Algoritmi care lucreaza cu tipuri de date definite de utilizatorPartea I

27.02.2021

Problema 1.

Enunţ.

Se dau 2 vectori de lungimi diferite, ce conţin numere întregi. Să se elaboreze o metodă care determină cea mai lungă secvenţă comună.

Exemplu: **X=(-5,4,-6,4,80,2,5,12,9,100)** de lungime m=10

şi **Y=(1,2,3,-5,4,-6,80,2,5,12,1,2,3,4)** de lungime n= 14

=> secvenţele colorate în **verde** constituie răspunsul, adică este o secvenţă comună de valori (prezentă în ambele tablouri), lungime maximă = 4.

Cum am putea să identificăm o secvenţă de numere dintr-un vector?

- Fie prin doi indici (p, q), p=indicele de început, q=indicele de final al secvenţei; (1,3) este secvenţa **roşie** din X, iar (7,10) este secvenţa **verde** din Y; evident (1,10) este secvenţa ce reprezintă vectorul X, etc;

- Fie prin cuplul (indice de început, lungime); exemplu (5,4) este secvenţa **verde** din X, iar (7,4) este secvenţa **verde** din Y; (secvenţa comună care evident are aceeaşi lungime).

Specificare (semiformală)

In: - $m,n \in N \setminus \{0,1\}$ (să avem vectori cu cel puţin 2 valori, fiecare);

- $X=(x_1,...,x_m)$, $x_i \in Z$, $Y=(y_1,...,y_n)$, $y_i \in Z$

Out:

- io: $io \in \{0,1,...,m\}$; jo: $jo \in \{0,1,...,n\}$;
- lungime: $lungime \in \{0,..., min\{m,n\}\}$ astfel încât:

  $lungime=0$, nu există elemente comune între cei doi vectori;

  sau $lungime>0$: $x_{io}=y_{jo},..., x_{io+lungime-1} = y_{jo+lungime-1}$ şi lungime e maximă

Observaţie: secvenţa comună este simbolizată prin cuplurile (io,lungime) în X şi (yo,lungime) în Y

```
//cea mai lunga secventa comuna dintre 2 vectori

void detSecCML(int m, int X[], int n, int Y[], int& io, int& jo, int& lungime){
   io=jo=lungime=0;
   for(int i=1;i<=m;i++){       /// i indice pentru X, (i,d) este secventa curenta
in X
      for (int j=1;j<=n;j++) {  /// j indice pentru Y, (j,d) este secventa curenta
in Y
        int d=0;                /// d pentru deplasament comun fata de i si j
                                /// verificam daca există secvente egale ce incep in
i si j
        while ((i+d<=m) && (j+d<=n) && (X[i+d]==Y[j+d]))
            d++;

        if (d>lungime)          ///  la iesire nu mai trebuie testat pe ce conditie
am iesit
                                ///  (cel putin o inegalitate  este falsa)
           { io     =i;         ///  se retine secventa mai lunga dintre
(io,lungime)
```

```
                lungime=d;          ///  si (i,d)
                jo     =j;          ///  in io se retine i (inceputul secventei din X)
                                    ///  iar in jo se retine j (inceputul secventei din
                        Y)
            }
          }
        }
}
```

Grila 1.

Se dă metoda A. Ce valori au parametri de ieşire B,D,C, (aprioric la apelul primar avem B=C=D=0) pentru:

```
void A(int m, int X[], int n, int Y[], int& B, int& D, int& C){
  for(int i=1;i<=m;i++){                    //B=D=C=0, la apelul primar
    for (int j=1;j<=n;j++) {
      int k=0;
      while ((i+k<=m) && (j+k<=n) && (X[i+k]==Y[j+k]))
              k++;
      if (k>D)
           { B  =i;
             D  =k;
             C  =j;
         }//end if
    }//end for
  }//end for
}//end A
```

B=5,D=3, C=7;

n=14, Y={-2,-7,1,5,-1,7,10,2,30,25,17,5,10,2}; ⇒ B=4,D=3, C=12;

C. m=5, X={1,1,1,1,1}
n=7, Y={1,1,1,1,1,1,1} ⇒ B=1,D=5,C=1;

D. m=5, X={1,1,1,1,1}
n=7, Y={1,1,1,1,1,1,1} ⇒ B=1,D=5,C=2;

Raspuns: B,C

Problema 2.

Enunţ.

Se dă un vector de lungime **n**, ce conţin numere întregi. Să se elaboreze o metodă care determină cea mai lungă secvenţă care se repetă (evident ele încep în poziţii diferite).

Exemplu: **x=(10,9,1,2,3,4,6,5,1,2 3,4 5)** de lungime n= 13

=> secvenţele colorate în **verde** constituie răspunsul, adică se repetă, lungimea maximă = 4.

Specificare (semiformală)

In: - $n \in N \setminus \{0,1\}$ (să avem vector cu cel puţin 2 valori);

- $X=(x_1,...,x_m)$, $x_i \in Z$;

Out:

- **io: io∈ {0,1,...,n}; jo: jo∈ {0,1,...,n};**
- **lungime: lungime∈ {0,...,n-1}** astfel încât:

lungime=0, nu există elemente care se repetă vector;

sau lungime>0 : $x_{io}=x_{jo}, \ldots, x_{io+lungime-1} = x_{jo+lungime-1}$ şi **lungime** e maximă

```
void secCareSeRepeta (int n, int X[], int& io, int& jo, int& lungime){
   io=jo=lungime=0;
   for(int i=1;i<n;i++){        /// i indice pentru X, (i,i+d-1) este secventa
curenta in X
      for (int j=i+1;j<=n;j++){ /// j indice pentru X, (j,j+d-1) este secventa
curenta in X
                                /// şi se repetă
        int d=0;                /// d pentru deplasament comun fata de i si j
                                /// verificam daca există secvente egale ce incep in
i si j
        while ((i+d<=n) && (j+d<=n) && (X[i+d]==X[j+d]))
            d++;

        if (d>lungime)          ///  la iesire nu mai trebuie testat pe ce conditie
am iesit
                                ///  (cel putin o inegalitate  este falsa)
           { io     =i;         ///  se retine secventa mai lunga dintre
(io,io+lungime-1)
             lungime=d;         ///  si (i,i+d-1)
             jo     =j;         ///  in io se retine i (inceputul secventei din X)
                                ///  iar in jo se retine j (inceputul secventei din
                          X)
           }
        }
      }
}
```

Grila 2.

Se dă metoda A. Ce valori au parametri de iesire B,D,C?

```
void A(int m, int X[], int& B, int& D, int& C){
  B=C=D=0;
  for(int i=1;i<m;i++){
    for (int j=i+1;j<=m;j++) {
      int k=0;
      while ((i+k<=m) && (j+k<=m) && (X[i+k]==X[j+k]))
              k++;
      if (k>D)
           { B  =i;
             D  =k;
             C  =j;
         }//end if
    }//end for
  }//end for
}//end A
```

Răspuns: A,B,C

Problema 3.

Enunţ.

Se dă un vector de lungime n. Să se sorteze crescător cu metoda quickSort.

Specificare (semiformală)

In: - $n \in N \setminus \{0,1\}$ (să avem vector cu cel puţin 2 valori);

- $V=(v_1,...,v_n)$, $v_i \in Z$;

Out:

- $v_1 \le v_2 \le ... \le v_n$ şi reprezintă o permutare a valorilor iniţiale ale vectorului **V**.

Rezolvare

Metoda a fost inventată de C.A.R. Hoare (în 1960) şi, în medie, efectuează **O(n*log$_2$n)** comparații pentru a sorta ***n*** elemente.

Quicksort efectuează sortarea bazându-se pe o strategie Divide et Impera. Astfel, se împarte secvenţa de sortat în două secvenţe mai ușor de sortat. Pașii algoritmului sunt:

1. Se alege un element al secvenţei, denumit **pivot**;
2. Se reconfigurează secvenţa astfel încât toate elementele strict mai mici decât **pivotul** să fie plasate înaintea **pivotului** și toate elementele mai mari să fie după **pivot**. După această partiționare, **pivotul se află în poziția sa finală**.
3. Se sortează apoi, **recursiv,** secvenţa de elemente strict mai mici decât pivotul și secvenţa de elemente mai mari decât pivotul, după acelaşi principiu.
4. O secvenţa de lungime 1 este considerată sortată.

Vom nota o secvenţa (după prima modalitate prezentată la problema 1) cu o pereche de indici **(st,dr);**

- **st reprezintă indicele de început al secvenţei;**
- **dr reprezintă indicele de final al secvenţei;**

Metoda de împărţire prin strategia Divide et Impera se poate scrie în C/C++ astfel:

```
void quickSort(int v[], int st, int dr){
  if(st<dr){
     int poz = pivotStanga(v,st,dr);
     quickSort(v,st, poz-1);
     quickSort(v,poz+1,dr);
  }
}
```

Trebuie apoi să definim metoda **pivStanga.** Vom lua drept pivot elementul egal cu **v[st]** apoi vom reorganiza secvenţa mutând acest element înspre dreapta până când obţinem cerinţa 2, de mai sus.

Exemplu: fie secvenţa **(9,3,15,12,-100,2), st=1, dr=6, deci secventa (1,6)**

- vom compara valoarea 9 cu 3 (primele 2 valori); pentru că **9 > 3**, facem o permutare circulară la dreapta a celor 2 valori astfel vom obţine secvenţa **(3,9,15,12,-100,2)**
- continuăm să comparăm apoi pe **9** pe rând cu valorile de după el, care sunt vecine între ele (ca indici) şi observăm că 15 şi 12 sunt ≥ 9 valorile rămân pe loc;
- în momentul când găsim o valoare strict mai mică ca 9 (pe -100), facem o nouă permutare circulară a valorilor colorate în roşu: **(3,9,15,12,-100,2) şi obţinem (3,-100,9,15,12,2)**
- în final **9>2** şi facem o nouă permutare circulară a valorilor cu roşu **(3,-100,9,15,12,2)**
- se obţine secvenţa **(3,-100,2,9,15,12); poziţia pivotului este 4**

- Concluzia:
  i=st; d=1, d=deplasament relativ fata de i
  se compara x[i] cu x[i+d]:
  a) cand x[i]> x[i+d] se face permutare circulara a secventei (i,i+d), creste i
  b) cand x[i]<=x[i+d] creste deplasamentul d

```
int pivotStanga(int v[], int st,int dr){
   int i=st;   ///in final i va deveni pozitia finala
pentru v[st]
   int d=1;    ///deplasament relativ fata de i
               ///se va compara v[i] cu v[i+d]
   while(i+d<=dr){
      if(v[i]<=v[i+d]) d++;   ///creste deplasamentul
relativ
      else{ permutCirc(v,i,i+d);
            i++;              ///i trebuie crescut
          }
   }
   return i;        ///pozitia finala a valorii initiale
v[st]
}
```

Se poate face şi o altă variantă, în care deplasamentul **d** nu este relativ la **i** ci devine indice:

```
int pivotStanga1(int v[], int st,int dr){
   int i=st;   ///in final i va deveni pozitia finala
pentru v[st]
   int d=st+1; ///d va indica indicele valorii cu care se
compara
               ///v[i], deci se va compara v[i] cu v[d]
   while(d<=dr){
      if(v[i]<=v[d]) d++;   ///creste indicele d
      else{ permutCirc(v,i,d);
            i++;            ///creste atat i cat si d
            d++;
```

```
            }
         }
         return i;        ///pozitia finala a valorii initiale
      v[st]
      }
```

Metoda care permută circular secvenţa (p,q) este următoarea:

```
     void permutCirc(int v[], int p, int q){ ///p<q !!!
          int aux=v[q];         ///se salveaza ultimul element in
aux
          for (int i=q;i>p;i--)///se muta toata secventa (p,q-1)
la drepta
               v[i]=v[i-1];     ///cu 1 pozitie
          v[p] = aux;           ///aux se mută in v[p] (prima
poziţie)
     }
```

Grila 3

Ce fac cele două metode? (schimb e metodă ce interschimbă 2 valori)

```
void b1(int n,int v[]){
   for(int k=1;k<n;k++){
     for(int i=1;i<n;i++)
       if(v[n-i+1]<v[n-i]) schimb(v[n-i+1],v[n-i]);
   }
}

void b2(int n,int v[]){
  for(int k=1;k<n;k++){
    for(int i=n;i>1;i--)
       if(v[i]<v[i-1]) schimb(v[i],v[i-1]);
   }
 }
```

a)b1 sortează crescător v, b2 sortează descrescator v;
b)b1 sortează desccrescător v, b2 sortează crescator v;
c)b1 sortează descrescător v, b2 sortează descrescator v;
d)b1 sortează crescător v, b2 sortează crescator v;

Raspuns d.