Algoritmi care lucrează cu tablouri unidimensionale
25.11.2023

1. Fie șirul $X=(x_1, x_2, \ldots, x_n)$ de numere întregi. Scrieți un subalgoritm care construiește șirul Y, format din elementele șirului X în ordinea $(x_1, x_3, x_5, \ldots, x_6, x_4, x_2)$.

Exemple:

a) X= (1, 2, 3), Y=(1, 3, 2)

b) X= (1, 2, 3, 4), Y=(1, 3, 4, 2)

c) X = (1, 2, 3, 4, 5, 6, 7), Y=(1, 3, 5, 7, 6, 4, 2)

d) X=(2, 6, 3, 9, 10, 12, 15, 1, 8), Y=(2, 3, 10, 15, 8, 1,12 ,9 ,6)

Observăm că putem împărți șirul X în **n div 2** perechi de forma (X_{2*i-1}, X_{2*i}), i=1,2,..., n/2. Primul element din pereche va fi adăugat în șirul Y în partea stângă (pe poziția i), iar al doilea în partea dreaptă (pe pozitia n-i+1). Pozițiile i și n-i+1 se află la egală depărtare de mijlocul șirului Y.

De exemplu, perechea 1 (i=1), formată din elementele (X_1, X_2) va fi adăugată în șirul Y pe poziția 1, respectiv n, iar perechea 2 (i=2), formată din elementele (X_3, X_4) va fi adăugată în șirul Y pe pozitia 2, respectiv n-1.

Pornind de la această observație încercăm să construim șirul Y punând elementele fiecărei perechi pe pozițiile corespunzătoare.

//Soluția 1

Parcurgem șirul X și încercăm să adăugăm elementul curent pe poziția aferentă din șirul Y. Se observă că toate elementele din șirul X aflate pe poziții impare vor fi în prima jumătate a șirului Y, și că toate elementele din șirul X aflate pe poziții pare vor fi adăugate în a doua jumătate a șirului Y.

Vom folosi un contor k pentru a determina numărul perechii care trebuie adăugate în șirul Y. Doar după adăugarea unui element de pe o poziție pară din șirul X, incrementăm contorul corespunzător numărului perechii.

Date de intrare: X, n

Rezultate: Y

```
Subalgoritmul construiesteSir1(X, n, Y) este:
       k=1;
       Pentru i=1, n, 1 executa:
               Daca i mod 2 == 1 atunci
                        Y[k]=X[i];
               altfel
                       Y[n-k+1]=X[i];
                        k=k+1;
               sfdaca
      stPentru
sfSubalgoritm
```

//Soluția 2

Folosim două variabile auxiliare (k, z) care păstrează poziția din stânga (variabila k), respectiv poziția din dreapta (variabila z) a șirului Y care trebuie completate. Parcurgem șirul X de la stânga la dreapta, iar dacă poziția curentă din șirul X este un număr impar, completăm poziția din stânga a șirului Y (pozitia k) și actualizăm următoarea poziție din stânga care trebuie completată, iar dacă poziția curentă din șirul X este un număr par, completăm poziția din dreapta a șirului Y (poziția z) și actualizăm următoarea poziție din dreapta care trebuie completată.

Date de intrare: X, n
Rezultate: Y

```
Subalgoritmul construiesteSir2(X,n,Y) este:
        k=1;   //prima poziție din stânga a șirului Y care va fi completată
        z=n;   //prima poziție din dreapta a șirului Y care va fi completată
       Pentru i=1, n, 1 executa:
                Daca i mod 2 = 1 atunci
                              Y[k]=X[i];
                              k=k+1;
                altfel
                           Y[z]=X[i];
                              z=z-1;
               sfdaca
       stPentru
Sfsubalgoritm
```

//Soluția 3

Ținem cont de observația că șirul X poate fi împărțit în n div 2 perechi și că pentru fiecare pereche i (i=1,2, …, n div 2) formată din elementele (Y_i, Y_{n-i+1}) putem obține pozițiile elementelor din șirul X care formează perechea respectivă (X_{2*i-1}, X_{2*i}):
Perechea 1: (Y_1, Y_n)=(X_1, X_2)
Perechea 2: (Y_2, Y_{n-1})=(X_3, X_4)
…
Perechea i: (Y_i, Y_{n-i+1})=(X_{2*i-1}, X_{2*i})

Date de intrare: X, n
Rezultate: Y

```
Subalgoritmul construiesteSir3(X,n,Y) este:
        k=n div 2;
        Pentru i=1, k, 1 executa:
                Y[i]=X[2*i-1];
                Y[n-i+1]=X[2*i];
        stPentru
        Daca n mod 2 ==1 atunci   //daca n este impar, ultimul element din sirul X va
                    Y[k+1]=X[n]      //aparea pe  pozitia din mijloc a sirului Y
        sfdaca
Sfsubalgoritm
```

2. Fie șirul $X=(x_1, x_2, \ldots, x_n)$ de numere întregi, introdus în ordinea $(x_1, x_3, x_5, \ldots, x_6, x_4, x_2)$. Scrieți un subalgoritm care determină ordinea inițială a elementelor din vector.
Exemplu:
a) Șirul introdus este (1,3,2), ordinea inițială a elementelor din vector este (1,2,3)
b) Șirul introdus este (4, 8, 3, 5), ordinea inițială a elementelor din vector este (4, 5, 8, 3)
c) Șirul introdus este (1,3,5,7,6,4,2), ordinea inițială a elementelor din vector este (1,2,3,4,5,6,7)

Notăm cu Y șirul introdus, $Y= (y_1, y_2, \ldots, y_n)=(x_1, x_3, x_5, \ldots, x_6, x_4, x_2)$. Observăm că această problemă este inversa problemei 1: fiind dat șirul Y, trebuie să obținem șirul inițial X și că fiecare pereche (Y_i, Y_{n-i+1}), i=1,2,..., n div 2 este formată din perechea (X_{2*i-1}, X_{2*i})

//Soluția 1
Parcurgem toate perechile (Y_i, Y_{n-i+1}), i=1,2,..., n div 2 și reconstituim perechea corespunzătoare din șirul X: (X_{2*i-1}, X_{2*i})

Date de intrare: Y, n
Rezultate: X

```
Subalgoritmul reconstruiesteSirA(Y,n, X) este
        k=n div 2;          //nr. de perechi din șir
        j=1;          //poziția curentă din șirul X
        Pentru i=1, k, 1 executa:              //parcurgem fiecare pereche din șirul Y
                X[j]=Y[i];
                j=j+1;     //trecem la următoarea poziție din șirul X
                X[j]=Y[n-i+1];
                j=j+1;      //trecem la următoarea poziție din șirul X
       sfPentru
       Daca n mod 2==1 atunci   //dacă n este impar, ultimul element din șirul X va fi
                X[j]=Y[k+1]       // elementul aflat pe poziția din mijloc în șirul Y
       sfDaca
sfSubalgoritm
```

//Soluția 2
Parcurgem șirul Y simultan de la stânga spre dreapta (folosind variabila st) și de la dreapta spre stânga (folosind variabila dr) si reconstituim șirul X:

Date de intrare: Y, n
Rezultate: X

```
Subalgoritmul reconstruiesteSirB(Y,n, X) este
      st=1;   //parcurge șirul Y  de la stânga spre dreapta
      dr=n;   //parcurge șirul Y  de la dreapta spre stânga
      i=1;      //poziția curentă din șirul X
     Cattimp (st<dr) executa:   //cat timp nu am parcurs toate elementele
            X[i]=Y[st];
             st=st+1;  //trecem la următoarea poziție din șirul Y
             i=i+1;     //trecem la următoarea poziție din șirul X
```

```
            X[i]=Y[dr];
            dr=dr-1;   //trecem la următoarea poziție din șirul Y
            i=i+1;   //trecem la următoarea poziție din șirul X
      sfCat
      Daca (st=dr) atunci   //daca n este impar, obținem valoarea ultimului
                X[i]=Y[st];     //element din sirul X
                   //sau X[i]=Y[dr]
      sfDaca
sfSubalgoritm
```

3. Fie ***a*** un șir de n numere naturale distincte ***(a_1,a_2, …, a_n)*** ordonat crescător și un numar natural ***S***. Scrieți un subalgoritm care determina numărul perechilor (a_i, a_j) care îndeplinesc condiția $S=a_i+a_j$ și i<j.
Exemplu:
a=(2,4,5,8,10,11), S=12 => 2 perechi (2,10) și (4,8)

//Soluția 1
O soluție simplă este parcurgerea șirului **a** cu două contoare `i,j` (`i`=1,2, …, n-1 și `j`=i+1,i+2, …, n) și verificarea dacă perechea (a_i,a_j) îndeplinește condiția $a_i+a_j=S$.

Date de intrare: `a, n, S`
Rezultate: `perechi`

```
Subalgoritm sumaS1(a,n, S) este:
       perechi=0; //la inceputul numărul de perechi este 0
       Pentru i=1, n-1, 1 este:
           Pentru j=i+1, n,1 este:
                 Daca (a[i]+a[j]==S) atunci perechi=perechi+1
                 sfDaca
           SfPentru
       SfPentru
      Returneaza perechi;
SfSubalgoritm
```

Această soluție are complexitatea $O(n^2)$ și nu ține cont de condițiile din enunț care precizează că șirul este ordonat crescător și că elementele șirului sunt distincte.

//Soluția 2
O altă soluție, care ține cont de faptul că șirul **a** este ordonat crescător este să folosim căutarea binară. Parcurgem șirul **a**, începând cu elementul cel mai mic. Pentru fiecare element a_i verificăm, folosind căutarea binară, dacă și valoarea $S-a_i$ apare în șirul **a** printre elementele a_{i+1}, …, a_n. Verificarea se face doar daca $S-a_i>a_i$.

Date de intrare: `a, n, S`
Rezultate: `perechi`

```
Subalgoritm sumaS2(a,n, S) este:
       perechi=0;   //la inceput numarul de perechi este 0
       Pentru i=1, n, 1 este:
```

```
            x=S-a[i];
            Daca  (x>a[i]) si (CautareBinara(a,x,i+1,n)==true) atunci
                          perechi=perechi+1;
            sfDaca;
         SfPentru
      Returneaza perechi;
SfSubalgoritm
```

Subalgoritmul `CautareBinara(a,x,poz_start,n)` caută valoarea **x** în șirul **a** cu **n** elemente, începând cu poziția `poz_start`, folosind algoritmul de căutare binară. Returnează `true` dacă valoarea **x** apare și `false` dacă nu a fost gasită. O posibila implementare a subalgoritmului `CautareBinara` este:

Date de intrare: `a, val, poz_start, n`
Rezultate: `gasit`

```
Subalgoritmul CautareBinara(a, val, poz_start, n){
        st=start;
        dr=n;
        gasit=false;
        Cattimp (st<dr) and (not gasit) executa:
                 mij=(st+dr) div 2;
                Daca a[mij]==val atunci
                        gasit=true;
                altfel
                        Daca a[mij]<val atunci
                                st=mij+1;
                        altfel
                                dr=mij-1;
                        sfDaca
                sfDaca
        sfCat
        return gasit;
SfSubalgoritm
```

Complexitatea acestei soluții este O(n*log n).

//Soluția 3
O altă soluție posibilă, care ține cont de condițiile din enunț (șirul **a** este ordonat crescător și elementele șirului sunt numere distincte) și încearcă să obțină numărul perechilor care satisfac condiția, este să parcurgem șirul o singură dată, dar să folosim două contoare: un contor **st** parcurge șirul de la pozitia 1 spre n, iar un contor **dr** de la poziția n spre 1. Pentru fiecare pereche `(st,dr)`, `st<dr`, astfel obținută, verificăm condiția ca $a_{st}+a_{dr}=S$. Dacă perechea îndeplinește condiția, incrementăm contorul ***st*** și decrementăm contorul ***dr*** pentru a căuta o altă pereche care îndeplinește condiția.
Dacă $a_{st}+a_{dr}>S$ înseamnă că trebuie să scădem valoarea sumei $a_{st}+a_{dr}$ și încercăm cu elementul aflat pe poziția `dr-1`, iar dacă $a_{st}+a_{dr}<S$ înseamnă că trebuie să mărim valoarea sumei $a_{st}+a_{dr}$ și încercăm cu elementul aflat pe pozitia `st+1`. Incercăm cu aceste valori pentru că în enunțul problemei s-a precizat că

șirul este ordonat strict crescator și atunci dacă $a_{st}<a_{st+1}$, respectiv $a_{dr-1}<a_{dr}$ => $a_{st}+a_{dr}<a_{st+1}+a_{dr}$ și $a_{st}+a_{dr-1}<a_{st}+a_{dr}$.
Subalgoritmul pentru această soluție este:

Date de intrare: a, n, S
Rezultate: perechi

```
Subalgoritm sumaS3(a,n, S) este:
       perechi=0; //la inceputul numarul de perechi este 0
       st=1;
       dr=n;
       Cattimp (st<dr) executa:
             suma=a[st]+a[dr];
             Daca (suma==S) atunci
                        st=st+1;
                        dr=dr-1;
                        perechi=perechi+1;
             altfel
                Daca (suma<S) atunci st=st+1
                              altfel dr=dr-1;
                sfDaca
             sfDaca
      sfCat
      Returneaza perechi;
SfSubalgoritm
```

Această soluție are complexitatea O(n) întrucât șirul **a** este parcurs o singură dată.

4. Fie ***a*** un șir de n numere naturale ***(a_1,a_2, …, a_n)***. Scrieti un subalgoritm care determină numărul tripletelor (a_i, a_j, a_k) care îndeplinesc condiția că unul dintre numere este egal cu suma celorlalte două numere și i<j<k ($a_i+a_j=a_k$ sau $a_i+a_k=a_j$ sau $a_j+a_k=a_i$).

a=(2,5,8,10,12,3,4) => 5 triplete (2, 5, 3), (2,8,10), (2,10,12), (5,8,3) și (8,12, 4)

O soluție este parcurgerea șirului **a** cu trei contoare i,j,k (i=1,2, …, n-2, j=i+1,i+2, …, n-1 și k=j+1, j+2,…, n) și verificarea dacă tripletul (a_i,a_j, a_k) îndeplinește condiția ($a_i+a_j=a_k$ sau $a_i+a_k=a_j$ sau $a_j+a_k=a_i$).

Date de intrare: a, n
Rezultate: triplete

```
Subalgoritm Triplete(a,n) este:
  triplete=0;
  Pentru i=1, n-2, 1 este:
     Pentru j=i+1, n-1, 1 este:
        Pentru k=j+1, n, 1 este:
          Daca (a[i]+a[j]==a[k]) sau (a[i]+a[k]==a[j]) sau (a[j]+a[k]==a[i]) atunci
                 triplete=triplete+1;
          sfDaca
         sfPentru
```

```
      sfPentru
   sfPentru
  Returneaza triplete;
SfSubalgoritm
```

Această soluție are complexitatea $O(n^3)$.

5. Fie ***a*** un șir de n numere naturale distincte ***$(a_1,a_2, \ldots, a_n)$*** ordonat crescător. Scrieți un subalgoritm care determină numărul tripletelor (a_i, a_j, a_k) care îndeplinesc condiția că unul dintre numere este egal cu suma celorlalte două numere și i<j<k ($a_i+a_j=a_k$ sau $a_i+a_k=a_j$ sau $a_j+a_k=a_i$).

a=(2,3,4,5,8,10, 12) => 5 triplete (2,3,5), (2,8,10), (2,10,12), (3,5,8) si (4,8,12)

//Soluția 1
Pentru a rezolva cerința acestei probleme putem folosi soluția de la problema 4, care are complexitatea $O(n^3)$ și care nu ține cont de toate condițiile din enunț (numerele din șir sunt distincte și sunt ordonate crescător).

//Soluția 2
Intrucât elementele șirului ***a*** sunt distincte și ordonate crescător, putem deduce că problema se rezumă la determinarea numărului de triplete (a_i, a_j, a_k), cu i<j<k care îndeplinesc condiția $a_i+a_j=a_k$, deoarece $a_i<a_j<a_k$. De asemenea, dacă alegem un element a_k din șirul ***a*** (k=n,n-1, …, 3), pentru a determina numărul de perechi (a_i, a_j), i<j care îndeplinesc condiția $a_i+a_j=a_k$, putem folosi una din soluțiile problemei 3 (preferabil cea cu complexitatea cea mai mică) unde înlocuim pe S cu valoarea a_k.
Pentru a determina numărul total de triplete, parcurgem șirul ***a*** de la dreapta spre stânga cu variabila k, k=n, n-1, …, 3, și pentru fiecare element a_k determinăm numărul de perechi (a_i, a_j), 1<=i<j<k, care îndeplinesc condiția $a_i+a_j=a_k$, folosind subalgoritmul sumaS3. Numărul total de triplete va fi suma tuturor numerelor perechilor determinate.

Date de intrare: a, n
Rezultate: triplete

```
Subalgoritmul TripleteB(a, n) este:
    triplete=0;
    k=n;
    Cattimp (k>2) executa:
          triplete=triplete+sumaS3(a, k-1, a[k]);
          k=k-1;
    sfCat
   Returneaza triplete;
SfSubalgoritm
```

Complexitatea acestei soluții este $O(n^2)$.

6. Fie șirul $X=(x_1,x_2, \ldots, x_n)$ de numere naturale distincte. Scrieți un subalgoritm care determină numărul tripletelor (x_i, x_j, x_k) din șirul X, care verifică teorema lui Pitagora și i<j<k.

Exemplu: X=(2,5,3,4,13,12,7) => 2 triplete (5 ,3, 4) și (5,13,12)

Teorema lui Pitagora: Dacă ***a*** și ***b*** sunt catetele unui triunghi dreptunghic și ***c*** este ipotenuza, atunci $a^2+b^2=c^2$.

//Soluția 1
Pentru a rezolva cerința acestei probleme putem scrie o soluție asemănătoare cu soluția de la problema 4, în care condiția de îndeplinit va fi $a_i^2+a_j^2=a_k^2$ sau $a_i^2+a_k^2=a_j^2$ sau $a_k^2+a_j^2=a_i^2$. Această soluție va avea complexitatea $O(n^3)$.

//Soluția 2
Observăm că problema 6, poate fi rezolvată folosind subalgoritmul de la problema 5 (`TripleteB`), dar sunt necesare câteva preprocesări pentru a îndeplini condițiile menționate în enunțul problemei 5: elementele șirului să fie ordonate crescător, iar valorile asupra cărora se va aplica subalgoritmul trebuie sa fie deja ridicate la patrat.
Pentru aceasta vom folosi un șir auxiliar Y, în care păstrăm pătratele numerelor din șirul X și apoi sortăm crescător șirul Y folosind unul din algoritmii de sortare studiați (bubble-sort, quicksort, etc). La final vom apela subalgoritmul `TripleteB` (solutia problemei 5), asupra vectorului Y pentru a determina numărul de triplete din șirul X care verifică teorema lui Pitagora.

Pentru exemplu dat, șirul `Y` inițial va fi `Y=(4,25,9,16,169,144,49)`, dupa sortare va deveni `Y=(4,9,16,25,49,144,169)`. Acest șir va fi folosit ca și date de intrare pentru subalgoritmul `TripleteB.`

Date de intrare: `x, n`
Rezultate: `triplete`

```
Subalgoritm SumaPitagora (x, n) este:
    Pentru i=1, n, 1 este:
        y[i]=x[i]*x[i];
    sfPentru
    ordoneaza(y,n);            //sortează crescător folosind un algoritm de sortare studiat
                               //(bubble sort, quicksort, etc)
    Returneaza TripleteB(y,n)
SfSubalgoritm
```

Complexitatea acestei soluții este $O(n^2)$ (sortarea șirului Y în cel mai rău caz va avea complexitatea $O(n^2)$, iar `TripleteB` are, de asemenea complexitatea $O(n^2)$).

Grile:

4. Se consideră algoritmul `f(n, x)`, unde n este număr natural ($3 \leq n \leq 10^4$), iar x este un vector de n numere naturale ($x[1], x[2], ..., x[n]$, $1 \leq x[i] \leq 10^4$, pentru $i = 1, 2, ..., n$).

```
Algorithm f(n, x):
    k ← 0
    For i ← 1, n - 1 execute
        If k = 0 then
            If x[i] = x[i + 1] then
                Return False
            EndIf
            If x[i] < x[i + 1] then
                k ← 1
            EndIf
        EndIf
        If k = 1 then
            If x[i] ≥ x[i + 1] then
                Return False
            EndIf
        EndIf
    EndFor
    If x[n - 1] ≥ x[n] then
        Return False
    EndIf
    Return True
EndAlgorithm
```

Pentru care din următoarele apeluri algoritmul va returna *True*?

A. `f(6, [1000, 512, 23, 22, 1, 2])`
B. `f(6, [6, 4, 1, 1, 2, 3])`
C. `f(8, [3000, 2538, 799, 424, 255, 256, 299, 1001])`
D. `f(3, [3, 2, 1])`

Admitere, sesiunea iulie 2023
Raspuns corect: A,C

6. Se consideră algoritmul `p(na, a, nb, b)`, unde na şi nb sunt numere naturale ($0 \leq na, nb \leq 10^4$), a şi b sunt vectori cu na, respectiv nb numere naturale ($a[1], a[2], ..., a[na]$, $1 \leq a[i] \leq 10^4$, pentru $i = 1, 2, ..., na$ şi $b[1], b[2], ..., b[nb]$, $1 \leq b[i] \leq 10^4$, pentru $i = 1, 2, ..., nb$). Variabila locală c este un vector.

```
Algorithm p(na, a, nb, b):
    i ← 1
    j ← 1
    nc ← 0
    While i ≤ na AND j ≤ nb execute
        nc ← nc + 1
        If a[i] < b[j] then
            c[nc] ← a[i]
            i ← i + 1
        Else
            c[nc] ← b[j]
            j ← j + 1
        EndIf
    EndWhile
    Return nc
EndAlgorithm
```

Care dintre următoarele afirmaţii sunt adevărate?

A. Dacă $na = 0$ şi $nb = 0$, atunci valoarea returnată prin nc este egală cu 0.
B. Dacă elementele din a şi b sunt sortate crescător, atunci elementele depuse în c sunt sortate crescător.
C. Valoarea returnată prin nc este întotdeauna egală cu $na + nb$.
D. Dacă $na, nb > 0$ şi cel mai mare element din a este mai mic decât toate elementele din b, atunci c va avea exact aceleaşi elemente ca şi a.

Exemple:
a=(1,5,2), b=(5,1,4,7,3)
a=(3,1,2), b=(7,5,6,8)

Admitere, sesiunea iulie 2023
Raspuns corect: A,B,D