Cluj-Napoca, 22 iulie 2002

ADMITERE UBB 2002

Militon Frențiu

În urma corectării lucrărilor scrise la proba de Informatică, dar și a examenelor de la disciplina "Algoritmică și programare" cu studenții anului I, am observat că absolvenții claselor de informatică nu sunt deprinși să folosească subprograme. De aceea, în urmă cu doi ani am publicat [1] un articol despre importanța conceptului de subalgoritm în programare. În acest articol veți putea citi despre examenul de admitere din acest an.

În enunțul problemei date în acest an la admitere, am menționat expres subprogramele care trebuie să apară în programul cerut. Cu toate acestea, mai mult de jumătate din candidații care au ales să dea examen la disciplina "Informatică" nu au avut în programul lor toate subprogramele menționate, iar dintre cei care le-au avut, o bună parte nu au respectat cerințele din enunțul dat. Cea mai frecventă greșeală, pentru cei care au scris subprogramele cerute, constă în lipsa parametrilor formali, deci folosirea variabilelor globale. Acest fapt dovedește o înțelegere greșită a conceptului de subprogram și a importanței subprogramelor în programare.

Pentru a putea fi refolosite în orice context subprogramele nu trebuie să folosească variabile globale. În consecință, variabilele care marchează datele presupuse cunoscute în subproblema corespunzătoare și cele care marchează rezultatele obținute trebuie să apară în lista parametrilor formali.

O altă greșeală, prezentă și la candidații care au luat note mari (care au gândit bine, dar n-au redat în scris nimic despre modul în care au gândit), este lipsa oricăror explicații referitoare la program. Din cei 183 de candidați doar trei au dat explicațiile necesare. O parte au dat unele explicații prin comentarii prezente în textul programului. Dar nici importanța comentariilor nu este cunoscută absolvenților de liceu. Doar 28 de lucrări au avut cel puțin un rând de comentarii, dar numai 7 dintre acestea au fost acceptabile, nici una perfectă.

Menționăm, fără a le analiza, alte neajunsuri, destul de frecvente: scriere necaligrafică, dezordonată, cu multe ștersături, cu multe greșeli, algoritmi incorecți, programe neclare sau cu artificii în dauna clarității, algoritmi complicați în locul unora destul de simpli, variabile inutile, construirea unor tablouri care nu sunt necesare în program. Convinși de faptul că acești candidați sunt elevii cei mai bine pregătiți la informatică, putem trage o concluzie despre nivelul de pregătire al absolvenților de liceu.

Problemele date în acest an au fost următoarele:

1. Descrieți în *pseudocod* algoritmi pentru rezolvarea următoarelor probleme:
   a. Determinați dacă o matrice pătratică care are elemente distincte poate fi transformată prin schimbări de linii într-una în care elementul minim al fiecărei coloane se află pe diagonala principală. Dacă răspunsul este da, rearanjați matricea.
   b. Ordonați crescător elementele de pe diagonala principală a unei matrice pătratice, prin schimbări de linii și coloane.
   c. Se dă un tablou unidimensional X cu componente numere reale. Eliminați din X toate componentele negative.

2. Scrieți un program *Pascal* care citește numerele naturale $n_1, n_2, \ldots, n_k$ și tipărește cea mai lungă secvență $n_s, n_{s+1}, \ldots, n_d$, cu $1 \le s \le d \le k$, care conține numai numere prime. Programul trebuie să conțină:
   - o procedură care citește numerele date;
   - o funcție care verifică dacă un număr natural este prim;
   - o procedură care determină indicii s și d, $1 \le s \le d \le k$, cu proprietatea că $n_s, n_{s+1}, \ldots, n_d$ sunt numere prime;
   - o procedură care tipărește numerele $n_s, n_{s+1}, \ldots, n_d$.

Diferența esențială între un algoritm și subalgoritmul corespunzător aceleiași probleme constă în faptul că algoritmul citește datele de intrare și tipărește rezultatele obținute, în timp ce subalgoritmul primește prin parametri datele de intrare și transmite, tot prin parametri, rezultatele obținute. Calculele efectuate sunt aceleași în ambele situații.

Deci, pentru aceeași problemă *P*, în care datele de intrare sunt depuse în *X*, iar rezultatele se depun în *R*, următorul algoritm rezolvă problema *P*:

```
Algoritmul AP este:
   Citește X;
   Cheamă SP(X, R);
   Tipărește R
sfârșit algoritm
```

unde *SP* este subalgoritmul corespunzător problemei *P*. El primește din locul apelului datele de intrare *X* și retransmite rezultatele *R*. În programe complicate, datele *X* pot fi rezultatul unor calcule anterioare (deci nu pot fi citite), iar rezultatele obținute și transmise la locul apelului, pot fi doar rezultate intermediare utile în calculele ulterioare; nefiind rezultate finale, ele nu vor fi *tipărite*.

Întrucât orice problemă poate apărea ca o subproblemă a unei probleme mai complicate, se recomandă scrierea unui subalgoritm pentru rezolvarea ei. De aceea, dăm în continuare subalgoritmii ceruți la punctele *a*, *b*, respectiv *c*. Pentru candidați au fost acceptați fie algoritmii fie subalgoritmii corespunzători.

a.

```
Funcția Minim(A,n,j) este:
{Găsește linia ce conține valoarea minimă de pe a j-a coloană a matricei A}
r:=1;
Pentru i:=2; n execută
   Dacă A[i,j]<A[r,j]
      atunci r:=i
   sfârșit dacă
sfârșit pentru
Minim:=r
sfârșit Minim;

Subalgoritmul InterSchimbLin(A,n,i,k)
                                          este:
{Interschimbă liniile i și k în matricea A}
Pentru j:=1;n execută
   temp:=A[i,j];
   A[i,j]:=A[k,j];
   A[k,j]:=temp
sfârșit pentru
sfârșit InterSchimbLin

Subalgoritmul Rearanjez(A,n,b) este:
{Efectuează rearanjarea cerută. Dacă aceasta este posibilă b=1, altfel b=0}
b:=1; {Ipoteza "se poate face rearanjarea"}
i:=1;
Cât timp i≤n și b=1 execută
   k:=Minim(A,n,i);
   Dacă i>k
      atunci b:=0
     {Rearanjarea nu se poate efectua!}
     altfel Dacă i<k atunci
               Cheamă InterSchimbLin(A,i,k);
            sfârșit dacă
            i:=i+1;
   sfârșit dacă
sfârșit cât timp
sfârșit Rearanjez
```

b. Pentru a descrie subalgoritmul de ordonare cerut avem nevoie de doi subalgoritmi auxiliari pentru interschimbarea a două linii, respectiv coloane.

Subalgoritmul `InterSchimbLin(A,n,i,k)` este dat deja la punctul a.

```
Subalgoritmul InterSchimbCol(A,n,j,k)
                                          este:
{Interschimbă coloanele j și k în matricea A}
Pentru i:=1; n execută
   temp:=A[i,j];
   A[i,j]:=A[i,k];
   A[i,k]:=temp
sfârșit pentru
sfârșit InterSchimbCol

Subalgoritmul Ordonare (A,n) este:
{Ordonează crescător elementele de pe diagonala matricei A prin interschimbări de linii și coloane}
Repetă
   Ordonat:=1;
   {Ipoteza "este ordinea dorită"}
   Pentru i:=1; n-1 execută
      Dacă A[i,i]>A[i+1,i+1] atunci
      {Elementele nu sunt în ordine, deci interschimbare}
         Cheamă InterSchimbLin(A,n,i,i+1);
         Cheamă InterSchimbCol(A,n,i,i+1);
         Ordonat:=0
      sfârșit dacă
pânacând Ordonat=1 sfârșit repetă
sfârșit Ordonare
```

c. Problema cere eliminarea componentelor negative dintr-un tablou `X`, nu copierea celorlalte componente într-un alt tablou. Deci, rezultatul obținut va fi tot `X`, specificarea subalgoritmului fiind:

```
DATE n,X; {n≥1, X fiind un tablou cu n componente
          numere întregi}
REZULTATE n,X; {X conține doar valorile nenegative din
                cele date inițial, iar n este numărul lor}
```

Mai subliniem că parcurgerea tabloului `X` nu poate fi făcută printr-un ciclu **Pentru** deoarece, eliminând o componentă x_i, să zicem x_2, pe această poziție vine următoarea valoare, x_3, care poate fi tot negativă și trebuie testată (ea numindu-se tot x_2), deci `i` nu poate fi încă modificat, în acest caz trebuind să rămână `2`. Aici nu se specifică ni-

mic despre păstrarea ordinii inițiale a elementelor care rămân în X. Dacă e permisă schimbarea acestei ordini (varianta aleasă în soluția dată mai jos), eliminarea va fi mai simplă, în locul unei valori negative putând aduce oricare altă valoare neprocesată. Subalgoritmul corespunzător este prezentat în continuare:

```
Subalgoritmul ELIMIN(n,X) este:
{Elimină din tabloul X cu n componente, valorile negative}
i:=1;
Cât timp i≤n execută
   Dacă x_i<0
      atunci x_i:=x_n; n:=n-1
            {x_i este șters, fiind înlocuit cu x_n}
      altfel i:=i+1
   sfârșit dacă
sfârșit cât timp
sfârșit ELIMIN
```

Dacă nu este permisă schimbarea ordinii elementelor (ceea ce ar fi trebuit menționat expres în textul problemei), atunci în locul atribuirii $x_i:=x_n$ erau necesare instrucțiuni care deplasau toate valorile din dreapta cu o poziție spre stânga, ceea ce se poate face prin instrucțiunile

```
Pentru p:=i+1; n execută
   x_{p-1}:=x_p
sfârșit pentru
```

Pentru rezolvarea problemei 2 propunem o abordare *bottom-up*, rezolvând separat subproblemele problemei date. Aceste subprobleme sunt: *citirea datelor*, *afișarea rezultatelor*, *verificarea primalității unui număr*, *stabilirea unei secvențe de numere prime începând cu un indice dat* și *determinarea celei mai lungi secvențe de numere prime dintre secvențele găsite*.

Având în vedere că citirea și afișarea necesită subalgoritmi simpli, iar stabilirea primalității se află în acest număr al revistei la pagina 41, în cele ce urmează vom prezenta succint subalgoritmii pentru stabilirea unei secvențe de numere prime începând cu un indice dat și determinarea celei mai lungi secvențe de numere prime dintre secvențele găsite.

Să observăm că dacă am găsit o secvență de numere vecine prime după care urmează un număr neprim, următoarea secvență va începe cu cel puțin o poziție spre dreapta. De aceea este necesar să parcurgem o singură dată vectorul N dat, și pentru aceasta avem nevoie de un indice i. În cazul în care N_i este prim, dorim să găsim cea mai lungă secvență de numere prime care începe pe poziția i, secvență precizată prin indicele j pe care se află ultimul număr prim. În continuare prezentăm acest subalgoritm:

```
Subalgoritmul SECVENTA(i,k,N,j) este:
{Pornim cu valoarea i care este indicele numărului prim N_i. Obținem cea mai mare valoare j pentru care secvența N_i, N_{i+1}, ..., N_j conține numai numere prime}
j:=i;
Cât timp (j≤k) și (N_j este prim) execută
   j:=j+1
sfârșit cât timp
j:=j-1;
sfârșit SECVENTA
```

Subalgoritmul LONGSECV trebuie să furnizeze indicii de la capetele celei mai lungi subsecvențe care conține doar numere prime. Vom avansa în șirul dat până când descoperim primul număr prim, moment în care apelăm subalgoritmul SECVENTA. La început, neavând nici o secvență de numere prime, valoarea inițială a variabilei d este 0, iar valoarea provizorie a variabilei s este 1. După revenirea din subalgoritmul SECVENTA, avem la dispoziție în j marginea din dreapta a secvenței de numere prime care începe pe poziția i. În concluzie, la pasul următor, dacă secvența curentă este mai lungă decât una găsită anterior, atribuim valorile i, j variabilor s, respectiv d.

```
Subalgoritmul LONGSECV(k,N,s,d) este:
{Se obțin indicii s și d pentru care secvența N[s..d] conține numai numere prime și este cea mai lungă}
i:=1
s:=1
d:=0 {codifică "Șirul nu conține numere prime"}
cât timp i<=k execută
   cât timp (i<=k) și (N[i] nu este prim) execută
      i:=i+1
   sfârșit cât timp
   Dacă (i<=k) și (N[i] este prim) atunci
      cheamă Secventa(i,k,N,j)
      dacă (j-i > d-s) atunci
      {s-a găsit o secvență mai lungă}
         s:=i
         d:=j
      sfârșit dacă
      i:=j+1
   sfârșit dacă
sfârșit cât timp
sfârșit LONGSECV
```

Celelalte explicații sunt date prin comentariile inserate în implementarea programului. Orice program trebuie să conțină un minim de comentarii explicative și trebuie să fie scris cu indentare pentru a mări claritatea textului. Textul sursă și versiunea integrală a acestui articol pot fi găsite la adresa:

http://www.cs.ubbcluj.ro/~mfrentiu/articole/adm2002.html

Bibliografie

1. M. Frențiu, *Conceptul de subalgoritm și importanța lui în programare*, GInfo 10/6.

Domnul prof. Militon Frențiu este cadru didactic la Universitatea "Babeș-Bolyai" și poate fi contactat prin e-mail la adresa mfrentiu@cs.ubbcluj.ro.