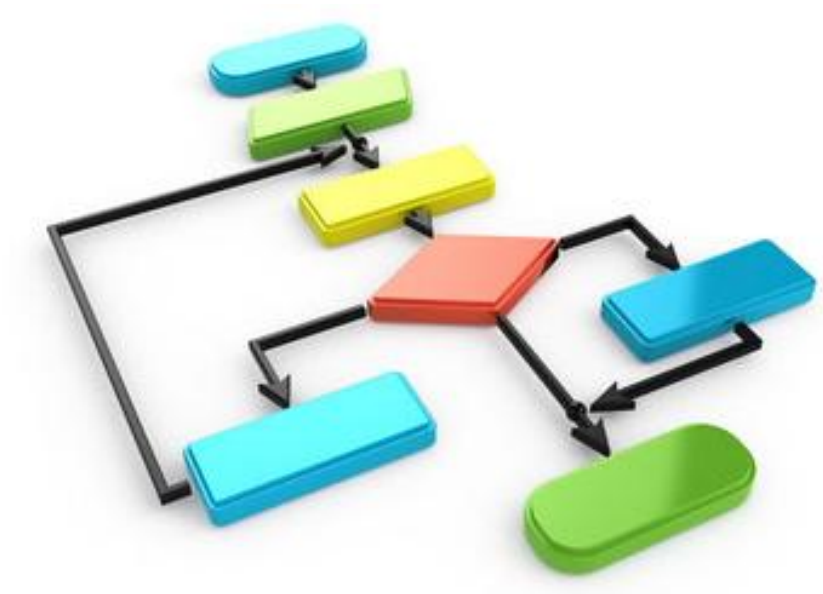
Liceul Teoretic Grigore Moisil



Vectori în C++

Îndrumar de laborator

Autor: Spătaru Mihaela

Cuprins

1. Tablouri unidimensionale (vectori)

Vectorii sau tablourile unidimensionale sunt structuri de date bine definite si organizate in memorie. Cu ajutorul acestora, se pot păstra in memorie si accesa ulterior mai multe variabile, fără a fi nevoie de reţinerea explicită a fiecăreia dintre ele. Cu alte cuinte, un vector reţine sub acelaşi nume mai multe valori de acelaşi tip. Fiecare valoare poate fi accesata folosind poziţia sa în vector (poziţiile sunt numere naturale cuprinse intre 0 si dimensiunea maximă a vectorului).

Declararea vectorului

Sintaxa este:

```
tip_elemente_vector   nume_vector [dimensiune_maxima_vector];
```

Exemple:

```
int  x[100];// tipul elementelor vectorului este int iar dimensiunea maxima 100
double a[50];
char  s[1000];
```

În ce secţiune a programului C++ declarăm vectorii? În mod normal vectorii se declară împreună cu celelalte variabile la începutul programului C++, imediat după main(). Dar noi vom face o excepţie de la regulă şi-i vom declara imediat înainte de main(), astfel:

Exemple:

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

int v[100];
int main() {
  ...
  return 0;
}
```

Iniţializarea vectorilor

Fiecare element al vectorului este o variabilă separată, care poate fi atribuită, citită sau scrisă întocmai ca o variabilă de tipul vectorului.

Exemple:

```
int v[4]={5, 6, 10, 21}; // iniţializarea la declarare

int numere[50];
for(i=0;i<numar_numere;i++)
{         numere[i]=i;              } // iniţializarea la atribuire

v[3] = 30;                  // atribuirea
v[i] = v[i] + 1;            // atribuirea
```

Citirea vectorilor

Citirea mai multor valori dintr-un fişier şi introducerea lor într-un vector se face similar cu citirea unei secvenţe: vom citi mai întâi numărul de elemente, n şi apoi cele n elemente, de la 0 la n-1 sau de la 1 la n.

int a[100];//un vector de maxim 100 elemente
int main()
{ int n,i;
cout<<"n= "; cin>>n; //n este lungimea vectorului
for(i=0;i<n;i++)
{
*cout<<"a["<<i<<"]= "; /*se afiseaza numarul de ordine al elementului, care este cu 1 mai mare decat indicele */*
cin>>a[i];
}
return 0; }
	Explicaţie
	
	Aici indicii tabloului sunt 0, 1 ,2, … 99.
	Deci daca tabloul are n elemente atunci primul indice este 0, iar ultimul indice este n-1;

int a[100]; //un vector de maxim 100 elemente
int main()
{ int n,i;
cout<<"n= "; cin>>n; //n este lungimea vectorului
for(i=1;i<=n;i++)
{
cout<<"a["<<i<<"]= "; //se afiseaza numarul de ordine al elementului,
cin>>a[i];
}
return 0;
}
	Aici indicii tabloului sunt 1, 2 ,… 99, 100.
	Deci daca tabloul are n elemente atunci primul indice de la care pornim este 1, iar ultimul indice este n, deşi este si elementul de indice 0 dar pe care nu îl folosim;

Afişarea vectorilor

Afişarea valorilor vectorului este similară cu citirea. Vom scrie în fişierul de ieşire fiecare valoare, pe rând: de la 0 la n-1 sau de la 1 la n.

int a[100]; *int main()* *{ int n,i;* *…….* *for(i=0;i<n;i++)* *{* *cout<<a[i+1]<<" ";* *}* *return 0;* *}*	*Explicaţie* *Aici indicii tabloului sunt 0,1,2,…99.* *Deci daca tabloul are n elemente atunci primul indice este 0, iar ultimul indice este n-1;*
int a[100]; *int main()* *{ int n,i;* *…….* *for(i=1;i<=n;i++)* *{* *cout<<a[i]<<" ";* *}* *return 0;* *}*	*Aici indicii tabloului sunt 1,2,…99,100.* *Deci daca tabloul are n elemente atunci primul indice de la care pornim este 1, iar ultimul indice este n, deşi este si elementul de indice 0 dar pe care nu îl folosim;*

2. Algoritmi elementari cu vectori

2.1 Suma elementelor unui vector

```
#include <iostream>

using namespace std;

int v[100];// se declară vectorul ca variabilă globală
int main()
{
        int  i, n, s=0;       //suma se inițializează cu 0
        cin>>n;
        for ( i = 1; i <= n; i++ )
        { cin>>v[i];
        s= s + v[i];//la suma anterioară se adaugă noua valoare
        }

        cout<<s;

        return 0;
}
```

2.2 Maximul dintr-un vector

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

        int v[101];//declararea vectorului de dimensiune maximă 100

int main()
{
        int i, vmax,n;
        cin>>n;
        for ( i = 1; i <= n; i++ )
        { cin>>v[i];}
        vmax = v[1];// se inițializează maximul cu valoarea primului element din vector
for ( i = 2; i <= n; i++ ){
        if (v[i] > vmax ) //daca se gaseste  un element mai mare
        vmax = v[i];  //vmax ia valoarea lui
        }

        cout<<vmax;


return 0;
}
```

Maxim multiplu: Dacă elementul maxim apare de mai multe ori în vector, afişaţi-l, precum şi numărul de apariţii ale acestuia.

Rezolvare:

```
        #include <iostream>
        using namespace std;
        int v[101];//declararea vectorului de dimensiune  maxima 100
int main()
{       int i, vmax,n;
        cin>>n;
        for ( i = 1; i <= n; i++ )
        {
       cin>>v[i];
        }
        vmax = v[1]; // se inițializează maximul cu primul element din vector
        int nr=1; // si contorul de elemente maxime cu 1;
for ( i = 2; i <= n; i++ ){
        if (v[i] > vmax ){  //daca elementul curent este mai mare decât valoarea maxima
        vmax = v[i];// se atribuie maximului noua valoare
        nr=1; //contorul se resetează la valoarea 1
        }
        else  if (vmax==v[i]){  //daca elementul curent este egal cu valoarea maxima
        nr++; // se adaugă la contor 1 unitate:  nr=nr+1;
        }
}
        cout<<vmax<<" apare de "<<nr<< " ori ";

return 0;
}
```

Min/Max

```
        int a[101], n, i;
        min=max=a[1];
for (i=2; i<=n; i++)
{       if(min>a[i])    min=a[i];
        if(max<a[i])    max=a[i];       }
```

Exersaţi:

1. Se introduc de la tastatură cel mult 10000 de numere întregi. Să se afişeze valoarea cea mai mică şi numărul de ordine (indicele) al elementelor care au valoarea minimă.
Indicaţie Pentru memorarea indicilor elementelor cu valoarea minimă se va folosi un vector.

2. Ionel s-a documentat şi a făcut o listă cu preturile ultimelor **n** smartphone-uri apărute şi altă listă cu preţurile celor mai vândute **m** tablete. Ajutaţi-l să determine cel mai scump smartphone şi cea mai ieftină tabletă. Dacă sunt mai multe, sa afişeze toate poziţiile din listele corespunzătoare.

3. Se citesc din fişierul text **medii.in** numărul de elemente n (n<40) şi apoi cele n medii la informatică ale unei clase de elevi a1, a2, ...an. Să se scrie în fişierul **medii.out**:

1. Indicii elevilor cu media 10
2. Numărul de elevi corigenţi (cu media<5)
3. Media clasei
4. Media cea mai mică şi media cea mai mare din clasă

2.3 Afişarea în ordine inversă a elementelor unui vector

for (i=n-1; i>=0; i--) *{ cout<<a[i]<<" " ; }*	*for (i=n; i>=1; i--)* *{ cout<<a[i]<<" " ; }*

2.4 Inversarea unui vector în el însuşi

for(i=0; i< n/2; i++) *{ x=a[i]; a[i]=a[n-i-1]; a[n-i-1]=x; }*	*for(i=1; i< =n/2; i++)* *{ x=a[i]; a[i]=a[n-i+1]; a[n-i+1]=x; }*

2.5 Permutare circulară

la stânga	**la dreapta**
aux=a[1]; *for(i=1;i<=n-1;i++)* *{ a[i]=a[i+1]; }* *a[n]=aux;*	*aux=a[n];* *for(i=n; i>1; i--)* *{ a[i]=a[i-1];}* *a[1]=aux;*

2.6 Inserarea valorii x pe poziţia k

```
for(i=n+1; i>k; i--)
a[i]=a[i-1];
a[k]=x;
n++;
```

2.7 Eliminarea elementului a[k] aflat pe poziţia k

```
for(i=k; i<=n-1; i++)
a[i]=a[i+1];
n--;
```

2.8 Căutarea unui element în vector

Exerciţiu: se citesc n şi x numere naturale, iar apoi se citesc n numere. Să se spună prima poziţie pe care apare elementul x. Dacă elementul x nu se află în vector vom afişa poziţia n (în afara vectorului).

```
#include <iostream>

using namespace std;

int v[100];
int main() {

        int n, i, x;
        cin>>n>>x;
 for ( i = 0; i < n; i++ )
  {     cin>>v[i] ;     }

         i = 0;
 while ( i < n && v[i] != x ) // câtă vreme suntem încă în vector
        { i++;      }       // şi nu l-am găsit pe x, avansăm cu un pas
        cout<<i ; // afisam pozitia pe care am gasit elementul x

        return 0;
}
```

2.8.1 Căutarea unui element în vector după poziţia k[1] **

Exerciţiu: se citesc n, x şi k numere naturale, iar apoi se citesc n numere. Să se spună prima poziţie după poziţia k pe care apare elementul x. Dacă ajungem la ultima poziţie, n-1, avem voie să începem din nou cu zero, deoarece se consideră că cele n numere sunt aşezate în cerc. Dacă elementul x nu se află în vector vom afişa poziţia n (în afara vectorului).

Soluţie: vom citi cele n valori într-un vector, iar apoi vom porni de la poziţia k şi vom înainta fie până ce găsim elementul x, fie până când ajungem la poziţia k din nou. Avansul indicelui i se va face % n. Iată soluţia:

```
#include <fstream>
using namespace std;
int v[100];
int main() {
          ifstream fin( "cautarek.in" );
          ofstream fout( "cautarek.out" );
int n, i, x,k;
fin>>n>>x>>k;
 for ( i = 0; i < n; i++ )
   {      fin>>v[i] ;       }
          fin.close();
           i = k;
 while (  v[i] != x )            // dacă pe poziţia k nu avem x
          {  i=(i+1)%n;       // avansăm cu un pas
          while ( i != k && v[i] != x )// câtă vreme nu ne-am întors la k
                    {i=(i+1)%n;   }
          }
          if ( v[i] != x )
          {         i=n;                }
          fout<<i ;
          fout.close();
          return 0;
}
```

Observăm că pentru a începe din nou de la zero atunci când i devine n este de ajuns să aplicăm operaţia ***%n*** *lui i. Acest lucru este posibil tocmai pentru că indicii vectorului v încep de la 0.*

Aplicaţie: Fiind dat un vector v şi un element x, să se elimine din vector prima apariţie a elementului x, în cazul în care acesta apare în vector.

```
i = 0;
while ( i < n && v[i] != x ){
i++;
}
if ( i < n ) { // dacă am găsit elementul, îl eliminăm
 for ( j = i + 1; j < n; j++ ){
   v[j-1] = v[j];
}
  n--;
}
```

Întrebare*: Pe ce poziţie se află primul element din vector? Modificaţi exerciţiul pentru cazul în care primul element este pe poziţia 1.*

[1] Problemă rezolvată cu fişiere

2.9 Secvenţe de numere

1. Să se afişeze cea mai lungă secvenţă de elemente consecutive de parităţi diferite.

Exemplu:

Date de intrare	**Date de ieşire**
8 2 4 3 3 4 7 2	3 4 7 8

Rezolvare:

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

int a[100];

int main() {

                int n, i, lmax, pmax, pc, lc;
                cin>>n;
        for ( i = 1; i <=n; i++ ){
        cin>>a[i] ;
}

        lmax = 0;
        lc=1;           //lungimea curenta se iniţializează cu 1
        pc=1;           //poziţia curenta este poziţia primului element

        for ( i = 2; i<=n; i++ ){
  if ((a[i]%2)!=(a[i-1]%2)){   //dacă resturile sunt diferite
    lc++;  // se incrementează  lungimea curenta cu 1
        }
  else
    {
         lc=1; // se resetează lungimea curenta la 1
        pc=i;
        }
  if (lmax<lc){
       lmax=lc;
       pmax=pc;
      }
        }
    for (i = pmax; i <= pmax+lmax-1; i++ ){
          cout<<a[i]<<" "; //se afişează toate elementele din secvenţa de lungime maxima
    }

        return 0;
}
```

Secvenţa de sumă maximă

Să se afişeze secvenţa de sumă maximă dintr-un şir de numere întregi şi valoarea acestei sume.

Exemplu:

Date de intrare	**Date de ieşire**
8 2 -4 -3 5 -4 7 8 -2	16 5 -4 7 8

Rezolvare:

```
#include <iostream>
using namespace std;
int a[100];
int main() {
                    int n, i, sc, smax, pc, ic,sf;
                    cin>>n;
          for ( i = 1; i <=n; i++ ){
          cin>>a[i] ;
}
   smax = sc=a[1];      // suma maximă si suma curenta se inițializează cu primul element
      ic=sf=1;          //poziția de început si poziția de sfârşit se inițializează cu 1
      pc=1;             //poziția curenta este poziția primului element
          for ( i = 2; i<=n; i++ ){
            if ( sc>0 ){   //dacă suma curentă este pozitivă
                    sc=sc+a[i];
          }
   else
      {    sc=a[i];  // se resetează suma curentă
          pc=i;  // se salvează poziția curentă
      }
   if (sc>smax){
         smax=sc;
         ic=pc;
         sf=i;       }
}
      cout<<smax<<'\n';
      for (i = ic; i <= sf; i++ ){
      cout<<a[i]<<" ";
      }
          return 0;         }
```

Exersaţi:

1. Rezolvaţi următoarele probleme de pe site-ul www.pbinfo.ro:
SumSec1 (#516), SecvZero (#518), SecvEgale (#523).

2. Se consideră două mulţimi de numere întregi A şi B. Să se calculeze:
   a. Reuniunea celor două mulţimi: X = A ∪ B
   b. Intersecţia celor două mulţimi: Y = A ∩ B
   c. Diferenţa celor două mulţimi Z = A - B

3.Vector de frecvenţe (vector caracteristic)

Vectorii de frecvenţă sunt vectori ale căror elemente au o semnificaţie specială: valoarea de pe poziţia i arată numărul de apariţii (sau frecvenţa) lui i într-o altă entitate, de obicei o secvenţă de numere, sau un număr.
Exemplu: Fie dat şirul de numere sau secvenţa: 5 4 4 0 3 2 2 2 0
Vectorul caracteristic va fi:

v[i] - frecventa	2	0	3	1	2	1	0	0	0	0
i - poziţia	0	1	2	3	4	5	6	7	8	9

Tot cu ajutorul acestui vector pot fi implementate operaţiile de bază cu mulţimi:

- **Căutarea** unui element
- **Intersecţia** a două (sau mai multe) mulţimi
- **Reuniunea** a două (sau mai multe) mulţimi

În orice mulţime elementele sunt unice, iar vectorul frecvenţelor are doar valori 0 sau 1. Acest vector este numit vectorul caracteristic al unei mulţimi. Vectorul de frecvenţe poate fi folosit pentru a obţine rapid mulţimea asociată ca un vector caracteristic astfel:

- 0 înseamnă că elementul nu aparţine mulţimii;
- 1 (o valoare diferită de 0) înseamnă că elementul aparţine mulţimii.

Exerciţiul 1

Fiind dat un număr natural n între 1 şi 2.000.000.000 să se afişeze cifrele numărului şi numărul de apariţii ale fiecărei cifre in număr (vezi exemplul de mai sus).

```
#include<iostream>

using namespace std;

int v[10];
int main()
{       int n,c;
        cin>>n;
while (n>0)
{       c=n%10;
        v[c]++;                 //construim vectorul frecventelor
        n=n/10;
}
for(c=0;c<10;c++)
{       if (v[c]!=0)            //afişam cifrele distincte si numărul de apariţii
cout<<c<<' '<<v[c]<<endl;
}
return 0;
}
```

Exerciţiul 2

Dat un număr natural n între 1 şi două miliarde să se afişeze numărul de elemente al mulţimii cifrelor sale. Acesta este un exerciţiu introductiv în folosirea vectorilor de frecvenţă.
Date de intrare: Numărul n.
Date de ieşire: Un singur număr, numărul de cifre distincte ale lui n.
Restricţii: $1 \le n \le 2.000.000.000$

```
#include<iostream>
using namespace std;
int v[10;
int main()
{ int n,c,nrc;
        cin>>n;
while(n>0)
{       c=n%10;
        v[c]=1;                 //construim vectorul frecventelor
        n=n/10;
}
nrc=0;
for(c=0;c<10;c++)
{
nrc=nrc+v[c];                   //suma elementelor vectorului de frecvență
cout<<nrc;
return 0;
}
```

Observație: *De remarcat că nu avem nevoie de numărul de apariții al cifrelor, de aceea stocăm unu, ori de câte ori apare o cifră. Aceasta ne permite ca la final să calculăm suma elementelor vectorului de frecvență, fără a mai testa dacă valorile sunt zero.*

Exerciţiul 3

Dat un număr natural n între 1 şi două miliarde să se afişeze cel mai mare număr care se poate forma cu cifrele sale. Acesta este un exerciţiu introductiv în folosirea vectorilor de frecvenţă.
Date de intrare: Numărul n. Exemplu: 234234234
Date de ieşire: Cel mai mare număr ce se poate forma cu cifrele numărului n. Exemplu: 444333222
Restricţii: 1 ≤ n ≤ 2.000.000.000

```
#include <fstream>

using namespace std;

int v[10];
int main() {
int n, c;
cin>>n;
 while ( n > 0 ) {
   c = n % 10;
   v[c]++;
   n = n / 10;
 }
 for ( c = 9; c >= 0; c-- )
   while ( v[c] > 0 ) {
     cout<<c; // n=n*10+c;
    v[c]--;
//cout<<n;
   }
 return 0;
}
```

Exerciţiul 4

Se citesc două numere naturale n şi m. Să se afişeze numărul de cifre distincte comune ambelor numere. Pentru a rezolva problema vom construi vectorul de frecvenţă al primului număr, a. El va avea numai elemente zero şi unu. Apoi vom extrage pe rând cifrele lui b. Dacă ele apar în vectorul de frecvenţă vom face două lucruri: vom aduna unu la contorul de cifre comune şi vom şterge acea cifră din vectorul de frecvenţă, setând acel element pe zero, pentru ca în viitor să nu o mai luăm din nou în considerare dacă cifra apare de mai multe ori în b. Iată o rezolvare bazată pe aceste idei:

```
#include <iostream>
using namespace std;
int v[10];
int main()
{ int a,b,uc,ncr;
cin>>a>>b;
// setam 1 pentru cifrele lui a
 while ( m > 0 ) {
   c= a % 10;
   v[c] = 1;
   a = a / 10;
 }
 nrc = 0;
 // testam cifrele lui n, resetând cifrele deja găsite, pentru a nu le
 // găsi de mai multe ori
 while ( n > 0 ) {
   c = b % 10;
   if ( v[c] == 1 ) {
     nrc++;
     v[c] = 0;
   }
   b = b / 10;
 }
cout<<nrc;
 return 0;
}
```

Exerciţiul 5[2]

Se dau două numere naturale a şi b cu maxim 9 cifre. Cerinţe:
a) Să se determine cifrele distincte, care apar în a şi nu apar în b.
b) Să se afişeze numărul cel mai mare format din cifrele distincte comune lui a şi b.

Date de intrare

Din fişierul de intrare cifre.in se citesc de pe prima linie, separate printr-un spaţiu, valorile a şi b.

Date de ieşire

Datele de ieşire se afişează în fişierul de ieşire cifre.out. Răspunsul la prima cerinţă se va afişa pe prima linie a fişierului, cifrele fiind scrise în ordine strict crescătoare, iar răspunsul la cea de a doua cerinţă pe linia a doua.

cifre.in	cifre.out
22987 1333992	8 7 92

[2] Exerciţiu rezolvat cu fişiere

Rezolvăm problema folosind un singur vector de zece elemente: v[10], pentru care stabilim:
v[i]=0 dacă elementul i (cifra i) nu apare nici în numărul a, nu apare nici în numărul b
v[i]=1 dacă elementul i (cifra i) apare numai în numărul a şi nu apare în numărul b
v[i]=2 dacă elementul i (cifra i) apare numai în numărul b şi nu apare în numărul a
v[i]=3 dacă elementul i (cifra i) apare în ambele numere a şi b

```
#include <fstream>
using namespace std;
ifstream fin("cifre.in");
ofstream fout("cifre.out");
int v[10];
int main()
{ int a,b,uc,nr=0;
fin>>a>>b;
while(a!=0)
{
uc=a%10;
v[uc]=1;
a=a/10;
}
while(b !=0)
{
        uc=b%10;
        if(v[uc]==1)
{
        v[uc]=3;
}
else    if ( v[uc]==0)
{
        v[uc]=2;
}
b=b/10;
}
for(i=0;i<10;i++)
{       if ( v[i]==1)
        fout <<i<<" " ;
}
for (i=9; i>=0; i--)
{       while(v[i]==3)
{       fout<<i;                // nr=nr*10+i;
        v[i]--;
}
}                               // fout<<nr;
return 0;
}
```

Exerciţiul 6
Dat un număr natural n între 1 şi două miliarde să se afişeze cel mai mic număr care se poate forma cu cifrele sale. Acesta este un exerciţiu în folosirea vectorilor de frecvenţă.
Date de intrare: Se citeşte de la tastatură numărul n. Exemplu: 80002452
Date de ieşire: Pe ecran se va afişa un singur număr. Exemplu: 20002458
Restricţii: 1 ≤ n ≤ 2.000.000.000

```
#include <iostream>
using namespace std;
int v[10];
int main()
{ int n,c;
cin>>n;
while ( n > 0 ) {
   c = n % 10;
   v[c]++;
   n = n / 10;
 }
 // cauta cea mai mica cifra diferita de zero
 c = 1;
 while ( v[c] == 0 )
   c++;
 cout<< c;                  // afiseaza prima cifra
 v[c]--;                    // scoate-o din numar
 // afiseaza restul cifrelor, in ordine crescatoare
 for ( c = 0; c < 10; c ++ )
   while ( v[c] > 0 ) {
     cout<<c;
     v[c]--;
   }
 return 0;
}
```

***Observaţie**: Trebuie să avem însă grijă deoarece nu putem să începem cu cifra 0! De aceea vom afişa mai întâi cea mai mică cifră diferită de zero din vectorul de frecvenţă. Apoi vom scădea unu de la acea poziţie, ca să ştim că am afişat-o. Abia apoi putem trece la afişarea întregului vector de la mic la mare fără să ne mai pese de cifrele 0. Iată o soluţie posibilă:*

Exerciţiul 7[3]

Se dă o secvenţă de n culori codificate cu numere între 1 şi 99. Să se afişeze culorile în ordinea lor crescătoare. Acesta este un exerciţiu în folosirea vectorilor de frecvenţă.
Date de intrare: Fişierul de intrare culori.in va conţine pe prima linie numărul n. Pe a doua linie va conţine n numere despărţite prin spaţiu.
Date de ieşire: În fişierul de ieşire culori.out veţi scrie o singură linie conţinând cele n culori în ordine crescătoare.
Restricţii: 1 ≤ n ≤ 1.000.000; 1 ≤ culoare ≤ 99
Exemplu:

culori.in	culori.out
10 4 7 3 1 3 7 4 1 2 2	1 1 2 2 3 3 4 4 7 7

***Observaţie**: Problema culori ne învaţă să ordonăm numere naturale, atunci când numerele naturale nu sunt foarte mari. În cazul nostru avem de ordonat culori, care sunt numere între 1 şi 99. Pentru aceasta vom folosi un vector de frecvenţă de 100 de elemente. Să nu uităm că elementele vor fi numerotate de la 1 la 99. Pe măsură ce citim culorile adunăm unu la poziţia acelei culori în vectorul de frecvenţă. Nu avem nevoie să păstrăm culorile în alt vector! Odată calculat vectorul de frecvenţă îl vom parcurge de la 1 la 99 afişând indicele i de câte ori ne arată elementul v[i]. Această ordonare (sau sortare) se mai numeşte şi sortare prin numărare*

[3] Cu fişiere

```
#include <fstream>
ifstream fin ( "culori.in" );
ofstream fout ( "culori.out" );
int v[100];
int main() {
 int n, i, cul;
 fin>>n;
 for ( i = 0; i < n; i++ ) {
  fin>>cul;
  v[cul]++;
 }
  for ( cul = 1; cul < 100; cul++ )
  while ( v[cul] > 0 ) {
   fout<< cul;
   v[cul]--;
  }
 return 0;
}
```

Exerciţiul 9 - Problema data la ONI 2002***

Gigel are o panglică alcătuită din benzi de 1 cm lăţime, colorate în diverse culori. Panglica are N benzi colorate cu C culori, culori pe care le vom numerota de la 1 la C. Gigel vrea ca la ambele capete ale panglicii să aibă aceeaşi culoare, dar cum nu poate schimba culorile benzilor, singura posibilitate rămâne tăierea unor bucăţi de la capete.
Cerinţă: Scrieţi un program care să determine modul de tăiere a panglicii astfel încât la cele două capete să fie benzi de aceeaşi culoare, iar lungimea panglicii obţinute să fie maximă.
Date de intrare: Fişierul de intrare panglica.in conţine:

- pe prima linie numerele naturale N şi C separate printr-un spaţiu;
- pe următoarele N linii descrierea panglicii: pe fiecare linie un număr natural de la 1 la C, reprezentând în ordine culorile fâşiilor ce alcătuiesc panglica.

Date de ieşire: Fişierul de ieşire panglica.out va conţine următoarele 4 numere:

- pe prima linie numărul de fâşii rămase
- pe linia a doua numărul culorii care se află la capete
- pe linia a treia câte fâşii trebuie tăiate de la începutul panglicii iniţiale
- pe linia a patra câte fâşii trebuie tăiate de la sfârşitul panglicii iniţiale

Restricţii

- $2 \le N \le 10000$
- $1 \le C \le 200$
- Dacă există mai multe soluţii alegeţi pe cea în care se taie cât mai puţin din partea de început a panglicii.

Exemplu:

panglica.in	panglica.out
6 3 1 2 1 3 2 3	4 2 1 1
5 2 1 2 1 2 2	4 2 1 0

Soluția eficienta se bazează pe vectori de frecvență. Ea nu stochează elementele inițiale, drept pentru care folosește mult mai puțină memorie. De asemenea, ea face o singură parcurgere prin elementele de la intrare, ceea ce duce la o soluție mai rapidă. Ideea este următoarea: pentru fiecare culoare c vom memora două poziții din vectorul de la intrare:
prim[c] este prima poziție în vectorul inițial pe care apare culoarea c.
ultim[c] este ultima poziție în vectorul inițial pe care apare culoarea c.

Dacă culoarea c nu apare la intrare atunci atât prim[c] cât și ultim[c] vor fi zero. Vom calcula aceste valori chiar în timp ce citim culorile la intrare: pentru poziția curentă i și culoarea citită c vom verifica dacă prim[c] este zero, caz în care vom păstra poziția i în el. De asemenea, întotdeauna vom păstra poziția i în ultim[c].

```
#include <fstream>
using namespace std;
 ifstream fin( "panglica.in");
 ofstream fout( "panglica.out");
int prim[200], ultim[200];
int main() {
        int n, c, i, cul, cmax, lungime, lmax;
        fin>>n>>c ;
        for ( i = 0; i < n; i++ ) {
         fin>>cul;
        if ( prim[cul-1] == 0 )   // daca culoarea nu a apărut până acum
         prim[cul-1] = i + 1;    // ii stocam in primul prima ei apariție
        ultim[cul-1] = i + 1;     // in ultimul vom păstra mereu ultima apariție
 }          // atenție: stocam i + 1 ca sa nu confundam 0 (nu apare)
 lmax = 0;
        for ( cul = 0; cul < c; cul++ )
         if ( prim[cul] > 0 ) {    // culoarea exista in vector
         lungime = ultim[cul] - prim[cul] + 1; // calculam lungimea benzii
        if ( lungime > lmax ) {  // daca lungimea este mai mare
         lmax = lungime;       // o pastram
        cmax = cul;          // pastram si culoarea de inceput si sfirsit
   }
  }
        fout<<lmax<<'\n';
        fout<<cmax + 1<<'\n'<<prim[cmax] – 1<<'\n'<<n - ultim[cmax];
 return 0;
}
```

Exersați:

1. Rezolvați următoarele probleme de pe site-ul www.pbinfo.ro:
Numere8 (#1005), Unice (#267), cifre comune (#365), e_palindrom (#1400)
Cod3 (#2177), MDiv (#2031), cadouri2 (#2342), Frecventa1(#301), Sortare Divizori (#1608).

2. Rezolvați următoarea problemă folosindu-vă de vectorii caracteristici.
Se consideră două mulțimi de numere întregi A și B. Să se calculeze:
   a. Reuniunea celor două mulțimi: X = A ∪ B
   b. Intersecția celor două mulțimi: Y = A ∩ B
   c. Diferența celor două mulțimi Z = A - B

4. Ciurul lui Eratostene

Eratostene a inventat un algoritm eficient de calcul al tuturor numerelor prime până la un număr dat. Algoritmul procedează astfel:

1. Creează o listă a întregilor consecutivi de la 2 la n: [2, 3, 4, ..., n].
2. Caută primul număr netăiat din listă (inițial nici un număr nu e tăiat). Fie p acel număr.
3. Mergi din p în p prin listă, începând cu 2*p și taie toate numerele întâlnite (unele vor fi deja tăiate)
4. Reia de la pasul 2, până ce depășim n.

În final numerele rămase netăiate sunt prime.
Vom folosi un vector de frecvență, ciur, care pentru fiecare număr între 2 și n ne va spune dacă este tăiat sau nu. Inițial vectorul ciur va fi inițializat cu zero, care semnifică că toate numerele sunt prime, iar pe măsură ce tăiem numere, ele vor fi marcate cu unu. În final ciur[x] va fi zero dacă numărul este prim, sau unu în caz contrar.

Iată implementarea acestei idei. Programul următor calculează vectorul ciur pentru numerele până la n, cu n maxim două milioane:

```
        char ciur[2000001];
...
        fin>>n;
        for ( d = 2; d < n; d++ )
        {       if ( ciur[d] == 0 ) // daca d este prim
                for ( i = d + d; i <= n; i = i + d ) // vom marca numerele din d in d
                        {       ciur[i] = 1;    }       }
```

Programul se poate optimiza dacă facem următoarele observații:
1. Deoarece elementele lui ciur sunt zero sau unu, vom folosi caractere, văzute ca numere mici. Să ne reamintim că un caracter ocupă un octet, pe când un întreg ocupă 4 octeți;memoria folosită va fi de patru ori mai mică.
2. În a doua buclă for: for (j = i * i; j <= dimmax; j = j + i) variabila j ia valorile 2·i, 3·i, 4·i, ..., k·i.
Pentru j < i toate valorile de forma k·i au fost deja tăiate de valorile j anterioare, drept pentru care nu are rost să le mai parcurgem. Putem să pornim cu j de la i·i.
3. Conform observației anterioare j pornește de la i ·i ; nu are rost să mergem cu d mai departe de SQRT(n).
Programul se poate optimiza dacă facem următoarele observații:
În a doua buclă for variabila i ia valorile 2·d, 3·d, 4·d, ..., k·d. Pentru k < d toate valorile de forma k·d au fost deja tăiate de valorile k anterioare, drept pentru care nu are rost să le mai parcurgem. Putem să pornim cu i de la d·d.
Conform observației anterioare i pornește de la d·d. De aceea nu are rost să mergem cu d mai departe de SQRT(n), deoarece nu vom mai găsi i astfel încât i ≤ n.
Iată implementarea optimizată, bazată pe aceste observații:

```
char ciur[2000001];
...
fin>>n;
for ( d = 2; d * d <= n; d++ )
  {       if ( ciur[d] == 0 ) // daca d este prim
    for ( i = d * d; i <= n; i = i + d ) // vom marca numerele din d in d
          {       ciur[i] = 1;    }
}
```

Prime **– varena.ro

Ajungând la capitolul de numere prime, Bianca încearcă sa își facă tema la matematica. Aceasta trebuie sa calculeze atât cel mai mare număr prim mai mic sau egal cu x, cat si suma primelor N numere prime mai mari decât x. Bianca, însa, nu ştie sa rezolve tema si, pentru a nu lua o nota proasta la şcoala, va cere ajutorul.

Cerinţa: Scrieţi un program care calculează atât cel mai mare număr prim mai mic sau egal cu un număr dat x, cat si suma primelor N numere prime mai mari decât x.

Date de intrare: Fişierul de intrare prime.in conţine pe prima linie separate printr-un spaţiu doua numere naturale x si N cu semnificaţia din enunţ.

Date de ieşire: În fişierul de ieşire prime.out se va afisa pe prima linie cel mai mare număr prim mai mic sau egal cu x, iar pe a doua linie suma primelor N numere prime mai mari decât x.

Restricţii: 3 ≤ x ≤ 1.000.000**;** 1 ≤ N ≤ 100.000

- Se garantează ca al N - lea număr prim mai mare decât x nu va depăşi niciodată 3.000.000

Exemplu

prime.in	prime.out
65 5	61 373

Explicaţie: 61 este cel mai mare număr prim ≤ 65
Suma primelor 5 numere prime mai mari decât x este 67 + 71 + 73 + 79 + 83 = 373

```
#include <fstream>
using namespace std;
ifstream  fin( "prime.in");
ofstream fout ( "prime.out");
char ciur[3000001];
int main() {
 int n, x, d, i, a;
 long long suma;
 fin>>x>>n;
 for ( d = 2; d * d <= 3000000; d++ ) // ciurul lui eratostene până la 3000000
  if ( ciur[d] == 0 )
   for ( i = d * d; i <= 3000000; i = i + d )
     ciur[i] = 1;
        a = x;  // cel mai mare număr mai mic sau egal cu x
 while ( ciur[a] == 1 )
  a--;
 // suma primelor n numere strict mai mari decât x
 suma = 0;
 for ( i = 0; i < n; i++ ) {
  x++;
  while ( ciur[x] == 1 )
   x++;
  suma = suma + x;
 }
 fout<< a <<"\n"<<suma;
 return 0;}
```

Kdiv** (clasa a 5-a) – varena.ro

Se dă *n* şi apoi *n* numere naturale, $a_1, a_2, ..., a_n$. Să se spună câte din cele *n* numere au fix *k* divizori numere prime, *k* citit.

Date de intrare: Fişierul de intrare kdiv.in conţine pe prima linie cele două numere, *n* şi *k*. Pe a doua linie se află cele *n* numere.

Date de ieşire: În fişierul de ieşire kdiv.out se va scrie un singur număr, numărul de numere din cele *n* care au exact *k* divizori numere prime.

Restricţii: $1 \le n \le 100\,000$; $0 \le k \le 1000$; $1 \le a_i \le 1\,000\,000$

Exemplu

kdiv.in	kdiv.out	Explicaţie
8 1 39 40 2 34 8 23 31 5	5	Sunt 5 numere cu exact un divizor prim: 2, 8, 23, 31, 5
10 2 33 12 13 10 3 33 11 8 23 45	5	Sunt 5 numere cu exact 2 divizori primi: 33, 12, 10, 33, 45

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

ifstream fin( "kdiv.in" );
ofsteam fout ( "kdiv.out");

// nrdiv[i] = 0 daca i este prim, sau numărul de divizori primi in caz contrar
char nrdiv[1000001];

int main() {
 int n, k, i, d, a, nrk;
 for ( d = 2; d <= 1000000; d++ )
   if ( nrdiv[d] == 0 ) // număr prim
    for ( i = d; i <= 1000000; i = i + d ) // marcam încă un divizor prim
          nrdiv[i]++;
         fin>>n>>k ;
          nrk = 0;
 for ( i = 0; i < n; i++ ) {
   fin>>a;
   if ( nrdiv[a] == k )
    nrk++;
 }

 fout<<nrk;

 return 0;
}
```

5. Sortarea vectorilor

Prin sortare se înţelege aranjarea elementelor unui vector în ordine crescătoare sau descrescătoare.

5.1 Sortarea prin selecţie

Pas 1. Determinam minimul din vector si poziţia sa.
Pas 2. Interschimba minimul cu primul element din vector si astfel minimul ajunge pe poziţia sa finală în vectorul ordonat.
Reluam Pas 1 si 2 pentru vectorul cu ultimele n-1 elemente, care trebuie sortat: determinam minimul dintre ultimele n-1 elemente si îl mutam pe a doua poziţie, apoi facem acelaşi lucru pentru şirul care începe cu al treilea element, până când vectorul este sortat.
Implementarea algoritmului descris:

```
                        // vectorul v de n elemente a fost citit anterior
                        // parcurgem vectorul de la stânga la dreapta
for ( i = 1; i <= n ; i++ ) {
mini = v[i];
p = i; // presupun ca mini este pe poziţia i
for ( j = i + 1; j < n; j++ )
if ( v[j] < mini ) / / memorăm noul mini şi poziţia lui
{ mini = v[j];
p = j; }
// interschimbăm minimul de pe poziţia p cu elementul curent, de pe poziţia i
v[p] = v[i];
v[i] = mini; }
```

5.2 Sortarea prin selecţie directă

O varianta a acestui algoritm care va compara fiecare element din vector cu toate de după el si le va interschimba pe cele care nu sunt în relaţia de ordine corecta.

```
// parcurgem vectorul de la stânga la dreapta
for ( i = 1; i < n ; i++ )
        for ( j = i + 1; j <= n; j++ )
        if ( v[j] < v[j] ) // daca elementele sunt cum nu trebuie
{        aux = v[i]; // interschimba elementele de pe poziţiile i si j
        v[i] = v[j];
        v[j] = aux;      }
```

*Observăm că varianta aceasta este mai simplă ca implementare dar face mai multe operaţii datorită interschimbărilor repetate. Timpul de execuţie al unui astfel de algoritm spunem ca este pătratic T(n) = O(n*n) datorita parcurgerilor imbricate (for în for) .*

Sortare folosind vector de frecvenţă
Se dă un număr n (1<=n<=10000) şi un şir a cu n numere (0<=ai<=100). Să se afişeze şirul sortat.
Soluţie:

```
        cin >> n;
for (i=1;i<=n;i++){
        cin >> x;
        fr[x]++;        }
for (x=0;x<=100;x++)
for (i=1;i<=fr[x];i++)
        cout<<x<<" ";
```

Problemă dată la selecţia pentru centru de pregătire "Hai La Olimpiadă!"

Cătălin are **N** creioane, fiecare creion **i** având lungimea **a[i].** El trebuie sa aleagă 2 creioane ca să scrie tema la mate. Ca un matematician adevărat, lui Cătălin îi place simetria: el vrea să aleagă cele două creioane astfel încât ele să fie cât mai asemănătoare (să aibă diferenţa dintre lungimi minimă). Rolul vostru este să îi ziceţi lui Cătălin care este aceasta diferenţă minimă.

Date de intrare

În fişierul creioane.in se află pe prima linie numărul **N** iar pe a doua line **N** numere.

Date de ieşire

Afişaţi în fişierul creioane.out diferenţa minimă dintre oricare două elemente.

Restricţii şi precizări

$2 \le N \le 100000$

$1 \le a[i] \le 1018$

Pentru 60% din punctaj : $2 \le N \le 1000$

creioane.in	creioane.out	Explicaţii
5 56 1 13 7 18	5	diferenţa minimă este intre creioanele 13 si 18 (18 - 13 = 5)
10 53 37 89 30 66 14 5 97 75 43	7	diferenţa minimă este intre creioanele 30 si 37 (37 - 30 = 7)

Pentru a ajunge la varianta optimă trebuie să facem o observaţie: nu are sens să calculăm diferenţele decât între elemente consecutive. Deci, o să sortăm elementele şi o să calculăm diferenţele doar între elemente de pe poziţii consecutive. De asemenea, trebuie să fim atenţi la limite: elementele sunt < 10^{18} , deci şi vectorul, dar şi variabila în care salvăm diferenţa trebuie să fie long long.

```
#include <fstream>
using namespace std;
ifstream fin("creioane.in");
ofstream fout("creioane.out");
long long v[100001];
int main()
{ int n,i;
long long aux;
   fin>>n;
   for(i=1;i<=n;i++)
     {      fin>>v[i];     }
   for ( i = 1; i < n ; i++ )
          for ( j = i + 1; j <= n; j++ ) // dorim să sortăm crescător
          if ( v[i] >v[j] ) // dacă e cum nu trebuie
{          aux = v[i]; // interschimbăm elementele de pe poziţiile i si j
          v[i] = v[j];
          v[j] = aux;
}
   long long s = v[2]- v[1];
   for(i=3;i<=n;i++){
      if ((v[i] - v[i-1])<s)
{          s= v[i] - v[i-1];            }
   fout<<s;
   return 0;
}
```

6. Şirul lui Fibonacci cu vectori

Şirul lui Fibonacci este un sir infinit de numere, care au la baza o formula simpla: $\mathbf{n_2 = n_1 + n_0}$. Pe baza acestei formule se generează elementele şirului.

Italianul Leonardo of Pisa (cunoscut in matematica drept Fibonacci) a descoperit un sir de numere destul de interesant: **0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597**, si aşa mai departe.

Primele doua elemente ale şirului sunt 0 si 1, iar al treilea element este obţinut adunând primele doua: 0 + 1 = 1. Al patrulea se obţine adunând al treilea număr cu cel de-al doilea (2 + 1 = 3).

Şirul lui Fibonacci in C++

Problema: Sa se genereze şi să se afişeze primii n termeni ai şirului lui Fibonacci. Şirul are primii doi termeni egali cu 1 si fiecare dintre următorii termeni este egal cu suma dintre precedentul si antecedentul.

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

ifstream fin ("fibonacci.in");
ofstream fout ("fibonacci.out");

int fib[1000];
int main()
{
          int  i, n;
          fin >> n;
          fib[1] = 1;
          fib[2] = 1;
  fout << fib[1]<< "  " << fib[2] << "  ";
  for(i = 3; i <= n; i++)
  {
     fib[i] = fib[i-1] + fib[i-2];
     fout << fib[i] << "  ";
  }
  return 0;
}
```

Exersaţi:

Rezolvaţi Problema #423 Fibonacci1 de pe site-ul www.pbinfo.ro.

7. Căutare binară

Căutarea unei valori într-un vector se poate face în două moduri:

- secvențial – presupune analizarea fiecărui element al vectorului într-o anumită ordine (de obicei de la stânga la dreapta). Când se găsește valoarea căutată parcurgerea vectorului se poate opri. În cel mai rău caz, pentru un vector cu n elemente parcurgerea face n pași, complexitatea timp a căutării secvențiale este O(n).
- binar. Căutarea binară se poate face într-un vector numai dacă elementele acestuia sunt în ordine (de obicei crescătoare) după un anumit criteriu (de obicei criteriul este chiar relația de ordine naturală între numere, cuvinte, etc.). Căutarea binară presupune împărțirea vectorului în secvențe din ce în ce mai mici, înjumătățindu-le și continuând cu jumătatea în care se poate afla valoarea dorită (conform ordinii elementelor din vector). Să ne folosim de o analogie: cum căutam un cuvânt în dicționar? Dicționarul are circa 60000 de cuvinte. Dacă am căuta la rând, liniar, ne-ar lua zile să-l găsim. Și atunci deschidem dicționarul undeva, să spunem la mijloc. Ne uităm la ce literă suntem, să spunem 'p'. Dacă noi căutăm cuvântul 'gogoriță', știm că 'g' < 'p', deci vom căuta la stânga. Vom continua la fel, deschidem undeva la jumate în jumătatea rămasă și vedem iarăși la ce literă suntem. Atunci când ajungem la litera 'g' vom continua cu a doua literă. În acest fel vom găsi cuvântul rapid, probabil în jumate de minut, în loc de zile.

Algoritmul căutării binare este următorul:

```
#include <iostream>
using namespace std;
int main()
{
  int a[1001], n, x, st, dr, poz,i;
  cin>>n>>x;
  for(i=1;i<=n;i++)
   {  cin>>a[i];  }
   st=1; dr=n; poz=0;
   while (st <= dr && poz==0)
{
   int mij=(st+dr)/2;
   if (a[mij]==x)  poz=mij;
   else
      if (a[mij]<x) st=mij+1;
            else dr=mij-1;
}
      if (poz!=0)
      cout<<"Valoarea "<< x <<" se afla pe pozitia "<< poz;
      else
      cout<<"Valoarea "<< x <<" nu exista in vector";
   return 0;
}
```

Exersați:

1. Rezolvați problema gogosi(#2297) de pe site-ul www.pbinfo.ro:

Problema "Căutare binară" de pe Varena**

Se dă un şir de numere ordonat crescător cu **N** elemente. Se cere să răspundeţi la **M** întrebări de tipul:

- 0 **x** - cea mai mare poziţie pe care se află un element cu valoarea **x**, sau -1 dacă această valoare nu se găseşte în şir
- 1 **x** - cea mai mare poziţie pe care se află un element cu valoarea mai mică sau egală cu **x** în şir. Se garantează că cel mai mic număr al şirului este mai mic sau egal cu **x**
- 2 **x** - cea mai mică poziţie pe care se află un element cu valoarea mai mare sau egală cu **x** în şir. Se garantează că cel mai mare număr din şir este mai mare sau egal cu **x**

Date de intrare

Pe prima linie a fişierului de intrare cautbin.in se află numărul **N** reprezentând numărul de elemente ale şirului. Pe următoarea linie se găsesc **N** numere reprezentând elementele şirului. Linia a treia conţine numărul **M** reprezentând numărul de întrebări. Apoi urmează **M** linii, fiecare cu unul dintre cele 3 tipuri de întrebări.

Date de ieşire

În fişierul de ieşire cautbin.out se vor afişa **M** linii reprezentând răspunsul la cele **M** întrebări.

Restricţii

- $1 \le \mathbf{N} \le 100\,000$
- $1 \le \mathbf{M} \le 100\,000$
- Elementele şirului vor fi numere strict pozitive şi se vor încadra pe 31 de biţi

Exemplu

cautbin.in	cautbin.out	Explicaţie
5 1 3 3 3 5 3 0 3 1 3 2 3	4 4 2	Ultima apariţie a numărului 3 în şir este pe poziţia 4. Prima apariţie pe poziţia 2.

Cerinţa 0

Se cere să găsim ultima apariţie a lui **x**. Cum procedăm? Mai întâi ne hotărâm asupra testului. Este destul de clar că dacă elementul curent din vector este strict mai mare decât x, atunci nu are sens să facă parte din intervalul nostru. De aici rezultă două lucruri:

1. Testul va fi *if (v[med] > x)*
2. Intervalul de căutare va fi deschis la dreapta, iar intervalul original de căutare va fi [0..n)

La final, când intervalul rămâne de lungime unu ne vom întreba dacă ***v[st] == x***.

Cerinţa 1

Se cere să găsim ultima poziţie a unui element mai mic sau egal cu **x**. În fapt, ea este aceeaşi cerinţă cu cerinţa zero. Dacă rezolvăm această cerinţă vom găsi ultima apariţie a lui **x**, dacă el există. Altfel vom găsi un element strict mai mic. Singura diferenţă faţă de cerinţa zero este că rezultatul va fi valoarea lui **st** indiferent de valoarea lui ***v[st]***.

Cerinţa 2

Se cere să găsim prima poziţie a unui element mai mare sau egal cu **x**. Cum procedăm? La fel, să ne hotărâm asupra testului. Ştim sigur că dacă elementul curent, la poziţia din mijloc, este strict mai mic decât **x**, atunci îl putem scoate în afara intervalului, în partea stângă, deoarece noi căutăm un element mai mare sau egal. De aici rezultă două lucruri:

1. Testul va fi *if (v[med] < x)*
2. Intervalul de căutare va fi deschis la stânga, iar intervalul original de căutare va fi (-1..n-1]

Iată și programul bazat pe aceste idei:

```
#include <fstream>
using namespace std;
int v[100000];
ifstream fin ( "cautbin.in" );
ofstream fout ( "cautbin.out" );
int main() {
int n, m, t, x, i, st, dr, med;
fin>>n;
 for ( i = 0; i < n; i++ )
          fin>>v[i];
          fin>>m;
 for ( i = 0; i < m; i++ ) {
   fin>>t>>x;
  if ( t < 2 ) { // cazurile 0 si 1 sunt similare
   // căutam cea mai mare poziție pe care se afla <= x in intervalul [0 n)
   st = 0; // capătul din stingă este primul in intervalul [st dr), deci 0
   dr = n; // capătul din dreapta este primul in afara intervalului, deci n
   while ( dr - st > 1 ) {
    med = (dr + st) / 2;
    if ( v[med] > x ) // element strict mai mare?
     dr = med; // putem scoate poziția med in dreapta afara intervalului
    else
     st = med; // altfel putem mari limita st către med
   }
   if ( t == 0 && v[st] != x ) // in cazul 0 x trebuie sa fie găsit
    st = -2; // vom aduna unu la afișare
  } else {    // căutam prima apariție a unui număr <= x in intervalul (-1 n-1]
   st = -1;   // capătul din stânga este ultimul in afara intervalului (-1 n-1]
   dr = n-1;  // capătul din dreapta este ultimul in interval, deci n-1
   while ( dr - st > 1 ) {
    med = (dr + st) / 2;
    if ( v[med] < x ) // element strict mai mic?
     st = med; // putem scoate poziția med in stânga in afara intervalului
    else
     dr = med; // altfel putem micșora limita dr către med
   }
   st = dr; // pentru a afișa totdeauna st ca rezultat
  }
  fout<<st + 1; // ajustam poziția cu unu, ca in cerința
 }
 return 0;
```

8. Interclasarea vectorilor

Considerăm două tablouri unidimensionale cu elemente numere întregi **ordonate crescător**. Se dorește construirea unui alt tablou, care să conțină valorile din cele două tablouri, în ordine.

O soluție **foarte eficientă** este **interclasarea**:

- considerăm două tablouri, cu n, respectiv m elemente, **ordonate crescător**
- cele două tablouri se parcurg concomitent;
- se alege valoarea mai mică dintre cele două elemente curente
  - se adaugă în al treilea tablou
  - se avansează numai în tabloul din care am ales valoarea de adăugat
- parcurgerea unuia dintre cele două tablouri se încheie
- toate elementele din celălalt tablou, neparcurse încă, sunt adăugate în tabloul destinație
- tabloul destinație are k = n + m elemente

```
int main()
{
int a[100001],b[100001], c[200002], i, j, k, n, m ;
//citire a[ ] cu n elemente
//citire b[ ] cu m elemente
i=1;j=1;k=0;
while(i<=n && j<=m)
if (a[i]<b[j]){
        k++;
c[k]=a[i];
        i++;
}
else{
        k++;
c[k]=b[j];
        j++;
}
while(i<=n){
        k++;
c[k]=a[i];
        i++;
}
while (j<=m){
        k++;
c[k]=b[j];
        j++;
}
```

Observație: k++,c[k]=a[i],i++ poate fi înlocuit cu c[++k]=a[i++]. (analog pentru b[j]) .

Observație: Doar una dintre instrucțiunile while(i < n)... și while(j < m)... se va executa, deoarece exact una dintre condițiile i < n și j < m este adevărată. În prima structură repetitivă, la fiecare pas, doar una dintre variabilele i și j se mărește, deci la final una dintre condiții este adevărată și una este falsă.

Algoritmul de interclasare este **foarte eficient**. El are complexitate O(n+m). De asemenea, este posibilă și interclasarea valorilor din două fișiere, singura condiție este ca valorile să fie ordonate.

9. Probleme date la olimpiadele de informatică

Problema 2 – flori* - OJI 2012

Lizuca are n flori ornamentale de înălţimi h_1, h_2, ..., h_n, exprimate în centimetri. Pentru a uda plantele, Lizuca stabileşte următorul program: în prima zi va alege o plantă pe care o va uda, în a doua zi va alege două plante pe care le va uda, în ziua a treia va alege trei plante pe care le va uda şi aşa mai departe. Dacă o plantă este udată într-o anumită zi, atunci creşte 1 centimetru până la sfârşitul acelei zile, iar dacă nu este udată, rămâne la înălţimea pe care o avea la sfârşitul zilei precedente.

Cerinţă

Scrieţi un program care determină:

a) un număr natural S, exprimat în centimetri, reprezentând suma înălţimilor finale ale tuturor plantelor, dacă Lizuca le-ar uda după procedeul descris, timp de n zile;
b) un număr natural K, reprezentând numărul maxim de zile în care Lizuca poate uda florile după procedeul descris anterior, astfel ca la sfârşitul celei de a K-a zi, nicio plantă ornamentală să nu atingă înălţimea H.

Date de intrare

Prima linie a fişierului flori.in conţine două numere naturale n şi H, separate printr-un spaţiu, având semnificaţia din enunţ.

Linia a doua conţine n numere naturale: h_1, h_2, ..., h_n, separate prin câte un singur spaţiu, reprezentând înălţimile iniţiale ale plantelor.

Date de ieşire

Fişierul flori.out va conţine pe prima linie un număr natural S având semnificaţia descrisă în cerinţa a).

A doua linie va conţine un număr natural K, având semnificaţia descrisă în cerinţa b).

Restricţii şi precizări

- $1 \le N, H \le 100$
- $1 \le h_1, h_2, \ldots h_n < H$
- O plantă poate fi udată o singură dată pe zi.

Exemple

flori.in	flori.out	Explicaţie - cerinţa b)
3 4 2 1 1	10 2	Dacă în prima zi se udă planta 3, atunci înălţimile devin: 2 1 2 Dacă în a doua zi se udă plantele 1 şi 2, atunci înălţimile devin: 3 2 2 Procedeul se opreşte aici, deoarece în ziua a treia, ar trebui să se ude toate plantele, iar planta 1 ar ajunge să aibă înălţimea 4

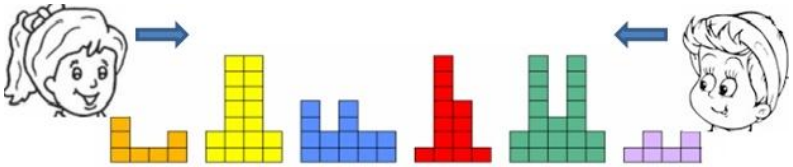
flori.in	flori.out	Explicaţie - cerinţa b)
4 5 1 3 2 1	17 3	Dacă în prima zi se udă planta 1, atunci înălţimile devin: 2 3 2 1 Dacă în a doua zi se udă plantele 1 şi 4, atunci înălţimile devin: 3 3 2 2 Dacă în a treia zi se udă plantele 1, 3 şi 4, atunci înălţimile devin: 4 3 3 3.

Rezolvare:

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

ifstream fin("flori.in");
ofstream fout("flori.out");

int h[100];
int main()
{       int n, k, H;
        int s=0;
        fin >> n >> H;
        for (   i = 0; i < n; i ++)
        {
                fin >> h[i];
                s =  s + h[i];
        }
        // Cerinta a)
        for (  i = 1; i <= n; i ++)
                s= s + i;
        fout << s << '\n';

        // Cerinta b)
/* Pentru a avea garanţia că maximul înălţimilor plantelor creşte în fiecare zi cu o valoare cât mai mică, trebuie ca în fiecare zi k trebuie să fie udate primele k plante, în ordinea crescătoare a înălţimilor (k = 1, 2, … K).  Se afişează cea mai mare valoare a lui k, pentru care nicio plantă nu atinge înălţimea H. */
        bool ok = true;
        while ( ok )
        {
                for ( i = 0; i < n - 1; i ++)
                        for (   j = i + 1; j < n; j ++)
                                if ( h[i] > h[j] )
                                {
                                        int aux = h[i];
                                        h[i] = h[j];
                                        h[j] = aux;
                                }

                for ( i = 0; i <= k && ok; i ++ )
                        if ( h[i] + 1 >= H )
                                ok = false;

                if ( ok ) {
                        for ( i = 0; i <= k; i ++)
                                h[i] ++;
                        k++;
                }
        }
        fout << k << '\n';

        return 0;
}
```

Problema 1– clădiri* - OJI 2013

Având mai multe cuburi la dispoziţie, Crina şi Rareş au hotărât să construiască clădiri prin alipirea a două sau mai multor turnuri. Turnurile au fost obţinute prin aşezarea cuburilor unul peste celălalt. Înălţimea unui turn este dată de numărul de cuburi din care este format. Clădirile construite au fost aşezate în linie, una lângă alta formând astfel o stradă, pe care cei doi copii se vor plimba.



Pentru numerotarea clădirilor Crina şi Rareş au stabilit următoarele reguli:

- Crina porneşte dintr-un capăt al străzii, iar Rareş din celălalt capăt al acesteia; fiecare dintre ei traversează strada complet, trecând prin dreptul fiecărei clădiri
- Crina lipeşte pe fiecare clădire câte un bileţel pe care scrie înălţimea turnurilor din care aceasta este construită, în ordinea în care ea le vede când trece prin dreptul lor (de exemplu, pentru imaginea de mai sus, Crina va lipi pe prima clădire un bileţel pe care va scrie numărul 3112 deoarece, primul turn e format din 3 cuburi, următoarele două turnuri ale acestei clădiri sunt formate din câte un cub iar cel de-al patrulea turn e format din 2 cuburi);
- Rareş va proceda la fel, dar începe plimbarea din celalalt capăt al străzii. În exemplul din imagine, el va lipi pe prima clădire pe care o întâlneşte un bileţel pe care scrie numărul 2121.

La finalul plimbării, Crina şi Rareş îşi dau seama că există clădiri pe care au lipit amândoi bileţele cu numere identice.

Cerinţe

a) Care este înălţimea celui mai înalt turn şi care este numărul clădirilor care au în construcţia lor un astfel de turn?
b) Care este numărul clădirilor pe care cei doi copii au lipit bileţele cu numere identice?
c) Care este cel mai mic număr de cuburi necesar pentru a completa clădirile astfel încât, pe fiecare clădire, bileţelul pe care îl va lipi Crina să conţină acelaşi număr cu cel pe care îl va lipi Rareş? Cuburile din care a fost construită iniţial clădirea nu se pot muta.

Date de intrare

Din fişierul cladiri.in se va citi de pe prima linie un număr natural N, reprezentând numărul clădirilor de pe stradă, iar de pe următoarele N linii câte un număr natural cu toate cifrele nenule, reprezentând numerele scrise de Crina pe bileţele, în ordinea în care au fost lipite de ea pe clădiri.

Date de ieşire

În fişierul cladiri.out se vor scrie pe prima linie două numere naturale despărţite printr-un singur spaţiu ce reprezintă, în ordine, valorile cerute la cerinţa a). Pe cea de-a doua linie a fişierului se va scrie un număr natural, mai mare sau egal cu zero, reprezentând răspunsul la cerinţa b). Pe cea de-a treia linie a fişierului se va scrie un număr natural, mai mare sau egal cu zero, reprezentând răspunsul la cerinţa c).

Restricţii şi precizări

- $1 \leq N \leq 10000$
- Fiecare clădire este alcătuită din cel mult 9 turnuri, iar înălţimea fiecărui turn este exprimată printr-o cifră nenulă.

Exemplu

cladiri.in	cladiri.out	Explicaţie
6 3112 2772 42422 1741 27372 1212	7 3 2 8	Cel mai înalt turn este format din 7 cuburi. Sunt 3 clădiri care au în construcţia lor turnuri cu această înălţime, cele pe care Crina lipeşte numerele: 2772, 1741 şi 27372. Rareş lipeşte pe clădiri bileţele cu numerele: 2121, 27372, 1471, 22424, 2772 şi 2113. Două dintre aceste clădiri au primit aceleaşi numere şi de la Crina: 2772 şi 27372. Valoarea determinată conform cerinţei c) este 8. Se adaugă un cub la clădirea cu numărul 3112, 2 cuburi la cea cu numărul 42422, 3 cuburi la clădirea cu numărul 1741 şi 2 cuburi la cea cu numărul 1212.

Rezolvare:

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

ifstream fin("cladiri.in");
ofstream fout("cladiri.out");

int main(){
int x, y, inv;
int i, p, j, n, cmax, maxim, nrmaxim, sum, nrpal;
   fin>>n;
   for (i=1;i<=n;i++){
      fin>>x;
      y = x;
      inv = 0;
      cmax = 0;
      p = 0;
      while (y) {
         if (y%10 > cmax)
            cmax = y%10;
         inv = inv*10 + y%10;
         p++;
         y = y/10;
      }
      if (cmax > maxim) {
         maxim = cmax;
         nrmaxim = 1;
      } else
         if (cmax == maxim)
            nrmaxim ++;
      if (x == inv)
         nrpal ++;
      for (j=1; j<=p/2; j++) {
         if (x % 10 > inv % 10)
            sum = sum + (x%10-inv%10);
         else
            sum = sum +(inv%10 - x%10);
         x = x/10;
         inv =inv/10;

      }
   }
   fout<<maxim<<" "<<nrmaxim<<" "<<nrpal<< " "<<sum;
   return 0;
}
```

Problema 2 – galbeni ** - OJI 2013

După ce au descoperit ascunzătoarea piratului Spânu, marinarii de pe corabia "Speranţa" au hotărât să ofere sătenilor o parte din comoara acestuia. Întrucât comoara avea un număr nelimitat de bani din aur, numiţi galbeni, singura problemă a marinarilor a fost regula după care să împartă banii. După îndelungi discuţii au procedat astfel: i-au rugat pe săteni să se aşeze în ordine la coadă şi să vină, pe rând, unul câte unul pentru a-şi ridica galbenii cuveniţi. Primul sătean a fost rugat să îşi aleagă numărul de galbeni, cu condiţia ca acest număr să fie format din exact **K** cifre. Al doilea sătean va primi un număr de galbeni calculat astfel: se înmulţeşte numărul de galbeni ai primului sătean cu toate cifrele nenule ale acelui număr, rezultatul se înmulţeşte cu **8** şi apoi se împarte la **9** păstrându-se doar ultimele **K** cifre ale câtului împărţirii. Dacă numărul obţinut are mai puţin de **K** cifre, atunci acestuia i se adaugă la final cifra 9, până când se completează **K** cifre. Pentru a stabili câţi galbeni primeşte al treilea sătean, se aplică aceeaşi regulă, dar pornind de la numărul de galbeni ai celui de-al doilea sătean. Regula se aplică în continuare fiecărui sătean, plecând de la numărul de galbeni primiţi de săteanul care a stat la coadă exact în faţa lui.

Cerinţa

Cunoscând numărul de galbeni aleşi de primul sătean, determinaţi numărul de galbeni pe care îl va primi al **N**-lea sătean.

Date de intrare

Fişierul **galbeni.in** conţine pe prima linie cele **3** numere naturale nenule **S, K, N** separate prin câte un spaţiu, unde **S** reprezintă numărul de galbeni ales de primul sătean, **K** este numărul de cifre ale numărului **S**, iar **N** reprezintă numărul de ordine al săteanului pentru care se cere să determinaţi numărul de galbeni primiţi.

Date de ieşire

Fişierului **galbeni.out** conţine pe unica sa linie un număr natural reprezentând rezultatul determinat.

Restricţii

- **2 ≤ N ≤ 1 000 000 000**
- **1 ≤ K ≤ 3**
- Se garantează că **S** are exact **K** cifre.

Exemplu

galbeni.in	galbeni.out	Explicaţie
51 2 3	**77**	Primul sătean a luat 51 de galbeni. Cel de al doilea sătean va primi 26 de galbeni (51 se înmulţeşte cu cifrele nenule 51*5*1=255, 255 se înmulţeşte cu 8 =2040. Câtul împărţirii lui 2040 la 9 = 226, ultimele două cifre fiind 26). Celui de al treilea sătean va primi 77 de galbeni (26 se înmulţeşte cu cifrele nenule 26*2*6=312, 312 se înmulţeşte cu 8 şi obţinem numărul 2496. Câtul împărţirii dintre 2469 şi 9 este 277, ultimele două cifre fiind 77)
10 2 3	**96**	Primul sătean primeşte 10 galbeni. Pentru a calcula câţi galbeni primeşte al doilea sătean procedăm astfel: înmulţim 10 cu cifele sale nenule: 10*1 = 10, apoi cu 8, 10*8 = 80. Câtul împărţirii lui 80 la 9 este 8. Acest număr având mai puţin de k=2 cifre, se adaugă la finalul său cifra 9 şi se obţine 89. Pentru al treilea sătean se pleacă de la 89 (89*8*9=6408, 6408*8=51264, câtul împărţirii lui 51264 la 9 este 5696, ultimele două cifre sunt 96)

Timp maxim de execuţie: 0.5 secunde/test
Memorie totală disponibilă: 32 MB
Dimensiunea maximă a sursei : 10 KB

Rezolvare soluția brută

Simulăm modul de construire a următoarei valori aşa cum este descris în enunţul problemei. O implementare corectă nu se va încadra în timp pe toate testele.

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

ifstream fin("galbeni.in");
ofstream fout("galbeni.out");

int n, s, k, x, p, c, inv, i, z;

int main() {

   fin>>s>>k>>n;
   z = 1;
   for (i=1;i<=k;i++)
      z = z * 10;
   for (i=2;i<=n;i++) {
      x = s;
      p = s;
      while (x!=0) {
         if (x%10 != 0) {
            p = p*(x%10);
         }
         x = x/10;
      }
      //p = numarul format dupa inmultire
      p = p*8;
      p = p/9;

      p = p%z;

      while (p < z/10)
         p = p*10 + 9;
      s = p;
   }

   fout<<s<<"\n";
   return 0;
}
```

Problema 1 – cifre OJI 2016

Elevii clasei pregătitoare se joacă la matematică cu numere. Învățătoarea are un săculeț plin cu jetoane, pe fiecare dintre ele fiind scrisă câte o cifră. Fiecare elev și-a ales din săculeț mai multe jetoane, cu care și-a format un număr. Pentru ca totul să fie mai interesant, elevii s-au grupat în perechi. Doamna învățătoare a oferit fiecărei perechi de elevi câte o cutiuță pentru ca cei doi să își pună împreună jetoanele. De exemplu, dacă unul din elevii unei echipe și-a ales jetoane cu care a format numărul 5137131 iar celălalt elev și-a ales jetoane cu care a format numărul 6551813, atunci cutiuța echipei va conține 5 jetoane cu cifra 1, câte 3 jetoane cu cifra 3 și 5 și câte un jeton cu cifrele 6, 7 și 8.

Doar Andrei stătea supărat pentru că numărul de elevi al clasei era impar iar el nu avea partener, motiv pentru care nu și-a mai ales jetoane. Din această cauză, doamna învățătoare i-a spus:
"- Alege o echipă din a cărei cutiuță poți lua o parte din jetoane, dar ai grijă ca fiecare dintre cei doi elevi să-și mai poată forma numărul lui din jetoanele rămase, iar tu să poți forma un număr nenul cu jetoanele extrase!".
Dar cum Andrei nu se mulțumea cu puțin, a vrut să aleagă acea echipă din a cărei cutiuță își poată forma un număr de valoare maximă folosind jetoanele extrase.

Cerință

Scrieți un program care să citească numărul N de cutiuțe și numerele formate de elevii fiecărei perechi și care să determine:
1) Numărul de cutiuțe din care Andrei poate lua jetoane respectând condiția pusă de doamna învățătoare;
2) Care este cel mai mare număr nenul pe care îl poate forma Andrei respectând aceeași condiție.

Date de intrare

Fișierul cifre.in conține pe prima linie numărul natural P reprezentând cerința din problemă care trebuie rezolvată. Pe a doua linie numărul natural N, iar pe următoarele N linii câte două numere naturale separate printr-un spațiu reprezentând numerele formate de elevii fiecărei perechi.

Date de ieșire

Dacă valoarea lui P este 1, fișierul de ieșire cifre.out va conține pe prima linie un număr natural reprezentând rezolvarea primei cerințe, adică numărul de cutiuțe din care Andrei poate lua jetoane.
Dacă valoarea lui P este 2, fișierul de ieșire cifre.out va conține pe prima linie un număr natural reprezentând rezolvarea celei de a doua cerințe, adică numărul maxim pe care îl poate forma Andrei.

Restricții și precizări

- $0 < N \leq 10000$
- 1 ≤ numărul de jetoane al fiecarui elev ≤ 9
- 0 ≤ cifra scrisă pe orice jeton ≤ 9
- Se garantează că există cel puțin o cutiuță din care Andrei își poate forma număr nenul

Exemplu

cifre.in	cifre.out	Explicație
1 3 1010 2000 12 34 1515 552	1	Cu jetoanele extrase din prima cutiuță Andrei nu poate forma un număr diferit de 0. Din a doua cutiuță Andrei nu poate lua jetoane astfel încât cei doi elevi să își mai poată forma numerele 12 și 34. Andrei poate extrage jetoane doar din a treia cutiuță(două jetoane cu cifra 5).
2 5 16815 38861 12 385 5137131 6551813 15033 11583 4704 240	5311	Numărul maxim pe care Andrei îl poate forma este 5311 și se obține din cutiuța a treia.

Descriere soluţie:

Se determină cifrele comune pentru cele 2 numere ce formează o pereche, utilizând doi vectori de apariţii. Pentru fiecare cifră comună se determină minimul dintre a si b, unde a este numărul de apariţii a cifrei în primul număr iar b este numărul de apariţii a cifrei în cel de-al doilea număr.
Dacă există cifre comune se formează valoarea maximă luând toate cifrele comune în ordinea descrescătoare a valorii lor.
Dacă valoarea obţinută este strict pozitiva, se numără ca soluţie, actualizându-se daca este cazul si valoarea maxima pe care o poate forma Andrei.

Rezolvare:

```
#include<fstream>
using namespace std;
ifstream f("cifre.in");
ofstream g("cifre.out");
int a[10],b[10],n,i,x,y,j,k,p,cod,cate,maxim, minim;
int main()
{
  f>>p>>n;
  for(i=1;i<=n;i++)
  {
    f>>x>>y;
    for(j=0;j<10;j++) a[j]=b[j]=0;
    while(x)
        {       a[x%10]++;
                x=x/10;         }
    while(y)
        {       b[y%10]++;
                y=y/10;         }
    cod=0;
    for(j=9;j>=0;j--)
      if(a[j]>0 && b[j]>0)
      {
        if(a[j]<b[j]) minim=a[j]; else minim=b[j];
        for(k=1;k<=minim;k++) cod=cod*10+j;
      }
    if(cod) cate++;
    if(cod>maxim) maxim=cod;
  }
  if(p==1) g<<cate;
    else g<<maxim;
  g.close();
  return 0;
}
```

Problema 2 – ordine* – *OJI 2016*

Gigel a primit de ziua lui un joc cu bile. Jocul conţine n bile numerotate cu numerele naturale distincte de la 1 la n. Jucându-se, Gigel a amestecat bilele astfel încât acum ele nu mai sunt în ordine. Ca să le pună înapoi în cutia jocului, Gigel ia de pe masă bilele una câte una, şi le pune în cutie formând un şir. Însă Gigel se joacă şi acum, astfel încât el nu pune bilele la rând, una după alta, ci are o regulă pe care o respectă cu strictețe. Astfel, Gigel încearcă să plaseze fiecare bilă pe care a luat-o de pe masă exact la mijlocul şirului de bile deja format. Dacă acest lucru nu este posibil (şirul are lungime impară), atunci el plasează bila la sfârşitul şirului de bile deja format. După ce toate bilele au fost puse în cutie, Gigel îşi dă seama că nu a notat ordinea în care a luat bilele de pe masă şi, în mod firesc, îşi pune problema dacă nu cumva poate deduce acest lucru din şirul de bile pe care tocmai l-a format.

Cerinţe

Cunoscându-se numărul de bile şi configuraţia finală a bilelor în şir să se determine:

1. numărul ultimei bile luate de pe masă;
2. ordinea în care bilele au fost luate de pe masă.

Date de intrare

Fişierul de intrare ordine.in conţine pe prima linie numărul n de bile. Pe linia a doua a fişierului de intrare se găsesc n numere naturale, cu valori între 1 şi n, separate prin câte un spaţiu, care reprezintă şirul de bile obţinut de Gigel în cutie. Linia a treia conţine una dintre valorile 1 sau 2 reprezentând cerinţa 1, dacă se cere determinarea ultimei bile luate de Gigel de pe masă, respectiv cerinţa 2, dacă se cere determinarea ordinii în care Gigel a luat bilele de pe masă.

Date de ieşire

Fişierul de ieşire ordine.out va conţine pe prima linie o valoarea naturală reprezentând numărul ultimei bile luate de Gigel, dacă cerinţa a fost 1, respectiv n numere naturale, cu valori cuprinse între 1 şi n, separate prin câte un spaţiu, care reprezintă ordinea în care Gigel a luat bilele de pe masă, dacă cerinţa a fost 2.

Restricţii

- 1 <= n <= 250000
- Pentru cerinţa 1 se acordă 30% din punctaj, iar pentru cerinţa 2 se acordă 70% din punctaj.

Exemple

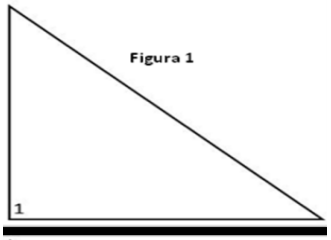
ordine.in	ordine.out
7 1 7 2 5 3 4 6 1	5
7 1 7 2 5 3 4 6 2	1 3 7 4 2 6 5

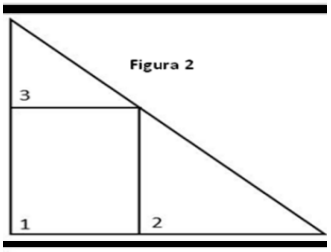
Rezolvare:

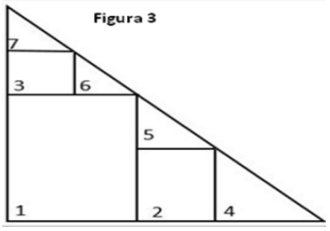
```
#include <fstream>
using namespace std;

#define DIM 1000001

ifstream fin("ordine.in");
ofstream fout("ordine.out");

int b[DIM];
int n, i, j, tip;

int main()
{
  //citire date de intrare
 //1   7   2   5   3   4   6

  fin >> n;
  j = 1;
  for (i = 1; i <= n; ++i)
  {
    fin >> b[j]; //1    3     7      4          2          6          5  - b[j]
    j = j + 2;     //1      2          3          4          5          6          7  - j
    if (j > n) j = 2;
  }
  fin >> tip;
  if (tip == 2)   //rezolv cerința 2
    //afișez rezultatul
    for (i = 1; i <= n; ++i) fout << b[i] << ' ';
  else   //rezolv cerința 1
    fout << b[n];       //e ultima bila

 fout << '\n';

  return 0;
}
```

Problema 1 - desen ONI 2018 – clasa a V-a

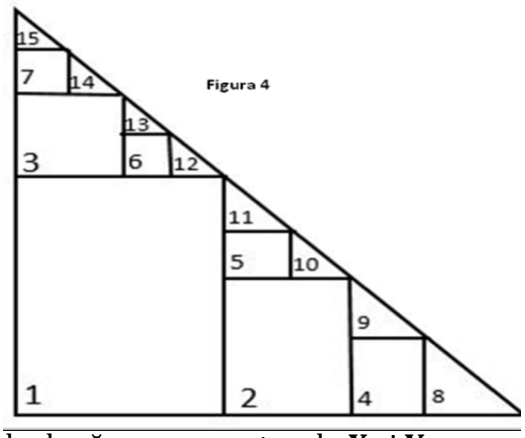
La ora de desen, Gigel a primit ca temă un desen care să fie realizat după următorul algoritm:
Pas 1: se desenează un triunghi, numerotat cu 1, ca în **Figura 1**;
Pas 2: se împarte triunghiul 1 în trei poligoane (un dreptunghi şi două triunghiuri numerotate cu 2 şi 3) trasând două segmente ca în **Figura 2** ;
Pas 3: fiecare triunghi dintre cele două obţinute la **Pas 2**, se împarte în câte un dreptunghi şi câte două triunghiuri (numerotate cu 4,5,6,7) trasând câte două segmente ca în **Figura 3** ;
Pas 4: fiecare triunghi dintre cele patru obţinute la **Pas 3**, se împarte în câte un dreptunghi şi câte două triunghiuri (numerotate cu 8,9,10, 11,12,13,14,15) trasând câte două segmente ca în **Figura 4** ;
.....
Pas N: fiecare triunghi dintre triunghiurile obţinute la **Pas N-1,** se împarte în câte un dreptunghi şi câte două triunghiuri trasând câte două segmente.
Dacă valoarea lui **K** este ultimul număr folosit pentru numerotarea triunghiurilor obţinute la **Pas N-1,** atunci triunghiurile rezultate la **Pas N** vor fi numerotate cu numerele naturale distincte consecutive **K**+1, **K**+2, **K**+3,... etc.







Cerinţe
Scrieţi un program care să citească numărul natural **K** şi să determine:
1. **cel mai mic** număr **X** şi **cel mai mare** număr **Y** dintre numerele folosite pentru numerotarea triunghiurilor obţinute la pasul în care este obţinut şi triunghiul numerotat cu **K**;
2. numerele triunghiurilor care au fost împărţite conform algoritmului din enunţ astfel încât să fie obţinut triunghiul numerotat cu **K**.



Date de intrare
Fişierul de intrare desen.in conţine pe prima linie un număr natural **C** reprezentând cerinţa din problemă care trebuie rezolvată (1 sau 2). Fişierul conţine pe a doua linie numărul natural **K**.
Date de ieşire
Dacă **C**=1, atunci prima linie a fişierului de ieşire desen.outconţine cele două numere naturale **X** şi **Y**, separate printr-un singur spaţiu, reprezentând răspunsul la cerinţa 1 a problemei.
Dacă **C**=2, atunci prima linie a fişierului de ieşire desen.out conţine un şir de numere naturale ordonate crescător, separate prin câte un spaţiu, reprezentând răspunsul la cerinţa 2 a problemei**.**
Restricţii şi precizări
- 2 ≤ K ≤ 9223372036854775807 (=263-1)
- doar triunghiurile sunt numerotate

Exemple desen.in	desen.out	***Explicaţie***
1 13	8 15	Cerinţa este 1, K=13. Aşa cum arată în Figura 4, la Pas 4 se obţin triunghiurile numerotate cu X=8, 9, 10, 11, 12, 13, 14, Y=15.
2 13	1 3 6	Cerinţa este 2, K=13. Aşa cum arată Figura 4, triunghiul numerotat cu K=13 se obţine din triunghiul 6. Triunghiul 6 este obţinut din triunghiul 3 care este obţinut din triunghiul 1.

Observăm că:	Cel mai mic număr folosit la numerotarea triunghiu-rilor obţinute la acest pas	Cel mai mare număr folosit la numerotarea triunghiuri-lor obţinute la acest pas	Număr triunghiuri obţinute la acest pas
Pas 1:	**1**	**1**	**1**
Pas 2:	**2**	**3**	**2**
Pas 3:	**4**	**7**	**4**
Pas 4:	**8**	**15**	**8**
.....			
Pas N:	2^{N-1}	$2^{N}-1$	2^{N-1}

Iată şi rezolvarea bazată pe aceste observaţii:

```
#include <iostream>
#include <fstream>
using namespace std;
ifstream f("desen.in");
ofstream g("desen.out");
unsigned long long C, n, p=1,k,v[100], i;
int main()
{
  f>>C>>n;
  if(C==1)
  {
    while(p<=n)
    {
      p=p*2;
    }
    g<<p/2<<" "<<p-1;
  }
  if(C==2)
  {
    while(n>1)
    {
      v[++k]=n/2; // x1=[K/2], x2=[x1/2], x3=[x2/2], ...., xN =1
      n=n/2;
    }
    for(i=k; i>=1; i--)
    {
      g<<v[i]<<" ";
    }
  }
  return 0;
}
```

Problema 3 – pyk* ONI 2018 – clasa a V-a (prima cerinţă)

Fie **k**, **n** şi **y** trei numere naturale.
Fie **X** un şir format din **n** numere naturale: **x1, x2, x3,..., xn.**

Cerinţe
Scrieţi un program care să citească numerele **n**, **x1, x2, x3,..., xn** şi care să determine:
1. **cel mai mic** număr şi **cel mai mare** număr din şirul **X,** formate doar din cifre identice;

Date de intrare
Fişierul de intrare **pyk.in** conţine:
- pe prima linie, numărul natural **n**;
- pe a doua linie, cele **n** numere naturale **x1, x2, x3,..., xn**, separate prin câte un singur spaţiu.

Date de ieşire
Prima linie a fişierului de ieşire **pyk.out** conţine două numere naturale, separate printr-un singur spaţiu, reprezentând răspunsul la cerinţa **1** a problemei. Dacă nu există astfel de numere, prima linie a fişierului va conţine valoarea **1**.
Scrierea în fişier a acestor factori primi se va face în ordinea crescătoare a valorii lor.

Restricţii şi precizări
- **2 ≤ n ≤ 50 000**
- **2 ≤ x1,x2,x3,...,xn ≤ 10 000**

Exemple: pyk.in	**pyk.out**	*Explicaţie*	
7 **122 1111 5 4 88 123 999**		**4 1111**	Cerinţa este **1, n=7.** Numerele din şirul **X** formate doar din cifre identice sunt: **1111, 5, 4, 88, 999** . Cel mai mic număr dintre acestea este **4**, iar cel mai mare este **1111**.

Descrierea soluţiei

Cerinţa 1. Se pot folosi două variabile xmin şi xmax pentru a memora valorile cerute. Se citesc succesiv numerele din fişier. Se compară cifrele fiecărui număr. Dacă cifrele sunt identice, atunci numărul curent se compară cu xmax şi xmin curent, actualizându-se valorile acestora***.***

Rezolvare:

```
#include <iostream>
#include <fstream>

using namespace std;

ifstream f("pyk.in");
ofstream g("pyk.out");
int  n, i, t, ok, x, maxim, minim=99999;
int main()
{
          f>>n;
      for(i=1; i<=n; i++)
      {
         f>>x;
         t=x;
         ok=1;
         while(t>0 and ok)
         {
            if(t%10!=x%10)
            {
               ok=0;
            }
            t=t/10;
         }
         if(ok==1 and x>maxim)
            maxim=x;
         if(ok==1 and x<minim)
            minim=x;
      }
      if(maxim!=0)
         g<<minim<<" "<<maxim;
      else
         g<<1;

          return 0;
  }
```

Bibliografie

1. LICA Dana, PAŞOI Mircea – "Informatica - Fundamentele Programării", Bucureşti, Editura L&S Soft, 2005;

2. BĂICAN Diana Carmen, CORITEAC Melinda Emilia, Informatică şi TIC, Manual pentru clasa a VI-a Editura Didactică şi Pedagogică;

3. MILOŞESCU Mariana, Informatică, Manual pentru clasa a IX-a, Editura Didactică şi Pedagogică R.A.

Bibliografie web:

www.pbinfo.ro

http://campion.edu.ro/index.php

www.varena.ro

Anexă

Programa clasei a V-a

- Tipuri simple de date: tipuri de date întregi, reale, caracter
- Structura liniara
- Structura alternativa
- Structura repetitiva cu test inițial
- Prelucrarea cifrelor unui număr
- Divizibilitate: divizori, multipli
- Calculul unor sume/produse cu termenul general dat
- Fişiere text.

Programa clasei a VI-a

- Tipuri simple de date: tipuri de date întregi, reale, character.
- Prelucrarea cifrelor unui număr.
- Divizibilitate: divizori, numere prime, descompunere în factori primi, cmmdc, cmmmc
- Şirul lui Fibonacci
- Tablouri unidimensionale
- Căutare secvențială în tablou
- Căutare binară
- Vector de apariții şi vector de frecvențe
- Ordonarea elementelor unui vector
- Fişiere text.

Programa clasei a VII-a si a VIII-a

- Prelucrarea datelor numerice (întregi şi reale) si de tip caracter
- Algoritmi elementari
- Prelucrarea cifrelor unui număr
- Divizibilitate (divizori, cmmdc, cmmc, numere prime, descompunere in factori primi)
- Şiruri recurente (Şirul lui Fibonacci, şiruri generate după o formulă matematică)
- Tablouri unidimensionale
- Vectori de frecvente
- Algoritmi de sortare
- Căutare binară
- Interclasare
- Inserarea/ştergerea elementelor dintr-un vector