Conceptul de ALGORITM - abordare MODERNĂ

Marin Vlada

În cadrul acestui ultim articol, dedicat unei abordări moderne a conceptului de algoritm, vom prezenta șase probleme simple care ilustrează modul de elaborare al algoritmilor de rezolvare pentru acestea.

Problema 1: Sumă de valori

Se consideră vectorul $X = (x_1, x_2, \ldots, x_n)$ ale cărui componente $x_1, x_2, \ldots, x_n$ sunt numere reale. Să se calculeze suma S a componentelor vectorului X, adică:

$$S = \sum_{i=1}^{n} x_i.$$

Date de intrare

Datele de intrare (***Di***) pentru această problemă sunt:
- n - numărul de componente ale vectorului X;
- $X = (x_1, x_2, \ldots, x_n)$ - vectorul cu componente reale.

Date de ieșire

Datele de ieșire (***De***) pentru această problemă sunt constituite de suma:

$$S = \sum_{i=1}^{n} x_i.$$

Rezolvare

Rezolvarea problemei constă în câțiva pași care sunt prezentați în continuare.

a) *Raționamentul de rezolvare* (***R***)
Pentru a rezolva această problemă vom defini următorul șir de valori:
- $S_0 = 0$;
- $S_i = S_{i-1} + x_i$, $1 \le i \le n$.

Prin inducție matematică se poate demonstra foarte ușor că $S_n = \sum_{i=1}^{n} x_i$. Prin urmare, este necesar să se elaboreze procesul de calcul ***P*** care să calculeze succesiv valorile S_0, S_1, …, S_n, valori care generează suma $S = S_n$.

Datorită faptului că valorile intermediare S_{i-1}, $1 \le i \le n$ nu vor mai fi folosite în alte scopuri în afară de calcularea valorilor S_i, aceste variabile se pot înlocui cu una singură și pentru că suma finală S este egală cu S_n, atunci vom înlocui toate variabilele S_i, $0 \le i \le n$, cu S și vom avea următorul raționament de rezolvare:
- $S = 0$, $i = 0$;
- $S = S + x_i$, $1 \le i \le n$.

b) *Mulțimea variabilelor* (***V***)
Variabilele definite de raționamentul ***R***, care vor fi utilizate în procesul de calcul ***P***, sunt:
- n - numărul de componente ale vectorului X;
- $X = (x_1, x_2, \ldots, x_n)$ - vectorul cu componente reale;
- S - variabila care stochează sumele parțiale și suma finală.

c) *Procesul de calcul* (***P***) *și reprezentarea algoritmului*
După *citirea* (`read`) unei valori pentru variabila n și *citirea* celor n valori care reprezintă componentele vectorului X, toate aceste valori vor fi stocate în memoria internă (prin intermediul mediului de intrare ***Mi***), în locații de memorie la care se poate avea acces prin intermediul instrucțiunilor (executate de o mașină virtuală).

Conform relației de recurență prezentată anterior, inițial trebuie ca în locația de memorie asociată lui S să fie stocată valoarea 0. Evident, pentru calculul valorilor intermediare se va utiliza intrucțiunea repetitivă cu contor `for`.

După încheierea execuției acestei instrucțiuni, în locația de memorie asociată variabilei S se va afla valoarea cerută de problemă (conform raționamentului de rezolvare), valoare care va fi citită din memorie prin intermediul mediului de ieșire ***Me*** și afișată pe ecran.

Prin urmare, vom avea următorul proces de calcul (și în același timp o reprezentare completă a algoritmului):

```
Algorithm Suma_valori;
   Integer n;
   Real s, x(500);
begin
   read n;
   read x1, x2, ..., xn;
   S ← 0;                          // starea inițială
   for i = 1, n do
       S ← S + xi;
   write S;
end
```

Problema 2: Configurația electronică

Dacă Z este numărul atomic al unui element chimic, să se elaboreze un algoritm pentru completarea straturilor energetice din învelișul electronic al atomului. Se presupune că straturile se completează în ordine cu numărul maxim de electroni $2 \cdot i^2$, unde i este numărul stratului, excepție ultimul strat care se completează cu restul de electroni. Să se determine numărul straturilor care se completează și pentru fiecare strat, numărul electronilor care se află pe acesta.

Date de intrare

Datele de intrare (***Di***) pentru această problemă sunt date de numărul atomic Z al unui element chimic.

Date de ieșire

Datele de ieșire (***De***) sunt constituite din:
- n - numărul de straturi completate;
- $S = (s_1, s_2, \ldots, s_n)$ - lista care va conține numărul de electroni pentru fiecare strat completat.

Rezolvare

a) *Raționamentul de rezolvare* (***R***)
Trebuie conceput un proces de calcul care să contorizeze numărul straturilor care s-au completat la un moment dat și să calculeze numărul total de electroni (sume parțiale) repartizați pe straturi până la un moment dat.

Conform enunțului, stratul i, $i > 0$, se completează cu $s_i = 2 \cdot i^2$ electroni. Vom nota cu $Sp = s_1 + s_2 + \ldots + s_i$ suma parțială (numărul electronilor repartizați) la momentul i.

La fiecare pas, procesul de calcul trebuie să realizeze:
- incrementarea contorului pentru stratul completat;
- completarea stratului i cu $s_i = 2 \cdot i^2$ electroni;
- calculul sumei parțiale $Sp = s_1 + s_2 + \ldots + s_i$;
- repetarea operațiilor precedente *cât timp* $Sp < Z$.

Pentru ca raționamentul să fie corect, în momentul în care nu mai este satisfacută condiția $Sp < Z$, se va realiza completarea ultimului strat cu numărul de electroni dați de diferența dintre numărul atomic Z și suma Sp, după ce din suma Sp se scade numărul de electroni adăugați ultima dată.

b) *Mulțimea variabilelor* (***V***)
Variabilele definite de raționamentul ***R***, care vor fi utilizate în procesul de calcul ***P***, sunt:
- Z - numărul atomic al elementului chimic;
- Sp - variabila care va conține suma parțială;
- n - numărul de strate completate;
- S - vectorul care va conține numărul electronilor pentru fiecare strat.

c) *Procesul de calcul* (***P***) *și reprezentarea algoritmului*
Conform raționamentului de rezolvare prezentat anterior, vom avea următorul proces de calcul (și în același timp o reprezentare completă a algoritmului):

```
Algorithm Configuratia_electronica;
   Integer Z,n,Sp,s(7);
begin
   read Z;  // nr. atomic pentru elementul chimic
   // inițializări
   n ← 0;  // contor pentru nr. stratului completat
   Sp ← 0;          // suma parțială a electronilor
   while (Sp < Z) do begin
       n ← n + 1;                  // incrementare
       sn ← 2 · n · n;            // completare strat
       Sp ← Sp + sn;               // suma parțială
   end;
   Sp ← Sp - sn;
   sn ← Z - Sp;    // completarea ultimului strat
   write "Nr. de straturi completate n =", n;
   write "Nr. electronilor pe straturi este:";
   write s1, s2, ..., sn;
end
```

Problema 3: Algoritmul lui Euclid

Dându-se două numere întregi a și b, se cere să se calculeze cel mai mare divizor comun al lor.

Date de intrare

Datele de intrare (***Dî***) pentru această problemă sunt constituite din cele două numere întregi a și b.

Date de ieșire

Datele de ieșire (***De***) sunt date de cel mai mare divizor comun al celor două numere.

Rezolvare

a) *Raționamentul de rezolvare* (***R***)
Pentru a rezolva această problemă trebuie ținut cont de teorema împărțirii cu rest.

Considerăm următorul șir de împărțiri succesive:

$a = b \cdot c_1 + r_1, b > r_1$
$b = r_1 \cdot c_2 + r_2, r_1 > r_2$
$r_1 = r_2 \cdot c_3 + r_3, r_2 > r_3$
$r_2 = r_3 \cdot c_4 + r_4, r_3 > r_4$
...
$r_{n-1} = r_n \cdot c_{n+1} + r_{n+1}, r_n > r_{n+1}$

Se poate observa că șirul resturilor $(r_i)_{i>0}$ este *mărginit*, *descrescător* și *convergent* către 0. Pentru cazul în care r_{n+1} este egal cu 0, *Euclid* a demonstrat că cel mai mare divizor comun al numerelor *a* și *b* este r_n, adică *ultimul rest nenul* din șirul de împărțiri succesive.

În concluzie, trebuie realizat un proces de calcul (și, implicit un algoritm) pentru a executa operațiile din șirul de împărțiri succesive. Procesul se va termina în momentul în care restul unei împărțiri din șirul de mai sus este 0.

Șirul împărțirilor de mai sus este corect doar în cazul în care *a* și *b* au același semn (amândouă numerele sunt pozitive sau negative în același timp). În caz contrar, trebuie să considerăm valoarea absolută a acestora, rezultatul final fiind același.

b) *Mulțimea variabilelor* (***V***)
Deoarece resturile împărțirilor intermediare (de la prima până la penultima) nu mai sunt folosite în alte scopuri, mulțimea variabilelor utilizate în procesul de calcul este formată doar din variabilele *a*, *b* și o variabilă auxiliară *r*, care, la fiecare pas, va conține restul împărțirii lui *a* la *b*.

c) *Procesul de calcul* (***P***) *și reprezentarea algoritmului*
Ținând seama că nu este necesară stocarea valorilor intermediare ale resturilor, rolul variabilelor *a* și *b* se va schimba la executarea fiecărei împărțiri din șirul împărțirilor succesive. Procesul de împărțire se repetă *până când r* devine 0. Se poate observa că valoarea câtului nu intervine în dinamica procesului de calcul.

În continuare prezentăm algoritmul de rezolvare a problemei:

```
Algorithm EUCLID_1;
  Integer a,b,r;
begin
  read a,b;
  repeat
    r ← a mod b;
    a ← b;     // împărțitorul devine deîmpărțit
    b ← r;           // restul devine împărțitor
  until r = 0;
  write "Cel mai mare divizor comun: ", a;
end
```

Folosind doar două variabile *a* și *b* (care, inițial vor stoca valorile datelor de intrare), se poate concepe un alt proces de calcul prin înlocuirea împărțirilor succesive cu scăderi repetate (operația de împărțire fiind, de fapt, un șir de scăderi repetate), obținându-se algoritmul lui *Nicomachus*.

```
Algorithm EUCLID_2;
  Integer a,b;
begin
  read a,b;
  repeat
    if a > b then
      a ← a - b;
    else
      b ← b - a;
  until a = b;
  write "c.m.m.d.c.=", a;
end
```

Problema 4: Algoritm de sortare

Dacă $x_1, x_2, \ldots, x_n$ sunt componentele reale ale vectorului $X = (x_1, x_2, \ldots, x_n)$, să se găsească o permutare σ a componentelor vectorului astfel încât acestea să fie sortate crescător, adică:
$X = (x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$ și $x_{\sigma(1)} \le x_{\sigma(2)} \le \ldots \le x_{\sigma(n)}$.

Date de intrare

Datele de intrare (***Di***) pentru această problemă sunt:
- n - numărul de componente ale vectorului X;
- $X = (x_1, x_2, \ldots, x_n)$ - vectorul cu componente reale.

Date de ieșire

Datele de ieșire (***De***) sunt date de vectorul:
$X = (x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$ și $x_{\sigma(1)} \le x_{\sigma(2)} \le \ldots \le x_{\sigma(n)}$.

Rezolvare

a) *Raționamentul de rezolvare* (***R***)
Există foarte mulți algoritmi de sortare, unii foarte performanți, alții mai puțin performanți [23]. Vom aplica un algoritm foarte utilizat, bazat pe metoda interschimbărilor, cunoscut sub numele de *metoda bulelor*. Metoda constă într-un proces de calcul principal în cadrul căruia, la fiecare pas, se parcurg componentele vectorului X (numărul de pași ai procesului fiind nedefinit, terminarea procesului fiind dată de obținerea sortării dorite). Prin parcurgerea componentelor vectorului X se înțelege, în acest context, un subproces de calcul care se execută în $(n - 1)$ pași și constă în analiza perechilor (x_i, x_{i+1}), $1 \le i \le n$, ceea ce înseamnă compararea valorilor celor două componente și efectuarea sau nu a interschimbării acestora în functie de tipul sortării (crescătoare sau descrescătoare).

Efectuarea interschimbării componentelor x_i, x_{i+1} se poate monitoriza prin utilizarea unei variabile intermediare (de lucru), notată, de exemplu, prin K și care poate să aibă valorile 0, dacă nu se realizează interschimbarea, sau 1, dacă se realizează interschimbarea. Această variabilă cu rol de indicator (semafor) se va utiliza pentru a controla terminarea procesului de calcul principal, și anume pentru a încheia parcurgerea componentelor vectorului X. Pentru acest scop, înainte de fiecare parcurgere, valoarea lui K se setează cu 0. În cazul în care în timpul parcurgerii componentelor vectorului (executării subprocesului de calcul pentru analiza celor n - 1 perechi pentru care se va utiliza instrucțiunea **for**) se va efectua cel puțin o interschimbare, indicatorul K se va seta cu valoarea 1. În cazul în care nu s-a efectuat nici o interschimbare, valoarea lui K rămâne neschimbată, adică 0, ceea ce este echivalent cu faptul că

valorile vectorului *X* sunt sortate. Prin urmare, la sfârșitul fiecărei parcurgeri se va verifica dacă valoarea indicatorului *K* este 0, caz în care procesul de calcul principal se va termina (se va utiliza instrucțiunea **repeat**). Evident, metoda utilizează structura vectorului *X* ca suport de stocare a componentelor sortate. Din acest motiv, pentru ca operația de interschimbare să se realizeze corect, se va folosi o variabilă intermediară (de lucru) notată *save*, variabilă care va stoca temporar valoarea x_i și care, ulterior, se va stoca în locul valorii x_{i+1}. Secvența corectă de instrucțiuni (*metoda celor trei scaune*) este următoarea (execuția secvențială a instrucțiunilor):

```
save ← x(i);
x(i) ← x(i+1);
x(i+1) ← save;
```

Operația de interschimbare prin simplele operații de atribuire `x(i) ← x(i+1)` și `x(i+1) ← x(i);` este incorectă, deoarece s-ar pierde valoarea `x(i)` și în final vectorul *X* ar conține alte valori și nu cele inițiale (citite).

În concluzie, procesul de calcul principal și subprocesul de parcurgere a componentelor vectorului *X*, se poate reprezenta schematic astfel:

Parcurgerea 1:
k ← 0
Analiza perechilor: (x_1, x_2), (x_2, x_3), …, (x_{n-1}, x_n).
Verificare condiție de oprire: k=0?

Parcurgerea 2:
k ← 0
Analiza perechilor: (x_1, x_2), (x_2, x_3), …, (x_{n-1}, x_n).
Verificare condiție de oprire: k=0?
…

Parcurgerea ?:
k ← 0
Analiza perechilor: (x_1, x_2), (x_2, x_3), …, (x_{n-1}, x_n).
Verificare condiție de oprire: k=0.

b) *Mulțimea variabilelor* (***V***)
Variabilele definite de raționamentul ***R***, care vor fi utilizate în procesul de calcul ***P***, sunt:
- *X* - vector cu componente reale;
- *n* - numărul de componente ale vectorului *X*;
- *K* - indicator pentru interschimbare;
- *save* - variabila folosită la interschimbări.

c) *Procesul de calcul* (***P***) *și reprezentarea algoritmului*
Conform raționamentului de rezolvare prezentat anterior, vom avea următorul proces de calcul (și în același timp o reprezentare completă a algoritmului):

```
Algorithm sortare_crescatoare;
  Integer n,k;
  Real save,X(500);
begin
  read n;
  read x1, x2, …, xn;
  // procesul de calcul principal
  repeat
    k ← 0;                          // inițializare
   // parcurgerea vectorului - subprocesul de calcul
    for i = 1,n-1 do begin
      if x_i > x_i+1 then begin// interschimbare
        k ← 1;                      // setare
        save ← xi;
        x_i ← x_i+1;
        x_i+1 ← save;
      end
    end
  until k=0;
  write "Vectorul sortat: ", x_1, x_2, …, x_n;
end
```

Problema 5: Puterile lui 2

Fie *k* un număr natural. Să se determine numărul cifrelor și cifrele numărului 2^k.

Date de intrare

Datele de intrare (***Di***) pentru această problemă sunt constituite din numărul *k*.

Date de ieșire

Datele de ieșire (***De***) pentru această problemă sunt:
- *n* - numărul cifrelor lui 2^k;
- $X = (x_1, x_2, \ldots, x_n)$ - vectorul cifrelor lui 2^k.

Rezolvare

a) *Raționamentul de rezolvare* (***R***)
Evident, problema ar fi fără sens dacă s-ar rezolva printr-o singură instrucțiune dintr-un limbaj de programare. Acest lucru se poate realiza doar dacă ar exista restricția $k < 31$. Ținând seama de reprezentarea tipului *integer* în memoria internă a calculatorului, majoritatea microprocesoarelor și limbajelor de programare pot stoca/reprezenta o valoare întreagă doar pe 4 bytes (32 biți).

Prin urmare, $2^{31} - 1 = 2147483647$ este cea mai mare valoare întreagă pe care o putem stoca. Este necesar să concepem un algoritm pentru calculul puterilor 2^k, $k > 30$. Vom pleca de la următorul tabel (generat cu ajutorul unui program simplu):

k	1	2	3	4	5	6	7	8	9	10	11	12
2^k	2	4	8	16	32	64	128	256	512	1024	2048	4096

Pentru a putea depăși obstacolul de mai sus, deci pentru a putea calcula valoarea 2^k pentru valori ale lui *k* mai mari decât 30, trebuie utilizat un vector cu elemente de tip întreg care să memoreze cifrele valorii care trebuie calculate (de fapt, acest lucru s-a cerut și în enunț în cadrul datelor de ieșire). Această decizie implică faptul că trebuie să se elaboreze un algoritm care să execute operațiile care rezultă din înmulțirea unui număr natural cu 2.

Prin urmare, dacă vom considera acest vector $X = (x_1, x_2, \ldots, x_n)$ - vectorul cifrelor lui 2^k, procesul de calcul prin-

cipal trebuie să aibă (k - 1) pași, și anume conform tabelului următor:

k	3	4	...	i	...	k
2^k	4	8	...	$x^i_n \dots x^i_2 x^i_1$	...	$x^k_n \dots x^k_2 x^k_1$

La fiecare pas $i \in \{2, 3, \dots, k - 1\}$ trebuie elaborat subalgoritmul (un subproces de calcul) care să realizeze operațiile pentru obținerea cifrelor valorii 2^{i+1} cu ajutorul cifrelor (memorate în componentele vectorului X) valorii $2^i = x^i_n \dots x^i_2 x^i_1$. Conform algoritmului elementar de înmulțire cu 2 a unui număr natural apare o cifră de transport. Vom nota cu c variabila care memorează la fiecare pas $j \in \{1, 2, \dots, n\}$ cifra de transport. Indicele i superior atașat fiecărui element din vectorul X semnifică faptul că în acesta se află valorile cifrelor calculate la pasul i. Cifrele valorii 2^{i+1}, respectiv valoarea de transport la fiecare pas, se vor calcula astfel:

```
x_j ← (x_j · 2 + c) mod 10;
c ← (x_j · 2 + c) div 10;      // cifra de transport
```

Numărul n de cifre ale valorii 2^k nu este cunoscut și se va modifica la fiecare pas. La început n = 1 și în continuare, la fiecare pas i, în cazul în care la sfârșitul subprocesului de calcul, care înmulțește pe 2^i cu 2, valoarea cifrei de transport este 1, atunci n se va incrementa cu 1 și valoarea lui x_n va fi 1. Deci, variabila n se va folosi pentru contorizarea cifrelor valorii 2^k.

Observație: Chiar dacă rationamentul prezentat este corect, în cazul elaborării unui algoritm și unui program pentru rezolvarea problemei, prin testarea și verificarea programului se vor obține rezultate greșite. Rezultatele incorecte provin de la simplul motiv că cifra de transport nu este calculată corect, și anume valoarea sa este calculată folosind cifra valorii 2^{i+1} și a valorii 2^i (avem în vedere execuția secvențială a instrucțiunilor și dinamica valorilor locațiilor din memoria internă). Este nevoie de utilizarea unei variabile auxiliare s pentru a reține la fiecare pas j al subprocesului considerat valoarea lui x_j și astfel vom avea:

```
s ← x_j;                       // variabila auxiliară
x_j ← (x_j · 2 + c) mod 10;
c ← (s · 2 + c) div 10;        // cifra de transport
```

b) *Mulțimea variabilelor* (***V***)
Variabilele definite de raționamentul ***R***, care vor fi utilizate în procesul de calcul ***P***, sunt:
- k - puterea pentru valoarea care trebuie calculată;
- $X = (x_1, x_2, \dots, x_n)$ - vectorul cifrelor lui 2^k;
- n - numărul de cifre ale lui 2^k;
- c - variabila care stochează valoarea cifrei de transport;
- s - variabila auxiliară folosită pentru calculul cifrelor lui 2^k.

c) *Procesul de calcul* (***P***) *și reprezentarea algoritmului*
În continuare prezentăm algoritmul de rezolvare (procesul de calcul) a problemei:

```
Algorithm Puterile_lui_2;
  Integer n,k,c,s,X(2000);
begin
  write "Program pentru calcularea valorii 2^k";
  write "Introduceți puterea k: ";
  read k;
  // inițializări
  n ← 1;
  x_n ← 2;
  for i = 2, k do begin
    // parcurgerea vectorului X
    c ← 0;                          // inițializare
    for j = 1, n do begin
      s ← x_j;                      // salvare
      x_j ← (x_j · 2 + c) mod 10;   // cifra puterii
      c ← (s · 2 + c) div 10;       // cifra de transport
    end
    if c = 1 then begin
      n ← n + 1;
      x_n ← 1;
    end
  end
  // scrierea cifrelor
  write "Numărul are",n,"cifre.";
  write "Numărul este:";
  for i = n, 1, -1 do
    write Xi;
end
```

Folosind un astfel de algoritm, se poate elabora un program corect pentru problema în studiu. De exemplu, dând valorile 1000 și 2000 lui k, rezultatele obținute au 302 cifre, respectiv 603 cifre.

În figura următoare se poate observa rezultatul furnizat de un program corect pentru valorile 1000 și 2000 date lui k, iar 2^{1000} și 2^{2000} au aceleași ultime 4 cifre, egale cu 9376.

```
Program pentru calcularea valorii 2^k
Introduceti puterea k: 1000

Numarul are 302 cifre.
Numarul este:
1071508607186267320948425049060001810561
4048117055336074437503883703511011249361
2249319837881569585812759467291755314682
5187145285692314043598457757469857480393
4567774824230985421074605062371141877954
1821530464749835819412673987675591655439
4607706291457119647768654216766042983165
2624386837205668069376
```

a) k = 1000

```
Program pentru calcularea valorii 2^k
Introduceti puterea k: 2000

Numarul are 603 cifre.
Numarul este:
1148130695274254524232833201177681984022
3177020886952004776427368257662613923703
1385665948631650626991844596463898746277
3447118960863055331425931356166653185391
2998914531228000068877914824004487142892
6990063486244781615463646388363947317026
0404663539709049965581623988089446296056
2331164953616422197033268134416890898445
8505602379484807914058900934776500429002
7167066258305220081322362812917612678833
1720659899539641812702177985840404215985
3183251540889433902091920554957783589672
0391600819572166305827553804255837260155
2834878641943205450891527578388262517543
5528800822842770817965453762184851149029
376
```

b) k = 2000

Figura 1

Problema 6: Suma puterilor unei matrice

Dacă $A = \|a_{ij}\|_{i,j=1,n}$ este o matrice pătratică de ordin n cu elemente reale, să se calculeze matricea $X = A + A^2 + \ldots + A^k$ pentru $k > 1$.

Date de intrare

Datele de intrare (*Dî*) pentru această problemă sunt:
- k - puterea;
- n - ordinul matricei A;
- $A = \|a_{ij}\|_{i,j=1,n}$ - matricea.

Date de ieșire

Datele de ieșire (*De*) pentru această problemă sunt constituite de matricea $X = \|x_{ij}\|_{i,j=1,n}$, $X = A + A^2 + \ldots + A^k$.

Rezolvare

a) *Raționamentul de rezolvare* (***R***)

Procesul de calcul principal trebuie să calculeze suma puterilor $A, A^2, ..., A^k$. Acest proces de calcul va avea (k - 1) pași. Inițial, procesul de calcul trebuie să aibă loc după atribuirea $X \leftarrow A$. La pasul $i \in \{2, 3, \ldots, k\}$ trebuie să se calculeze A^i (este necesară definirea unei proceduri pentru determinarea produsului a două matrice pătratice) și să se execute instrucțiunea $X \leftarrow X + A^i$.

Ținând seama că algoritmul este folosit ca "sursă" pentru codificarea într-un limbaj de programare, calculul lui A^i trebuie realizat de o procedură. Este necesară folosirea unei variabile care să salveze valoarea lui A^i și care va fi utilizată de procedura de determinare a valorii $A^{i+1} = A \cdot A^i$. Dacă această variabilă este S, inițial, înaintea procesului de calcul principal, trebuie ca S să fie inițializat cu A.

Dacă presupunem că este definită procedura `produs (n, A, B, C)` pentru calculul produsului `C = A · B`, unde n este ordinul matricelor A, B și C, atunci structura generală a procesului de calcul principal este următoarea (matricea P se va utiliza pentru returnarea produsului $A \cdot S$):

```
begin
   read n,k;
   read A = ||a_ij||_i,j=1,n;
   // inițializări
   X ← A;
   S ← A;
   // procesul de calcul principal
   for i = 2, k do begin
      produs(n, A, S, P);
      X ← X + P;
      S ← P;                                  // salvare
   end
   write X = ||x_ij||_i,j=1,n;
end
```

Observație: Există limbaje de programare care oferă operații (atribuirea și suma) cu tablouri.

Pentru elaborarea procedurii `produs(n, A, B, C)`, vom porni de la faptul că elementele matricei C se calculează după regula (linia i a lui A se "înmulțește" ca produs scalar cu coloana j a lui B): $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$, unde $i, j \in \{1, 2, \ldots, n\}$.

Pentru determinarea elementelor matricei C se vor utiliza trei instrucțiuni **for** imbricate (un proces de calcul principal care să parcurgă liniile $i \in \{1, 2, \ldots, n\}$, un subproces de calcul care să parcurgă coloanele $j \in \{1, 2, \ldots, n\}$ și pentru fiecare pas al acestui subproces, un subproces de calcul pentru determinarea sumei din formula de calcul pentru elementul c_{ij}).

b) *Mulțimea variabilelor* (***V***)

Variabilele definite de raționamentul ***R***, care vor fi utilizate în procesul de calcul ***P***, sunt:
- k - puterea;
- n - ordinul matricei A;
- $A = \|a_{ij}\|_{i,j=1,n}$ - matricea inițială;
- $X = \|x_{ij}\|_{i,j=1,n}$ - matricea rezultat;
- $S = \|s_{ij}\|_{i,j=1,n}$, $P = \|p_{ij}\|_{i,j=1,n}$ - matrici de lucru.

c) *Procesul de calcul* (***P***) *și reprezentarea algoritmului*

În continuare prezentăm algoritmul de rezolvare (procesul de calcul) al problemei:

```
Algorithm Suma_puterilor;
   Integer n, k;
   Real A(30,30), X(30,30),
        S(30,30), P(30,30);

   procedure produs(n, A, B, C)
      Integer i,j,k;
      Real A(n,n), B(n,n), C(n,n);
   begin
      for i = 1, n do
         for j = 1, n do begin
            c_ij ← 0;
            for k= 1, n do
               c_ij ← c_ij + a_ik · b_kj;
            end
   end

begin
   read n,k;
   read A = ||a_ij||_i,j=1,n;
   // inițializări
   X ← A;
   S ← A;
   // procesul de calcul principal
   for i = 2, k do begin
      produs(n, A, S, P);
      X ← X + P;
      S ← P;                                  // salvare produs
   end
   write X = ||x_ij||_i,j=1,n;
end
```

Bibliografie

1. Albeanu Gr., *Algoritmi și limbaje de programare*, Universitatea "Spiru Haret", Editura România de Mâine, București, 2000
2. Apostol C., Roșca I. Gh., Roșca V., Ghilic-Micu B., *Introducere în programare. Teorie și aplicații*, Editura București, 1993
3. Georgescu H., *Programare concurentă. Teorie și aplicații*, Editura Tehnică, București, 1996
4. Kinston J. H., *Algorithms and Structures Design. Correctness, Analysis*, Addison Wesley, Longman, 1998
5. Mitrana V., *Provocarea algoritmilor. Probleme pentru concursurile de informatică*, Editura Agni, București, 1994
6. Odăgescu I., *Optimizarea algoritmilor*, Editura Militară, București, 1991
7. Perjeriu E., Văduva I., *Îndrumar pentru lucrări de laborator la cursul de Bazele Informaticii*, Universitatea din București, 1986
8. Popovici C., Georgescu H., State L., *Bazele Informaticii*, vol. I, Universitatea din București, 1990
9. Skiena S., *The Algorithm Design Manual*, Springer Verlag, New York, Inc., 1998
10. Vlada M., *Informatica*, Universitatea din București, Editura Ars Docendi, București, 1999
11. Vlada M., *Poligoane stelate. Problema lui Hopf și Pannwitz*, Gazeta de matematică, nr. 8/1995, pag. 339-348
12. Cristea V., Giumale C., Kalisz E., Păunoiu Al., *Limbajul C standard*, Editura Teora, București, 1992
13. Zaharia M. D., *Structuri de date și algoritmi. Exemple în limbajele C și C++*, Editura Albastră, Cluj-Napoca, 2002
14. Popovici M. D., Popovici M. I., *C++. Tehnologia orientată spre obiecte. Aplicații*, Editura Teora, București, 2000
15. www.math.gatech.edu/~thomas/FC/fourcolor.html
16. mathworld.wolfram.com/Four-ColorProblem.html
17. spicerack.sr.unh.edu/~student/tutorial/fourColor/FourColor.html
18. mathcentral.uregina.ca/RR/database/RR.09.97/fisher1.html
19. www.uccs.edu/~asoifer/book5.html
20. www.uccs.edu/~asoifer/solve98.html
21. www-groups.dcs.st-and.ac.uk/~history/BigPictures/Guthrie.jpeg
22. mathworld.wolfram.com/Algorithm.html
23. www.cs.rit.edu/~atk/Java/Sorting/sorting.html
24. faculty.juniata.edu/rhodes/cs2/ch129.htm
25. www.scism.sbk.ac.uk/law/Section5/chap1/ s5c1p2.htm
26. www.ics.uci.edu/~eppstein/161/people.html
27. www.cs.pitt.edu/~kirk/algorithmcourses/
28. www.gnacademy.org/text/cc/
29. java.sun.com/docs/books/tutorial/java/concepts/
30. www.accu.org/acornsig/public/articles/oop_c.html
31. loki.cs.brown.edu:8081/webae/full.html

Domnul conf. dr. Marin Vlada este cadru didactic la Universitatea București și poate fi contactat prin e-mail la adresa vlada@chem.unibuc.ro.