Secvența de SUMĂ MAXIMĂ

Clara Ionescu

Frumusețea problemelor de programare se ascunde în rezolvarea lor. O problemă care se rezolvă în multe linii sursă, cu un algoritm banal, nu prea are șanse să se remarce în rândul rezolvitorilor de probleme de algoritmică. Răsfoind notițe făcute "de demult" am (re)descoperit o problemă a cărui enunț este cât se poate de simplu, rezolvarea în schimb este foarte frumoasă.

Enunț

Se consideră un șir de `n` numere întregi, printre care există cel puțin un element pozitiv. Scrieți un program care determină *secvența având suma elementelor maximă*.

Exemplu

`n=10`
Șirul: `1 2 -6 3 4 5 -2 10 -5 -6`
Rezultate:
Suma maximă: `20`
Lungimea secvenței: `5`
Poziția de început a secvenței: `4`
Poziția de sfârșit a secvenței: `8`
Secvența având suma `20`: `3 4 5 -2 10`

Varianta I

Cea mai simplă idee care conduce la o rezolvare a acestei probleme ar fi de a genera toate perechile de numere întregi `st` și `dr` având proprietatea: $1 \le st \le dr \le n$, calculând pentru fiecare pereche suma elementelor secvenței `T[st..dr]` și determinând secvența având suma maximă.

În acest caz algoritmul ar avea următorul nucleu:

```
Suma_Max:=T[1];
            { Suma_Max: suma maxima cautata }
for st:=1 to n do
       { st: pozitia de inceput a secventei }
  for dr:=st to n do
       { dr: pozitia de sfarsit a secventei }
  begin
    sum:=0;
              { sum: suma secventei actuale }
    for i:=st to dr do
      sum:=sum+T[I];
    if Suma_Max<sum
      then Suma_Max:=sum
  end;
```

Complexitatea acestui algoritm, format din trei cicluri imbricate este $O(n^3)$. Pentru $n = 1000$ un calculator cu performanțe medii va lucra aproape o oră. Această complexitate nu poate fi acceptată, de aceea vom căuta modalități de micșorare a complexității, astfel încât să avem mai întâi un algoritm pătratic, apoi unul liniar.

Varianta II

Căutând optimizări posibile, observăm că suma secvenței `T[st..dr]` poate fi determinată în funcție de suma calculată anterior, pentru secvența `T[st..dr-1]`. Evident, orice sumă `T[st..dr]` poate fi comparată cu suma maximă obținută până în acel moment, imediat după ce a fost calculată. Rezultă un nucleu de complexitate $O(n^2)$, deoarece în algoritm vom avea doar două cicluri imbricate:

```
Suma_Max:=T[1];
for st:=1 to n do
begin
  sum:=0;
  for dr:=st to n do
  begin
    sum:=sum+T[dr];
    if Suma_Max<sum then Suma_Max:=sum
  end
end;
```

Varianta III

Al doilea algoritm pătratic are la bază o idee nouă, dar care folosește spațiu suplimentar, dublând necesarul de memorie. În rezolvarea pe care o propunem în această variantă, presupunem că dimensiunea șirului nu va fi așa de mare încât acest inconvenient să facă imposibilă obținerea soluției pe această cale.

Vom nota cu `Sum_Part[i]` suma elementelor secvenței `T[1..i]`. Vom calcula aceste sume parțiale pentru valorile `i=0,1,…,n` la începutul algoritmului și le vom memora în tabloul `SumPart[0..n]`. În consecință, suma elementelor unei secvențe `T[st..dr]` se calculează simplu, folosind elementele acestui tablou auxiliar:

```
T[st..dr]=Sum_Part[dr]-Sum_Part[st-1].
```

Programul în care s-a implementat algoritmul care are la bază ideea prezentată, este următorul:

Listing VAR3.PAS

```
Program Secventa_de_suma_maxima;
const n=100;
var T:array[1..n] of Integer;  { sirul dat }
    Sum_Part:array[0..n] of Integer;
                 { sirul sumelor partiale }
    inceput,sfarsit,st,dr,i:Byte;
    Suma_Max,sum:Integer;
Begin
  Writeln('Sirul dat: ');
  Sum_Part[0]:=0;
  { este nevoie de Sum_Part[i-1] cand i=1 }
  for i:=1 to n do
          { in paralel cu citirea sirului }
          { se calculeaza sumele partiale }
  begin
    Read(T[i]);
    Sum_Part[i]:=Sum_Part[i-1]+T[i]
  end;
  Writeln;
  Suma_Max:=T[1];
     { initializarea sumei maxime cautate }
  for st:=1 to n do
    for dr:=st to n do
    begin
      sum:=Sum_Part[dr]-Sum_Part[st-1];
                          { sume curente }
      if Suma_Max<sum then
      begin
        Suma_Max:=sum;
                   { suma maxima curenta }
        inceput:=st;
         { pozitia de inceput a secventei }
        sfarsit:=dr
         { pozitia de sfarsit a secventei }
      end
    end;
  Writeln('Secventa avand suma maxima: ');
  for i:=inceput to sfarsit do
    Write(T[i],' ');
  Writeln
End.
```

Evident, există și alte posibilități de a rezolva această problemă în complexitate pătratică; propunem cititorilor să caute un algoritm recursiv, folosind observațiile prezentate.

Varianta IV

În continuare urmează descrierea unui algoritm având complexitatea $O(n)$. Această performanță se obține datorită faptului că tabloul se parcurge o singură dată.

În paralel cu parcurgerea șirului de la stânga la dreapta vom determina și memora la fiecare pas suma maximă posibilă, în variabila `Suma_Max`. Să presupunem că am aflat suma maximă și secvența `T[1..i-1]` respectiv. Se pune problema extinderii metodei pentru calcularea sumei maxime pentru `T[1..i]`.

Se observă că după parcurgerea primelor `i` elemente ale șirului, suma maximă este egală cu suma maximă a primelor `i-1` elemente (`Suma_Max`) sau este mai mare decât aceasta. Dacă este mai mare, înseamnă că din această sumă maximă face parte și elementul de pe poziția `i`, deci avem un subșir care are elementul de sfârșit cel de-al `i`-lea element din șirul dat (această sumă o vom numi `Suma_C`).

Suma `Suma_C` nu va fi calculată de fiecare dată de la început, ci vor fi aplicate raționamentele deja cunoscute: se va utiliza suma maximă a secvenței ce se termină pe poziția `i-1`. În consecință obținem:

```
Suma_Max:=T[1];
Suma_C:=Suma_Max;
for i:=1 to n do
begin
  Suma_C:=max(Suma_C+T[i],0);       {1}
  Suma_Max:=max(Suma_Max,Suma_C)    {2}
end;
```

La examinarea acestei secvențe o deosebită atenție merită acordată variabilei `Suma_C`. Înainte de a fi executată instrucțiunea de atribuire `{1}` din corpul ciclului, valoarea acestei variabile este egală cu suma maximă a elementelor secvenței care se termină în poziția `i-1`. În urma execuției acestei instrucțiuni de atribuire în `Suma_C` va fi depusă suma maximă a elementelor secvenței care se termină în poziția `i` și începe într-o poziție memorată în momentul începerii însumării elementelor secvenței începând cu poziția `poz_inceput`. Evident, la primul pas această variabilă are valoarea `1`, indicele primului element din șir.

Este momentul să ne aducem aminte că șirul dat conține *cel puțin un element de valoare pozitivă*, (proprietate neexploatată în nici una din variantele de rezolvare prezentate). Din această particularitate rezultă că dacă șirul ar avea `n-1` elemente negative și unul singur pozitiv, secven-

ța având suma maximă s-ar reduce la un singur element și anume la acest număr pozitiv care ar da naștere la o sumă pozitivă. Rezultă că suma maximă căutată în orice situație va fi o sumă pozitivă. În consecință, cât timp suma curentă este pozitivă, prin atribuirea {1}, lui Suma_C i se adaugă T[i]. Dacă, la un moment dat se constată că Suma_C este negativă, atunci acestei variabile i se atribuie valoarea elementului T[i].

În ambele cazuri variabila Suma_C va fi comparată cu Suma_Max, actualizând valoarea Suma_Max după caz. În continuare urmează varianta finală a programului *Pascal*.

Listing VAR4.PAS

```
Program Secventa_de_suma_maxima;
const n=100;
var T:array[1..n] of Integer;  { sirul dat }
    Suma_Max,       { suma maxima cautata }
    Suma_C:Integer;
                    { suma maxima curenta }
    inceput,
         { pozitia de inceput a secventei }
    sfarsit,
         { pozitia de sfarsit a secventei }
    poz_inceput,    { pozitia de inceput a }
                    {   secventei curente }
    i:Byte;
Begin
  Writeln('Sirul dat: ');
  for i:=1 to n do Read(T[i]);
  Writeln;
            { initializarea sumei maxime  }
  Suma_Max:=T[1];
            { initializarea sumei curente }
  Suma_C:=Suma_Max;
      { initializarea pozitiei de inceput }
      {               a secventei curente }
  poz_inceput:=1;
      { initializarea pozitiei de inceput }
      {        a secventei de suma maxima }
  inceput:=1;
      { initializarea pozitiei de sfarsit }
      {        a secventei de suma maxima }
  sfarsit:=1;
  for i:=2 to n do
  begin
    if Suma_C<0 then
              { daca Suma_C este negativa }
    begin
     { aceasta se reinitializeaza cu T[i] }
      Suma_C:=T[i];
       { noua secventa posibila incepe la }
       {                          pozitia i }
      poz_inceput:=i
    end
    else
              { lui Suma_C i se adauga T[i] }
      Suma_C:=Suma_C+T[i];
    if Suma_Max<Suma_C then
                { se actualizeaza Suma_Max }
    begin
      Suma_Max:=Suma_C;
       { actualizarea pozitiei de inceput }
      inceput:=poz_inceput;
       { actualizarea pozitiei de sfarsit }
      sfarsit:=i
    end
  end;
  Writeln('Suma maxima: ',Suma_Max);
  Writeln('Lungimea secventei: ',
                         sfarsit-inceput+1);
  Writeln('Incepe din pozitia: ',inceput);
  Writeln('Sfarsitul in pozitia: ',sfarsit);
  Writeln('Secventa avand suma maxima: ');
  for i:=inceput to sfarsit do
    Write(T[i],' ');
  Writeln
End.
```

În urma executării programului pentru șirul T=(4,2,-7,9,-10,3,10,-3,5,-15), rezultatele furnizate de program sunt:

```
Suma maxima: 15
Lungimea secventei: 4
Incepe din pozitia: 6
Sfarsitul in pozitia: 9
Secventa avand suma maxima: 3 10 -3 5
```

În urma executării programului pentru șirul T=(1,2,3,-1,-2,-3,6,7,-7,8), rezultatele furnizate sunt:

```
Suma maxima: 14
Lungimea secventei: 10
Incepe din pozitia: 1
Sfarsitul in pozitia: 10
Secventa avand suma maxima: 1 2 3 -1 -2 -3 6
                                    7 -7 8
```

Se observă că secvența obținută ca rezultat are lungimea maximă printre secvențele care au aceeași sumă maximă. În acest exemplu secvența 6 7 -7 8 are, de asemenea, suma 14. Lăsăm pe seama cititorului să caute o soluție care să furnizeze ca rezultat secvența de lungime minimă și având suma maximă. De asemenea, se poate pune întrebarea: cum procedăm dacă ne interesează prima secvență având proprietatea cerută, ultima sau eventual, toate secvențele.

Bibliografie:

Gh. Bostan, I. Mocanu, *Al VII-lea concurs republican de informatică*, Chișinău, 1993, nepublicat

D-na lect. univ. Clara Ionescu este redactor-șef al GInfo. Poate fi contactată prin e-mail la adresa clara@cs.ubbcluj.ro.