Secvențe

Deseori este util să studiem fenomene care se întâmplă unul după altul, sau într-o perioadă continuă, am putea spune. Formal, pe calculator, avem elemente stocate într-un tablou și ne interesează să aflăm informații despre grupuri aflate pe o plajă de indici consecutivi, cu anumite proprietăți. Pentru un șir de numere v_1, v_2, ..., v_n, numim secvență ca fiind un set de elemente v_i, v_{i+1}, v_{i+2}, ... v_{j-1}, v_j, unde `i` și `j`, cu `i<=j`, sunt poziții ale elementelor din șirul `v`.

Următorul set de probleme rezolvate se referă la cerințe de forma: să se determine cea mai lungă secvență din șir cu o anumită proprietate.

1. Dat fiind un șir cu `n` elemente întregi, să se determine lungimea maximă a unei secvențe formate doar din valori pare.

Rezolvare

Soluția 1. Cei mai mulți oameni au la început tendința de a aborda problema în următorul mod: traversează șirul de numere (deci cu o structură repetitivă – de regulă `for`) iar la găsirea unei valori pare folosesc altă structură repetitivă ce avansează la dreapta cât timp se găsesc valori pare, așadar prima structură dă startul secvenței iar a doua o identifică în întregime.

```
for (i=1;i<=n;i++)
    cin>>v[i];
for (i=1;i<=n;i++)
    if (v[i]%2 == 0) {
        j = i;
        while (j <= n && v[j] % 2 == 0)
            j++;
        if (j-i > maxim) {
            maxim = j-i;
            st = i;
            dr = j-1;
        }
    }
cout<<st<<" "<<dr;
```

Soluția 2. Caracteristica principală a acestei abordări este că folosește o singură structură repetitivă. Este totodată o subtilă abordare de programare dinamică, tehnică foarte utilă în rezolvarea multor probleme grele, dar pe care o vom studia detaliat altă dată.

Concret, păstrăm într-o variabilă informații despre *secvența ce are finalul la poziția curentă*. Fiind vorba despre secvențe, când trecem la elementul următor, fie el poate fi alipit secvenței despre care aveam deja calculate informații (deci noua secvență se termină acum pe această următoare poziție) și atunci creștem cu `1` valoarea variabilei, fie elementul curent nu poate fi alipit secvenței terminate pe poziția anterioară. În acest caz resetăm valoarea variabilei despre care vorbeam: fie o facem `1`, dacă putem începe o nouă secvență cu elementul curent, fie o facem `0` dacă elementul curent nu poate face parte din vreo secvență în condițiile probemei (nu uităm că valoarea variebilei indică lungimea cea mai bună a unei secvențe terminate pe poziția curentă).

Să luăm un exemplu pentru problema enunțată:

Indice	1	2	3	4	5	6	7	8	9	10	11
Element al șirului	2	4	3	5	6	8	2	6	7	6	4
Valoarea variabilei (lungimea maximă a unei secvențe terminate cu elementul curent)	1	2	0	0	1	2	3	4	0	1	2
Explicație	Primul element este par deci avem secvență de lungime 1 terminată cu el	Al doilea element tot par, deci extinde secvența anterioară	Avem element impar, deci nici nu extinde secvența anterioară nici nu începe una nouă								

Iată și codul:

```
for (i=1;i<=n;i++)
    cin>>v[i];
L = 0;
for (i=1;i<=n;i++) {
    if (v[i] % 2 == 0)
        L++;
    else
        L = 0;
    if (L > maxim)
        maxim = L;
}
cout<<maxim;
```

Observăm că la fiecare pas, după actualizarea variabilei pentru acea poziție (în cazul nostru `L`), aceasta este comparată cu un maxim. Astfel, la final se obține valoarea cea mai mare a unei secvențe de elemente pare care apare în șir.

Despre o abordare ca mai sus se spune că este una bazată pe evenimente. La astfel de probleme ne stabilim mai întâi o stare, în cazul nostru ea este dată de valoarea curentă a variabilei L (cu semnificația prezentată, adică de optim pentru secvența terminată în elementul curent) Apoi ne stabilim ce se poate întâmpla când ajungem la un element (tipul de eveniment) și, parcurgând șirul, tratăm unul câte unul evenimentele apărute. Aici am avea:

- Eveniment de apariție a unui număr par: modul de tratare este incremntarea lui `L`;
- Eveniment de apariție a unui număr impar: modul de tratare este resetarea lui `L`;

În continuarea materialului soluțiile propuse vor fi pe acest model, prezentat al doilea.

O cerință mai generală care apare este de a localiza secvența de lungime maximă. Codul următor afișează indicii de început și de final pentru cea mai lungă secvență de elemente pare. Dacă sunt mai multe secvențe de lungime maximă se afișează date despre cea cu indici mai mici (cea mai din “stânga”).

```
for (i=1;i<=n;i++)
    cin>>v[i];
L = 0;
for (i=1;i<=n;i++) {
    if (v[i] % 2 == 0)
        L++;
    else
        L = 0;
    if (L > maxim) {
        maxim = L;
        u = i;
    }
}
cout<<u-maxim+1<<" "<<u;
```

La fiecare actualizare a maximului am reținut și poziția de final pentru secvența cea mai lungă. Este suficient doar finalul deoarece această informație împreună cu lungimea secvenței ne permit obținerea și a poziției de început.

La compararea lui `L` cu `maxim` am folosit semnul "mai mare strict". Dacă am fi folosit "mai mare sau egal" ar fi rămas în `u` poziția de final a celei mai din dreapta secvențe de lungime maximă, în caz că ar fi fost mai multe.

Trebuie remarcat că la această problemă nu era necesară stocarea elementelor într-un vector. Întrucât din urmă nu ne sunt necesare decât informațiile despre secvența terminată la pasul anterior (`maxim`, `L` și `u`), când citim un element nou doar actualizăm eventual cele trei variabile.

2. Dat fiind un șir de numere întregi, să se găsească secvența cu elemente în ordine strict crescătoare care are lungimea maximă. Se vor afișa indicii de început și de final pentru secvența găsită. Dacă sunt mai multe astfel de secvențe se vor afișa informații despre "cea mai din stânga".

Rezolvare

Aici aplicăm strategia învățată astfel: putem spune că primul element formează o secvență de lungime `1`, cu elemente ordonate crescător. De la al doilea, stabilim dacă îl includem pe cel curent în secvența anterioară sau dacă începem cu el o secvență nouă, în funcție de rezultatul comparării valorii lui cu valoarea celui de pe poziția anterioară. Așadar, dacă `v[i] > v[i-1]` vom mări le `L` cu `1` iar în caz contrar `L` se resetează, dar nu la `0` ca în problema anterioară ci la `1` (dacă elementul curent nu se lipește la secvența anterioară, cu el începe totuși o nouă secvență de lungime `1`). Și această problemă admite soluție fără folosirea tablourilor, fiind nevoie de elementul curent și de cel anterior doar.

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

ifstream fin ("secvcresc.in");
ofstream fout("secvcresc.out");
int v[10010];
int n, i, L, maxim, dr;
int main () {
    fin>>n;
    for (i=1;i<=n;i++)
        fin >> v[i];
    L = 1
    maxim = 1;
    u = 1
    for (i=2;i<=n;i++) {
        if (v[i] > v[i-1])
```

```
            L++;
        else
            L = 1;

        if (L > maxim) {
            maxim = L;
            u = i;
        }

    }
    fout<<u-maxim+1<<" "<<u;
    return 0;
}
```

Probleme diverse, rezolvate, cu secvențe

1. Se dau doi vectori, `a` cu `n` elemente și `b` cu `m` elemente, ambii cu valorile memorate începând cu poziția `1`. Să se determine dacă elementele lui `b` apar în `a` pe poziții consecutive (ca o secvență). În caz afirmativ se va determina cea mai mică poziție pe care elementele lui `b` apar în `a` ca secvență (*pbinfo.ro, #519*).

Rezolvare

Vom prezenta o soluție brut adică încercăm fircare loc din `a` unde `b` poate începe ca secvență, și cu o altă strucură repetitivă vom testa element cu element dacă toate valorile se potrivesc.

Astfel, pentru un indice `i` din `a`, va fi o potrivire a lui `b` care începe acolo dacă: `a[i+0] == b[1]`, `a[i+1] == b[2]` ... `a[i+m-1] == b[m]`. Așadar indicele `i` trebuie căutat în `v` între pozițiile `1` și `n-m+1` (ultimul loc de unde mai sunt în a cel puțin `m` elemente).

```
#include <iostream>
using namespace std;
int a[1001], b[1001];
int n, m, i, j, diferite, k;
int main () {
    cin>>n;
    for (i=1;i<=n;i++)
        cin>>a[i];
    cin>>m;
    for (i=1;i<=m;i++)
        cin>>b[i];
    for (i=1;i<=n-m+1;i++) {
        /// verifc daca sirul b este secventa in a incepand din pozitia
i
        diferite = 0;
        for (j=1,k=i;j<=m;j++,k++) {
            if (a[k] != b[j]) {
                diferite++;
                break;
            }
        }
        if (diferite == 0) {
            cout<<i;
            return 0;
```

```
            }
        }
        cout<<"NU";
}
```

Se observă din codul de mai sus că în cazul în care se găsește o potrivire programul se încheie (`return 0`). Am fi putut renunța la ieșire și astfel s-ar fi determinat toate potrivirile lui `b` ca secvență în `a`. Timpul de executare al acestui algoritm este de ordin $n \cdot m$ deoarece avem două repetiții una în alta care pot merge pana la capăt. Există algoritmi performanți precum KMP sau Rabin-Karp care rezolvă această problemă cu timp de rulare de ordin $n+m$.

2. Se dă un vector cu `n` elemente, numere naturale, și un număr `k`, divizor al lui `n`. Se împarte vectorul în `k` secvențe disjuncte, numerotate de la `1` la `k`. Să se stabilească dacă există două secvențe identice (*pbinfo.ro, #521*).

Rezolvare

Folosim principiul de la sortarea prin comparare. Vom verifica fiecare secvență cu oricare alta de după ea. O secvență o identificăm prin poziția de început, așadar dacă `i` este însepul secvenței curente, pentru a trece la începutul următoarei avem `i+=L`.

```
/// timp de executare de ordin n
#include <iostream>
using namespace std;
int v[1002], s[100010], n, k, i, p, suma, maxim, j;
int L, st, dr, ok, t;
int main () {

    cin>>n>>k;
    for (i=1;i<=n;i++) {
        cin>>v[i];
    }
    L = n/k;
    for (i=1/*inceputul oricărei secvențe*/;i+L <= n;i+=L)
        for (j=i+L; j<=n; j+=L) {
            /// j e pe rand inceputul fiecarei secvente de lungime L dupa
            /// secventa care incepe la pozitia i
            /// verific daca secventa de lungime L ce incepe la indicele i
            /// este egala cu cea de lungime L ce incepe la indicele j

            ok = 1;
            for (t = 0;t<L; t++) {
                if (v[i+t] != v[j+t]) {
                    ok = 0;
                    break;
                }
            }
            if (ok == 1) {
                if (st == 0) {
                    cout<<i/L+1<<" "<<j/L+1;
                    return 0;
                }
            }
        }
    cout<<"NU";
}
```

3. Se dă un vector cu n elemente, numere naturale. Determinați câte secvențe ale vectorului au toate elementele egale (*pbinfo.ro, #578*).

Rezolvare

O primă soluție, cu timpul de executare de ordin n^3 este de a fixa toate secvențele posibile (cu două foruri, primul pentru indicele de început și al doilea pentru indicele de final) și de a verifica apoi pentru secvența curentă dacă are toate elementele egale.

```
/// timp de calcul de ordinul n^3
#include<iostream>
#include<fstream>
using namespace std;
int v[100010], i, j, n, egale, k;
long long sol;
int main()
{
    cin>>n;
    for (i=1;i<=n;i++)
        cin>>v[i];
    for (i=1;i<=n;i++)
        for (j=i;j<=n;j++) {
            /// in acest loc perechea (i,j) reprezinta
            /// indicii din capetele cate unei secvente posibile
            /// verific deci daca intre pozitiile
            /// i si j toate elementele sunt egale

            egale = 1;/// toate elementele de la i la j sunt egale
            /// fac testul comparand
            for (k=i; k<j; k++)
                if (v[k] != v[k+1]) {
                    egale = 0;
                    break;
                }
            if (egale == 1)
                sol++;
        }
    cout<<sol;
}
```

O altă soluție, cu timp de calcul de ordin n^2 este să fixăm începutul unei secvențe și apoi să avansăm de acolo cu un alt indice în dreapta, cât timp elementele sunt egale, numărând la fiecare pas finalul câte unei secvențe.

```
/// timp de calcul de ordinul n^2
#include<iostream>
#include<fstream>
using namespace std;
int v[100010], i, j, n, egale, k;
long long sol;
int main()
{
    cin>>n;
    for (i=1;i<=n;i++)
        cin>>v[i];

    for (i=1;i<=n;i++) /// fixez inceputul
        for (j=i;j<=n;j++) { /// avansez cu finalul, numarand,
                             /// dar ma opresc la primul diferint de inceput
            if (v[j] != v[i])
                break;
```

```
            sol++;
        }
    cout<<sol;
}
```

În fine, există și soluție cu timp de calcul de ordin `n`, bazată pe rezolvarea cunoscută a celei mai lungi secvențe de elemente egale și pe următoarea observație: dacă `L` este lungimea maximă a unei secvențe terminate pe poziția curentă, sunt totodată `L` secvențe *terminate chiar acolo* (cea de lungime `1`, cea de lungime `2`, ... cea de lungime `L`). Așadar la fiecare pas adunăm valoarea lui `L`.

```
#include <iostream>
using namespace std;
long long n, i, ant, L, crt, sol;
int main () {
    cin>>n;
    cin>>ant;
    L = 1;
    sol = 1;
    for (i=2;i<=n;i++) {
        cin>>crt;
        /// mereu aici ant e penultimul numar citit si crt ultimul
        /// aici L = lungimea maxima a unei secvente de elemente egale
        /// terminata cu ultimul numar citit, adica la pozitia i
        if (ant == crt)
            L++;
        else
            L = 1;
        sol += L;
        ant = crt;
    }
    cout<<sol;
}
```

În soluția prezentată am arătat și că putem renunța la folosirea tablourilor (fiind necesară doar valoarea de la pasul curent și cea de la pasul anterior).

4. Dat fiind un tablou unidimensional cu elemente întregi, să se determine secvența de lungime `k` (dat și el) cu suma elementelor maximă (*pbinfo.ro, #134*).

Rezolvare

O primă soluție, cu timp de executare de ordin `n · k` este să fixăm începutul fiecărei secvențe posibile și apoi, cu altă repetiție în `k` pași, să calculăm suma elementelor secvenței fixate.

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

int v[100001];
int n, k, i, st, maxim, j;
ifstream fin ("secvk.in");
ofstream fout("secvk.out");
int main() {
    fin>>n>>k;
    for (i=1;i<=n;i++)
        fin>>v[i];
    /// vom calcula suma fiecarei secvente de lungime k
    /// fixam fiecare pozitie posibila de inceput pentru o secventa
    for (i=1;i<=n-k+1;i++) {
        /// calculam suma unei secvente de lungime k care incepe la pozitia i
```

```
        int suma = 0;
        for (j=i;j<=i+k-1;j++)
            suma += v[j];
        if (suma > maxim) {
            maxim = suma;
            st = i;
        }
    }
    for (i=st;i<=st+k-1;i++)
        fout<<v[i]<<" ";
}
```

Problema admite și o soluție liniară în `n`, folosind următoarea observație: dacă avem în `s` suma secvenței de lungime `k` cu indicele de final la poziția `i-1`, când trecem la secvența următoare trebuie doar să adunăm la `s` elementul din tablou de la poziția `i` și să îl scădem pe cel de pe poziția `i-k`. Așadar actualizarea lui `s` se face doar prin simple atribuiri fără a mai fi nevoie de încă o repetiție.

```
#include <fstream>
#include <cmath>
using namespace std;
int v[100001];
int n, k, i, dr, maxim, j;
ifstream fin ("secvk.in");
ofstream fout("secvk.out");
int main() {
    fin>>n>>k;
    for (i=1;i<=n;i++)
        fin>>v[i];
    int suma = 0;
    for (i=1;i<=k;i++)
        suma += v[i];
    maxim = suma;
    dr = k;
    /// am calculat mai sus suma primei secvente de lungime k
    /// observam ca de la o secventa a alta nu este necesara
recalcularea
    /// intregii sume ci doar adunam un element (care intra in
secventa)
    /// si il scadem pe cel din spate care iese
    for (i=k+1;i<=n;i++) {
        suma -= v[i-k];
        suma += v[i];
        if (suma > maxim) {
            maxim = suma;
            dr = i;
        }
    }
    for (i=dr-k+1;i<=dr;i++)
        fout<<v[i]<<" ";
}
```

Problema admite și o soluție bazată pe *sume parțiale*. Chiar dacă vom aloca un capitol special studierii acestei tehnici, putem prezenta încă de acum și acest mod de rezolvare pentru problema curentă.

Odată calculat șirul sumelor parțiale (așa cum este descris în codul ce urmează), suma oricărei secvențe din șirul dat se obține prin diferența a două elemente ale acestui șir. Vom obține și la această soluție timp de calcul de ordin n.

```
#include <fstream>
#include <cmath>
using namespace std;
int v[100001], s[100001];
int n, k, i, st, maxim, j;
ifstream fin ("secvk.in");
ofstream fout("secvk.out");
int main() {
    fin>>n>>k;
    for (i=1;i<=n;i++) {
        fin>>v[i];
        s[i] = s[i-1] + v[i];
    }
    /// s[i] se obtine ca fiind suma v[1] + v[2] + ... + v[i]
    /// adica suma tuturor elementelor din v de dinaintea pozitiei i
    /// se spune ca s este sirul sumelor partiale pentru v
    /// observam ca nu este necesar timp (repetitie) suplimentar pentru
    /// calculul elementelor lui s, asta facandu-se la citire
    /// printr-o singura atribuire
    for (i=1;i<=n-k+1;i++) {
        /// suma secventei de lungime k care incepe la poztia i
        /// folosindu-l pe s
        if (s[i+k-1] - s[i-1] > maxim) {
            maxim = s[i+k-1] - s[i-1];
            st = i;
        }
    }
    for (i=st;i<=st+k-1;i++)
        fout<<v[i]<<" ";
}
```

5. O secvenţă a unui vector se numeşte palindromică dacă primul element ale secvenţei este egal cu ultimul, al doilea cu penultimul, etc. Se dă un vector cu n elemente, numere naturale. Determinaţi secvenţa palindromică de lungime maximă (*pbinfo.ro, #310*).

Rezolvare

O primă soluție presupune fixarea tuturor secvențelor posibile și verificrea, pentru fiecare în parte, dacă este palindromică. Pentru a verifica dacă o secvență este palindromică avem nevoie de o repetiție. Soluția descrisă mai sus are complexitate în timp de ordin n^2. Optimizări de forma: testăm secvențele în ordine descrescătoare a lungimii lor și ne oprim la prima lungime găsită, nu reduce complexitatea pentru cazul defavorabil (când soluția este o secvență de lungime mică).

```
#include <fstream>
#include <cmath>
using namespace std;
int v[100001], s[100001];
int n, i, maxim, j, p, u, st, dr;
ifstream fin ("secvpal.in");
ofstream fout("secvpal.out");
```

```
int main() {
    fin>>n;
    for (i=1;i<=n;i++) {
        fin>>v[i];
    }
    for (i=1;i<=n;i++)
        for (j=i+1;j<=n;j++) {
            /// verificam daca secventa dintre
            /// indicii i si j este palindromica
            int este = 1;
            st = i; dr = j;
            while (st <= dr) {
                if (v[st] != v[dr]) {
                    este = 0;
                    break;
                }
                st++;
                dr--;
            }
            if (este == 1)
                if (j-i+1 > maxim) {
                    maxim = j-i+1;
                    p = i;
                    u = j;
                }
        }
    fout<<p<<" "<<u;
}
```

A doua soluție permite obținerea de timp de executare de ordin n^2 prin următoarea schimbare de optică: nu mai fixăm capetele secvenței (ce ar însemna deja n^2) ci fixăm mijlocul. Apoi, ne extindem în același timp în stânga și în dreapta cât timp pe pozițiile curente sunt valori egale. Când nu mai putem extinde oprim căutarea pentru mijlocul curent. Acest lucru funcționează întducât, cu un mijloc fixat, dacă nu avem o secvență palindromică de o anumită lungime, nu avem nici una de lungime mai mare, iar dacă avem de o anume lungime, avem și de lungimi mai mici. La fixarea mjlocului avem de tratat două cazuri: al secvenței de lungime impară (deci cu un singur mijloc) și al secvenței de lungime pară (în care elementul fixat este unul dintre cele două din mijloc).

```
#include <fstream>
using namespace std;
ifstream fin("secvpal.in");
ofstream fout("secvpal.out");
int n, a[1001], st, dr, i, j, L, maxim, p, u;
int main()
{
    fin>>n;
    for(int i=1; i<n; i++)
        fin>>a[i];
    for (i=1;i<=n;i++) {
        /// calculez lungimea maxima a unui palindrom cu centrul in i
        /// si de lungime impara
        st = i;
        dr = i;
```

```
        L = 1;
        while (st-1 >= 1 && dr + 1 <= n && a[st-1] == a[dr+1]) {
            st--;
            dr++;
            L = dr-st+1;
        }
        if (L > maxim) {
            maxim = L;
            p = st;
            u = dr;
        } else
            if (L == maxim && st < p) {
                p = st;
                u = dr;
            }
        /// calculez lungimea maxima a unui palindrom cu centrul in i
        /// si de lungime para
        st = i;
        dr = i+1;
        L = 1;
        while (st-1 >= 1 && dr + 1 <= n && a[st-1] == a[dr+1]) {
            st--;
            dr++;
            L = dr-st+1;
        }
        if (L > maxim) {
            maxim = L;
            p = st;
            u = dr;
        } else
            if (L == maxim && st < p) {
                p = st;
                u = dr;
            }
    }
    fout<<p <<" "<<u;

    return 0;
}
```