TEOREME de programare

Clara Ionescu

În prima fază a analizei unei probleme de algoritmică, de regulă, căutăm să stabilim clasa de probleme în care ar putea fi încadrată aceasta. Pentru un începător această etapă este dificilă, deoarece nu există încă o "arhivă" mentală în care poate avea loc căutarea. Vom expune, pe parcursul a câtorva articole teoremele de programare, încercând astfel să oferim un mod posibil de clasificare a problemelor, precum și a rezolvării lor eficiente.

Prelucrarea unui șir de date

O clasă importantă de probleme cu care ne întâlnim frecvent, cere ca pe baza unui șir de date de intrare să determinăm un singur rezultat. Vom începe tratarea primului subiect cu prezentarea unor probleme!

P1. Cunoscând mediile semestriale ale celor *n* elevi dintr-o clasă, să se determine media semestrială a clasei!
P2. Să se construiască o variabilă de tip *string* formată din *m* caractere date!
P3. Cercetând lumea păsărilor din Delta Dunării *k* specialiști au notat, fiecare, tipurile de păsări observate. Să se stabilească speciile de păsări din Delta Dunării!
P4. Să se calculeze produsul primelor *n* numere naturale (factorialul)!

Să analizăm problemele de mai sus și să încercăm să stabilim ce anume au în comun! În fiecare avem mai multe date de intrare de același tip, deci ele formează un șir (vector) și fiecare cere obținerea unui singur rezultat. Acest rezultat se poate determina prin aplicarea unei funcții care prelucrează întreg șirul de date (suma a *n* numere, concatenarea a *m* caractere, reuniunea a *k* mulțimi, produsul a *n* numere). Dar această funcție poate fi descompusă într-un șir de funcții, astfel încât argumentele lor să fie două entități (suma a două numere, concatenarea a două caractere, reuniunea a două mulțimi, produsul a două numere).

În proiectarea oricărui algoritm există o fază preliminară în care se stabilesc variabilele și semnificațiile lor:

n: *întreg* { *numărul elementelor șirului de prelucrat* }
x: *tablou(1..n): tip element* { *elementele șirului dat* }
*F*0: *tip element* { *element nul față de operație* }
S: *tip element* { *rezultatul* }

Aici, prin operație se înțelege oricare din operațiile realizabile cu instrumente matematice: adunare, produs, reuniune, intersecție, operații logice, concatenare etc. Oricare dintre aceste operații poate fi redusă la o operație binară și se poate stabili pentru ea un element neutru, astfel încât operația efectuată asupra unui element *y*, ales arbitrar și acesta are ca rezultat *y*.

Exemple

$F = \Sigma, \prod, \cup, \cap, \wedge, \vee$, & (concatenare)
$F0$ = 0, 1, {}, mulțimea valorilor posibile, Adevărat, Fals, '' (șirul vid).

Rezolvarea se construiește prin inducție matematică folosindu-se operația binară și elementul nul.

- dacă prelucrăm 0 elemente, rezultatul este $F0$,
- dacă se cunoaște rezultatul prelucrării primelor i - 1 elemente (F^{i-1}), atunci rezultatul prelucrării primelor i elemente (F^i) se calculează astfel: $F^i = f(F^{i-1}, x_i)$.

Funcția *F*, exprimată recursiv este:

$$F_i := F(X_1, \cdots, X_i) := \begin{cases} F_0 & \text{, dacă } i = 0 \\ f(F^{i-1}, x_i), & \text{altfel} \end{cases}$$

Algoritmul în pseudocod este:

```
subalgoritm Prelucrare șir(n,x,S):
  S ← F0
  pentru i ← 1, n execută
    S ← f(S,x[i])
  sfârșit pentru
sfârșit subalgoritm
```

Vom aplica această rezolvare generală pentru problemele enunțate la începutul articolului, dând implementările în limbajul *Pascal*.

P1. Prima problemă cere determinarea unei medii aritmetice. Nu vom număra termenii sumei *S*, deoarece toate elementele șirului participă la sumă, în concluzie se cunoaște numărul cu care aceasta se împarte pentru a calcula media.

```
procedure P1(n:Byte;x:note;var medie:Real);
{ există definit tipul note = array[1..MaxN] of Real}
var i:Byte; S:Real;
begin
  s:=0;      { element neutru pentru adunare }
  for i:=1 to n do
    S:=S+x[i];      { operatia este adunarea }
  medie:=s/n
end;
```

P2. Cea de-a doua problemă cere concatenarea unor caractere într-o variabilă de tip *string*. Rezultă că operația va fi `'+'`, iar elementul neutru va fi șirul vid: `''`.

```
procedure P2(m:Byte;x:car;var S:string);
{ există definit tipul car = array[1..MaxM] of Char}
var i:Byte;
begin
  s:='';{ element neutru pentru concatenare }
  for i:=1 to m do
    S:=S+x[i];                { concatenare }
end;
```

P3. În această problemă se cere determinarea reuniunii a *m* mulțimi. În *Pascal* există posibilitatea declarării unui tip utilizator *mulțime*, respectând restricția ca tipul de bază să fie ordinal. Pentru a nu încărca prezentarea cu elemente de programare inutile, vom considera că tipurile de păsări sunt notate cu numere naturale mai mici decât 255. Operatorul cu care implementăm reuniunea este `'+'`, iar elementul neutru este mulțimea vidă: `[]`.

```
procedure P3(k:Byte;x:mult;var S:o_multime);
  { există definite tipurile o_multime = set of Byte;}
    { și mult = array[1..Maxk] of set of Byte;}
var i:Byte;
begin
  s:=[];   { element neutru pentru reuniune }
  for i:=1 to k do
    S:=S+x[i];                  { reuniune }
end;
```

P4. Această problemă pare să fie cea mai simplă. Rezolvarea este similară cu prima, dar se finalizează după determinarea produsului.

```
procedure P4(n:Byte;var f:Integer);
var i:Byte;
begin
  f:=1;  { element neutru pentru inmultire }
  for i:=1 to m do
    f:=f*i;                   { inmultire }
end;
```

Ne ocupăm de această problemă, deoarece în multe manuale se pot vedea diferite enunțuri în care este nevoie de calcularea factorialului, dar rezolvările nu tratează detaliile de implementare "interesante" care se leagă de această problemă. În cele ce urmează vom intra în... laborator.

Evitarea depășirii

Să ne imaginăm un laborator într-o școală unde profesorul cere elevilor să scrie programul care calculează produsul primelor *n* numere naturale. Elevii scriu programul și fiind curioși, îl verifică. Introduc pe rând valori pentru *n*: 5, 8, ..., 25, ... poate și 50. În funcție de tipul variabilei declarate pentru rezultat se pot întâmpla lucruri intersante. De exemplu, dacă rezultatul este declarat ca fiind de tipul `Byte`, elevul respectiv obține că `6!=208`, iar `8!=128`. Dar orice elev știe că `8!` trebuie să fie mai mare decât `6!`. Încep căutările disperate în programul sursă. Unde am greșit? se întreabă elevul nostru. Să presupunem în continuare că își dă seama că trebuie să schimbe tipul rezultatului. Va efectua modificarea tipului rezultatului din `Byte` în `Word` sau direct în `Longint`. În acest caz, elevul nostru devine curajos, rulează programul pentru $n = 20$, apoi pentru $n = 50$. Și din nou surpriză: în cazul declarării rezultatului de tip `Longint`, `20!=-2102132736` (număr negativ!), iar `50!=0`. Până la urmă și cel mai răbdător începător se enervează și își declară variabila respectivă de tipul `Real`. În schimb, acum programul lui se întrerupe pentru $n > 33$ cu mesajul de eroare: `Floating point overflow`! Ce-i de făcut?

Să luăm în considerare situația în care se preferă ca rezultatul să fie de tip `Longint`. Evident, încercările ne-au convins, că începând cu o anumită valoare a lui *n* se produce depășire. Mai ales atunci când trebuie calculat factorialul unui *n*, care constituie un rezultat calculat în program, ne trebuie "un instrument" care să ne protejeze de neplăceri, cum ar fi obținerea unui rezultat fals. Cel mai mare număr de tipul `Longint` posibil de reprezentat în calculator este `2147483647`, reținut de constanta simbolică a *unit*-ului *System* numită `MaxLongint`. Ar urma ca atare să putem pune o întrebare de genul `f > MaxLongint`... Dar este absurd să întrebăm dacă un număr este mai mare decât cel mai mare număr posibil...

În consecință, vom deveni preventivi și vom pune întrebarea cu un pas înainte de efectuarea înmulțirii curente care oricând ar putea produce o depășire în felul următor:

```
procedure P4_bis(n:Byte;var f:Longint);
var i:Byte;
begin
  f:=1; i:=1;
  while (f<>0) and (i<=n) do begin
    if f>MaxLongint div i then f:=0
    else begin f:=f*i; Inc(i) end
  end
end;
```

Valoarea lui `f` se va testa în blocul apelant pentru a decide dacă s-a putut calcula factorialul cerut, sau determinarea lui s-a întrerupt deoarece s-ar fi produs o depășire.