UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

Consultații admitere

16.01.2021

Contents

Algoritmi care lucrează cu tipuri de date definite de utilizator Partea I

Prof. Dr. Czibula Istvan

Problemă de programare

Enunț

Scrieți o aplicație care să suporte operații de adunare pe numere mari (pâna la 10000 de cifre). Aplicația va citi de la tastatură câte două astfel de numere și va afișa rezultatul adunării. Acest lucru se repetă până când utilizatorul închide aplicația.

Exemplu:

- Primul numar: 2345235324532345325345345432
- Al doilea număr: 3254435312352353252352532
- Rezultatul adunării: 2348489759844697678597697964

Analiză

- Problema nu este triviala: nu putem folosi tipuri de date existente, deoarece acestea sunt limitate la maxim 64 de biți, adică aproximativ 19 cifre.
- Trebuie să implementăm algoritmul de adunare, cel folosit când adunăm două numere pe foaie.
- Ne trebuie o reprezentare convenabilă a unui număr. Putem folosi un tablou de cifre, dar atunci implementarea devine anevoioasa, deoarece nu putem returna rezultatul adunării decât prin intermediul unui parametru. Mai mult, trebuie să păstrăm lungimea tabloului într-un alt parametru.
- Pentru a rezolva aceste neajunsuri vom folosi o structură.

Implementare

C++

```
#include <iostream>
#include <cstring>
```

```
using namespace std;

const int MAX_CIFRE = 10000;

/*
Retine un numar mare, impreuna cu lungimea sa (numarul de cifre).
Pentru usurinta implementarii, numarul este retinut invers, adica cifra unitatilor se
afla pe prima pozitie a tabloului cifre.
*/

struct NumarMare
{
    int lungime = 0;
    int cifre[MAX_CIFRE];
};

/*
Construieste un NumarMare pe baza unui sir de caractere.
Input:
- cifre: un sir de caractere ce retine cifrele asa cum s-ar citi de la tastatura
Output:
- un NumarMare construit pe baza sirului de caractere si respectand definitia
  structurii
*/
NumarMare construiesteDinSirDeCaractere(char cifre[])
{
    NumarMare rezultat;
    rezultat.lungime = strlen(cifre);
    for (int i = 0; i < rezultat.lungime; ++i)
    {
        rezultat.cifre[rezultat.lungime - i - 1] = cifre[i] - '0';
    }
    return rezultat;
}

/*
Afiseaza un numar mare, tinand cont de definitia structurii.
Input:
- numarMare: numarul mare pe care dorim sa-l afisam.
Output:
- afiseaza numarul pe ecran.
*/
void afisareNumarMare(const NumarMare &numarMare)
{
    for (int i = numarMare.lungime - 1; i >= 0; --i)
    {
        cout << numarMare.cifre[i];
    }
    cout << endl;
}

/*
Aduna doua numere mari.
Input:
- nr1, nr2: doua numere mari
Output:
- returneaza nr1+nr2
```

```
*/
NumarMare aduna(const NumarMare &nr1, const NumarMare &nr2)
{
    NumarMare rezultat;
    int minCifre = min(nr1.lungime, nr2.lungime);
    int tinMinte = 0;
    for (int i = 0; i < minCifre; ++i)
    {
        int suma = nr1.cifre[i] + nr2.cifre[i] + tinMinte;
        int cifraRezultat = suma % 10;
        tinMinte = suma / 10;
        rezultat.cifre[rezultat.lungime++] = cifraRezultat;
    }
    // completez ce a ramas
    for (int i = minCifre; i < nr1.lungime; ++i)
    {
        int suma = nr1.cifre[i] + tinMinte;
        rezultat.cifre[rezultat.lungime++] = suma % 10;
        tinMinte = suma / 10;
    }
    for (int i = minCifre; i < nr2.lungime; ++i)
    {
        int suma = nr2.cifre[i] + tinMinte;
        rezultat.cifre[rezultat.lungime++] = suma % 10;
        tinMinte = suma / 10;
    }
    if (tinMinte)
        rezultat.cifre[rezultat.lungime++] = tinMinte;

    return rezultat;
}

int main()
{
    char nr[MAX_CIFRE + 1];
    while (true)
    {
        cout << "Dati primul numar:" << endl;
        cin >> nr;
        NumarMare nr1 = construiesteDinSirDeCaractere(nr);

        cout << "Dati al doilea numar:" << endl;
        cin >> nr;
        NumarMare nr2 = construiesteDinSirDeCaractere(nr);

        cout << "Rezultatul adunarii celor doua numere este:" << endl;
        afisareNumarMare(aduna(nr1, nr2));
        cout << endl;
    }

    return 0;
}
```

Pascal

```
program adunareNumarMare;
const MAX_CIFRE = 10000;
```

```
{Retine un numar mare, impreuna cu lungimea sa (numarul de cifre).
Pentru usurinta implementarii, numarul este retinut invers, adica cifra unitatilor se
afla pe prima pozitie a tabloului cifre.}
type NumarMare = record
    cifre : array [1..MAX_CIFRE] of byte;
    lungime: integer;
end;

   {Initializeaza numarul}
procedure initializeaza(var nr:NumarMare);
begin
    nr.lungime :=1;
end;

{Adauga o cifra la sfarsit}
procedure adaugaCifra(var nr:NumarMare; cifra:byte);
begin
    nr.cifre[nr.lungime] := cifra;
    nr.lungime := nr.lungime + 1;
end;

{Construieste un NumarMare pe baza unui sir de caractere.
    Input:- cifre: un sir de caractere ce
        retine cifrele asa cum s-ar citi de la tastatura
    Output:- un NumarMare construit pe baza sirului de caractere si respectand
        definitia structurii}
function construieste(cifre : string):NumarMare;
    var i:integer;
        nr:NumarMare;
begin
    initializeaza(nr);
    for i:=length(cifre) downto 1 do begin
        adaugaCifra(nr, ord(cifre[i]) - ord('0'));
    end;
    construieste := nr;
end;

{Afiseaza un numar mare, tinand cont de definitia structurii.
    Input:- numarMare: numarul mare pe care dorim sa-l afisam.
    Output:- afiseaza numarul pe ecran.}
procedure afisare(var nr:NumarMare);
    var i:integer;
begin
    for i:= nr.lungime - 1 downto 1 do begin
        write(nr.cifre[i]);
    end;
    writeln;
end;

{Aduna doua numere mari.
    Input:- nr1, nr2: doua numere mari
    Output:- returneaza nr1+nr2}
function aduna(var nr1,nr2:NumarMare):NumarMare;
    var sum, nrAux : NumarMare;
        transport:byte;
        pozCifra, s:integer;
begin
```

```
    initializeaza(sum);
    transport := 0;
    {parcurgem cifrele comune. Obs: cifra unitatilor e pe pozitia 1}
    while (sum.lungime < nr1.lungime) and (sum.lungime < nr2.lungime) do
    begin
        s := nr1.cifre[sum.lungime] + nr2.cifre[sum.lungime] + transport;
        adaugaCifra(sum, s mod 10);
        transport:= s div 10;
    end;
    {daca nr1, nr2 au numar diferit de cifre atunci mai avem cifre de adunat}
    if nr1.lungime>nr2.lungime then nrAux := nr1
    else nrAux := nr2;

    {punem si restul cifrelor, trebuie sa tinem cont de transport}
    while sum.lungime < nrAux.lungime do begin
        s := nrAux.cifre[sum.lungime] + transport;
        adaugaCifra(sum, s mod 10);
        transport:= s div 10;
    end;
    {poate mai am transport si dupa ce am adunat cifrele}
    if transport>0 then adaugaCifra(sum, transport);
    aduna:=sum;
end;

var nrString:string;
    nr1,nr2, sum: NumarMare;
begin
    writeln('Dati primul numar:');
    readln(nrString);
    nr1 := construieste(nrString);
    writeln('Dati al doilea numar:');
    readln(nrString);
    nr2 := construieste(nrString);
    afisare(nr1);
    afisare(nr2);
    writeln('suma este:');
    sum := aduna(nr1,nr2);
    afisare(sum);
    readln;
end.
```

Problemă de modelare

Enunț

Considerăm o matrice A de numere naturale de dimensiune 2×2. Fie următoarea suma:

$$S(n) = (A + A^2 + A^3 + \ldots + A^n) \text{ modulo } p \qquad (1)$$

unde p poate fi orice numar prim. Pentru simplitate, vom considera $p = 666013$.

Avem $1 \leq n \leq 10^{18}$.

Operația X modulo p, unde X este o matrice, are ca rezultat o matrice cu elementele lui X care se iau fiecare modulo p.

Să se calculeze suma $S(n)$.

Analiza

- O primă rezolvare în $O(n)$ presupune un for de la 2 la n în care actualizăm suma (inițial A) și termenul curent (inițial A^2): la sumă adăugăm termenul curent, iar termenul curent îl înmulțim cu A. Acest lucru este implementat în funcția **calculeazaSFolosindBruteForce**.
- Observăm că n poate fi prea mare pentru o rezolvare în $O(n)$. Ne propunem să găsim rezolvări de complexitate mai bună.
- Observăm că avem o progresie geometrică de rație A.
- O primă idee poate fi să folosim formula de sumă pentru o progresie geometrică.
- Dar avem de-a face cu matrici și cu modulo, ceea ce complică folosirea formulei (dar o face imposibilă?)
- Încercăm să manipulăm suma astfel încât să obținem o expresie care se poate calcula mai eficient.
- Ne gândim că dacă putem exprima $S(n)$ în funcție de $S(n/2)$ (deocamdată nu ne gândim la detalii gen paritatea lui n), atunci putem obține o rezolvare în $O(log\ n)$. Complexitatea s-ar reduce deoarece, la fiecare pas, l-am înjumatați pe n, pâna ajunge la 1. De câte ori îl putem înjumatați pe n pâna sa ajunga la 1? De aproximativ $log\ n$ ori.
  Aceasta exprimare a lui $S(n)$, dacă e posibilă, s-ar realiza, probabil, printr-o operație gen "factor comun".
- Dar dacă îl dăm factor comun pe A, nu ne ajuta cu nimic...
- O altă idee ar fi să înjumătățim numărul termenilor din suma printr-o factorizare de genul:

$$A + A^2 + A^3 + \ldots + A^n = (X + Y)(A + \ldots + A^{n/2}) \qquad (2)$$

- Astfel, ne propunem ca, înmulțind cu X să obținem prima jumătate a sumei, iar înmulțind cu Y să obținem a doua jumatate.
- Se observă destul de ușor că funcționează $X = I$ și $Y = A^{n/2}$
- Rezultă:

$$S(2k) = (I + A^k)(A + \ldots + A^k) \qquad (3)$$

$$= (I + A^k)\,S(k) \qquad (4)$$

$$S(2k + 1) = S(2k) + A^{2k+1} \qquad (5)$$

Considerente de implementare

- Avem nevoie de o funcție care ridică o matrice (de 2×2) la o anumită putere.

- Vom folosi algoritmul de exponențiere logaritmică - implementat atât iterativ cât și recursiv. Implementarea iterativă ține cont de reprezentarea binară a exponentului. Pentru a înțelege mai bine modul de lucru al algoritmului, este recomandat să îl rulați manual pe un exemplu (pentru ușurință, se pot folosi numere, nu este necesară o matrice).
- Vom folosi o structură $Matrice2\times2$, pentru a putea lucra cu matrici așa cum am lucra și cu tipuri de date primitive.
- O implementare directă duce la complexitatea $O(log^2\ n)$, deoarece apelăm funcția de ridicare la putere la fiecare pas al recursivității principale. Această abordare este implementată în funcția **calculeazaSInLogPatrat**.
- Implementând exponențierea logaritmică în funcția recursivă principală, putem obține complexitatea $O(log\ n)$. Această abordare este implementată în funcția **calculeazaSInLog**.

Implementare

C++

```cpp
#include <iostream>

using namespace std;

const int p = 666013;

/*
Retine o matrice de 2x2 de numere intregi.

*/
struct Matrice2x2
{
    // Elementele matricei
    // trebuie long deoarece p*p > INT_MAX
    long x[2][2];

    /*
    Initializeaza o matrice de 2x2, cu elementele luate modulo p.
    Input:
    - xij: elementul de pe linia i si coloana j (mod p)
    Output: -
    */
    Matrice2x2(long x00, long x01, long x10, long x11)
    {
        x[0][0] = x00 % p;
        x[0][1] = x01 % p;
        x[1][0] = x10 % p;
        x[1][1] = x11 % p;
    }
};

/*
Returneaza matricea unitate de 2x2.
*/
Matrice2x2 getUnitate()
```

```
{
    return Matrice2x2(1, 0, 0, 1);
}

/*
Inmulteste doua matrici mod P.
Input:
- X: o matrice de 2x2 de intregi
- Y: o matrice de 2x2 de intregi
Output:
- X*Y (mod p)
*/
Matrice2x2 inmulteste(Matrice2x2 X, Matrice2x2 Y)
{
    return Matrice2x2(
        (X.x[0][0]*Y.x[0][0] + X.x[0][1]*Y.x[1][0]) % p,
        (X.x[0][0]*Y.x[0][1] + X.x[0][1]*Y.x[1][1]) % p,
        (X.x[1][0]*Y.x[0][0] + X.x[1][1]*Y.x[1][0]) % p,
        (X.x[1][0]*Y.x[0][1] + X.x[1][1]*Y.x[1][1]) % p
    );
}

/*
Aduna doua matrici mod P.
Input:
- X: o matrice de 2x2 de intregi
- Y: o matrice de 2x2 de intregi
Output:
- X + Y (mod p)
*/
Matrice2x2 aduna(Matrice2x2 X, Matrice2x2 Y)
{
    return Matrice2x2(
        (X.x[0][0] + Y.x[0][0]) % p,
        (X.x[0][1] + Y.x[0][1]) % p,
        (X.x[1][0] + Y.x[1][0]) % p,
        (X.x[1][1] + Y.x[1][1]) % p
    );
}

/*
Ridica o matrice la o putere data (mod p).
Input:
- X: matrice de 2x2 de intregi
- exponent: puterea la care sa se ridice X
Output:
- X la puterea exponent (mod p)
*/
Matrice2x2 ridicaLaPutere(Matrice2x2 X, long exponent)
{
    if (exponent == 0)
    {
        // matricea I
        return getUnitate();
    }

    Matrice2x2 jumatate = ridicaLaPutere(X, exponent / 2);
```

```
    Matrice2x2 jumatatePatrat = inmulteste(jumatate, jumatate);

    if (exponent % 2 == 0)
    {
        return jumatatePatrat;
    }

    return inmulteste(jumatatePatrat, X);
}
Matrice2x2 ridicaLaPutereIterativ(Matrice2x2 X, long exponent)
{
    Matrice2x2 rezultat = getUnitate();
    while (exponent)
    {
        if (exponent % 2)
        {
            rezultat = inmulteste(rezultat, X);
            --exponent;
        }

        X = inmulteste(X, X);
        exponent /= 2;
    }


    return rezultat;

}

/*
Afiseaza o matrice de 2x2.
Input:
- X: matricea de afisat
Output: -
*/
Matrice2x2 afiseazaMatrice(Matrice2x2 X)
{
    cout << X.x[0][0] << " " << X.x[0][1] << endl << X.x[1][0] << " " << X.x[1][1] <<
        endl;
}

/*
Citeste o matrice de 2x2.
Input: -
Output:
- matricea citita
*/
Matrice2x2 citesteMatrice()
{
    Matrice2x2 X(0, 0, 0, 0);
    for (int i = 0; i < 2; ++i)
    {
        for (int j = 0; j < 2; ++j)
        {
            cout << "Dati elementul de pe linia " << i << " si coloana " << j  << ": ";
            cin >> X.x[i][j];
        }
    }
```

```
    return X;
}

/*
Citeste n-ul din enunt.
Input: -
Output:
- valoarea citita
*/
long citesteN()
{
    cout << "Dati n (pana la cat se calculeaza suma): ";
    long n;
    cin >> n;
    return n;
}

/*
Calculeaza suma S(n), in complexitate O(n)
Input:
- A: matricea A din enunt.
- n: limita din enunt.
Output:
- S(n) (mod p).
*/
Matrice2x2 calculeazaSFolosindBruteForce(Matrice2x2 A, long n)
{
    Matrice2x2 rez = A;
    Matrice2x2 curent = inmulteste(A, A);
    for (int i = 2; i <= n; ++i)
    {
        rez = aduna(rez, curent);
        curent = inmulteste(curent, A);
    }

    return rez;
}

/*
Calculeaza suma S(n), in complexitate O(log^2 n)
Input:
- A: matricea A din enunt.
- n: limita din enunt.
Output:x
- S(n) (mod p).
*/
Matrice2x2 calculeazaSInLogPatrat(Matrice2x2 A, long n)
{
    // n >= 1 in enunt. Daca poate fi si 0?
    if (n == 1)
    {
        return A;
    }

    long k = n / 2;
    Matrice2x2 jumatate = calculeazaSInLogPatrat(A, k);
```

```
    // apeleaza ridicaLaPutere, care are complexitatea O(log n)
    // rezulta complexitate totala O(log^2 n)
    Matrice2x2 unitatePlusALaNpe2 = aduna(getUnitate(), ridicaLaPutere(A, k));
    Matrice2x2 tot = inmulteste(unitatePlusALaNpe2, jumatate);

    if (n % 2 == 0)
    {
        return tot;
    }

    return aduna(tot, ridicaLaPutere(A, n));
}

/*
Helper pentru calculeaza suma S(n), in complexitate O(log n).
Input:
- A: matricea A din enunt.
- n: limita din enunt.
- AlaPutereaNpe2: folosit pentru a calcula A^n in acelasi timp cu suma.
Output:
- S(n) (mod p).
*/
Matrice2x2 calculeazaSInLogHelper(Matrice2x2 A, long n, Matrice2x2 &
   AlaPutereaNpe2)
{
    if (n == 1)
    {
        AlaPutereaNpe2 = A;
        return A;
    }

    long k = n / 2;
    Matrice2x2 jumatate = calculeazaSInLogHelper(A, k, AlaPutereaNpe2);

    // nu se mai apeleaza functia de ridicare la putere => complexitatea totala ramane
    O(log n)
    Matrice2x2 unitatePlusALanPe2 = aduna(getUnitate(), AlaPutereaNpe2);
    Matrice2x2 tot = inmulteste(unitatePlusALanPe2, jumatate);

    Matrice2x2 patrat = inmulteste(AlaPutereaNpe2, AlaPutereaNpe2);

    if (n % 2 == 0)
    {
        AlaPutereaNpe2 = patrat;
        return tot;
    }

    AlaPutereaNpe2 = inmulteste(patrat, A);
    return aduna(tot, AlaPutereaNpe2);
}
/*
Calculeaza suma S(n), in complexitate O(log n)
Input:
- A: matricea A din enunt.
- n: limita din enunt.
Output:
- S(n) (mod p).
```

```
*/
Matrice2x2 calculeazaSInLog(Matrice2x2 A, long n)
{
    Matrice2x2 temp(0, 0, 0, 0);
    return calculeazaSInLogHelper(A, n, temp);
}

int main()
{
    Matrice2x2 A = citesteMatrice();
    long n = citesteN();

    cout << "Matricea data este:" << endl;
    afiseazaMatrice(A);
    cout << endl << endl;

    cout << "Sumele calculate sunt:" << endl;
    cout << "-------Cu brute force-------" << endl;
    afiseazaMatrice(calculeazaSFolosindBruteForce(A, n));
    cout << "-------In O(log^2 n)--------" << endl;
    afiseazaMatrice(calculeazaSInLogPatrat(A, n));
    cout << "-------In O(log n)----------" << endl;
    afiseazaMatrice(calculeazaSInLog(A, n));
    cout << "--------------" << endl;

    return 0;
}
```

Pascal

```
Program structuri;
const p = 666013;

{Retine o matrice de 2x2 de numere intregi.}
type Matrice2x2 = record
{ Elementele matricei trebuie long deoarece p*p > INT_MAX}
    elems:array[0..10,0..10] of longint;
    linii: longint;
    coloane: longint;
end;

{Initializeaza o matrice de 2x2, cu elementele luate modulo p.
    Input:
    - xij: elementul de pe linia i si coloana j (mod p)
    Output: -}
function initializareMatrice (x00, x01, x10, x11:longint):Matrice2x2;
    var x: Matrice2x2;
begin
    x.elems[0,0] := x00 mod p;
    x.elems[0,1] := x01 mod p;
    x.elems[1,0] := x10 mod p;
    x.elems[1,1] := x11 mod p;

    initializareMatrice:=x;
end;
```

```
{Returneaza matricea unitate de 2x2.}
function getUnitate():Matrice2x2;
var result : Matrice2x2;
begin
    result:=initializareMatrice(1, 0, 0, 1);
    getUnitate := result;
end;

{Inmulteste doua matrici mod P.
    Input:
    - X: o matrice de 2x2 de intregi
    - Y: o matrice de 2x2 de intregi
    Output:
    - X*Y (mod p)}
function inmulteste(X : Matrice2x2;Y : Matrice2x2) : Matrice2x2;
    var Res: Matrice2x2;
begin
    Res := initializareMatrice(0, 0, 0, 0);
    Res.elems[0,0] := (X.elems[0,0]*Y.elems[0,0] + X.elems[0,1]*Y.elems [1,0]) mod p;
    Res.elems[0,1] := (X.elems[0,0]*Y.elems[0,1] + X.elems[0,1]*Y.elems [1,1]) mod p;
    Res.elems[1,0] := (X.elems[1,0]*Y.elems[0,0] + X.elems[1,1]*Y.elems [1,0]) mod p;
    Res.elems[1,1] := (X.elems[1,0]*Y.elems[0,1] + X.elems[1,1]*Y.elems [1,1]) mod p;

    inmulteste := Res;
end;

{Aduna doua matrici mod P.
    Input:
    - X: o matrice de 2x2 de intregi
    - Y: o matrice de 2x2 de intregi
    Output:
    - X + Y (mod p)}
function aduna( X : Matrice2x2; Y: Matrice2x2):Matrice2x2 ;
    var Res : Matrice2x2;
begin
    Res := initializareMatrice(0, 0, 0, 0);
    Res.elems[0,0] := (X.elems[0,0] + Y.elems[0,0]) mod p;
    Res.elems[0,1] := (X.elems[0,1] + Y.elems[0,1]) mod p;
    Res.elems[1,0] := (X.elems[1,0] + Y.elems[1,0]) mod p;
    Res.elems[1,1] := (X.elems[1,1] + Y.elems[1,1]) mod p;

    aduna:= Res;
end;

{Ridica o matrice la o putere data (mod p).
    Input:
    - X: matrice de 2x2 de intregi
    - exponent: puterea la care sa se ridice X
    Output:
    - X la puterea exponent (mod p)}
function ridicaLaPutere( X : Matrice2x2; exponent : integer): Matrice2x2;
    var jumatate, jumatatePatrat, Res : Matrice2x2;
begin
    if (exponent = 0) then ridicaLaPutere := getUnitate()
    else begin

        jumatate := ridicaLaPutere(X, exponent div 2);
```

```
        jumatatePatrat := inmulteste(jumatate, jumatate);

        if (exponent mod 2 = 0)
            then Res := jumatatePatrat
            else Res := inmulteste(jumatatePatrat, X);

        ridicaLaPutere:=Res;
    end;
end;

{Citeste o matrice de 2x2.
    Input: -
    Output:
    - matricea citita}
procedure citesteMatrice(var m:Matrice2x2);
    var i,j:integer;
begin
    for i:=0 to 1 do begin
        for j:=0 to 1 do begin
            write('m[',i,',',j,']=');
            readln(m.elems[i,j]);
        end;
    end;
end;

{Afiseaza o matrice de 2x2.
    Input:
    - X: matricea de afisat
    Output: -}
procedure afiseazaMatrice(var m:Matrice2x2);
    var i,j:integer;
begin
    writeln;
    for i:=0 to 1 do
    begin
        for j:=0 to 1 do
            write(m.elems[i,j]:2,',');
        writeln;
    end;
end;

{Citeste n-ul din enunt.
    Input: -
    Output:
    - valoarea citita }
procedure citesteNumar(n:integer);
begin
    write('"Dati n (pana la cat se calculeaza suma): ');
    readln(n);
end;

{Calculeaza suma S(n), in complexitate O(n)
    Input:
    - A: matricea A din enunt.
    - n: limita din enunt.
    Output:
    - S(n) (mod p).}
function calculeazaSFolosindBruteForce(var A: Matrice2x2; n:longint): Matrice2x2;
```

```
    var res,res1,curent1, curent : Matrice2x2; i:integer;
begin
    res := A;

    curent := inmulteste(A, A);
    for i := 2 to n do
    begin
        res1 := aduna(res, curent);
        curent1 := inmulteste(curent, A);
        res:=res1;
        curent:=curent1;
    end;

    calculeazaSFolosindBruteForce:=res;
end;

{Calculeaza suma S(n), in complexitate O(log^2 n)
    Input:
    - A: matricea A din enunt.
    - n: limita din enunt.
    Output:
    - S(n) (mod p).}
function calculeazaSInLogPatrat(var A : Matrice2x2; n : longint) : Matrice2x2;
    var res, jumatate, unitatePlusALaNpe2, tot : Matrice2x2;
        k: longint;
begin
    { n >= 1 in enunt. Daca poate fi si 0?}
    if (n = 1) then calculeazaSInLogPatrat := A
    else begin
        k := n div 2;
        jumatate := calculeazaSInLogPatrat(A, k);
        { apeleaza ridicaLaPutere, care are complexitatea O(log n) rezulta
            complexitate totala O(log^2 n)}
        unitatePlusALaNpe2 := aduna(getUnitate(), ridicaLaPutere(A, k));
        tot := inmulteste(unitatePlusALaNpe2, jumatate);
        if (n mod 2 = 0) then res := tot
            else res := aduna(tot, ridicaLaPutere(A, n));

        calculeazaSInLogPatrat := res;
    end;
end;

{Helper pentru calculeaza suma S(n), in complexitate O(log n).
    Input:
    - A: matricea A din enunt.
    - n: limita din enunt.
    - AlaPutereaNpe2: folosit pentru a calcula A^n in acelasi timp cu suma.
    Output:
    - S(n) (mod p).}
function calculeazaSInLogHelper(var A : Matrice2x2; n :longint; var
    AlaPutereaNpe2:Matrice2x2) : Matrice2x2;
    var jumatate, unitatePlusALanPe2, tot, patrat, result : Matrice2x2;
        k : longint;
begin
    if (n = 1) then
    begin
        AlaPutereaNpe2 := A;
        calculeazaSInLogHelper := AlaPutereaNpe2;
```

```
    end

    else begin
        k := n div 2;
        jumatate := calculeazaSInLogHelper(A, k, AlaPutereaNpe2);
        { nu se mai apeleaza functia de ridicare la putere => complexitatea  totala
            ramane O(log n)}
        unitatePlusALanPe2 := aduna(getUnitate(), AlaPutereaNpe2);
        tot := inmulteste(unitatePlusALanPe2, jumatate);

        patrat := inmulteste(AlaPutereaNpe2, AlaPutereaNpe2);

        if (n mod 2 = 0) then
        begin
            AlaPutereaNpe2 := patrat;
            calculeazaSInLogHelper := tot;
        end
        else
        begin
            AlaPutereaNpe2 := inmulteste(patrat, A);
            calculeazaSInLogHelper := aduna(tot, AlaPutereaNpe2);
        end;
    end;
end;

{Calculeaza suma S(n), in complexitate O(log n)
    Input:
    - A: matricea A din enunt.
    - n: limita din enunt.
    Output:
    - S(n) (mod p).}
function calculeazaSInLog(var A:Matrice2x2; n:longint): Matrice2x2;
    var temp, res : Matrice2x2 ;
begin
    temp:=initializareMatrice(0, 0, 0, 0);
    res := calculeazaSInLogHelper(A, n,temp);
    calculeazaSInLog := res;
end;

var A,aux: Matrice2x2;
    n : longint;

begin
    citesteMatrice(A);
    citesteNumar(n);
    write('Matricea data este:');
    afiseazaMatrice(A);
    writeln('Sumele calculate sunt:');
    write ( '-------Cu brute force-------' );
    aux:=calculeazaSFolosindBruteForce(A, n);
    afiseazaMatrice(aux);
    write('-------In O(log^2 n)--------');
    aux:=calculeazaSInLogPatrat(A, n);
    afiseazaMatrice(aux);
    write ( '-------In O(log n)----------' );
    aux:=calculeazaSInLog(A, n);
    afiseazaMatrice(aux);
    writeln ( '--------------');
```

```
end.
```

Algoritmi care lucrează cu tipuri de date definite de utilizator Partea II

Lector. Dr. Pop Andreea-Diana

Întrebări grilă

Grila 1

Se dă următorul tip de date definit de utilizator:

C++

```
struct Nota
{
    float valoare;
    char numeStudent[50];
};
```

Pascal

```
type Nota = record
    valoare: real;
    numeStudent: string[50];
end;
```

Selectați variantele care conțin cod compilabil, presupunând că bibliotecile necesare sunt incluse:

C++

```
a) Nota note[100];
b) Nota n1;
   n1[valoare] = 10;
   n1[numeStudent] = "Ion";
c) Nota n1;
   n1.valoare = 10;
   n1.numeStudent = "Ion";
d) Nota n1;
   n1.valoare = 10;
   strcpy(n1.numeStudent,"Ion“);
```

Pascal

```
a) var note: array [1..100] of Nota;
b) var n1:Nota;
   begin
       n1[valoare] := 10;
       n1[numeStudent] := ’Ion’;
   end.
c) var n1:Nota;
   begin
       n1.valoare:= 10;
       n1.numeStudent := ’Ion’;
   end.
d) var n1:Nota;
   begin
       n1.valoare:= 10;
       n1.numeStudent := copy(’Ion Ion’,0,3);
   end.
```

Răspuns: a,c și d.

Justificare: Varianta b) consideră Nota ca fiind un vector de note, ceea ce nu e cazul, Nota este un tip de date definit de utilizator ce conține două câmpuri: valoare – valoarea notei studentului, respectiv numeStudent. În a) se construiește un vector de astfel de Note, în c) respectiv d) se lucrează cu câte o notă.

Grila 2

Se dă următorul tip de date definit de utilizator:

Pseudocod
```
Vector = Înregistrare
        lungime : Întreg
        elemente: Întreg [10]
sf_înregistrare
```

Care dintre următoarele funcții sunt corecte pentru a returna distanța Euclideană dintre doi vectori de lungime egală?

Pseudocod

a) Funcția distanța (v_1, v_2:Vector): Întreg
   distanţa ← $\sqrt{(v_1.x - v_2.x)^2 + (v_1.y - v_2.y)^2}$
   sf_funcție

b) Funcția distanța(v_1, v_2:Vector): Întreg
   Pentru $i \leftarrow \overline{1, v_1.n}$
   distanţa ← $(v_1.elemente_i - v_2.elemente_i)^2$
   sf_Pentru
   sf_funcție

c) Funcția distanța(v_1, v_2:Vector): Real
   dist ← 0;
   Pentru $\mathrm{i} \leftarrow \overline{1, \mathrm{v_1.lungime}}$
   $\mathrm{dif} \leftarrow \mathrm{v_1.elemente_i} - \mathrm{v_2.elemente_i}$
   $\mathrm{dist} \leftarrow \mathrm{dist} + \mathrm{dif}^2$
   sf_Pentru
   distanţa ← $\sqrt{\mathrm{dist}}$
   sf_funcție

d) Funcția distanța(v_1, v_2:Vector): Real
   dist ← 0;
   Pentru $\mathrm{i} \leftarrow \overline{1, \mathrm{v_1.n}}$
   $\mathrm{dif} \leftarrow (\mathrm{v_1})_\mathrm{i} - (\mathrm{v_2})_\mathrm{i}$
   $\mathrm{dist} \leftarrow \mathrm{dist} + \mathrm{dif}^2$
   sf_Pentru
   distanţa ← $\sqrt{\mathrm{dist}}$
   sf_funcție

Răspuns: c).

Justificare: a) reprezintă distanța euclidiană dintre doi vectori bidimensionali, dar tipul de date definit inițial este compus dintr-un vector cu maxim 10 elemente și lungimea care reprezintă câte elemente are de fapt vectorul. Oricum, în varianta a), se consideră că cele două câmpuri ale vectorului sunt *x*, respectiv *y*, ceea ce nu e adevărat.
Varianta d) de asemenea nu utilizează corect tipul de date definit, considerând că v_1 respectiv v_2 sunt pe de o parte vectori, pe de alta tipuri de date definite de utilizator ce au câmpul n.
Nici varianta b) nu utilizează corect tipul de date, deoarece vectorul nu are câmpul n, ci *lungime*. Și nici nu calculează corect distanța euclidiană, deoarece în variabila *distanța*, la ieșirea din ciclul

”Pentru” se păstrează doar pătratul diferenței ultimelor două elemente ale celor doi vectori, nu suma tuturor pătratelor diferențelor.

Grila 3

Se dă următorul tip de date definit de utilizator:

Pseudocod
```
triunghiDr = Înregistrare
        AB,BC : Întreg
sf_înregistrare
```

Unde AB și BC reprezintă lungimile celor 2 catete ale triunghiului dreptunghic.

Care dintre următoarele funcții returnează valoarea cos(A)?

a) Funcția cosA (t: triunghiDr): Real
   $\cos A \leftarrow \frac{t.AB}{t.BC}$
   sf_funcție
b) Funcția cosA (t: triunghiDr): Real
   $\cos A \leftarrow \frac{t.AB}{\sqrt{t.AB^2+t.BC^2}}$
   sf_funcție
c) Funcția cosA (t: triunghiDr): Real
   $\cos A \leftarrow \frac{t.BC}{t.AC}$
   sf_funcție
d) Funcția cosA (t: triunghiDr): Real
   $\cos A \leftarrow \frac{t.BC}{\sqrt{t.AB^2+t.BC^2}}$
   sf_funcție

Răspuns: b).

Justificare: cosinusul se obține împărțind lungimea catetei opuse unghiului A la ipotenuză. Cateta opusă este BC, iar ipotenuza se poate obține prin teorema lui Pitagora din lungimile celor două catete.

Varianta a) returnează cotangenta unghiului A, c) nu e corectă deoarece triunghiul nu are câmpul AC, iar d) returnează sinusul unghiului A.

Grila 4

Se dă următorul tip de date definit de utilizator:

Pseudocod
```
triunghiDrA = Înregistrare
        AB,AC,BC : Întreg
sf_înregistrare
```

Unde AB,AC și BC reprezintă lungimile laturilor unui triunghi dreptunghic în A.

Care dintre următoarele funcții nu returează valoarea sin(B)?

a) Funcția sinB (t: triunghiDrA): Întreg

$\text{sinB} \leftarrow \frac{t.AC}{t.AB}$
sf_funcție

b) Funcția sinB (t: triunghiDrA): Întreg
$\text{sinB} \leftarrow \frac{t.AC}{\sqrt{t.AB+t.AC}}$
sf_funcție

c) Funcția sinB (t: triunghiDrA): Întreg
$\text{sinB} \leftarrow \frac{t.AC}{t.BC}$
sf_funcție

d) Funcția sinB (t: triunghiDrA): Întreg
$\text{sinB} \leftarrow \frac{t.AC}{\sqrt{t.AB^2+t.AC^2}}$
sf_funcție

Răspuns: a) și b).

Justificare: S-a cerut ”care dintre următoarele funcții **nu** returează valoarea sin(B)”. Varianta a) returnează tangenta unghiului B, în varianta b) ar fi trebuit ca la numitor lungimile catetelor să fie ridicate la pătrat.

Grila 5

Dorim să mergem în vacanță și vrem să punem în rucsac cât mai multe obiecte vestimentare de culori diferite, știind că rucsacul are un volum constant.

Să se selecteze componentele esențiale tipului de date ”ObVest” din lista de mai jos:

a) greutate
b) denumire
c) culoare
d) tip (îmbrăcăminte/încălțăminte)
e) volum
f) vreme (tipul de vreme pentru care este potrivit frig/cald)

Selectați metoda optimă de rezolvare a problemei:

a) Greedy
b) backtracking

Răspunsuri: c) și e), respectiv a).

Justificare: În problemă se discută doar despre culoare și volum.
Întrucât avem disponibil un volum constant și trebuie să alegem cât mai multe obiecte vestimentare de culori diferite, vom sorta obiectele vestimentare crescător în funcție de volum, și vom pune în rucsac inițial obiectul cu cel mai mic volum. Căutăm apoi următorul obiect ce are culoare diferită de cele din rucsac și care are cel mai mic volum și încă mai încape în rucsac. Îl adăugăm, și repetăm procesul până ce obiectul vestimentar următor are un volum prea mare și nu mai intră în rucsac. În acel moment reluăm obiectele vestimentare de la început și adăugăm în rucsac cât de multe obiecte vestimentare încap, indiferent ce culoare au. Deci se aplică un algoritm de tip Greedy, care este liniar, cu o complexitate de ordin *n*, deci e de preferat unui algoritm de tip backtracking care are complexitate factorială.

Grila 6

Dorim să mergem în vacanță și vrem să punem în rucsac cât mai multe obiecte vestimentare de culori diferite, dar cel puțin un obiect de îmbrăcăminte și unul de încălțăminte, unul pentru frig și unul pentru cald, știind că rucsacul are un volum constant și rezistă la o greutate dată.

Să se selecteze componentele esențiale tipului de date ”ObVest” din lista de mai jos:

a) greutate
b) denumire
c) culoare
d) tip (îmbrăcăminte/încălțăminte)
e) volum
f) vreme (tipul de vreme pentru care este potrivit frig/cald)

Selectați metoda optimă de rezolvare a problemei:

a) Greedy
b) backtracking

Răspuns: a),c),d),e),f), respectiv b).

Justificare: Doar de denumire nu avem nevoie în rezolvarea problemei.
În acest caz, o simplă ordonare nu ne ajută să rezolvăm problema, astfel că, deși metoda Greedy este mai eficace, nu poate fi aplicată, deoarece ea ne va furniza soluții incomplete.

Grila 7

Se dau următoarele tipuri de date definite de utilizator:

Pseudocod
```
Casa = Înregistrare
        mp: Real
        urm,pred: Întreg
sf_înregistrare
Cartier = Înregistrare
        adr_inc, adr_sf,dim,nr: Întreg
        case: casa[20]
sf_înregistrare
```

unde adr_inc respectiv adr_sf reprezintă indicii primei respectiv ultimei case din cartier, dim este dimensiunea vectorului de case, nr este numărul caselor, vectorul pornește de la indicele 1
iar urm și pred resprezintă indicii caselor vecine: următoare respectiv anterioare.

Care dintre următoarele funcții parcurge casele în ordine inversă?

a) Funcția parcurgere (*c*: Cartier):
```
i ← c.adr_sf
Cât timp  i ≠ c.adr_inc
   @afișează(c.case_i.mp)
   i ← c.case_i.pred
sf_Cât timp
@afișează(c.case_i.mp)
```

sf_funcție

b) Funcția parcurgere (*c*: Cartier):
   Pentru $i \leftarrow \overline{c.nr, 1}$
      @afişează($c.case_i$.mp)
   sf_Pentru
   sf_funcție

c) Funcția parcurgere (*c*: Cartier):
   $i \leftarrow c.nr$
   Cât timp $i \neq 1$
      @afişează($c.case_i$.mp)
      $i \leftarrow$i-1
   sf_Cât timp
   @afişează($c.case_i$.mp)
   sf_funcție

d) Funcția parcurgere (*c*: Cartier):
   $i \leftarrow c.adr\_inc$
   Cât timp $i \neq c.adr\_sf$
      @afişează($c.case_i$.mp)
      $i \leftarrow c.case_i.urm$
   sf_Cât timp
   @afişează($c.case_i$.mp)
   sf_funcție

Răspuns: a).

Justificare: Tipurile de date definite de utilizator corespund unei reprezentări înlănțuite a caselor din cartier. Astfel, casa de pe poziția *i* și casa de pe poziția *i*+1 nu sunt vecine decât în cazul în care indicele *pred* al casei *i*+1 este *i*. De aceea variantele b) și c) nu sunt corecte. În varianta d) parcurgerea se face de la prima la ultima casă, deci nici ea nu corespunde cerinței.

Pentru a înțelege mai bine cum are loc înlănțuirea pe vector, puteți considera următorul exemplu scurt:

Cartierul are:

adr_inc = 4, adr_sf = 5, dim = 6, nr=3

iar vectorul "case":

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
mp	0	50	0	70	100	0														
urm	-1	5	-1	2	-1	-1														
pred	-1	4	-1	-1	2	-1														

Parcurgerea caselor în ordine inversă va afișa: 100, 50, 70.

Deci, se începe cu casa de pe poziția adr_sf, adică casa cu indicele 5, care are 100mp, această casă are casa predecesoare cu indicele 2, deci casa cu 50mp, care are casa predecesoare cu indicele 4, deci cea cu 70mp, care este de fapt și prima casă din cartier.