COMPLEXITATEA ALGORITMILOR

Determinarea perechilor cu suma elementelor dată

Considerăm un șir format din n produse sortate strict crescător în funcție de prețul lor. Directorul de marketing al firmei BuyAll s-a gândit să lanseze o campanie de promoții în care să cupleze câte două produse distincte astfel încât suma prețurilor lor să fie egală cu o valoare dată s. Scrieți un program care să-l ajute!

Exemplu:

produse.in	produse.out
11 2 5 7 8 10 12 15 17 22 25 40 20	5 15 8 12

Explicație: Sunt $n = 9$ produse având prețurile indicate și se dorește găsirea perechilor de produse având prețul total egal cu $s = 20$. Există două soluții: se pot grupa produsele 2 și 7 (având prețurile 5 și 15 RON), respectiv produsele 4 și 6 (având prețurile 8 și 12 RON).

Algoritm general:

```
for(i = 0; i < n && p[i] < s/2; i++)
     caut s-p[i] în secvența p[i+1],..., p[n-1]
```

Complexitate:

$$\mathcal{O}(\underbrace{n}_{\substack{citire \\ date}} + n \cdot complexitate\_căutare) \approx \mathcal{O}(n \cdot complexitate\_căutare)$$

- Dacă folosim căutarea liniară, cu complexitatea $\mathcal{O}(n)$, obținem complexitatea $\mathcal{O}(n^2)$
- Dacă folosim căutarea binară, cu complexitatea $\mathcal{O}(\log_2 n)$, obținem complexitatea $\mathcal{O}(n \log_2 n)$
- Dacă folosim un vector de marcaje, cu complexitatea $\mathcal{O}(1)$, obținem complexitatea $\mathcal{O}(n)$

  Observație: Un vector de marcaje este un vector de frecvențe cu elemente de tip `bool` sau `char`!

Mici "optimizări" globale (pentru orice variantă):

- eliminăm din tabloul p toate valorile $p[i] >= s$ (din dreapta tabloului)
- eliminăm din tabloul p toate valorile $p[i] + p[n-1] < s$ (din stânga tabloului)

1. Forță brută (căutare directă) - complexitate $\mathcal{O}(n^2)$

Pentru fiecare element $p[i]$ cu $0 <= i < n-1$ verificăm dacă există un element $p[j]$ cu $i+1 <= j < n$ astfel încât $p[i] + p[j] == s$.

Mici "optimizări" locale (pentru această variantă):

- oprim for-ul după j la prima pereche $p[i] + p[j] >= s$
- oprim for-ul după i la primul element $p[i] >= s/2$

Generarea datelor de test (un șir de numere ordonate strict crescător):

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <algorithm>
using namespace std;

//declaram tabloul global, deoarece are multe elemente (1000000)
const unsigned int n = 1000000;
unsigned int v[n];

int main()
{
    unsigned int i, vinit, r;

    vinit = time(NULL);
    srand(vinit);

    double t = clock();

//    v[0] este cuprins intre 1 si 100
    v[0] = 1 + (rand() * rand()) % 100;

    cout << v[0] << endl;


//    restul elementelor vor fi cuprinse intre v[i-1]+1  si v[i-1]+1000
    for(i = 1; i < n; i++)
    {
        r = rand() * rand();
        r = v[i-1] + 1 + r % 1000;
        v[i] = r;
    }

    t = (clock() - t) / CLOCKS_PER_SEC;
```

```
    cout << endl << "Timpul pentru generarea datelor de test: " << t
         << " secunde!" << endl;

    for(i = 0; i < n; i++)
        cout << v[i] << " ";

    return 0;
}
```

Variante de rezolvare:

2. Forță brută (căutare binară) - complexitate $\mathcal{O}(n \log_2 n)$

Pentru fiecare element $p[i]$ cu $0 <= i < n-1$ verificăm dacă există elementul $s - p[i]$ în secvența $p[i+1], \ldots, p[n-1]$.

Mici "optimizări" locale (pentru această variantă):

- oprim for-ul după i la primul element $p[i] >= s/2$

Căutarea binară

- se poate aplica pentru căutarea unei valori într-un tablou sortat (de obicei, crescător)

- se compară elementul căutat cu elementul aflat în mijlocul tabloului/secvenţei curente din tablou:
  - dacă $t[mij] == x$ => x apare pe poziția mij => STOP
  - dacă $x < t[mij]$ => îl voi căuta pe x în secvența $t[st], \ldots, t[mij-1]$
  - dacă $x > t[mij]$ => îl voi căuta pe x în secvența $t[mij+1], \ldots, t[dr]$

- numărul maxim de comparări este $\log_2 n$

Exemplu 1: $n = 11$, $t = (2, 5, 5, 10, 25, 43, 55, 78, 78, 85, 96)$ și $x = 55$

$$t = (\underbrace{2}_{\sout{st=0}}, 5, 5, 10, 25, \underbrace{43}_{mij=5}, \underbrace{55}_{st=6}, 78, 78, 85, \underbrace{96}_{dr=10})$$

$$t = (\underbrace{2}_{\sout{st=0}}, 5, 5, 10, 25, \underbrace{43}_{mij=5}, \underbrace{55}_{st=6}, \underbrace{78}_{dr=7}, \underbrace{78}_{mij=8}, 85, \underbrace{96}_{\sout{dr=10}})$$

$$t = (\underbrace{2}_{\sout{st=0}}, 5, 5, 10, 25, \underbrace{43}_{mij=5}, \underbrace{55}_{mij=6}, \underbrace{78}_{dr=7}, \underbrace{78}_{mij=8}, 85, \underbrace{96}_{\sout{dr=10}})$$

Exemplu 2: $n = 11$, $t = (2, 5, 5, 10, 25, 43, 55, 78, 78, 85, 96)$ și $x = 60$

$$t = (\underbrace{2}_{\sout{st=0}}, 5, 5, 10, 25, \underbrace{43}_{mij=5}, \underbrace{55}_{st=6}, 78, 78, 85, \underbrace{96}_{dr=10})$$

$$t = (\underbrace{2}_{\sout{st=0}}, 5, 5, 10, 25, \underbrace{43}_{mij=5}, \underbrace{55}_{st=6}, \underbrace{78}_{dr=7}, \underbrace{78}_{mij=8}, 85, \underbrace{96}_{\sout{dr=10}})$$

$$t = (\underbrace{2}_{\sout{st=0}}, 5, 5, 10, 25, \underbrace{43}_{mij=5}, \underbrace{55}_{\substack{st=6\\mij=6}}, \underbrace{78}_{dr=7}, \underbrace{78}_{mij=8}, 85, \underbrace{96}_{\sout{dr=10}})$$

$$t = (\underbrace{2}_{\sout{st=0}}, 5, 5, 10, 25, \underbrace{43}_{mij=5}, \underbrace{55}_{\sout{st=6}}, \underbrace{78}_{\substack{dr=7\\st=7\\mij=7}}, \underbrace{78}_{mij=8}, 85, \underbrace{96}_{\sout{dr=10}})$$

$$t = (\underbrace{2}_{\sout{st=0}}, 5, 5, 10, 25, \underbrace{43}_{mij=5}, \underbrace{55}_{\substack{\sout{st=6}\\mij=6\\dr=6}}, \underbrace{78}_{\substack{\sout{dr=7}\\st=7\\mij=7}}, \underbrace{78}_{mij=8}, 85, \underbrace{96}_{\sout{dr=10}})$$

Dacă $dr < st$ => valoarea x nu apare în tabloul t!!!

Explicație: La fiecare pas, caut valoarea x curentă în secvența curentă $t[st], \dots, t[dr]$, deci am unde să-l caut pe x dacă $st \leq dr$!

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

int main()
{
    int n, x, i, st, dr, mij;
    int v[1000];

    cout << "n = ";
    cin >> n;

    for (i = 0; i <= n - 1; i++)
    {
        cout << "v[" << i << "] = ";
        cin >> v[i];
    }

    cout << "x = ";
    cin >> x;

    st = 0;
    dr = n-1;
    while(st <= dr)
    {
        mij = (st + dr) /2;
        if(v[mij] == x)
            break;
        else
            if(x < v[mij])
                dr = mij - 1;
            else
                st = mij + 1;
    }

    if(st > dr)
        cout << "Valoarea " << x << " nu apare in tablou!";
    else
        cout << "Valoarea " << x << " apare in tablou pe pozitia " << mij;

    return 0;
}
```

Numărul maxim de comparări (când x nu apare în tabloul t sau este ultimul element)**:**

- la pasul 1 îl caut pe *x* într-o secvenţă de lungime n
- la pasul 2 îl caut pe *x* într-o secvenţă de lungime $\frac{n}{2}$
- la pasul 3 îl caut pe *x* într-o secvenţă de lungime $\frac{n}{2^2}$
- .....

- la pasul k îl caut pe x într-o secvență de lungime $\frac{n}{2^{k-1}}$
- .....
- căutarea se termină pentru prima valoare a lui k pentru care $\frac{n}{2^{k-1}} < 1$ =>
  căutarea se termină pentru prima valoare a lui k pentru care $n < 2^{k-1}$ =>
  căutarea se termină pentru prima valoare a lui k pentru care $k > \log_2 n - 1$ =>
  căutarea se termină pentru $k = [\log_2 n - 1] + 1$

Numărul maxim de comparări: $\approx \log_2 n$

Căutare liniară pentru $n = 10^6$: maxim 10^6 comparări
Căutare binară pentru $n = 10^6$: maxim $\log_2 10^6 = 6 \log_2 10 \in (18, 24) \approx 18$ comparări

Varianta optimă:

11
2 5 7 8 10 12 15 17 22 25 40
S = 20

st=0	st=1	st=2			dr=5	dr=6	dr=7	dr=8	dr=9	dr=10
2	5	7	8	10	12	15	17	22	25	40

- **v[st=0]+v[dr=10] = 42 > 20 = s => dr--** (suma elementelor curente este prea mare, deci trebuie să o micșorăm)
- **v[st=0]+v[dr=9] = 27 > 20 = s => dr--** (suma elementelor curente este prea mare, deci trebuie să o micșorăm)
- **v[st=0]+v[dr=8] = 24 > 20 = s => dr--** (suma elementelor curente este prea mare, deci trebuie să o micșorăm)
- **v[st=0]+v[dr=7] = 19 < 20 = s => st++** (suma elementelor curente este prea mică, deci trebuie să o creștem)
- **v[st=1]+v[dr=7] = 22 > 20 = s => dr--** (suma elementelor curente este prea mare, deci trebuie să o micșorăm)
- **v[st=1]+v[dr=6] = 20 = 20 = s => st++** și **dr--** (pentru elementele curente 5 și 15 nu mai pot exista alte soluții)
- .....

Complexitate: $\mathcal{O}(n)$ – fără memorie suplimentară!!!

Metoda se numește *two pointers* (arderea lumânării la două capete).

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <algorithm>

using namespace std;

void varianta_1(unsigned int v[], unsigned int n, unsigned int s)
{
    for(unsigned int i = 0; i < n; i++)
    {
        if(v[i] >= s/2)
            break;

        unsigned int x = s - v[i];

        for(unsigned int j = i + 1; j < n; j++)
        {
            if(v[j] == x)
            {
                cout << v[i] << ' ' << x << endl;
                break;
            }

            if(v[j] > x)
                break;
        }
    }
}

void varianta_2(unsigned int v[], unsigned int n, unsigned int s)
{

    for(unsigned int i = 0; i < n; i++)
    {
        if(v[i] >= s/2)
            break;

        unsigned int x = s - v[i];

        unsigned int st = i + 1, dr = n-1;
        while (st <= dr)
        {
            unsigned int mij = (st + dr)/2;
            if (v[mij] == x)
            {
                cout << v[i] << ' ' << x << endl;
                break;
            }
            else
            {
                if (x < v[mij])
                {
```

```
                    dr = mij-1;
                }
                else
                {
                    st = mij + 1;
                }
            }
        }
    }
}

bool marcaje[10000000];
// Presupunem ca preturile sunt numere cuprinse intre 1 si 100000
void varianta_3(unsigned int v[], unsigned int n, unsigned int s)
{

    for(unsigned int i=0; i<n; i++)
        marcaje[v[i]]=true;

    for(unsigned int i = 0; i < n; i++)
    {
        if(v[i] >= s/2)
            break;

        unsigned int x = s - v[i];

        if(marcaje[x]==true)
        {
            cout << v[i] << ' ' << x << endl;
            break;
        }
    }
}

void varianta_4(unsigned int v[], unsigned int n, unsigned int s)
{

    unsigned int st = 0, dr = n-1;

    while(st < dr)
        if(v[st] + v[dr] == s)
        {
            cout << v[st] << ' ' << v[dr] << endl;
            st++;
            dr--;
        }
        else
            if(v[st] + v[dr] < s)
                st++;
            else
                dr--;
}

void generareTeste(const char nume_fisier[], unsigned int n)
```

```
{
    unsigned int i, vcrt, r, x, y;

    srand(time(NULL));

    ofstream fout(nume_fisier);

    fout << n << endl;
    //primul numar este cuprins intre 1 si 100
    vcrt = 1 + (rand() * rand()) % 100;
    fout << vcrt << " ";

    //restul elementelor vor fi cuprinse intre vcrt+1  si vcrt+1000
    x = y = 0;
    for(i = 1; i < n; i++)
    {
        r = 1 + (rand() * rand()) % 100;
        vcrt = vcrt + r;

        //sansele ca r >= 50 sunt, teoretic, 50%
        if(r >= 50)
            if(x == 0)
                x = vcrt;
            else if(y == 0)
                y = vcrt;

        fout << vcrt << " ";
    }

    fout << endl << x + y << endl;

    fout.close();
}

void citireDate(const char nume_fisier[], unsigned int v[], unsigned int &n,
unsigned int &s)
{
    ifstream fin(nume_fisier);

    fin >> n;
    for(int i = 0; i < n; i++)
        fin >> v[i];
    fin >> s;

    fin.close();
}
unsigned int v[1000000];

int main()
{
    unsigned int n, s;

    generareTeste("preturi.in", 100000);
```

```
    citireDate("preturi.in", v, n, s);

    varianta_1(v, n, s);
    cout << endl;
    varianta_2(v, n, s);
    cout << endl;
    varianta_3(v, n, s);
    cout << endl;
    varianta_4(v, n, s);

    return 0;
}
```