TEOREME de programare

Clara Ionescu

Sfătuim programatorii începători (dar și pe cei care se consideră "avansați") să respecte aceste modele de rezolvare a diferitelor probleme, considerate clasice și care apar frecvent ca subprobleme în multe probleme de rezolvat. Astfel, vor putea depana mult mai simplu programele scrise, le vor "citi" ușor și peste mult timp, dar le vor înțelege și alții. Nu este doar o recomandare din partea GInfo, ci și a majorității firmelor dezvoltatoare de software...

Selectarea elementului maxim

Fie următoarele enunțuri de probleme!

P15. Într-un spital s-a măsurat temperatura fiecărui bolnav internat. Să se stabilească cea mai mare temperatură măsurată!

P16. Având la dispoziție un tabel cu numele elevilor participanți la un concurs, să se stabilească acel nume cu care va începe tabelul nominal (în ordine alfabetică)!

P17. Într-o clasă se face o statistică, privind veniturile familiilor elevilor. Să se determine venitul minim, existent printre venituri!

Observăm că rezolvările acestor probleme - probabil - vor fi asemănătoare, deoarece în fiecare se cere stabilirea unei valori care este fie cea mai mare, fie cea mai mică în șirul dat. Bineînțeles, pot exista mai multe astfel de valori egale cu valoarea minimă sau maximă. Dar în problemele considerate nu se cere, de exemplu, să numărăm câți bolnavi au acea temperatură maximă și nu ne interesează nici numele acelui bolnav care este cel mai "fierbinte". Subalgoritmul care rezolvă această problemă (în termeni generali) va lucra cu o buclă de tip **pentru**, deoarece fiecare valoare trebuie prelucrată. În pseudocodul următor lucrăm cu entitățile:

n: *întreg* {*numărul elementelor șirului de prelucrat*}
x: *tablou(1..n): tip element* {*elementele șirului dat*}
val: *tip element* {*rezultatul*}

```
subalgoritm Maxim(n,x,val):
  val ← x[1]
  pentru i ← 2, n execută:
    dacă val < x[i] atunci
      val ← x[i]
    sfârșit dacă
  sfârșit pentru
sfârșit subalgoritm
```

Înainte de a trece la rezolvarea problemelor enunțate, să analizăm pseudocodul subalgoritmului. De ce nu am inițializat valoarea maximului (variabila `val`) cu 0, gândindu-ne că sigur va exista un bolnav cu temperatură pozitivă, deci nu există pericolul ca maximul să rămână 0? În rezolvarea problemei **P15** am putea lucra cu această valoare inițială, dar dacă nu ar fi vorba de temperatura unor bolnavi, ci de temperatura aerului în timpul unei luni într-o iarnă geroasă, ar fi posibil să nu avem nici o valoare pozitivă, și astfel maximul ar rămâne 0, păcălindu-ne... Rezultă că se recomandă inițializarea maximului (minimului) cu o valoare existentă sigur în șir, și, deoarece, de regulă, prelucrăm șirurile începând cu primul element, cel mai firesc este ca în inițializarea respectivă să folosim această valoare. Să analizăm oportunitatea de a implementa acest subalgoritm cu o procedură sau cu o funcție. Avem de determinat o singură valoare, deci funcția "ne face cu ochiul". Dacă am folosi o funcție denumită "`val`", în instrucțiunea **if** am avea un apel recursiv nedorit, deci ne-ar mai trebui o variabilă suplimentară, iar la sfârșit am atribui valoarea acesteia identificatorului de funcție. Presupunem că în programul apelant am declarat tipul:

```
type temp = array[1..50] of Real;
procedure P15(n:Byte; T:temp; var max:Real);
var i:Byte;
begin
  max:=T[1];
  for i:= 2 to n do
    if max < T[i] then
      max:=T[i]
end;
```

Având în vedere simplitatea implementării problemelor **P16** și **P17**, nu le vom detalia. În schimb, le vom modifica enunțul, astfel încât să vedem modul de rezolvare dacă se cere și poziția pe care se află minimul cerut. De exem-

plu, în problema **P16** se cere "cel mai mic nume". Să ne imaginăm că în problemă nu se cere numele, ci doar indicele în șirul dat al aceluia care va ajunge în tabelul nominal pe prima poziție! Este evident că rezultatul va fi nu o valoare dată în șir (de data aceasta o valoare de tip **string**), ci un număr de ordine. Presupunem că în programul apelant am declarat tipul:

```
type tabel = array[1..30] of string;
procedure P16_1(n:Byte; t:tabel; var ind:Byte);
var i:Byte;
    min:string;
begin
  min := t[1];    {presupunem că min este primul}
  ind := 1;       {dacă min rămâne primul, ind e 1}
  for i := 2 to n do
    if min > t[i] then begin
      min := t[i];            {cel mai mic nume}
      ind := i               {indicele acestuia}
    end
end;
```

Nu ne-am pus problema cu privire la "care" minim ne interesează, (respectiv, primul găsit, al doilea sau ultimul) deoarece într-un grup de elevi, probabil nu vor exista doi copii cu nume identic. Dar dacă transpunem problema la un șir de cuvinte luate dintr-un text oarecare, atunci este posibil să existe mai multe asemenea valori minime. Observăm că, dacă am utiliza acest subprogram în cazul respectiv, acesta ne-ar furniza indicele *primului* element având valoarea minimă. Dacă ni s-ar cere *ultimul* astfel de element, atunci ar fi suficient să schimbăm operatorul relațional ">" în ">=". Dar se mai poate schimba ceva pentru a scrie un subprogram care să fie optimizat la "sânge". Vom rezolva problema **P16**, modificând implementarea în acest sens.

```
procedure P16_m(n:Byte; t:tabel; var min:Byte);
var i:Byte;
begin
  min := 1;       {presupunem că min este primul}
  for i := 2 to n do
    if t[min] >= t[i] then
      min := i;
end;
```

Dacă, după apelarea subprogramului vrem să știm și numele elevului (având valoare minimă) vom cere afișarea elementului `t[min]`.

Să analizăm problema **P17** într-o variantă modificată, în care ni s-ar cere numele tuturor copiilor care trăiesc în familii având același venit minim. Rezolvarea acestei probleme se realizează în două etape. Mai întâi stabilim minimul în șirul veniturilor, apoi parcurgem din nou întreg șirul și afișăm elementele (numele copiilor) în cazul cărora venitul familiei este egal cu minimul găsit în prima etapă. În acest program trebuie să declarăm tipurile utilizator:

```
type elev = record
              nume : string;
              venit : Real
            end;
     elevi = array[1..30] of elev;
procedure P17_1(n:Byte; e:elevi; var j:Byte;
                                 var rez:elevi);
var i:Byte;
    min:Real;
begin
  min := e[1].venit;
  for i := 2 to n do
    if min > e[i].venit then
      min := e[i].venit;
  j := 0;
  for i := 1 to n do
    if min = e[i].venit then begin
      Inc(j);
      rez[j] := e[i];     {atribuire la nivel}
    end                           {de articol}
end;
```

Dacă nu ne place faptul că trebuie să parcurgem de două ori șirul, se poate scrie un subprogram care rezolvă aceeași problemă cu o singură parcurgere a lui:

```
procedure P17_2(n:Byte; e:elevi; var j:Byte;
                                 var rez:elevi);
var i:Byte;
    min:Real;
begin
  min := e[1].venit;
  j := 1;
  rez[j] := e[1];
  for i := 2 to n do
    if min > e[i].venit then begin
      j := 1;
      rez[j] := e[i];
      min := e[i].venit
    end else
      if min = e[i].venit then begin
        Inc(j);
        rez[j] := e[i];
      end
end;
```

Dar astfel am ajuns la o problemă de *selectare a unor elemente dintr-un șir dat pe baza unei proprietăți*. Aici proprietatea a fost simplă: elementele selectate au avut un câmp egal cu minimul din șirul veniturilor. Pe baza observației avem o nouă teoremă de programare.

Selectarea elementelor

Acest model de algoritm pe de o parte seamănă cu *căutarea liniară*, pe de altă parte cu *numărarea*. Aici trebuie să furnizăm pozițiile a mai multor elemente, având o proprietate dată, deci nu este suficient să le numărăm.

```
subalgoritm Selectare_1(n,X,buc,Y):
  buc ← 0
  pentru i ← 1, n execută:
    dacă P(X[i]) atunci
      buc ← buc + 1
      Y[buc] ← i
    sfârșit dacă
  sfârșit pentru
sfârșit subalgoritm
```

Se observă că pentru păstrarea rezultatului vom avea nevoie de un tablou nou în cazul căruia nu cunoaștem exact câte elemente va cuprinde. Dar, din moment ce nu este exclus ca fiecare element al șirului dat să aibă proprietatea cerută, acesta va fi declarat având atâtea elemente câte are șirul dat. Modelul de mai sus "adună" indicii elementelor având proprietatea `P` în vectorul `Y`, și totodată obține și numărul acestora în `buc`. Dacă nu avem nevoie de elementele respective colecționte într-un vector nou, deoarece trebuie doar să le furnizăm ca rezultat (de exemplu, afișându-le) algoritmul este și mai simplu:

```
subalgoritm Selectare_2(n,X):
  pentru i ← 1, n execută:
    dacă P(X[i]) atunci
      scrie X[i] {sau i,în funcție de cerințe}
    sfârșit dacă
  sfârșit pentru
sfârșit subalgoritm
```

Vom da și a treia variantă a acestui algoritm. În aceasta, considerăm că după selectarea elementelor căutate nu mai avem nevoie de șirul dat. Rezultă că, în loc să folosim un al doilea șir pentru elementele selectate, putem folosi chiar șirul dat. Pseudocodul acestui algoritm este următorul:

```
subalgoritm Selectare_3(n,X,buc,Y):
  buc ← 0
  pentru i ← 1, n execută:
    dacă P(X[i]) atunci
      buc ← buc + 1
      X[buc] ← X[i]
                {sau i, în funcție de cerințe}
    sfârșit dacă
  sfârșit pentru
sfârșit subalgoritm
```

Oare cum rezolvăm problema, dacă se cere păstrarea, nu a elementelor având proprietatea `P`, ci a acelora care *nu* au această proprietate? Vom nega condiția din **if**, veți răspunde! Așa este. Dar dacă vrem să păstrăm aceste elemente "nemișcate" pe pozițiile lor? În acest caz, putem *anula prezența* acestor elemente, suprascriindu-le cu o valoare specială, aleasă în acest scop. Astfel, vom avea un algoritm care realizează o selectare *pe loc*. Această soluție va fi avantajoasă doar dacă "evitarea" prelucrării acestor elemente, având valoarea aleasă va fi mai simplu de realizat decât verificarea proprietății `P`.

```
subalgoritm Selectare_4(n,X):
  pentru i ← 1, n execută:
    dacă P(X[i]) atunci
      X[i] ← valoare
    sfârșit dacă
  sfârșit pentru
sfârșit subalgoritm
```

Să considerăm câteva enunțuri de probleme:

P18. Să se realizeze un tabel cu fetele înscrise la olimpiada de informatică!

P19. Dat fiind un număr natural, să se genereze șirul tuturor divizorilor săi!

P20. Fie șirul mediilor semestriale ale elevilor unei clase în care s-a comis o eroare, în sensul că s-au calculat și mediile copiilor care au rămas corigenți. Ar trebui modificat șirul, astfel încât media elevilor corigenți să fie înlocuită cu 0!

Rezolvarea problemei **P18** se poate realiza aplicând algoritmul `Selectare_1(n,X,buc,Y)`, presupunând că tipul `sir` desemnează un tablou unidimensional de elemente de tip **record**, unde printre câmpuri există și câmpul `sex`, care în cazul fetelor are valoarea `'F'`.

```
procedure Tabel(n:Byte; elevi:sir;
                        var nr:Byte; var fete:sir);
var i:Byte;
begin
  nr := 0;
  for i := 1 to n do
    if elevi[i].sex = 'F' then begin
      Inc(nr);
      fete[nr] := elevi[i]
    end
end;
```

Rezolvarea problemei P20 aparent este mai complicată, dar numai din cauza că proprietatea "corigent" nu este memorată explicit printre câmpurile articolelor, ci trebuie s-o stabilim pe baza mediilor pe discipline. Presupunem că programul conține următoarele declarații:

```
const nrdiscipline = 15;
type unelev = record
                nume:string[30];
                medii:array[1..15] of 3..10;
                media:Real     {media generală}
              end;
type catalog = array[1..30] of unelev;
procedure Sterge(n:Byte; var c:catalog);
var i,j:Byte;
begin
  for i := 1 to n do begin
    j:=1;
    while (j <= nrdiscipline) and
                (c[i].medii[j] >= 5) do Inc(j);
    if j <= nrdiscipline then
      c[i].media := 0
  end
end;
```