

Dan Pracsiu

Probleme de informatică

pentru liceu, facultate
și interviuri de angajare



- > Standard Template Library (STL)
- > Tablouri unidimensionale

Editura Paralela 45

Cuprins



<i>Argument</i>	3
Partea I – Containere STL	
1. vector	6
2. set.....	18
3. multiset	24
4. map	30
5. unordered_map	35
6. priority_queue.....	44
7. stack	53
8. queue.....	61
Partea a II-a – Tablouri unidimensionale	
9. Ciurul lui Eratostene	70
10. Sume parțiale. Maxime și minime parțiale	77
11. Pivot.....	86
12. Sortări	89
13. Interclasări	94
14. Two Pointers	98
15. Vectori de frecvență, vectori caracteristici.....	112
16. Parcurgerea vectorilor.....	129
17. Căutare binară.....	149
18. Generări combinatoriale	159
19. Meet in the Middle.....	165
20. Potrivirea șirurilor.....	170
<i>Bibliografie</i>	175

PARTEA I

Containere STL

Standard Template Library, sau prescurtat STL, pune la dispoziție componente software eficiente și generice și conține structuri de date și algoritmi de bază implementați eficient și care pot fi apelați cu ușurință, astfel că STL devine și o bibliotecă de algoritmi clasici care pot fi utilizați cu ajutorul containerelor.

STL se bazează pe containere, colecții de obiecte ce gestionează optim memoria și care au definite funcții specifice pentru un acces rapid la componentele sale. Un container este un obiect care memorează obiecte. Containerul este, de fapt, o clasă șablon care gestionează spațiul de memorie pentru elementele sale și permite accesul la acestea direct sau prin intermediul iteratorilor.

Componentele STL sunt deci:

- A** **containerele**, care pot fi containere secvențiale (și din această categorie vom studia clasa `vector`), containere asociative (din care vom analiza `set`, `multiset`, `map`, `unordered_map`) și containere adaptor (din această categorie studiem `priority_queue`, `stack`, `queue`).
- B** **algoritmii**, care sunt funcții sau metode definite în interiorul containerelor și cu ajutorul cărora putem efectua, de exemplu, operații de sortare, căutare liniară sau binară etc. Acești algoritmi sunt optimizați și este mai ușoară apelarea lor decât implementarea de la zero.
- C** **iteratorii**, asemănători pointerilor, care sunt definiți pentru a parcurge mai simplu obiectele unui container.



1. vector

1.1. Parcurgeri ale vectorilor STL. Funcția size()

Începând cu versiunea C++11 se poate folosi o modalitate simplă de iterare (parcure) a fiecărui element al unui vector. În exemplul de mai jos, variabila `elem` memorează pe rând fiecare valoare din vector. Cuvântul cheie `auto` este utilizat pentru a ușura utilizarea structurii repetitive, în sensul că nu este necesar să reținem tipul vectorului. Aplicația de mai jos declară un vector de constante și ilustrează două moduri de parcure a elementelor sale. Se utilizează funcția `size()`, care returnează dimensiunea logică a vectorului, adică numărul componentelor sale.

```
#include <bits/stdc++.h>
using namespace std;

vector<int> a = {5,3,1,7,8};
int main()
{
    // afisare cu auto
    for (auto elem : a) //la fel de corect este for(int elem : a)
        cout << elem << " ";
    cout << "\n";

    // alta modalitate de afisare:
    // functia size() furnizeaza numarul elementelor vectorului
    for (int i = 0; i < a.size(); i++)
        cout << a[i] << " ";
    return 0;
}
```

1.2. Funcția for_each()

Această funcție, care se regăsește ca instrucțiune în limbaje precum C#, Java, PHP, parcurge sistematic fiecare element al unui vector. În cazul exemplului de mai jos, utilizând funcția `Afis()`, se realizează într-o altă formă afișarea valorilor din vector. Se observă că `Afis()` este apelată ca parametru al funcției `for_each()`. Complexitatea funcției este $O(n)$, unde n este dimensiunea vectorului.

```
#include <bits/stdc++.h>
using namespace std;

void Afis(int x)
{
    cout << x << " ";
}
}
```

```
int main()
{
    vector<int> a = {1,2,3,4,5};
    for_each(a.begin(), a.end(), Afis);
    cout << "\n";
    return 0;
}
```

1.3. Funcțiile push_back(), pop_back(), random_shuffle

Funcția push_back() adaugă la sfârșit un element într-un vector, iar pop_back() elimină ultimul element din vector. Funcția random_shuffle() amestecă elementele vectorului, astfel că în aplicația de mai jos se obține în vectorul a o permutare a mulțimii {1, 2, ..., n}.

```
int main()
{
    vector <int> a;
    int i, n;
    cin >> n;
    // se adauga in vectorul STL numerele de la 1 la n
    for (i = 0; i < n; i++)
        a.push_back(i + 1);
    // se amesteca elementele vectorului STL
    random_shuffle(a.begin(), a.end());
    // afisare cu auto; elementele vectorului a sunt parcurse
    // de la primul pana la ultimul
    for (auto elem : a)
        cout << elem << " ";
    cout << "\n";
    // elimina ultimul element, apoi afiseaza vectorul
    a.pop_back();
    for (auto elem : a)
        cout << elem << " ";
    cout << "\n";
    return 0;
}
```

1.4. Ordonare crescătoare, ordonare descrescătoare

Fie un vector a de numere întregi. Să se ordoneze mai întâi crescător, apoi să se ordoneze descrescător. De exemplu, dacă a=(5, 3, 4, 1, 2), atunci după sortarea crescătoare, a=(1, 2, 3, 4, 5), iar după ordonarea descrescătoare, a=(5, 4, 3, 2, 1).

Rezolvare:

Se utilizează funcția sort(). Complexitatea algoritmului sortării este $O(n \cdot \log n)$, unde n este numărul de elemente din vector.

```
#include <bits/stdc++.h>
using namespace std;
```



```
void Sortari(vector <int> a)
{
    int n = a.size(); // numarul de elemente din vectorul a
    // sorteaza crescator elementele vectorului si le afiseaza
    sort(a.begin(), a.end());
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << "\n";

    // sorteaza descrescator elementele vectorului si le afiseaza
    sort(a.begin(), a.end(), greater<int>());
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << "\n";
}

int main()
{
    vector<int> a = {5,3,4,1,2};
    Sortari(a);
    return 0;
}
```

1.5. Sortarea punctelor în plan

Dacă vectorul are un tip special, de exemplu elementele sale sunt puncte în plan, pentru a face o sortare după criterii particulare, este nevoie să definim o funcție de comparare a elementelor. De exemplu, avem la dispoziție n puncte în plan date prin abscisă și ordonată, numere întregi. Vrem să ordonăm crescător punctele, mai întâi după abscisă, apoi, în caz de egalitate, crescător după ordonată. Mai jos este definită funcția `Compara` care furnizează funcției `sort()` modalitatea de ordonare a elementelor vectorului.

```
#include <bits/stdc++.h>
using namespace std;

struct Punct
{
    int x, y;
};

int n;
Punct a[1001];

inline bool Compara(const Punct A, const Punct B)
{
    if (A.x == B.x) return A.y < B.y;
    return A.x < B.x;
}

int main()
```

PARTEA A II-A

Tablouri unidimensionale

EDITURA PARALELA 45



9. Ciurul lui Eratostene

9.1. Generarea numerelor prime

Se dă un număr natural n , $2 \leq n \leq 10^6$. Să se genereze numerele prime cuprinse între 1 și n . De exemplu, dacă $n=20$, atunci numerele prime mai mici decât 20 sunt 2, 3, 5, 7, 11, 13, 17, 19.

Rezolvare:

Vom construi un vector a de lungime $n+1$ în care, pentru orice $i=0..n$, $a[i]=0$, dacă i este prim, sau $a[i]=1$, dacă i nu este prim. Inițial, $a[i]=0$, pentru $i=2..n$, iar $a[0]=a[1]=1$. Pentru fiecare $i=2..n$, se verifică dacă i este prim ($a[i]=0$) și, în caz afirmativ, toți multiplii săi, începând cu $2 \cdot i$, vor fi marcați în a cu 1 deoarece nu sunt primi. De exemplu, $i=5$ este prim, dar multiplii săi, 10, 15, 20, 25..., nu sunt.

Funcția de mai jos construiește vectorul P de lungime k în care se depun numerele prime mai mici sau egale cu n .

```
void Ciur_1(int n, int P[], int &k)
{
    int i, j;
    int a[n + 1] = {0};
    a[0] = a[1] = 1;
    for (i = 2; i <= n; i++)
        if (a[i] == 0) // i este prim
            for (j = 2 * i; j <= n; j += i)
                a[j] = 1; // j este multiplul lui i
    k = 0;
    for (i = 2; i <= n; i++)
        if (a[i] == 0) P[k++] = i;
    for (i = 0; i < k; i++)
        cout << P[i] << " ";
}
```

Care este complexitatea algoritmului din funcția `Ciur_1`? Sunt $n-1$ pași și pentru fiecare număr prim i se parcurg multiplii săi, deci n/i numere. În total sunt $n/2+n/3+\dots+n/i+\dots$ pași, adică $n(1/2+1/3+\dots+1/i+\dots)$, adică aproximativ $n \cdot \log n$ pași (deoarece șirul $1/2+1/3+\dots+1/i+\dots - \log n$ are limita un număr strict mai mic decât 1). Deci complexitatea este $O(n \cdot \log n)$.

Cum îmbunătățim algoritmul de mai sus? În primul rând, numerele pare diferite de 2 se marchează separat cu 1 în a . Parcurgem apoi doar numerele impare mai mari sau egale cu 3 și, dacă i este prim, atunci vom începe să marcăm multiplii impari ai lui i , începând cu $i \cdot i$. De ce $i \cdot i$? Pentru că multiplii lui i mai mici decât $i \cdot i$ au fost marcați la pașii anteriori. De exemplu, dacă $i=7$, primul multiplu nemarcat este $7 \cdot 7=49$, deoarece $3 \cdot 7$ și $5 \cdot 7$ au fost marcați când s-au parcurs multiplii lui 3 și ai lui 5. Pentru că $i \cdot i$ este primul marcat, atunci i nu mai trebuie să parcurgă toate numerele impare până la n , ci doar până la \sqrt{n} . Cu aceste modificări, complexitatea algoritmului devine $O(n \cdot \log(\log n))$. De menționat că în primele 10^6 numere naturale se află 78498 numere prime.

```

void Ciur_2(int n, int P[], int &k)
{
    int i, j;
    int a[n + 1] = {0};
    a[0] = a[1] = 1;

    for (i = 4; i <= n; i += 2)
        a[i] = 1;

    for (i = 3; i * i <= n; i += 2)
        if (a[i] == 0)
            for (j = i * i; j <= n; j += 2*i)
                a[j] = 1;

    P[0] = 2;
    k = 1;
    for (i = 3; i <= n; i += 2)
        if (a[i] == 0) P[k++] = i;
}

```

9.2. Cele mai îndepărtate numere prime consecutive

Dându-se numărul natural n , $4 \leq n \leq 105$, să se determine diferența maximă dintre două numere prime consecutive din intervalul $[1, n]$. De exemplu, pentru $n=20$, numerele prime sunt 2, 3, 5, 7, 11, 13, 17, 19, iar diferența maximă este 4 (11-7, sau 17-13).

Rezolvare:

Se construiește vectorul caracteristic a de lungime n în care, pentru orice $i=1..n$, $a[i]=0$ dacă i este număr prim, sau $a[i]=1$ dacă i nu este prim. Se determină apoi două poziții i și j cu $a[i]=0$, $a[j]=0$, între pozițiile $i+1..j-1$ să fie doar valori de 1, iar i și j să fie la distanță maximă.

```

int DifMaxPrime(int n)
{
    int a[100001] = {0};
    int j, i, difMax = 0;
    // ciur
    for (i = 4; i <= n; i += 2)
        a[i] = 1;
    for (i = 3; i * i <= n; i += 2)
        if (a[i] == 0)
            for (j = i * i; j <= n; j += 2*i)
                a[j] = 1;
    // diferenta maxima
    j = 2;
    difMax = 1;
    for (i = 3; i <= n; i++)
        if (a[i] == 0)
        {
            difMax = max(difMax, i - j);
            cout << j << " " << i << "\n";
            j = i;
        }
    return difMax;
}

```



9.3. Numărul divizorilor

Să se determine numărul divizorilor fiecărui număr cuprins între 1 și n ($1 \leq n \leq 10^4$). De exemplu, 12 are 6 divizori, aceștia fiind 1, 2, 3, 4, 6, 12.

Rezolvare:

Utilizăm aceeași tehnică de la ciurul lui Eratostene. Parcurgem numerele de la 1 la n și fiecare $i=1..n$ este divizor pentru toți multiplii săi, deci i este multiplu pentru $i, 2 \cdot i, 3 \cdot i, \dots$. Complexitatea algoritmului este $O(n \cdot \log n)$.

```
#include <bits/stdc++.h>
using namespace std;

int a[10001], n;

int main()
{
    int i, j;
    cin >> n;
    for (i = 1; i <= n; i++)
        // parcurge multiplii lui i
        for (j = i; j <= n; j += i)
            a[j]++;
    for (i = 1; i <= n; i++)
        cout << a[i] << " ";
    return 0;
}
```

9.4. Indicatorul lui Euler

Indicatorul lui Euler, $\varphi(n)$, este numărul de numere mai mici decât n și prime cu n . De exemplu, dacă $n=12$, atunci $\varphi(n)=4$, deoarece numerele 1, 5, 7, 11 sunt prime cu 12 (două numere sunt prime între ele dacă cel mai mare divizor comun al lor este 1). Dându-se un număr natural n , să se construiască vectorul a în care pentru fiecare $i=1..n$, $a[i]$ memorează numărul de numere mai mici decât i și prime cu i . De exemplu, dacă $n=8$, atunci $a=(1, 1, 2, 2, 4, 2, 6, 4)$.

Rezolvare:

De menționat faptul că, dacă x este număr prim, atunci $\varphi(x)=x-1$, deoarece x este prim cu toate numerele cuprinse între 1 și $x-1$. Formula matematică pentru determinarea lui $\varphi(x)$ este:

$$\varphi(x)=x \cdot (p_1-1) \cdot (p_2-1) \cdot \dots \cdot (p_k-1) / (p_1 \cdot p_2 \cdot \dots \cdot p_k)$$

unde p_1, p_2, \dots, p_k sunt factorii primi ai lui x din descompunerea în factori ai lui x .

Plecând de la formula dată, inițial $a[i]=i$, pentru fiecare $i=1..n$. Se parcurg apoi numerele de la 2 la n și pentru fiecare număr i care este prim se parcurg multiplii j ai lui i începând chiar cu acesta și fiecare $a[j]$ se împarte la i , apoi se înmulțește cu $i-1$, conform cu formula. Cum se stabilește dacă un număr este prim sau nu? Dacă i este prim, atunci $a[i]=i$, adică până atunci nu a fost modificat deoarece nu are divizori.

```

void Phi(int a[], int n)
{
    int i, j;
    for (i = 1; i <= n; i++)
        a[i] = i;
    for (i = 2; i <= n; i++)
        if (a[i] == i) // i este prim
            for (j = i; j <= n; j += i)
                a[j] = a[j] * (i - 1) / i;
    for (i = 1; i <= n; i++)
        cout << a[i] << " ";
}

```

9.5. Secvențe având suma un număr prim

Fie un vector $a=(a_1, a_2, \dots, a_n)$ de numere naturale ($1 \leq n \leq 1000$, $1 \leq a_i \leq 1000$). Să se determine numărul secvențelor care au suma egală cu un număr prim. De exemplu, dacă $a=(2, 1, 4, 2)$, atunci numărul secvențelor este 6, acestea fiind (2) , $(2, 1)$, $(2, 1, 4)$, $(1, 4)$, $(1, 4, 2)$ și (2) .

Rezolvare:

Deoarece suma maximă posibilă a tuturor elementelor din vector este 10^6 , vom construi cu ajutorul Ciurului lui Eratostene vectorul c în care $c[x]=0$ dacă x este prim și $c[x]=1$, dacă x nu e prim. În continuare, pentru fiecare poziție i ($i=0..n-1$), se calculează orice sumă a unei secvențe care începe cu $a[i]$, parcurgând cu j toate elementele vectorului de la poziția i la $n-1$. Complexitatea algoritmului este $O(M \log(\log M) + n^2)$, unde M memorează suma elementelor din vectorul a .

```

#include <iostream>
using namespace std;

int c[1000001], n, a[1001];

void Ciur(int M)
{
    int i, j;
    c[0] = c[1] = 1;
    for (i = 4; i <= M; i += 2)
        c[i] = 1;
    for (i = 3; i * i <= M; i += 2)
        if (c[i] == 0)
            for (j = i * i; j <= M; j += 2*i)
                c[j] = 1;
}

int main()
{
    int i, j, s, nrSecv, M = 0;
    cin >> n;
    for (i = 0; i < n; i++)
    {
        cin >> a[i];

```