax / n + 1;
  while (lg <= dg) {
    int dubina = lg + (dg - lg) / 2;
    if (pritisakOK(h, dubina, pMax))
      lg = dubina + 1;
    else
      dg = dubina - 1;
  }
  cout << lg - 1 << endl;
  return 0;
}
```

Задатак: Четвороугао

Аутор: Милан Вуīелија

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

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



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

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

Пример 1

Улаз

```
9
9
```

```
9
9
```

Излаз

```
ROMB
```

Пример 2

Улаз

```
5
6
6
7
```

Излаз

```
OBICAN
```

Решење

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

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

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

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

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

```
#include <iostream>

using namespace std;

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

   return 0;
}
```

Задатак: Играчка

Аутор: Небојша Варница

Играчка је облика квадрата странице 3 поља (3 реда и 3 колоне). У 8 од 9 поља је плочица са једним од бројева 1, 2, 3, …, 8 а преостало девето поље је празно. Потез се састоји у померању једне плочице у суседно лево, десно, доње или горње празно поље. Написати програм који за дато стање играчке одиграва један потез.

Улаз: Уноси се тренутно стање играчке (прва 3 реда) и потез који покушавамо да одиграмо (четврти ред).

Тренутно стање играчке се уноси тако што се у три реда уносе бројеви 1, 2, …, 8 (сваки тачно по једном) а на месту празног поља уноси се знак *. На пример:

```
245
*38
167
```

Потез чије се одигравање покушава може бити:

1 - плочица са бројем се помера у празно суседно поље са леве стране

2 - плочица са бројем се помера у празно суседно поље са десне стране

3 - плочица са бројем се помера у празно суседно поље са горње стране

4 - плочица са бројем се помера у празно суседно поље са доње стране

Излаз: Исписује се стање играчке након одиграног потеза. Испис треба да буде у облику у коме је унето почетно стање.

Ако потез није могуће одиграти исписује се учитано стање играчке.

Пример

Улаз

```
245
*38
167
1
```

Излаз

```
245
3*8
167
```

Решење

Опис главног решења

Сваки ред играчке је један стринг. Померања лево и десно се своде на замену места унутар једне врсте а померања горе и доле на замену знакова између суседних редова.

```
#include <iostream>

using namespace std;

int main()
{
  // polje je predstavljeno pomocu tri niske
  string a[3];
  cin >> a[0] >> a[1] >> a[2];

  // pronalazimo koordinate zvezdice
  int pv = 0, pk = 0;
  for (int i = 0; i < 3; i++) {
    int j;
    if ((j = a[i].find("*")) != string::npos) {
      pv = i;
      pk = j;
      break;
    }
  }

  int op;
  cin >> op;
  switch (op) {
  case 1: // levo
    if (pk<2) {
      a[pv][pk] = a[pv][pk+1];
      a[pv][pk+1] = '*';
```

```
    }
    break;
  case 2: // desno
    if (pk>0) {
      a[pv][pk] = a[pv][pk-1];
      a[pv][pk-1] = '*';
    }
    break;
  case 3: // gore
    if (pv<2) {
      a[pv][pk] = a[pv+1][pk];
      a[pv+1][pk] = '*';
    }
    break;
  case 4: // dole
    if (pv>0) {
      a[pv][pk] = a[pv-1][pk];
      a[pv-1][pk] = '*';
    }
  }

  // ispisujemo polje nakon pomeranja
  cout << a[0] << endl;
  cout << a[1] << endl;
  cout << a[2] << endl;
  return 0;
}
```

Задатак: Множење и кореновање

Аутор: Филип Марић

Природни бројеви се могу трансформисати коришћењем следеће две операције:

- множење било којим другим природним бројем (може се применити увек)
- кореновање (може се применити само ако број потпун квадрат тј. ако је његов корен поново природан број).

Напиши програм који за дати број n одређује најмањи број који се може добити применом ове две операције.

Улаз: Са стандардног улаза се учитава природан број n ($1 \leq n \leq 10^12$).

Излаз: На стандардни излаз исписати тражени најмањи број.

Пример 1

Улаз

```
20
```

Излаз

```
10
```

Објашњење

Број 20 се може помножити бројем 5, а затим се може кореновати и тако добити број 10. Ниједан број мањи од 10 није могуће добити.

Пример 2

Улаз

```
540
```

Излаз

```
30
```

Објашњење

Број 540 се може помножити бројем 1500, а затим се може два пута кореновати и тако добити број 30. Ниједан број мањи од 30 није могуће добити.

Решење

Растављање на просте чиниоце

Кључ решења задатака је да се на основу основне теореме аритметике број представи као производ простих чинилаца $n = p_1^{k_1} \cdot \ldots \cdot p_m^{k_m}$.

Након множења неким бројем сви ови чиниоци остају присутни и у производу (а могу се евентуално појавити неки нови). Број се може кореновати ако и само ако су сви експоненти k_1 до k_m парни и тада се они смањују на пола, али ни при тој операцији ниједан прост чинилац p_i не може нестати из резултата кореновања.

Дакле, применом ових операција увек се добија резултат који садржи све просте чиниоце p_1 до p_m.

Са друге стране, применом операција је могуће добити број $p_1 \cdot \ldots \cdot p_m$. Заиста, нека је k најмањи степен двојке који је већи или једнак од свих експонената k_1 до k_m. Множењем полазног броја бројем $p_1^{k-k_1} \cdot \ldots \cdot p_m^{k-k_m}$ добијамо број $p_1^k \cdot \ldots \cdot p_m^k$. Пошто је k степен броја 2 коренивањем тог броја се експоненти смањују на пола, све док не стигну до 1.

Дакле, минимални број који се може добити је производ различитих простих фактора броја n и он се може одредити уобичајеним алгоритмом факторизације броја.

```
#include <iostream>

using namespace std;

typedef unsigned long long ull;

int main() {
  ull n;
  cin >> n;
  ull rez = 1;
  ull d = 2;
  while (d * d <= n) {
    if (n % d == 0) {
      rez *= d;
      while (n % d == 0)
        n /= d;
    }
    d++;
  }
  if (n > 1)
    rez *= n;
  cout << rez << endl;
  return 0;
}
```

Задатак: Сечење

Аутори: Иван Дрецун, Душан Пођадић

Дат је штап дужине n метара. Потребно је пресећи га на неколико места тако да добијени делови буду целобројне дужине и ниједан од добијених делова не буде дужи од k метара. Написати програм који одређује на колико начина је могуће извршити овакво сечење.

Улаз: Са стандардног улаза се уносе бројеви n ($1 \leq n \leq 1000$) и k ($1 \leq k \leq 300$).

Излаз: На стандардни излаз исписати број могућих сечења. Како тај број може бити велик, исписати његов остатак при дељењу са $10^9 + 9$.

Пример

Улаз

```
4 2
```

Излаз

```
5
```

Објашњење: Могућа су следећа сечења представљена дужинама добијених делова штапа:

```
1 1 1 1
2 1 1
1 2 1
1 1 2
2 2
```

Решење

Основно решење

Обележимо број могућих сечења штапа дужине d са $broj(d)$. Штап дужине нула се може исећи на један начин. Посматрајмо штап дужине $d > 0$. Ако фиксирамо дужину последњег исеченог дела на 1, остатак штапа је дужине $d - 1$ и њега можемо исећи на $broj(d - 1)$ начина. Наравно, дужина последњег дела не мора бити 1. Ако фиксирамо дужину последњег исеченог дела на 2, остатак штапа је дужине $d - 2$ и њега можемо исећи на $broj(d - 2)$ начина. Слично можемо фиксирати дужину последњег дела на 3, на 4 итд. Дужина последњег дела може бити највише k по условима задатка, па је тада дужина остатка штапа $d - k$, а број могућности за сечење је $broj(d - k)$. Укупан број начина да исечемо штап дужине d је збир броја могућности када је последњи део 1, броја могућности када је последњи део 2 и тако даље. Наравно, уколико је дужина d мања од k, онда се сабирање не врши до k, него само до d. Дакле можемо записати

$$broj(d) = \sum_{i=1}^{\min(k,d)} broj(d - i).$$

Директно рекурзивно решење на основу ове рекурентне формуле би било веома неефикасно, због понављања рекурзивних позива, те стога примењујемо технику динамичког програмирања. У низу $broj$ ћемо чувати број сечења за сваку могућу дужину штапа. Потребно проћи кроз све вредности дужина штапа од 1 до n и на основу претходно добијених резултата које чувамо у низу лако можемо израчунати број могућности за тренутну дужину.

Треба водити рачуна да се при сваком сабирању и множењу бројева, узима остатак при дељењу бројем 10^9+9 као што је сугерисано у задатку, да не би дошло до прекорачења.

Анализа сложености. Сложеност овог алгоритма је $O(n \cdot k)$ јер за сваку од дужина треба проћи кроз k (или мање за првих k дужина) претходних резултата и сабрати их. Како важи $k \leq n \leq 10^6$, ово решење се не може извршити у датим временским ограничењима (1,5 секунди) на свим тест-примерима.

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

using namespace std;

constexpr int mod = 1000000009;

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

```
  vector<int> broj(n + 1);
  broj[0] = 1;
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= k && j <= i; j++)
      broj[i] = (broj[i] + broj[i - j]) % mod;

  cout << broj[n] << '\n';
  return 0;
}
```

Оптимизовано решење

Претходно решење можемо побољшати тако што користимо инкрементално рачунање збира претходних k елемената. Када рачунамо решење за $broj(d)$, ми пролазимо кроз све чланове низа од $broj(d-k)$ до $broj(d-1)$. Када рачунамо решење за $broj(d+1)$, пролазимо кроз све чланове од $broj(d-k+1)$ до $broj(d)$, односно тако израчунат збир се разликује од збира који смо рачунали за $broj(d)$ само у члановима $broj(d-k)$ и $broj(d)$. Ако ово уочимо, јасно нам је да не морамо сваки пут да пролазимо кроз свих k последњих чланова низа, већ можмо да памтимо претходни збир и да је једноставно мењамо у сваком кораку. Наравно, опет треба водити рачуна за дужине штапа које су мање од k.

Анализа сложености. Сложеност овог решења је $O(n)$ и оно доноси максималан број поена.

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

using namespace std;

const int mod = 1000000009;

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

  vector<int> broj(n + 1);
  broj[0] = 1;
  int zbirPoslednjihK = 1;
  for (int i = 1; i <= n; i++) {
    broj[i] = zbirPoslednjihK;
    zbirPoslednjihK = (zbirPoslednjihK + broj[i]) % mod;
    if (i >= k)
      zbirPoslednjihK = (zbirPoslednjihK + mod - broj[i - k]) % mod;
  }

  cout << broj[n] << '\n';
  return 0;
}
```

Задатак: Прозор и ограда

Аутор: Милан Вуīелија

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

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

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

Пример

Улаз

```
10 20
5 15
12 17
```

Излаз

```
2
```

Решење

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

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

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

```
#include <iostream>

using namespace std;

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

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

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

Задатак: Чинилац мање

Аутор: Милан Вуīделија

Написати програм, који за дати низ целих бројева одређује који број треба изоставити из низа, тако да производ осталих буде што већи. Ако има више решења наћи најмањи број који представља решење.

Улаз: У првом реду стандардног улаза број N, $2 \leq N \leq 50000$. У другом реду N целих бројева раздвојених размацима. Сваки од ових N бројева је из интервала $[-10^9, 10^9]$.

Излаз: На стандардни излаз исписати само један цео број, најмањи број чијим изостављањем се добија највећи производ.

Пример 1

Улаз

```
6
0 -4 7 0 -4 0
```

Излаз

```
-4
```

Пример 2

Улаз

```
5
-5 9 -4 -7 3
```

Излаз

```
-4
```

Решење

Ако постоје две или више нула у низу, производ не зависи од елемента који избацујемо, тако да можемо да избацимо најмањи.

Ако се у низу налази само једна нула, треба проверити да ли се избацивањем те нуле производ смањује или повећава. Уколико се у низу налази паран број негативних елемената, треба избацити нулу, јер тиме производ постаје позитиван (док избацивањем било којег другог елемента производ остаје нула). У случају да се у низу налази непаран број негативних елемената, треба избацити најмањи елемент, јер би се избацивањем нуле производ смањио, док у свим осталим случајевима производ остаје нула, па је од тих осталих случајева најбоље да се избаци најмањи елемент.

Остао је случај када у низу нема нула. Поново треба размотрити подслучајеве са парним и непарним бројем негативних елемената, јер од тога зависи знак производа.

У случају да се у низу налази паран број негативних елемената, производ свих елемената је позитиван. Уколико је могуће, треба избацити позитиван број да би производ остао позитиван. Дакле, ако постоје позитивни бројеви у низу, избацујемо најмањи позитиван број, а ако не постоје, производ ће бити негативан па треба избацити најмањи (највећи по апсолутној вредности) негативан број.

У случају да се у низу налази непаран број негативних елемената, производ свих елемената је негативан. Уколико је могуће, треба избацити негативан број да би производ постао позитиван. Дакле, ако постоје негативни бројеви у низу, избацујемо највећи (најмањи по апсолутној вредности) негативан број, а ако не постоје, производ ће бити негативан па треба избацити највећи позитиван број.

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

using namespace std;

const int POZ_BESK = numeric_limits<int>::max();
const int NEG_BESK = numeric_limits<int>::min();

int main() {
    int n, a, brNula = 0, brNeg = 0;
    int minn = POZ_BESK, minNeNula = POZ_BESK, maxPoz = NEG_BESK,
        maxNeg = NEG_BESK, minPoz = POZ_BESK, minNeg = POZ_BESK;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> a;
        minn = min(minn, a);
        if (a == 0) brNula++;
        else minNeNula = min(minNeNula, a);
```

```
        if (a < 0) { brNeg++; maxNeg = max(maxNeg, a); minNeg = min(minNeg, a); }
        if (a > 0) { minPoz = min(minPoz, a); maxPoz = max(maxPoz, a);}
    }

    if (brNula > 0) {
        if (brNula == 1 && brNeg % 2 == 0) cout << 0 << endl;
        else cout << minn << endl;
    }
    else if (brNeg % 2 == 0) {
        if (minPoz != POZ_BESK) cout << minPoz << endl;
        else cout << minNeg << endl;
    }
    else if (maxNeg != NEG_BESK) cout << maxNeg << endl;
    else cout << maxPoz << endl;

    return 0;
}
```

Задатак: Експеримент

Аутор: Иван Дрецун

Професор Прока је коначно направио времеплов! Сад је преостало само да га активира и отпутује t година у будућност. Времеплов се састоји од n гомила истоветних металних куглица поређаних у низ, сваке две суседне гомиле на размаку по 1 метар. За покретање времеплова потребно је поставити по један магнет у тачно две гомиле у том низу. Свака куглица која се налази између та два магнета ће одлетети до ближег и за њега се залепити. Времеплов ће професора одвести онолико година у будућност колики је укупан број метара који су све куглице прешле у збиру. Професор жели да изврши експеримент како би одредио где је потребно поставити магнете тако да отпутује што више година у будућност, али никако више од t година. Напиши програм који помаже професору да одреди где је потребно поставити магнете.

Улаз: Са стандардног улаза се уносе цели бројеви t ($1 \leq t \leq 10^{18}$) и n ($2 \leq n \leq 10^5$). Након тога се уноси n вредности s_i ($1 \leq s_i \leq 10^6$) које представљају број куглица у гомили i.

Излаз: У првом реду стандардног улаза исписати цео број који представља колико највише година у будућност професор може да отпутује. У наредном реду исписати две вредности i и $(0 \leq i < j < n)$ одвојене размаком, које представљају редне бројеве гомила где је потребно поставити магнете. Уколико постоји више могућих решења, исписати оно где је i најмање.

Пример

Улаз

```
16 6
2 1 3 4 3 2
```

Излаз

```
14
1 5
```

Објашњење: Укупна удаљеност коју куглице прелазе је $0 \times 1 + 1 \times 3 + 2 \times 4 + 1 \times 3 + 0 \times 2$.

Решење

Означимо са $R_{x,y}$ укупно растојање које пређу куглице када су магнети на позицијама x и y.

Груба сила

Решење грубом силом подразумева да се испробају све могуће позиције магнета (оне се једноставно могу набројати унгежђеним петљама). За сваки пар позиција $0 \leq i < j < n$ изнова израчунавамо $R_{i,j}$ и ако је оно мање од задате границе, ажурирамо максимум ако је то потребно.

Анализа сложености. Сложеност овог решења је $O(n^3)$ и оно је недовољно ефикасно.

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

using namespace std;

int main() {
  int64_t t;
  cin >> t;

  int n;
  cin >> n;

  vector<int64_t> a(n);
  for(int i = 0; i < n; i++)
    cin >> a[i];

  int maxi = 0, maxj = 1;
  int64_t maxZbir = 0;
  for(int i = 0; i < n; i++) {
    for(int j = i + 1; j < n; j++) {
      int64_t zbir = 0;
      for(int k = i; k <= j; k++)
        zbir += a[k] * min(k - i, j - k);
      if(zbir <= t && zbir > maxZbir) {
        maxZbir = zbir;
        maxi = i;
        maxj = j;
      }
    }
  }

  cout << maxZbir << '\n';
  cout << maxi << ' ' << maxj << '\n';
  return 0;
}
```

Инкременталност

Боље решење се добија ако се приликом померања магнета збир не израчунава сваки пут изнова, већ се добије *инкрементално*, на основу претходне вредности. Заиста, ако знамо $R_{i,j}$ тада $R_{i,j+1}$ не морамо израчунати из почетка већ ажурирањем познате вредности $R_{i,j}$. То значи да за фиксирану позицију левог магнета i све вредности $R_{i,j}$ можемо рачунати инкрементално.

На пример, претпоставимо да је дат низ `1 5 2 2 4 3 2` и да смо израчунали збир пређених растојања куглица за сегмент низа `5 2 2 4`. Вредност тог збира је $s = 0 \cdot 5 + 1 \cdot 2 + 1 \cdot 2 + 0 \cdot 4$. Посматрајмо сада збир на сегменту `5 2 2 4 3` који добијамо померањем десне границе за једно место удесно. Вредност тог збира је $s' = 0 \cdot 5 + 1 \cdot 2 + 2 \cdot 2 + 1 \cdot 4 + 0 \cdot 3$. Разлика тих збирова је $s' - s = (0-0) \cdot 5 + (1-1) \cdot 2 + (2-1) \cdot 2 + (1-0) \cdot 4 + 0 \cdot 3 = 2 + 4$, што значи да вредност s' можемо да добијемо додавањем збира елемената десне половине сегмента `5 2 2 4`.

Овако нешто важи и у општем случају. Претпоставимо, да је познат $R_{i,j} = 0 \cdot a_i + 1 \cdot a_{i+1} + 2 \cdot a_{i+2} + \ldots + 2 \cdot a_{j-2} + 1 \cdot a_{j-1} + 0 \cdot a_j$. Када се магнет са позиције j помери на позицију $j+1$, тада је укупан пут једнак $R_{i,j+1} = 0 \cdot a_i + 1 \cdot a_{i+1} + 2 \cdot a_{i+2} + \ldots + 3 \cdot a_{j-2} + 2 \cdot a_{j-1} + 1 \cdot a_j + 0 \cdot a_{j+1}$ тј. увећава се у односу на претходни за $D_{i,j} = a_s + \ldots + a_{j-2} + a_{j-1} + a_j$, где је s прва позиција у десној половини дела низа између позиција i и j тј. $s = \left\lfloor \frac{i+j}{2} \right\rfloor + 1$ (ако тај део низа садржи непарни број елемената, лева половина је за један елемент дужа од десне).

Током рада алгоритма одржаваћемо у посебној променљивој збир $D_{i,j-1} = a_p + \ldots + a_{j-2} + a_{j-1}$, за $p = \left\lfloor \frac{i+j-1}{2} \right\rfloor + 1 = \left\lceil \frac{i+j}{2} \right\rceil$. За брзо одређивање вредности $D_{i,j-1}$ могуће је користити *низ збирова префикса*, али за тим нема потребе пошто је и тај збир могуће збир рачунати инкрементално, тј. на основу $D_{i,j-1}$ пре

увећавања j можемо лако израчунати $D_{i,j}$ и затим $R_{i,j}$ увећати за ту ажурирану вредност да бисмо добили $R_{i,j+1}$. Непосредно пре повећања j познати збир $D_{i,j-1}$ увећавамо за a_j, проверавамо да ли је $i-j$ парно и ако јесте, тада ту увећану вредност $D[i,j-1]$ додатно умањујемо за $a_p = a_{\frac{i+j}{2}}$.

Пример. Нека је низ $2,1,3,4,3,2$. Тада се збирови $R(0,j)$ инкрементално израчунавају на следећи начин.

```
i  j    [i, j]              p   [p, j)   D[i,j-1] R[i,j]     s   [s, j]     D[i, j]
0  1    [2, 1]              1   []       0        0          1   [1]        1
0  2    [2, 1, 3]           1   [1]      1=0+1    1=0+1      2   [3]        3
0  3    [2, 1, 3, 4]        2   [3]      3=1+3-1  4=1+3      2   [3, 4]     7
0  4    [2, 1, 3, 4, 3]     2   [3, 4]   7=3+4    11=4+7     3   [4, 3]     7
0  5    [2, 1, 3, 4, 3, 2]  3   [4, 3]   7=7+3-3  18=11+7    3   [4, 3, 2]  9
```

На основу претходне анализе је једноставно написати програмски код.

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

using namespace std;

int main() {
  int64_t t;
  cin >> t;

  int n;
  cin >> n;

  vector<int64_t> a(n);
  for(int i = 0; i < n; i++)
    cin >> a[i];

  int maxi = 0, maxj = 1;
  int64_t maxZbir = 0;
  for (int i = 0; i < n; i++) {
    int64_t zbir = 0, zbirDesno = 0;
    for (int j = i+1; j < n; j++) {
      if (zbir <= t && zbir > maxZbir) {
        maxZbir = zbir;
        maxi = i;
        maxj = j;
      }
      zbirDesno += a[j];
      if ((j - i) % 2 == 0)
        zbirDesno -= a[(i + j) / 2];
      zbir += zbirDesno;
    }
  }

  cout << maxZbir << '\n';
  cout << maxi << ' ' << maxj << '\n';
  return 0;
}
```

Техника два показивача

Задатак се ефикасније може решавати техником два показивача, слично задатку у коме се тражи проналажење сегмента узастопних елемената датог низа природних бројева чији је збир елемената једнак датом броју (тај задатак је често детаљно описан у литератури).

Означимо са $R(x,y)$ укупно растојање које пређу куглице када су магнети на позицијама x и y. Нека је један сегмент низа одређен позицијама a и b и неки је његов његов подсегмент одређен позицијама c и d тј. нека

важи $a \leq c \leq d \leq b$. Пошто су сви бројеви у низу ненегативни, важи да је $R(c,d) \leq R(a,b)$. Због овога, ако је збир пређених растојања куглица на неком сегменту мањи или једнак t, нема потребе разматрати његове подсегменте зато што тражимо највећи такав збир. Аналогно, ако је збир на неком сегменту већи од t, нема потребе разматрати његове надсегменте зато што тражимо збир који је мањи или једнак t. Ово нам омогућава да решење оптимизујемо техником два показивача.

Дакле, пошто су сви бројеви у низу ненегативни, померањем десног магнета надесно тј. проширивањем обухваћеног сегмента се повећава укупно растојање које куглице пређу, док се померањем левог магнета надесно тј. скраћивањем обухваћеног сегмента смањује укупно растојање које куглице пређу.

Ако установимо да је $R_{i,j}$, за неко $i < j$ већи од дате границе t, тада нема потребе рачунати и проверавати $R(i,j')$ за $j < j'$, јер ће и у њима збир пређених растојања бити превелики.

Ако установимо да је $R_{i,j}$, за неко $i < j$ мањи или једнак од дате границе t, тада нема потребе рачунати и проверавати $R(i',j)$ за $i < i' < j$, јер је $R(i',j) \leq R_{i,j}$, па $R(i',j)$ не може да поправи максимум (а ако је збир једнак, тражи сегмент чији је леви крај мањи).

Зато одржавамо текући сегмент $[i,j]$ (иницијализујући га на $[0,1]$) рачунамо збир пређених растојања куглица у том сегменту $R_{i,j}$ и ако је он мањи или једнак од дате границе померамо десни крај (ажурирајући пре тога максимум ако је то потребно), а ако је строго већи од дате границе померамо леви крај. При том одржавамо вредност максимума и позиције магнета на којима се он постиже.

Анализа сложености. Пошто се оба показивача i и j крећу у једном смеру кроз низ, број корака спољашње петље линеарно зависи од димензије низа n. У сваком кораку се збир изнова израчунава (поново у линеарној сложености), па је сложеност ове имплементације $O(n^2)$.

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

using namespace std;

int main() {
  int64_t t;
  cin >> t;

  int n;
  cin >> n;

  vector<int64_t> a(n);
  for(int i = 0; i < n; i++)
    cin >> a[i];

  // maksimalni zbir rastojanja i pozicije magneta na kojima se
  // postize taj maksimalni zbir
  int64_t maksZbir = 0;
  int maxi = i, maxj = j;

  // pozicije levog i desnog magneta - pomeraju se kuglice u intervalu [i, j]
  int i = 0, j = 1;
  while (j < n) {
    // izracunavamo ukupno predjeno rastojanje kada su magneti na
    // pozicijama [i, j]
    int64_t zbir = 0;
    for(int k = i; k <= j; k++)
      zbir += a[k] * min(k - i, j - k);

    if (zbir <= t) {
      // tekuci zbir je u granicama, pa azuriramo maksimum ako je to potrebno
      if (zbir > maksZbir) {
        maksZbir = zbir;
        maxi = i;
```

```
        maxj = j;
      }
      // zbir nije veci od ogranicenja, pa pomeramo desni kraj i
      // prosirujemo interval [i, j]
      j++;
    }
    else
      // zbir je veci od ogranicenja, pa pomeramo levi kraj i
      // skracujemo interval [i, j]
      i++;
  }

  cout << maksZbir << '\n';
  cout << maxi << ' ' << maxj << '\n';
  return 0;
}
```

Техника два показивача и инкременталност

Најефикасније решење се добија ако искомбинујемо технику два показивача и инкрементално рачунање збира. Већ је описано како се на основу познате вредности $R_{i,j}$ ефикасно израчунава вредност $R_{i,j+1}$. На сличан начин се може израчунати и вредност $R_{i+1,j}$ која нам је потребна када се леви магнет помера за једно место удесно.

Претпоставимо, да је познат $R_{i,j} = 0 \cdot a_i + 1 \cdot a_{i+1} + 2 \cdot a_{i+2} + \ldots + 2 \cdot a_{j-2} + 1 \cdot a_{j-1} + 0 \cdot a_j$. Када се магнет са позиције i помери на позицију $i+1$ тада је укупан пут једнак $R_{i+1,j} = 0 \cdot a_{i+1} + 1 \cdot a_{i+2} + 2 \cdot a_{i+3} \ldots + 2 \cdot a_{j-2} + 1 \cdot a_{j-1} + 0 \cdot a_j$ тј. умањује се у односу на претходни за $L_{i,j} = a_{i+1} + a_{i+2} + \ldots + a_s$, где је s последњи елемент леве половине дела низа између i и тј. $s = \left\lfloor \frac{i+j}{2} \right\rfloor$ (ако тај део низа има непаран број елемената, лева половина је за један елемент дужа). Стога током рада алгоритма у посебној променљивој одржавамо додатно и збир елемената леве половине $L_{i,j} = a_{i+1} + a_{i+2} + \ldots + a_s$. И њега можемо ажурирати инкрементално. Да бисмо израчунали $L_{i+1,j} = a_{i+2} + \ldots + a_{\left\lfloor \frac{i+j+1}{2} \right\rfloor}$, непосредно након повећања i тј. одређивања вредности $i' = i+1$ познати збир $L_{i,j}$ умањујемо за вредност $a_{i'} = a_{i+1}$, проверавамо да ли је $j - i' = j - (i+1)$ паран број и ако јесте, тада збир додатно увећавамо за вредност $a_{\frac{i'+j}{2}} = a_{\frac{i+1+j}{2}}$.

Анализа сложености. Ако се инкременталност употреби у комбинацији са грубом силом, добија се алгоритам сложености $O(n^2)$.

Ако се инкременталност употреби у комбинацији са техником два показивача, добија се алгоритам сложености $O(n)$.

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

using namespace std;

int main() {
  int64_t t;
  cin >> t;

  int n;
  cin >> n;

  vector<int64_t> a(n);
  for(int i = 0; i < n; i++)
    cin >> a[i];

  // pozicije levog i desnog magneta - pomeraju se kuglice u intervalu [i, j]
  int i = 0, j = 1;

  // maksimalni zbir rastojanja i pozicije magneta na kojima se
```

```
  // postize taj maksimalni zbir
  int64_t maksZbir = 0;
  int maxi = i, maxj = j;

  //  ukupno predjeno rastojanje kada su magneti na pozicijama [i, j]
  int64_t zbir = 0, zbirLevo = 0, zbirDesno = 0;
  while (j < n) {
    if (zbir <= t) {
      // tekuci zbir je u granicama, pa azuriramo maksimum ako je to potrebno
      if (zbir > maksZbir) {
        maksZbir = zbir;
        maxi = i;
        maxj = j;
      }
      // zbir nije veci od ogranicenja, pa pomeramo desni kraj i
      // prosirujemo interval [i, j], azurirajuci pri tom zbir
      zbirDesno += a[j];
      j++;
      zbir += zbirDesno;
    } else {
      // zbir je veci od ogranicenja, pa pomeramo levi kraj i
      // skracujemo interval [i, j], azurirajuci pri tom zbir
      zbir -= zbirLevo;
      i++;
      zbirLevo -= a[i];
    }
    // prebacujemo jedan element iz desne u levu polovinu
    if ((j - i) % 2 == 0) {
      zbirLevo  += a[(i + j) / 2];
      zbirDesno -= a[(i + j) / 2];
    }
  }

  cout << maksZbir << '\n';
  cout << maxi << ' ' << maxj << '\n';
  return 0;
}
```