TEOREME de programare

Clara Ionescu

Continuăm prezentarea unor algoritmi simpli care vor conferi programelor claritate, iar programatorilor siguranța unei rezolvări corecte.

Partiționarea unui șir în două subșiruri

În numărul precedent am rezolvat câteva probleme în care se dădea un șir și se cerea selecționarea mai multor elemente, având o aceeași proprietate dată. Se pune întrebarea: ce se întâmplă cu elementele neselecționate? În acest episod al serialului nostru vom da răspuns la această întrebare.

P21. Se consideră *n* numere naturale. Să partiționăm șirul în două subșiruri astfel încât unul să conțină numerele pare, iar celălalt numerele impare!

P22. Să se creeze subșirul fetelor, respectiv al băieților pe baza șirului numelor elevilor unei clase!

P23. Cunoscând mediile anuale dintr-un catalog, să se creeze subșirul elevilor care au media mai mare decât 9, din cei rămași subșirul celor cu media mai mare decât 8 și așa mai departe!

Prima particularitate pe care o observăm este că, de data aceasta, ***se dă un șir*** și ***se cer mai multe***. Practic, va trebui să efectuăm una după alta mai multe *selecționări*. De exemplu, în cazul problemei **P23**, mai întâi vom selecționa acele elemente care au prima proprietate, apoi din șirul elementelor puse deoparte (rămase după selecționare) selecționăm elementele având cea de a doua proprietate și așa mai departe. Rezultă că algoritmul constă din aplicarea repetată a *partiționării în două șiruri* a șirului dat. Dacă partiționarea se face în două subșiruri, este clar că elementele neselecționate pentru primul șir îl vor forma pe cel de al doilea.

Vom prezenta mai multe modalități de a realiza cerința acestui tip de problemă. În prima variantă elementele având proprietatea cerută și cele care nu au această proprietate le vom așeza în două șiruri noi. Aceste șiruri le vom declara având dimensiunea șirului dat, deoarece nu se poate anticipa numărul exact de elemente. Este posibil ca toate elementele să fie selecționate în primul șir sau nici unul. În *pseudocod*, lucrăm cu următoarele entități:

n: *întreg* {*numărul elementelor șirului de prelucrat*}
X: *tablou(1..n): tip element* {*elementele șirului dat*}
bucY, bucZ: *întregi* {*numărul elementelor subșirurilor*}
Y, Z: tablou(1..n): întreg { *elementele șirurilor noi*}

```
subalgoritm Partiționare1(n,X,bucY,Y,bucZ,Z):
  bucY ← 0
  bucZ ← 0
  pentru i ← 1, n execută
    dacă P(X[i]) atunci
      bucY ← bucY +1
      Y[bucY] ← X[i]
    altfel
      bucZ ← bucZ +1
      Z[bucZ] ← X[i]
    sfârșit dacă
  sfârșit pentru
sfârșit subalgoritm
```

Dacă am dori să economisim spațiul ripisit cu o astfel de soluție putem utiliza un singur șir. Elementele selecționate le reținem în acesta, așezându-le de la început spre sfârșit, iar pe cele rămase le așezăm de la ultimul element spre primul. Evident, nu există pericolul să "ne ciocnim", deoarece avem exact *n* elemente care trebuie așezate pe *n* locuri.

```
subalgoritm Partiționare2(n,X,bucY,Y):
  bucY ← 0
  pentru i ← 1, n execută
    dacă P(X[i]) atunci
      bucY ← bucY +1
      Y[bucY] ← X[i]
    altfel
      Z[n + 1 - i + bucZ] ← X[i]
    sfârșit dacă
  sfârșit pentru
sfârșit subalgoritm
```

Să observăm că această rezolvare ne furnizează șirul elementelor neselecționate în ordine inversă față de pozițiile elementelor sale în șirul dat. În cazul în care după partiționare nu avem nevoie de șirul dat în forma sa originală, partiționarea, exact ca selecționarea, poate fi realizată folosind spațiul alocat șirului dat, adică *pe loc*.

Ne putem imagina o soluție care ar folosi un algoritm conform căruia am avansa în șir până la descoperirea primului element care nu are proprietatea cerută și l-am interschimbat cu primul găsit care, în schimb, are proprietatea cerută. Această soluție nefiind suficient de eficientă, propunem următorul algoritm. Punem primul element deoparte păstrându-i valoarea într-o variabilă auxiliară și căutăm un element, având proprietatea cerută, pornind de la capătul șirului și îl punem pe locul eliberat la începutul șirului. Acum vom căuta, pornind de la această poziție, un element care nu are proprietatea cerută și îl așezăm pe locul eliberat la sfârșitul șirului. Continuăm acest procedeu până când, în urma avansării în cele două direcții, ne vom întâlni.

```
subalgoritm Partiționare3(n,X,buc):
  înc ← 1
  sf ← n
  aux ← X[înc]
  cât timp înc < sf
    cât timp (înc < sf) and not (P(X[sf]))
      sf ← sf - 1
    sfârșit cât timp
    dacă înc < sf atunci
      X[înc] ← X[sf]
      înc ← înc + 1
      cât timp (înc < sf) and (P(X[înc]))
        înc ← înc + 1
      sfârșit cât timp
      dacă înc < sf atunci
        X[sf] ← X[înc]
        sf ← sf - 1
      sfârșit dacă
    sfârșit dacă
  sfârșit cât timp
  X[înc] ← aux
  dacă P(X[înc]) atunci
    buc ← înc
  altfel
   buc ← înc - 1
  sfârșit dacă
sfârșit subalgoritm
```

Vom rezolva problemele enunțate la începutul articolului. În **P21** se cere șirul numerelor pare și cel al numerelor impare luate dintr-un șir de numere naturale. Vom rezolva problema cu algoritmul `Partiționare2` care reține cele două subșiruri în locul rezervat șirului dat.

Presupunem că în programul apelant am declarat tipul:

```
type tablou=array[1..50] of Word;
procedure P21(n:Byte; X:tablou; var buc:Byte;
                                  var Y:tablou);
begin
  buc:=0;
  for i:= 1 to n do
    if Odd(X[i]) then begin
      Inc(buc);
      Y[buc]:=X[i]
    end else
      Y[n+1-i+buc]:=X[i]
end;
```

Problema **P22** este foarte asemănătoare cu **P21**, deci rezolvarea ei o lăsăm pe seama cititorului.

Să vedem cum se rezolvă problema **P23**. În cazul acestei probleme vom aplica algoritmul de partiționare de mai multe ori, șirul de partiționat la pasul doi fiind cel pus deoparte după prima partiționare. Pentru a economisi spațiu, nu vom declara mai multe șiruri, ci vom realiza partiționarea și de data aceasta pe loc, folosind algoritmul `Partiționare3` cu parametri care se vor referi la șirul curent care trebuie partiționat.

Presupunem că avem declarate tipurile și variabilele:

```
type unelev=record
               nume:string[30];
               media:Real
             end;
     catalog=array[1..15] of unelev;
var n,i,primul,buc,limita:Byte;
    X:catalog;
procedure part(n,primul,limita:Byte; var
                      X:catalog; var buc:Byte);
var aux:unelev;
begin
  aux:=X[primul];
  while primul < n do begin
    while (primul < n) and
          not (X[n].media > limita) do Dec(n);
    if primul < n then begin
      X[primul]:=X[n];
      Inc(primul);
      while (primul < n) and
     (X[primul].media > limita) do Inc(primul);
      if primul < n then begin
        X[n]:=X[primul];
        Dec(n)
      end
    end
  end;
  X[primul]:=aux;
  if X[primul].media > limita then buc:=primul
  else buc:=primul - 1
end;
```

În programul apelant vom avea următoarea secvență în care apelăm repetat subprogramul `part`:

```
primul:=1     {primul indice al șirului curent}
limita:=9;{se caută medii mai mari decât limita}
 while limita >= 5 do begin     { medii posibile }
   part(n,primul,limita,x,buc);
   primul:=buc+1;     {indicele de început al}
         {subșirului care trebuie partiționat}
   Dec(limita)       {următorul grup de medii}
 end;
```