Tablouri bidimensionale (matrici)

Numere prime in spirala

Enunt

Sa se scrie un program care citeste de la tastatura dimensiunile n si m (n,m<=100) a unei matrici (n x m). Programul va forma matricea din numere prime consecutive aranjate in spirala: incepand de la stanga la dreapta, apoi de sus in jos, apoi de la dreapta la stanga si inapoi de jos in sus. La final programul va afisa matricea formata.

Se cere să se utilizeze subprograme care să comunice între ele şi cu programul principal prin parametri. Fiecare subprogram trebuie specificat.

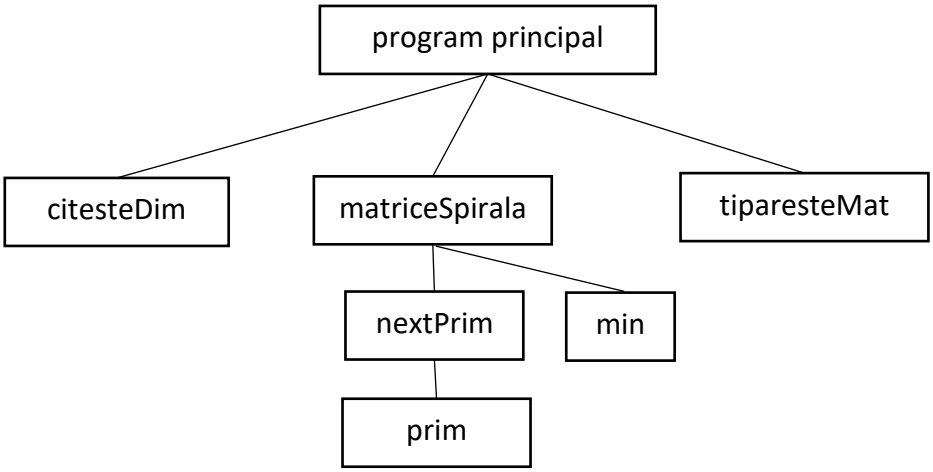
Exemplu

Date de intrare	Date de iesire
n = 5 m = 7	2 3 5 7 11 13 17 71 73 79 83 89 97 19 67 131 157 151 149 101 23 61 127 113 109 107 103 29 59 53 47 43 41 37 31

Pasii algorimului principal

Algoritm matriceSpirala
    @ citeste dimensiuni matrice
    @ formeaza matrice in spirala
    @ afiseaza matrice
Sf.Algoritm

Identificarea subalgoritmilor



Programul

Implementare C++

```
// Rezolvarea nu este optimizata pentru viteza de executie
// Rezolvarea exemplifica o abordare a problemei bazata pe descompunerea in subprobleme
// Programul a fost compilat cu Visual Studio Community 2015
#include <iostream>
using namespace std;

typedef struct {
	int n, m;
	int elem[100][100];
} Matrice;

//Date de intrare: -
//Date de iesire: a matrice cu dimensiunile n,m –dimensiunile matricii, n,m din N, 1<=n,m<=100
void citesteDim(Matrice& a) {
	cout << "Introduceti dimensiunile matricii" << endl;
	cout << "Linii=";
	do {
		cin >> a.n;
		if (!(a.n >= 1 && a.n <= 100))
			cout << "Va rog sa introduceti un numar intre 1 si 100";
	} while (!(a.n >= 1 && a.n <= 100));
	cout << "Coloane=";
	do {
		cin >> a.m;
		if (!(a.m >= 1 && a.n <= 100))
			cout << "Va rog sa introduceti un numar intre 1 si 100";
	} while (!(a.m >= 1 && a.m <= 100));
}

//Date de intrare: a-matricea cu dimensiunile n,m –dimensiunile matricii, n,m din N, 1<=n,m<=100
//Date de iesire: - (se afiseaza matricea pe ecran)
void tiparesteMat(Matrice a)
{
	for(int i = 0;i < a.n;i++)
	{
		for(int j = 0;j < a.m;j++)
```

```
                cout << a.elem[i][j] << " ";
            cout << endl;
        }
}

//Date de intrare: x din Z
//Date de iesire: 0 daca x nu e prim si 1 daca x este prim
int prim(int x)
{
        if (x < 2)
            return 0;
        if (x == 2)
            return 1;
        if (x % 2 == 0)
            return 0;
        for (int i = 3; i*i <= x; i += 2)
            if (x%i == 0)
                return 0;
        return 1;
}


//Date de intrare: nr-numar intreg
//Date de iesire: p – primul nr. prim mai mare decat p
int nextPrim(int nr) {
        nr++;
        while (!prim(nr))
            nr++;
        return nr;
}

//Date de intrare: x,y nr. intregi
//Date de iesire> min(x,y)
int min(int x, int y)
{
        if (x < y)
            return x;
        else
            return y;
}

//Date de intrare: a - matricea la care i se cunosc doar dimensiunile n si m, n,m din N,
1<=n,m<=100
//Date de iesire: a - matricea de numere consecutive asezate in spirala
void matriceSpirala(Matrice& a) {
        int i, j;
        int nrPrim = 1;
        //cu i parcurgem "cercurile" de nr. prime din martice. Sunt min(a.n,a.m) / 2 + min(a.n,
a.m) % 2 astfel de cercuri
        for (i = 0; i < (min(a.n,a.m) / 2 + min(a.n, a.m) % 2); i++) {
            //parcurgem de la stanga la dreapta - marginea de sus a cercului
            for (j = i; j < a.m - i; j++) {
                nrPrim = nextPrim(nrPrim);
                a.elem[i][j] = nrPrim;
            }

            //parcurgem de sus in jos - marginea din dreapta a cercului
            for (j = 1 + i; j < a.n - i; j++) {
                nrPrim = nextPrim(nrPrim);
```

```
                a.elem[j][a.m - i - 1] = nrPrim;
            }

            //parcurgem de la dreapta la stanga- marginea de jos a cercului
            for (j = a.m - i - 2; j >= i; j--) {
                nrPrim = nextPrim(nrPrim);
                a.elem[a.n - i - 1][j] = nrPrim;
            }

            //parcurgem de jos in sus - marginea din stanga a cercului
            for (j = a.n - i - 2; j >= i + 1; j--) {
                nrPrim = nextPrim(nrPrim);
                a.elem[j][i] = nrPrim;
            }
        }
}

int main() {
        Matrice a;
        citesteDim(a);
        matriceSpirala(a);
        tiparesteMat(a);
        return 0;
}
```

Implementare Pascal

```
{ Rezolvarea nu este optimizata pentru viteza de executie
Rezolvarea exemplifica o abordare a problemei bazata pe descompunerea in subprobleme
Programul a fost compilat cu Turbo Pascal 7 + Dos Box}
Type
     Matrice = Record
       n,m: Integer;
       elem: Array[1..10,1..10] Of Integer; {la Turbo Pascal 7 i se umple stiva de lucru la
transmiterea prin parametru de tip valoare a Array[1..100,1..100]}
     End;

{Date de intrare: -
Date de iesire: a matrice cu dimensiunile n,m –dimensiunile matricii, n,m din N, 1<=n,m<=100}
Procedure citesteDim(Var a:Matrice);
Begin
  Writeln('Introduceti dimensiunile matricii:');
  Writeln('Linii=');
  Repeat
    Readln(a.n);
    If Not((a.n >= 1) And (a.n <= 100)) Then
      Writeln('Va rog sa introduceti un numar intre 1 si 100');
  Until (a.n >= 1) And (a.n <= 100);
  Writeln('Coloane=');
  Repeat
    Readln(a.m);
    If Not((a.m >= 1) And (a.n <= 100)) Then
      Writeln('Va rog sa introduceti un numar intre 1 si 100');
  Until (a.m >= 1) And (a.m <= 100);
End;

{Date de intrare: a-matricea cu dimensiunile n,m –dimensiunile matricii, n,m din N, 1<=n,m<=100
Date de iesire: - (se afiseaza matricea pe ecran)}
```

```
Procedure afisare(a: matrice);
Var
  i,j: Integer;
Begin
  For i:=1 To a.n Do
    Begin
      For j:=1 To a.m Do
        Write(a.elem[i,j],' ');
      Writeln;
    End;
End;

{Date de intrare: x din Z
Date de iesire: False daca x nu e prim si True daca x este prim}
Function prim(x:Integer): Boolean;
Var
  i: Integer;
Begin
  prim := True;
  If x < 2 Then
    prim := False
  Else
    If x = 2 Then
      prim := True
  Else
    If x Mod 2 = 0 Then
      prim := False
  Else
    Begin
      i := 3;
      While i*i<=x Do
        Begin
          If x Mod i =0 Then
            Begin
              prim := False;
              i := x+1;
            End;
          i := i+2;
        End;
    End;
End;

{Date de intrare: nr-numar intreg
Date de iesire: p – primul nr. prim mai mare decat p}
Function nextPrim(nr:Integer): Integer;
Begin
  nr := nr+1;
  While Not prim(nr) Do
    nr := nr+1;
  nextPrim := nr;
End;
{Date de intrare: x,y nr. intregi
Date de iesire> min(x,y)}
Function min(x,y:Integer): Integer;
Begin
  If x < y Then
    min := x
  Else
    min := y;
```

```
End;

{Date de intrare: a - matricea la care i se cunosc doar dimensiunile n si m, n,m din N,
1<=n,m<=100
Date de iesire: a - matricea de numere consecutive asezate in spirala}
Procedure matriceSpirala(Var a:Matrice);
Var
  i,j,nrPrim: Integer;
Begin
  nrPrim := 1;

{cu i parcurgem "cercurile" de nr. prime din martice. Sunt min(a.n,a.m) / 2 + min(a.n, a.m) % 2
astfel de cercuri}
  For i := 1 To min(a.n,a.m) Div 2 + min(a.n, a.m) Mod 2 Do
    Begin
 {parcurgem de la stanga la dreapta - marginea de sus a cercului}
      For j := i To a.m - i + 1 Do
        Begin
          nrPrim := nextPrim(nrPrim);
          a.elem[i,j] := nrPrim;
        End;
 {parcurgem de sus in jos - marginea din dreapta a cercului}
      For j := 1 + i To a.n - i+1 Do
        Begin
          nrPrim := nextPrim(nrPrim);
          a.elem[j,a.m - i+1] := nrPrim;
        End;
 {parcurgem de la dreapta la stanga- marginea de jos a cercului}
      For j := a.m - i Downto i Do
        Begin
          nrPrim := nextPrim(nrPrim);
          a.elem[a.n - i+1,j] := nrPrim;
        End;
 {parcurgem de jos in sus - marginea din stanga a cercului}
      For j := a.n - i Downto i + 1 Do
        Begin
          nrPrim := nextPrim(nrPrim);
          a.elem[j,i] := nrPrim;
        End;
    End;
End;

Var
  a: Matrice;
Begin
  citesteDim(a);
  matriceSpirala(a);
  afisare(a);
  Readln;
End.
```

Spiderman

Enunt

Omul păianjen (Spiderman) sare de pe o clădire pe alta, aflată în imediata vecinătate, în nord, est, sud sau vest. Clădirile din cartierul omului păianjen au o înălţime exprimată în numere naturale şi sunt aşezate pe `m` rânduri, câte `n` pe fiecare rând. Spiderman va alege să sară pe una dintre clădirile vecine, care are înălţimea mai mică sau egală, iar diferenţa de înălţime este minimă. Dacă există mai multe clădiri vecine de aceeaşi înălţime, omul păianjen aplică ordinea preferenţială nord, est, sud, vest, dar nu sare încă o dată pe o clădire pe care a mai sărit. Scopul omului păianjen este acela de a reuşi să facă un număr maxim de sărituri succesive.

Cerință

Scrieţi un program care determină numărul maxim de sărituri succesive, pe care îl poate efectua, pornind de la oricare dintre clădiri, precum şi coordonatele cladirii care reprezinta punctul de start pentru drumul maxim.

Date de intrare

n, m: 1<= n,m <= 100
a – matricea cu n linii si m coloane reprezentând înălţimile clădirilor
înălţimile clădirilor (valorile matricii) sunt numere naturale din intervalul [1,10.000]

Date de ieşire

numărul maxim de sărituri, coordonatele (i,j) punctului de start

Exemplu

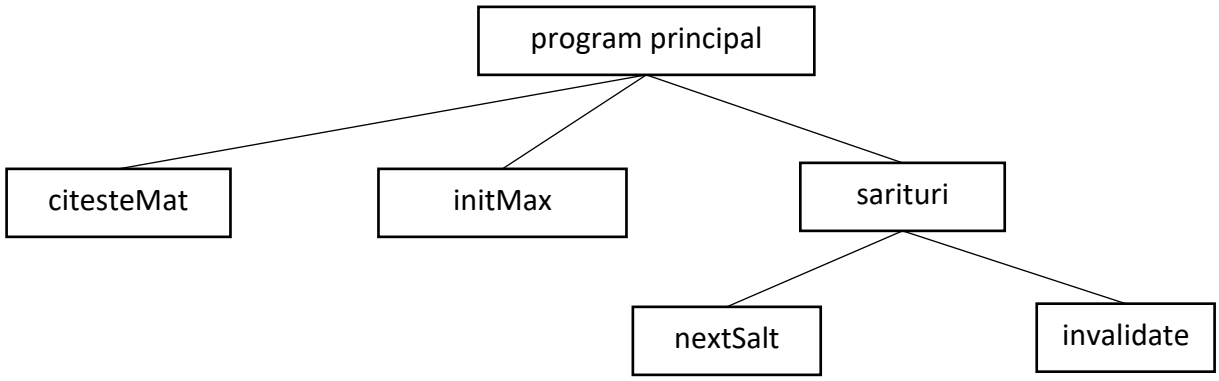
Date de intrare	Date de iesire
n = 5 , m = 5 35 38 42 40 50 34 38 30 75 50 70 78 88 86 30 39 90 88 23 25 35 80 89 90 34	8 (numarul maxim de sarituri) linia = 4, coloana = 3 – pentru numerotarea de la 0 SAU linia = 5, coloana = 4 – pentru numerotarea de la 1

Pasii algorimului principal

Algoritm matriceSpirala
@ se initialieaza un numar max de sarituri si un punct de plecare pentru maximul respectiv
@ Pentru fiecare punct din matrice
@ se alege punctul de start ca si punctul curent

@ se calculeaza numarul de sarituri pentru punctul de start respectiv
@ Daca numar sarituri > numar max sarituri
@se suprascriu numar max de sarituri si punctul de plecare cu noile valori
@ Sf.Daca
@ Sf.Pentru
@ se tipareste numar maxim sarituri si punctul de start
Sf.Algoritm

Identificarea subalgoritmilor



Programul

Implementare C++

```
// Rezolvarea nu este optimizata pentru viteza de executie
// Rezolvarea exemplifica o abordare a problemei bazata pe descompunerea in subprobleme
// Programul a fost compilat cu Visual Studio Community 2015
#include <iostream>
using namespace std;

// Tipul de data matrice
typedef struct {
	int n, m;
	int elem[100][100];
} Matrice;

typedef struct {
	int i;
	int j;
} Punct;

//Date de intrare: -
//Date de iesire: a matrice cu dimensiunile n,m –dimensiunile matricii, n,m din N, 1<=n,m<=100 si
elemente intre 1 si 10000
Matrice citesteMat() {
	Matrice a;
	cout << "Introduceti dimensiunile matricii" << endl;
	cout << "Linii=";
	do {
		cin >> a.n;
		if (!(a.n >= 1 && a.n <= 100))
```

```
                cout << "Va rog sa introduceti un numar intre 1 si 100";
        } while (!(a.n >= 1 && a.n <= 100));
        cout << "Coloane=";
        do {
                cin >> a.m;
                if (!(a.m >= 1 && a.n <= 100))
                        cout << "Va rog sa introduceti un numar intre 1 si 100";
        } while (!(a.m >= 1 && a.m <= 100));
        cout << "Introduceti elementele matricii linie cu linie" << endl;
        for (int i = 0;i < a.n;i++)
                for (int j = 0;j < a.m;j++)
                        do {
                                cin >> a.elem[i][j];
                                if (!(a.elem[i][j] >= 1 && a.elem[i][j] <= 10000))
                                        cout << "Va rog sa introduceti un numar intre 1 si 100";
                        } while (!(a.elem[i][j] >= 1 && a.elem[i][j] <= 10000));
        return a;
}

//Date de intrare: Matricea a si punctul de pornire p
//Date de iesire: punctul pe care va sari Spiderman. Acest punct va avea coordonatele -1,-1 daca
Spiderman nu mai are unde sa sara
Punct nextSalt(Matrice a, Punct p) {
        Punct next;
        next.i = -1;
        next.j = -1;
        int diferentaMinima = -1;
        int diferenta;
        //verific daca pot sari la nord
        //daca valoarea e -1 inseamna ca am fost deja pe cladirea respectiva si nu mai pot sari
acolo

        if (p.i > 0 && a.elem[p.i - 1][p.j] != -1 && a.elem[p.i - 1][p.j] <= a.elem[p.i][p.j]) {
                next.i = p.i - 1;
                next.j = p.j;
                diferentaMinima = a.elem[p.i][p.j] - a.elem[p.i - 1][p.j];
        }

        //verific daca pot sari la est
        if (p.j < a.m - 1 && a.elem[p.i][p.j + 1] != -1 && a.elem[p.i][p.j + 1] <=
a.elem[p.i][p.j]) {
                diferenta = a.elem[p.i][p.j] - a.elem[p.i][p.j + 1];
                if (diferentaMinima == -1 || diferenta < diferentaMinima) {
                        next.i = p.i;
                        next.j = p.j + 1;
                        diferentaMinima = diferenta;
                }
        }

        //verific daca pot sari la sud
        if (p.i < a.n - 1 && a.elem[p.i + 1][p.j] != -1 && a.elem[p.i + 1][p.j] <=
a.elem[p.i][p.j]) {
                diferenta = a.elem[p.i][p.j] - a.elem[p.i + 1][p.j];
                if (diferentaMinima == -1 || diferenta < diferentaMinima) {
                        next.i = p.i + 1;
                        next.j = p.j;
                        diferentaMinima = diferenta;
                }
        }
```

```
	//verific daca pot sari la vest
	if (p.j > 0 && a.elem[p.i][p.j - 1] != -1 && a.elem[p.i][p.j - 1] <= a.elem[p.i][p.j]) {
		diferenta = a.elem[p.i][p.j] - a.elem[p.i][p.j - 1];
		if (diferentaMinima == -1 || diferenta < diferentaMinima) {
			next.i = p.i;
			next.j = p.j - 1;
			diferentaMinima = diferenta;
		}
	}

	return next;
}

//Date de intrare: Matricea cladirilor a, Punctul de start punctStart
//Date de iesire: Matricea cladirilor a, in care s-a marcat cu -1 cladirea de pe care a plecat
Spiderman, pentru a nu mai reveni pe ea
void invalidate(Matrice& a, Punct punctStart) {
	a.elem[punctStart.i][punctStart.j] = -1;
}

//Date de intrare: Matricea cladirilor a, Punctul de start punctStart
//Date de iesire: Numarul total de sarituri pe care le poate efectua Spiderman pornind din
punctul de Start punctStart
int sarituri(Matrice a, Punct punctStart) {
	int contor = 0;
	Punct next = nextSalt(a, punctStart);
	while (next.i != -1) {
		contor++;
		invalidate(a, punctStart);
		punctStart = next;
		next = nextSalt(a, punctStart);
	}
	return contor;
}

//Date de intrare:-
//Date de iesire:punctul maxStartPunct se initializeaza cu coordonatele -1, -1 si numarul maxim
de sarituri efectuate pana in acest moment, max, se initializeaza cu -1
void initMax(int& max, Punct& maxStartPunct) {
	max = -1;
	maxStartPunct.i = -1;
	maxStartPunct.j = -1;
}

int main() {
	Matrice a = citesteMat();

	int i, j, nr, max;
	Punct start;
	Punct maxStartPunct;

	initMax(max, maxStartPunct);

	for (i = 0; i < a.n; i++)
		for (j = 0; j < a.m; j++) {
			start.i = i;
			start.j = j;
```

```
                    nr = sarituri(a, start);
                    if (nr > max) {
                        max = nr;
                        maxStartPunct.i = start.i;
                        maxStartPunct.j = start.j;
                    }
                }
        cout << "Maxim: din punctul (" << maxStartPunct.i << "," << maxStartPunct.j << ")";
        cout << " a facut " << max << " sarituri" << endl;

        return 0;
}
```

Implementare Pascal

```
{ Rezolvarea nu este optimizata pentru viteza de executie
Rezolvarea exemplifica o abordare a problemei bazata pe descompunerea in subprobleme
Programul a fost compilat cu Turbo Pascal 7 + Dos Box}
{ Tipul de data matrice}
Type
  Matrice = Record
    n, m: Integer;
    elem: Array[1..10,1..10] Of Integer;
 {la Turbo Pascal 7 i se umple stiva de lucru la transmiterea prin parametu de tip valoare a
Array[1..100,1..100]}
  End;
Type
  Punct = Record
    i,j: Integer;
  End;

{Date de intrare: -
Date de iesire: a matrice cu dimensiunile n,m - dimensiunile matricii, n,m din N,
1<=n,m<=100 si elemente intre 1 si 10000}

Procedure citesteMat(Var a : Matrice);
Var
  i,j: Integer;
Begin
  Writeln('Introduceti dimensiunile matricii');
  Writeln('Linii:=');
  Repeat
    Readln(a.n);
    If Not ((a.n >= 1) And (a.n <= 100)) Then
      Writeln('Va rog sa introduceti un numar intre 1 si 100');
  Until (a.n >= 1) And (a.n <= 100);
  Writeln('Coloane:=');
  Repeat
    Readln(a.m);
    If Not((a.m >= 1) And (a.n <= 100)) Then
      Writeln('Va rog sa introduceti un numar intre 1 si 100');
  Until (a.m >= 1) And (a.m <= 100);
  Writeln('Introduceti elementele matricii linie cu linie');
  For i := 1 To a.n Do
    For j := 1 To a.m Do
      Repeat
        Readln(a.elem[i,j]);
        If Not(a.elem[i,j] >= 1) And (a.elem[i,j] <= 10000) Then
```

```
         Writeln('Va rog sa introduceti un numar intre 1 si 10000');
      Until (a.elem[i,j] >= 1) And (a.elem[i,j] <= 10000);
End;

{Date de intrare: Matricea a si punctul de pornire p
Date de iesire: punctul pe care va sari Spiderman. Acest punct va avea coordonatele -1,-1
daca
Spiderman nu mai are unde sa sara}

Procedure nextSalt(a:Matrice; p:Punct;Var next:Punct);
Var
  diferentaMinima,diferenta: Integer;
Begin
  next.i := -1;
  next.j := -1;
  diferentaMinima := -1;

{verific daca pot sari la nord
daca valoarea e -1 inseamna ca am fost deja pe cladirea respectiva si nu mai pot sari acolo}
  If (p.i > 1) And (a.elem[p.i - 1,p.j] <> -1) And (a.elem[p.i - 1,p.j] <= a.elem[p.i,p.j]) Then
    Begin
      next.i := p.i - 1;
      next.j := p.j;
      diferentaMinima := a.elem[p.i,p.j] - a.elem[p.i - 1,p.j];
    End;
 {verific daca pot sari la est}
  If (p.j < a.m) And (a.elem[p.i,p.j + 1] <> -1) And (a.elem[p.i,p.j + 1] <= a.elem[p.i,p.j])
Then
    Begin
      diferenta := a.elem[p.i,p.j] - a.elem[p.i,p.j + 1];
      If (diferentaMinima = -1) Or (diferenta < diferentaMinima) Then
        Begin
          next.i := p.i;
          next.j := p.j + 1;
          diferentaMinima := diferenta;
        End;
    End;
 {verific daca pot sari la sud}
  If (p.i < a.n) And (a.elem[p.i + 1,p.j] <> -1) And (a.elem[p.i + 1,p.j] <= a.elem[p.i,p.j])
Then
    Begin
      diferenta := a.elem[p.i,p.j] - a.elem[p.i + 1,p.j];
      If (diferentaMinima = -1) Or (diferenta < diferentaMinima) Then
        Begin
          next.i := p.i + 1;
          next.j := p.j;
          diferentaMinima := diferenta;
        End;
    End;
 {verific daca pot sari la vest}
  If (p.j - 1 > 0) And (a.elem[p.i,p.j - 1] <> -1) And (a.elem[p.i,p.j - 1] <= a.elem[p.i,p.j])
Then
    Begin
      diferenta := a.elem[p.i,p.j] - a.elem[p.i,p.j - 1];
      If (diferentaMinima = -1) Or (diferenta < diferentaMinima) Then
        Begin
          next.i := p.i;
          next.j := p.j - 1;
```

```
          diferentaMinima := diferenta;
        End;
    End;
End;

{Date de intrare: Matricea cladirilor a, Punctul de start punctStart
Date de iesire: Matricea cladirilor a, in care s-a marcat cu -1 cladirea de pe care a plecat Spiderman,
pentru a nu mai reveni pe ea}

Procedure invalidate(Var a:Matrice; punctStart:Punct);
Begin
  a.elem[punctStart.i,punctStart.j] := -1;
End;

{Date de intrare: Matricea cladirilor a, Punctul de start punctStart
{Date de iesire: Numarul total de sarituri pe care le poate efectua Spiderman pornind din punctul de Start punctStart}

Function sarituri(a:Matrice; punctStart:Punct): Integer;
Var
  contor: Integer;
  next: Punct;
Begin
  contor := 0;
  nextSalt(a, punctStart,next);
  While (next.i <> -1) Do
    Begin
      contor := contor+1;
      invalidate(a, punctStart);
      punctStart := next;
      nextSalt(a, punctStart,next);
    End;
  sarituri := contor;
End;

{Date de intrare:-
Date de iesire:punctul maxStartPunct se initializeaza cu coordonatele -1, -1 si numarul maxim de sarituri efectuate
pana in acest moment, max, se initializeaza cu -1}

Procedure initMax(Var max:Integer; maxStartPunct:Punct);
Begin
  max := -1;
  maxStartPunct.i := -1;
  maxStartPunct.j := -1;
End;
Var
  a: Matrice;
  i, j, nr, max: Integer;
  start,maxStartPunct: Punct;
Begin
  citesteMat(a);
  initMax(max, maxStartPunct);
  For i := 1 To a.n Do
    For j := 1 To a.m Do
      Begin
        start.i := i;
```

```
        start.j := j;
        nr := sarituri(a, start);
        If nr > max Then
          Begin
            max := nr;
            maxStartPunct.i := start.i;
            maxStartPunct.j := start.j;
          End;
      End;
  Writeln('Maxim: din punctul (' , maxStartPunct.i , ',' , maxStartPunct.j , ')');
  Writeln(' a facut ' , max , ' sarituri');
  Readln;
End.
```

Planul casei

Enunt

Părinții Corinei au cumparat o casă nouă și la cumpărare au primit planul casei. Corina și-a propus ca, înainte să vadă casa, să ghicească din plan care este cea mai mare încăpere din casă.

Cerință

Scrieţi un program care determină aria maximă a unei încăperi din casă.

Date de intrare

n, m: 1<= n,m <= 100
a – matricea cu n linii si m coloane reprezentând planul casei astfel:

- valoarea 0 pentru pereți
- valoarea -1 pentru spațiu gol (unde nu e perete)

Date de ieşire

Aria maximă a unei încăperi din casă. Prin încăpere înțelegem spațiu gol înconjurat de perete (delimitat de valori 0).

Se cere să se utilizeze subprograme care să comunice între ele şi cu programul principal prin parametri. Fiecare subprogram trebuie specificat.

Intrebare suplimentara*:

Se schimba complexitatea daca adaugam in plus restrictia ca toate camerele sa fie convexe? Modificati corespunzator algoritmul.

Exemplu

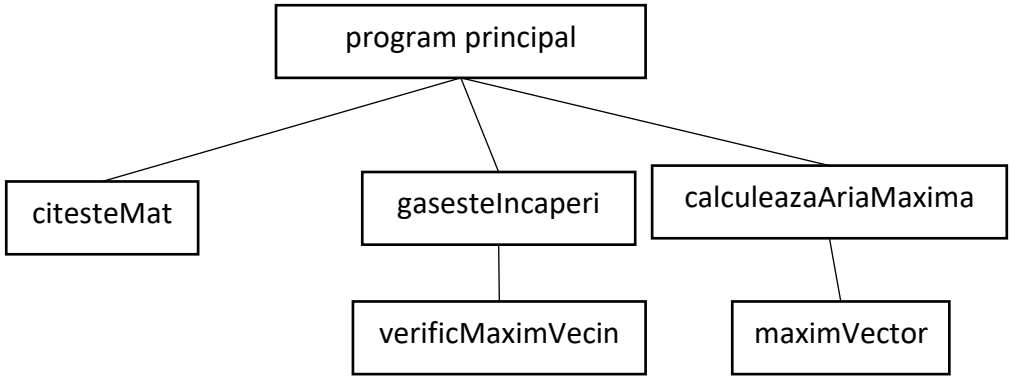
Date de intrare	Date de iesire
n = 6, m = 7	10 (aria maximă a încăperii din colțul dreapta sus)

-1 -1 0 -1 -1 0 -1 -1 -1 0 -1 -1 0 -1 -1 -1 0 -1 -1 -1 -1 0 0 0 0 0 0 0 -1 -1 0 -1 -1 -1 -1 -1 -1 0 -1 -1 -1 -1	

Pasii algoritmului principal

Algoritm matriceSpirala
    @ citeste matrice
    @ identifica incaperi
    @ calculeaza arii pentru incaperi
    @ determina aria maxima
    @ afiseaza aria maxima
Sf.Algoritm

Identificarea subalgoritmilor



Programul

Implementare Iterativă C++

```
// Rezolvarea nu este optimizata pentru viteza de executie
// Rezolvarea exemplifica o abordare a problemei bazata pe descompunerea in subprobleme
// Programul a fost compilat cu Visual Studio Community 2015
#include <iostream>
#include "Matrice.h"
using namespace std;

//verific daca valoare vreunui vecin este >0 si o returnez.
```

```
//Inseamna ca e o camera deja detectata.
int verificMaximVecin(Matrice a, int i, int j) {
	int max = -1;

	//daca am un vecin in directia respectiva si nu e perete
	if (i > 0 && a.elem[i - 1][j] != 0)
		max = a.elem[i - 1][j];

	if (i < a.n - 1 && a.elem[i + 1][j] != 0)
		if (a.elem[i + 1][j] > max)
			max = a.elem[i + 1][j];

	if (j > 0 && a.elem[i][j - 1] != 0)
		if (a.elem[i][j - 1] > max)
			max = a.elem[i][j - 1];

	if (j < a.m - 1 && a.elem[i][j + 1] != 0)
		if (a.elem[i][j + 1] > max)
			max = a.elem[i][j + 1];

	return max;
}

//returneaza true daca au mai fost schimbari
bool gasesteIncaperi(Matrice& a, int& contorIncaperi) {
	int i, j;

	int max;
	bool schimbari = false;

	for (i = 0; i < a.n; i++)
		for (j = 0; j < a.m; j++) {
			if (a.elem[i][j] != 0) {
				max = verificMaximVecin(a, i, j);
				//daca minimul e -1, atunci e o incapere inca nedescoperita
				if (max == -1) {
					contorIncaperi++;
					a.elem[i][j] = contorIncaperi;
					schimbari = true;
				}
				//altfel, e o camera detectata deja si completez cu numarul ei
				//iar daca cumva are mai multe numere, il aleg pe cel mai mare
				else
					if (a.elem[i][j] != max) {
						a.elem[i][j] = max;
						schimbari = true;
					}
			}
		}
	return schimbari;
}

//returneaza maximul de pe primele l pozitii din vectorul v
int maximVector(int v[], int l) {
	int max = 0;
	for (int i = 0; i < l; i++)
```

```
			if (v[i] > max)
				max = v[i];
	return max;
}

int calculeazaAriaMaxima(Matrice a, int contorIncaperi) {
	int ariiCamere[200];
	int i, j;

	//initializez toate ariile cu 0;
	for (i = 0; i < contorIncaperi; i++)
		ariiCamere[i] = 0;

	for (i = 0; i < a.n; i++)
		for (j = 0; j < a.m; j++) {
			int idCamera = a.elem[i][j];
			//daca e Camera si nu perete ii cresc cu 1 aria
			if (idCamera > 0)
				ariiCamere[idCamera - 1]++;
		}
	return maximVector(ariiCamere, contorIncaperi);
}

int main() {
	Matrice a = citire("3.in");
	afisare(a);

	bool schimbari = true;
	int contorIncaperi = 0;

	//Cat timp mai sunt schimbari nu putem fi siguri ca o camera e umpluta cu acelasi
	//numar, se poate sa nu fi fost detectata din prima parcurgere ca o singura incapere.
	//De aceea parcurgem de mai multe ori si daca detectam numere diferite in aceeasi
//incapere le suprascriem cu cel mai mare dintre cele intalnite
	while (schimbari)
		schimbari = gasesteIncaperi(a, contorIncaperi);

	int aria = calculeazaAriaMaxima(a, contorIncaperi);
	cout << "Aria maxima a unei incaperi este: " << aria << endl;
	return 0;
}
```

Implementare iterativă Pascal

```
{ Rezolvarea nu este optimizata pentru viteza de executie
Rezolvarea exemplifica o abordare a problemei bazata pe descompunerea in subprobleme
Programul a fost compilat cu Turbo Pascal 7 + Dos Box}
{ Tipul de data matrice}
Type
  Matrice = Record
    n, m: Integer;
    elem: Array[1..10,1..10] Of Integer;
 {la Turbo Pascal 7 i se umple stiva de lucru la transmiterea prin parametu de tip valoare a
Array[1..100,1..100]}
  End;
```

```
Type Vector = Array[1..100] of Integer;

{Date de intrare: -
Date de iesire: a matrice cu dimensiunile n,m - dimensiunile matricii, n,m din N, 1<=n,m<=100 si elemente intre 1 si 10000}
Procedure citesteMat(Var a : Matrice);
Var
  i,j: Integer;
Begin
  Writeln('Introduceti dimensiunile matricii');
  Writeln('Linii:=');
  Repeat
    Readln(a.n);
    If Not ((a.n >= 1) And (a.n <= 100)) Then
      Writeln('Va rog sa introduceti un numar intre 1 si 100');
  Until (a.n >= 1) And (a.n <= 100);
  Writeln('Coloane:=');
  Repeat
    Readln(a.m);
    If Not((a.m >= 1) And (a.n <= 100)) Then
      Writeln('Va rog sa introduceti un numar intre 1 si 100');
  Until (a.m >= 1) And (a.m <= 100);
  Writeln('Introduceti elementele matricii linie cu linie');
  For i := 1 To a.n Do
    For j := 1 To a.m Do
      Repeat
        Readln(a.elem[i,j]);
        If (a.elem[i,j] <> -1) And (a.elem[i,j] <> 0) Then
          Writeln('Va rog sa introduceti -1 (nu e perete) sau 0 (e perete)');
      Until (a.elem[i,j] = -1) Or (a.elem[i,j] = 0);
End;

{verific daca valoarea vreunui vecin este >0 si o returnez.
Inseamna ca e o camera deja detectata.
Date de intrare: Matricea cu planul casei a si i,j coordonatele punctului de analizat
Date de iesire: valoarea celui mai mare vecin al punctului analizat}
Function verificMaximVecin(a:Matrice;i,j:Integer):Integer;
Var max:integer;
begin
      max := -1;

      {daca am un vecin in directia respectiva (sus) si nu e perete  }
      if (i > 1) and (a.elem[i - 1,j] <> 0) then
            max := a.elem[i - 1,j];

      {daca am un vecin in directia respectiva (jos) si nu e perete}
      if (i < a.n) and (a.elem[i + 1,j] <> 0) then
            if a.elem[i + 1,j] > max then
                  max := a.elem[i + 1,j];
```

```
	{daca am un vecin in directia respectiva (stanga) si nu e perete}
	if (j > 1) and (a.elem[i,j - 1] <> 0) then
		if a.elem[i,j - 1] > max then
			max := a.elem[i,j - 1];

	{daca am un vecin in directia respectiva (dreapta) si nu e perete}
	if (j < a.m) and (a.elem[i,j + 1] <> 0) then
		if a.elem[i,j + 1] > max then
			max := a.elem[i,j + 1];

	verificMaximVecin:=max;
end;

{returneaza true daca au mai fost schimbari
Date de intrare: Matricea cu planul casei a, cu elemente: -1 (zona nedetectata), 0 -perete,
k din [1,contor incaperi]
- care indica ca punctul curent apartine de camera k
si contorIncaperi := nr. de incaperi identificate deja
Date de iesire: false daca nu s-au mai modificat incaperile detectate,
true daca s-au mai modificat incaperile detectate, a - planul actualizat si contorIncaperi
actualizat}
Function gasesteIncaperi(Var a:Matrice; Var contorIncaperi:Integer):Boolean;
Var i,j,max:integer;
	schimbari:Boolean;
begin
	schimbari := false;

	for i := 1 to a.n do
		for j := 1 to a.m Do
		 begin
			if a.elem[i,j] <> 0 then
			 begin
				max := verificMaximVecin(a, i, j);
				{daca minimul e -1 si a.elem[i,j]=-1, atunci e o incapere inca
				nedescoperita si se va marca cu un numar nou}
				if (a.elem[i,j]=-1) and (max = -1) then
				begin
					contorIncaperi:=contorIncaperi+1;
					a.elem[i,j] := contorIncaperi;
					schimbari := true;
				end
				{altfel, e o camera detectata deja si completez cu numarul ei
				iar daca cumva are mai multe numere, il aleg pe cel mai mare
				si punctul analizat se va lipi de camera cu cel mai mare indice
				astfel, incet, incet, unele camere sa fie absorbite de altele cu
				indice mai mare}
				else
					if max>a.elem[i,j] then
					begin
						a.elem[i,j] := max;
						schimbari := true;
```

```
                                    end;
                        end;
                end;
        gasesteIncaperi:=schimbari;
end;

{returneaza maximul de pe primele l pozitii din vectorul v
Date de intrare: v vector de nr. intregi, l- nr. de elemente ale vectorului
Date de iesire: cea mai mare valoare din vector}
Function maximVector(v:Vector; l:integer):Integer;
Var i,max:integer;
begin
        max := 0;
        for i := 1 to l do
                if v[i] > max then
                        max := v[i];
        maximVector:=max;
end;

{Date de intrare: Matricea cu planul casei a cu elemente: 0 -perete, k din [1,contor
incaperi]
- care indica ca punctul curent apartine de camera k
si contorIncaperi := cel mai amre indice al unei incaperi din casa
Date de iesire: cea mai mare arie a unei incaperi din plan}
Function calculeazaAriaMaxima(a:Matrice; contorIncaperi:Integer): Integer;
Var i,j,idCamera:integer;
        ariiCamere:Vector;
begin
        {initializez toate ariile cu 0;}
        for i := 1 to contorIncaperi do
                ariiCamere[i] := 0;

        for i := 1 to a.n do
                for j := 1 to a.m Do
                begin
                        idCamera := a.elem[i,j];
                        {daca e Camera si nu perete ii cresc cu 1 aria}
                        if idCamera > 0 then
                                ariiCamere[idCamera - 1]:=ariiCamere[idCamera - 1]+1;
                end;
        calculeazaAriaMaxima:= maximVector(ariiCamere, contorIncaperi);
end;

Var schimbari:Boolean;
        contorIncaperi,aria:Integer;
        a: Matrice;
begin
        citesteMat(a);

        schimbari := true;
```

```
        contorIncaperi := 0;

        {Cat timp mai sunt schimbari nu putem fi siguri ca o camera e umpluta cu acelasi
        numar, se poate sa nu fi fost detectata din prima parcurgere ca o singura incapere.
        De aceea parcurgem de mai multe ori si daca detectam numere diferite in aceeasi
        incapere le suprascriem cu cel mai mare dintre cele intalnite}
        while schimbari=True do
                    schimbari := gasesteIncaperi(a, contorIncaperi);

        aria := calculeazaAriaMaxima(a, contorIncaperi);
        Writeln('Aria maxima a unei incaperi este: ',aria);
end.
```

Implementare recursivă Pascal

```
// definim tipul de date matrice
type
  matrice = record
  elem:array[0..100,0..100] of integer;
  n,m : integer;
end;

// citim datele de intrare
function readfile(s : string) : matrice;
var f:text; i,j,val:integer;
m : matrice;
begin
assign(f,'plancasa.in');
reset(f);
read(f,m.n);
read(f,m.m);
for i:=0 to m.n-1 do
  for j:=0 to m.m-1 do
    read(f,m.elem[i][j]);
close(f);
readFile := m;
end;

// functia recursiva de umplere
function umplere(var m : matrice; l,c:integer) : integer;
begin
  if (l<0) or (l>=m.n ) or (c<0) or (c>=m.m) then
    umplere := 0
    else begin
    if m.elem[l][c] <> 0 then
      umplere :=0
      else begin
      m.elem[l][c] := 1;
```

```
      umplere := 1 + umplere(m,l-1,c) + umplere(m,l+1,c) + umplere(m,l,c-1)
      + umplere(m,l,c+1);
      end;
    end;
end;

// functia unde determinam dimensiunea camerei maxime
function cameraMaxima(casa:matrice) : integer;
var l,c,v,cameraMax : integer;
begin
  cameraMax := 0;
  for l:=0 to casa.n-1 do
    for c:=0 to casa.m-1 do
    begin
     v := umplere(casa,l,c);
     if v>cameraMax then cameraMax := v;
    end;
  cameraMaxima := cameraMax;
end;

var m : matrice;
begin
m:=readfile('plancasa.in');
writeln('Camera cea mai mare are dimensiunea ',cameraMaxima(m));
end.
```

Implementare recursivă C++

-- matrice.h --

```
const int MAX = 200;

struct Matrice {
      int m;
      int n;
      int elem[MAX][MAX];
};

void afisare(Matrice m);
Matrice citire(char*);
```

-- matrice.cpp –

```
#include "Matrice.h"
#include <iostream>
#include <iomanip>
using namespace std;

void afisare(Matrice m) {
```

```
	cout << "linii=" << m.n << ", coloane=" << m.m << endl;
	for (int i = 0; i < m.n; i++) {
		for (int j = 0; j < m.m; j++) {
			cout << setw(4) << m.elem[i][j];
		}
		cout << endl;
	}
}

Matrice citire(char* fisier) {
	FILE *fin;
	Matrice m;

	fopen_s(&fin, fisier, "r");
	fscanf_s(fin, "%d", &m.n);
	fscanf_s(fin, "%d", &m.m);

	int v;
	for (int i = 0; i < m.n; i++) {
		for (int j = 0; j < m.m; j++) {
			fscanf_s(fin, "%d ", &m.elem[i][j]);
		}
	}
	return m;
}

#include <iostream>
#include "Matrice.h"
using namespace std;

// Umplem casutele legate ale matricii m cu valoarea '1', incepand cu pozitia (l,c)
int umplere(Matrice& m, int l, int c) {
	// am iesit din matrice
	if (l < 0 || l >= m.n|| c < 0 || c >= m.m) return 0;

	// am dat de un perete, sau o camera deja detectata
	if (m.elem[l][c] != 0) {
		return 0;
	}

	// marchez locatia, apoi verific recursiv vecinii
	m.elem[l][c] = 1;
	return 1 + umplere(m, l - 1, c) + umplere(m, l + 1, c) + umplere(m, l, c - 1) +
umplere(m, l, c + 1);
}

int cameraMaxima(Matrice casa) {
	int cameraMaxima = 0;
	for (int l=0;l<casa.n;l++)
		for (int c = 0; c < casa.m; c++) {
			int v = umplere(casa, l, c);
			if (v > cameraMaxima) {
				cameraMaxima = v;
			}
		}
	return cameraMaxima;
```

```
}

void main() {
	Matrice casa = citire("3a.in");
	cout << "Dimensiunea camerei maxime: " << cameraMaxima(casa);
}
```

Ferma[1]

Enunț

Un fermier deține o fermă de formă dreptunghiulară cu lungimea **m** metri și lățimea **n** metri. Respectând principiul rotației culturilor, fermierul și-a realizat un plan pentru semănarea culturilor în noul an. Astfel ,el a desenat un dreptunghi pe care l-a împărțit în **m** * **n** celule, fiecare corespunzând unui metru pătrat, și a colorat în culori diferite zonele care corespund unor culturi diferite. O cultură poate fi semănată pe mai multe parcele. Două celule care au o latură comună aparțin aceleiași parcele dacă au aceeași culoare (sunt însămânțate cu aceeași cultură). Fermierul are posibilitatea să irige o sigură parcelă și dorește să aleagă parcela cu cea mai mare suprafață. Nefiind mulțumit de suprafața rezultată, s-a întrebat dacă ar putea schimba cultura de pe o singură celulă, astfel încât să obțină o parcelă de suprafață mai mare.

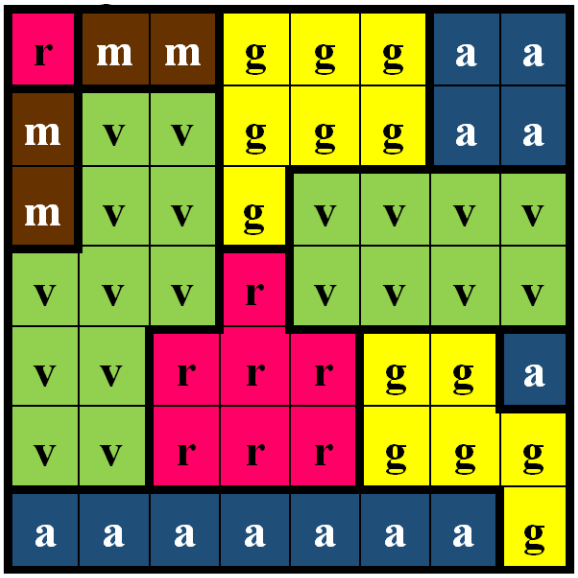


Figura 1 - Exemplu culturi ferma

Cerință

[1] Enunț adaptat pornind de la OJI 2014, 1 martie.

Dându-se dimensiunile fermei și pentru fiecare celulă culoarea corespunzătoare culturii semănate, determinați dimensiunea maximă a parcelei ce poate fi obținută prin schimbarea tipului de cultură într-o singură parcelă.

Date de intrare

Fișierul de intrare **ferma.in** va conține:

- pe prima linie un număr natural **v** (**1 ≤ v ≤ 2**) indicând varianta cerinței de rezolvare;
- pe a doua linie două numere naturale **m** şi **n** separate printr-un spațiu, cu semnificația din enunț;
- pe fiecare dintre următoarele **m** linii se găsesc câte **n** caractere (litere mici), reprezentând codurile culturilor ce vor fi semănate pe cele **n** celule corespunzătoare fiecărei linii.

Date de ieşire

Dimensiunea parcelei maxime care se poate obține prin semănarea altei culturi

Restricţii şi precizări

- 2 ≤ m ≤ 400
- 2 ≤ n ≤ 400
- Numărul de culturi distincte este cel puţin 2 şi cel mult 26.

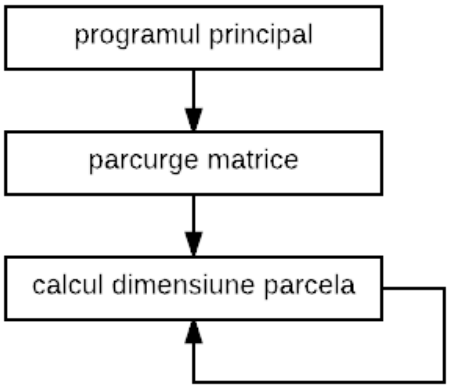
Exemplu

ferma.in	Explicații
1 7 8 rmmgggaa mvvgggaa mvvgvvvv vvvrvvvv vvrrrgga vvrrrggg aaaaaaag	Schimbând în **verde** (v) culoarea celulei de pe linia **3** şi coloana **4,** se obține o parcelă cu suprafața 11+8+1=20 (se unesc parcelele cu numărul 6 respectiv 8).

Pasii algoritmului principal

Algoritm Ferma
    @ citeste matrice
    @ parcurge matricea
    @ schimbare cultura pentru fiecare casuta
    @ calculeaza aria obtinuta
    @ afiseaza aria maxima
Sf.Algoritm

Identificarea subalgoritmilor



Programul

```
// Rezolvarea nu este optimizata pentru viteza de executie
// Rezolvarea exemplifica o abordare a problemei bazata pe descompunerea in subprobleme
// Programul a fost compilat cu Visual Studio Community 2015
#include <iostream>
struct Parcela {
	int linie;
	int coloana;
};

struct Stiva {
	int varf;
	Parcela parcele[15000];
};

struct Ferma {
	int linii;
	int coloane;
	char celule[402][402];
};

bool egal(Parcela& p1, Parcela& p2) {
	return p1.coloana == p2.coloana && p1.linie == p2.linie;
}

// verificam daca stiva data contine parcela
bool contine(Stiva& stiva, Parcela& p) {
	for (int i = 0; i < stiva.varf; i++)
		if (egal(p, stiva.parcele[i]) == true)
			return true;
	return false;
}

// adaugarea unei noi parcele in stiva
void push(Stiva& stiva, Parcela& p) {
```

```
	stiva.parcele[stiva.varf++] = p;
}

// citirea datelor despre ferma
Ferma citire(char* fisier) {
	FILE *fin;
	Ferma m;
	fopen_s(&fin, fisier, "r");
	fscanf_s(fin, "%d %d\n", &m.linii, &m.coloane);
	for (int i = 0; i < m.linii; i++)
		fgets(m.celule[i], 400, fin);
	return m;
}

// apelul recursiv pentru calculul dimensiunii parcelei, incepand cu pozitia (l,c)
// valoarea 'v' retine cultura pe care o cautam
// celulele memorate le pastram intr-o stiva
int dimParcelaRec(Ferma& f, int l, int c, char v, Stiva& stiva) {
	if (l < 0 || l >= f.linii || c < 0 || c >= f.coloane) return 0;
	if (f.celule[l][c] != v) {
		return 0;
	}

	Parcela p;
	p.linie = l;
	p.coloana = c;

	// daca stiva memoreaza parcela curenta, nu o mai numaram
	if (contine(stiva, p))
		return 0;
	push(stiva, p);

	// 1 pentru celula curenta + valoarea apelului recursiv pentru celulele adiacente
	return 1 + dimParcelaRec(f, l - 1, c, v, stiva) + dimParcelaRec(f, l + 1, c, v, stiva)
+ dimParcelaRec(f, l, c - 1, v, stiva) + dimParcelaRec(f, l, c + 1, v, stiva);
}

// calculam dimensiunea maxima a parcelei de pe pozitia (l,c)
int dimParcela(Ferma& f, int l, int c) {
	Stiva s;
	s.varf = 0;
	return dimParcelaRec(f, l, c, f.celule[l][c], s);
}

int parcurgere(Ferma& ferma) {
	int max = -1;
	//parcurgem fiecare celula a fermei
	for (int i = 0; i < ferma.linii; i++)
		for (int j = 0; j < ferma.coloane; j++)
		{
			//retinem cultura originala a celulei curente
			char original = ferma.celule[i][j];
			for (char c = 'a'; c <= 'z'; c++) {
				//incercam sa inlocuim cultura existenta cu alta
				//de fiecare data calculam dimensiunea parcelei obtinute
				ferma.celule[i][j] = c;
```

```
                        int parcela = dimParcela(ferma, i, j);
                        if (parcela > max) {
                            max = parcela;
                        }
                    }
                    ferma.celule[i][j] = original;
                }
        return max;
}

void main() {
        Ferma ferma = citire("5a.in");
        std::cout << "Dimensiunea parcelei maxime care se poate obtine: " << parcurgere(ferma)
<< std::endl;
}
```

Suma Matrici Rare

Enunț

O matrice A(n,m) cu elemente întregi se numește rară dacă majoritatea elementelor sale sunt egale cu 0. O matrice rară A(n,m) având k elemente nenule poate fi memorată folosind un șir X conținând k triplete de forma (linie, coloană, valoare) corespunzătoare valorilor nenule ale matricei – fără a folosi un tablou bidimensional. Elementele șirului X se memorează în ordine lexicografică (crescătoare) după (linie, coloană).

Să se scrie un program care citește de la tastatură valorile n,m și două matrice rare A(n,m) și B(n,m), calculează sub forma unei matrice rare suma C(n,m) a celor două matrice A și B și afișează sub forma unui tablou bidimensional matricea C(n,m).

Citirea unei matrice se va face prin citirea numărului n de linii, numărului m de coloane și citirea repetată a unor triplete (linie, coloană, valoare) – corespunzătoare valorilor nenule din matrice, până la citirea tripletului (-1,-1,-1). În cazul citirii mai multor triplete cu aceeași linie și coloană, se ia în considerare doar primul triplet citit.

Exemplu

De exemplu, pentru n=m=3, matricea A

0	**5**	**2**
0	**2**	**0**
2	**0**	**3**

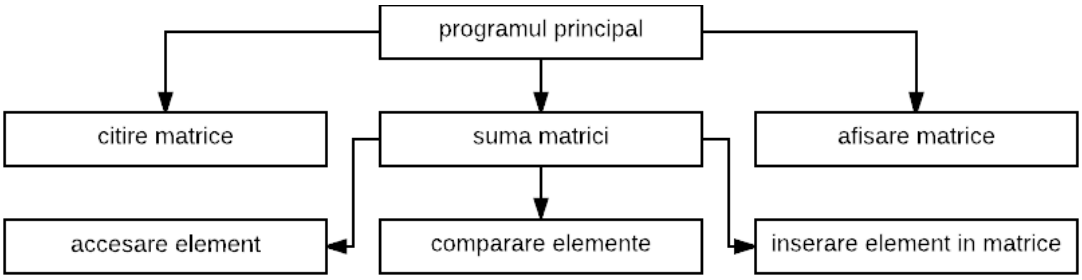
Figura 2 - Exemplu matrice rara

se va memora sub forma șirului X=((1,2,5), (1,3,2), (2,2,2), (3,1,2), (3,3,3))

Pasii algorimului principal

Algoritm Suma Matrici
    @ citire matrici
    @ determinare suma
    @ afisare matrice suma
Sf.Algoritm

Identificarea subalgoritmilor



Programul

Implementare C++

```
// Rezolvarea nu este optimizata pentru viteza de executie
// Rezolvarea exemplifica o abordare a problemei bazata pe descompunerea in subprobleme
// Programul a fost compilat cu Visual Studio Community 2015
#include <iostream>
#include <iomanip>
#include <stdio.h>

using namespace std;

#define MAX_N 100

struct Triplet {
	int linie, coloana, valoare;
};

struct MatriceRara {
	int nrLinii, nrColoane;
	int nrElemente;
	Triplet elemente[MAX_N];
};

int compara(Triplet t1, Triplet t2)
{
	if (t1.linie < t2.linie)
		return 1;
	if (t1.linie == t2.linie && t1.coloana < t2.coloana)
		return 1;
```

```
	return 0;
}

// Inserarea unui nou triplet in matricea rara
void inserare(MatriceRara &m, Triplet t)
{
	// daca matricea nu are elemente, noul triplet este primul
	if (m.nrElemente == 0)
	{
		m.elemente[0] = t;
		m.nrElemente = 1;
		return;
	}

	int i = m.nrElemente;
	while (i > 0 && compara(t, m.elemente[i - 1])) {
		m.elemente[i] = m.elemente[i - 1];
		i--;
	}
	// se insereaza tripletul
	m.elemente[i] = t;
	m.nrElemente++;
}

int element(MatriceRara &m, int linie, int coloana)
{
	for (int k = 0; k < m.nrElemente; k++)
		if (m.elemente[k].linie == linie && m.elemente[k].coloana == coloana)
			return m.elemente[k].valoare;
	// elementul implicit al unei matrici rare este 0
	return 0;
}

// se vor citi doar valorile nenule ale matricii
// subprogramul se terminla la citirea tripletului (-1,-1,-1)
void citire(MatriceRara &m)
{
	// presupunem datele valide
	std::cout << "Numarul de linii = ";
	std::cin >> m.nrLinii;
	std::cout << "Numarul de coloane = ";
	std::cin >> m.nrColoane;
	m.nrElemente = 0;

	Triplet t;
	std::cin >> t.linie >> t.coloana >> t.valoare;

	while (t.linie != -1 || t.coloana != -1 || t.valoare != -1) {
		if (element(m, t.linie, t.coloana) == 0)
			inserare(m, t);
		std::cin >> t.linie >> t.coloana >> t.valoare;
	}
}

// la calcularea sumei utilizam faptul ca sirul de triplete este ordonat 'lexicografic'
MatriceRara suma(MatriceRara& a, MatriceRara& b)
```

```
{
	MatriceRara c;
	// presupunem ca numarul de linii si coloane ale matricilor coincide
	c.nrColoane = a.nrColoane;
	c.nrLinii = a.nrLinii;
	c.nrElemente = 0;

	//indicii pentru tripleta curenta in cadrul matricilor a si b
	int i = 0;
	int j = 0;
	while (i < a.nrElemente && j < b.nrElemente) {
		if (compara(a.elemente[i], b.elemente[j]))
			c.elemente[c.nrElemente++] = a.elemente[i++];
		else
			if (compara(b.elemente[j], a.elemente[i]))
				c.elemente[c.nrElemente++] = b.elemente[j++];
			else {
				int s = a.elemente[i].valoare + b.elemente[j].valoare;
				//adaugam doar valorile nenule
				//este posibil ca sumarea sa rezulte intr-o valoare nula
				if (s != 0) {
					c.elemente[c.nrElemente] = a.elemente[i];
					c.elemente[c.nrElemente++].valoare = s;
				}
				i++;
				j++;
			}
	}
	// se copiaza tripletele ramase
	while (i < a.nrElemente)
		c.elemente[c.nrElemente++] = a.elemente[i++];

	while (j < b.nrElemente)
		c.elemente[c.nrElemente++] = b.elemente[j++];

	return c;
}


void tiparire(MatriceRara &m)
{
	cout << "Linii=" << m.nrLinii << ", Coloane=" << m.nrColoane << endl;
	for (int i = 1; i <= m.nrLinii; i++) {
		for (int j = 1; j <= m.nrColoane; j++)
			cout << setw(4) << element(m, i, j) << " ";
		cout << endl;
	}
}

void main()
{
	MatriceRara a, b;
	citire(a);
	citire(b);
	tiparire(a);
	tiparire(b);
```

```
        MatriceRara c = suma(a, b);
        tiparire(c);
}
```

Implementare Pascal

```
type triplet = record
  linie, coloana, valoare : integer;
end;

type matricerara = record
  nrLinii,nrColoane,nrElemente : integer;
  elemente : array[0..100] of triplet;
end;

function compara(t1,t2:triplet) : integer;
begin
 if (t1.linie < t2.linie) then
   compara := 1
 else if (t1.linie = t2.linie) and (t1.coloana<t2.coloana) then
  compara := 1
 else compara := 0;
end;

// inserarea unui nou triplet in matricea rara
procedure inserare(m:matricerara;t:triplet);
var i:integer;
begin
// daca matricea nu are elemente, noul triplet este primul
if (m.nrElemente = 0) then
  begin
  m.elemente[0]:=t;
  m.nrElemente:=1;
  end;
  i := m.nrElemente;
  while (i>0) and (compara(t,m.elemente[i-1])=1) do
  begin
    m.elemente[i]:=m.elemente[i-1];
    dec(i);
  end;
  // se insereaza tripletul
  m.elemente[i] := t;
  inc(m.nrElemente);
end;

function element(m:matricerara;linie,coloana:integer) : integer;
var aux,k:integer;
begin
```

```
  aux:=0;
  for k:=0 to m.nrElemente-1 do
    if (m.elemente[k].linie = linie) and (m.elemente[k].coloana=coloana) then
      aux := m.elemente[k].valoare;
  element := aux;
end;

procedure citire(var m:matricerara);
var t:triplet;
begin
  // presupunem datele de intrare valide
  write('Numarul de linii=');
  readln(m.nrLinii);
  write('Numarul de coloane=');
  readln(m.nrColoane);
  m.nrElemente := 0;

  write('triplet>');
  readln(t.linie,t.coloana,t.valoare);
  while (t.linie<>-1) or (t.coloana<>-1) or (t.valoare<>-1) do
  begin
    if element(m,t.linie,t.coloana) = 0 then
      inserare(m,t);

  write('triplet>');
  readln(t.linie,t.coloana,t.valoare);
  end;
end;

// la calcularea sumei utilizam faptul ca sirul de triplete
// este ordonat 'lexicografic'
function suma(a,b:matricerara) : matricerara;
var c:matricerara; s,i,j:integer;
begin
  // presupunem ca numarul de linii si coloane al matricilor coincide
  c.nrColoane := a.nrColoane;
  c.nrLinii := a.nrLinii;
  c.nrElemente := 0;

  // indicii pentru tripleta curenta in cadrul matricilor a si b
  i := 0;
  j := 0;
  while (i<a.nrElemente) and (j<b.nrElemente) do
  begin
    if compara(a.elemente[i],b.elemente[j])<>0 then
      begin
      c.elemente[c.nrElemente] := a.elemente[i];
```

```
      inc(c.nrElemente); inc (i);
      end else
      if compara(b.elemente[j],a.elemente[i])<>0 then
      begin
      c.elemente[c.nrElemente] := b.elemente[j];
      inc(c.nrElemente); inc(j);
      end else
        begin
        s := a.elemente[i].valoare + b.elemente[j].valoare;
        // adaugam doar valorile nenule
        if s<>0 then
          begin
            c.elemente[c.nrElemente] := a.elemente[i];
            c.elemente[c.nrElemente].valoare := s;
            inc(c.nrElemente);
          end;
        inc(i); inc(j);
        end;
  end;

  // se copiaza valorile ramase
  while i<a.nrElemente do
    begin
    c.elemente[c.nrElemente] := a.elemente[i];
    inc(c.nrElemente); inc(i);
    end;
  while j<b.nrElemente do
    begin
    c.elemente[c.nrElemente] := b.elemente[j];
    inc(c.nrElemente); inc(j);
    end;
suma := c;
end;

procedure tiparire(m:matricerara);
var i,j:integer;
begin
  writeln('Linii=',m.nrLinii,', Coloane=',m.nrColoane);
  for i:=1 to m.nrLinii-1 do
  begin
    for j:=1 to m.nrColoane-1 do
      write(element(m,i,j),' ');
  writeln();
  end;
end;
```

```
var a,b,c : matricerara;
begin
  citire(a);
  citire(b);
  tiparire(a);
  tiparire(b);
  c := suma(a,b);
  tiparire(c);
end.
```

Elicoptere[2]

Enunț

Un teren de fotbal este folosit pentru aterizarea elicopterelor. Gazonul de pe stadion este parcelat în pătrăţele de aceeaşi dimensiune, cu laturile paralele cu marginile terenului. Liniile cu pătrăţele de gazon sunt numerotate de sus în jos cu numerele 1, 2, ..., **m**, iar coloanele cu pătrăţele de gazon sunt numerotate de la stânga la dreapta cu numerele 1, 2, ..., **n**. Din cauza tipului diferit de iarbă se ştie care dintre pătrăţele de gazon sunt afectate sau nu de umbră. Acest lucru este precizat printr-un tablou bidimensional **a** cu **m** linii şi **n** coloane, cu elemente 0 şi 1 (a_{ij}= 0 înseamnă că pătrăţelul aflat pe linia **i** şi coloana **j** este afectat de umbră, iar a_{ij}= 1 înseamnă că pătrăţelul aflat pe linia **i** şi coloana **j** nu este afectat de umbră). Fiecare elicopter are 3 roţi pe care se sprijină. Roţile fiecărui elicopter determină un triunghi dreptunghic isoscel. Elicopterele aterizează, astfel încât triunghiurile formate să fie cu catetele paralele cu marginile terenului. În exemplul următor avem patru elicoptere.

1	1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	0
0	0	1	0	1	1	1	0	0
1	1	1	0	1	1	0	1	1
0	0	1	1	1	1	0	1	1
1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	0	0	1

Figura 3 - Exemplu problema elicoptere

Pentru a preciza poziţia unui elicopter pe teren este suficient să cunoaştem linia şi coloana vârfurilor ipotenuzei şi poziţia vârfului deasupra (codificată prin 1) sau dedesubtul ipotenuzei (codificată prin -1). Pentru exemplu, elicopterul din stânga sus este dat prin (1,1), (3,3) şi -1, cel din dreapta sus prin (1,9), (5,5) şi 1, cel din stânga jos

[2] Enunț adaptat pornind de la OJI 2012

prin (5,1), (6,2) şi 1, iar cel din dreapta jos prin (5,9), (6,8) şi 1. Un elicopter se consideră că a aterizat *greşit*, dacă triunghiul format sub el (definit mai sus) are mai mult de jumătate din pătrăţele afectate de umbră.

Cerinţă

Administratorul terenului de fotbal doreşte să determine elicopterele care au aterizat ***greșit***.

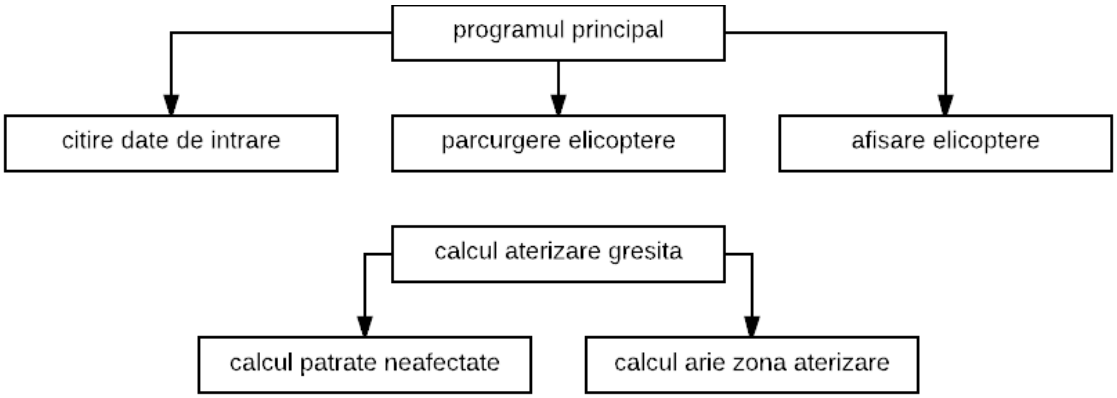
Exemplu

În exemplul de mai sus, elicopterele date prin coordonatele ((1,1),(3,3),-1) și ((5,1),(6,2),1) au aterizat greșit.

Pasii algorimului principal

Algoritm Elicoptere
    @ citire date intrare (teren de aterizare și elicoptere)
    @ determinare elicoptere aterizate greșit
    @ afișare rezultat
Sf.Algoritm

Identificarea subalgoritmilor



Programul

```
// Rezolvarea nu este optimizata pentru viteza de executie
// Rezolvarea exemplifica o abordare a problemei bazata pe descompunerea in subprobleme
// Programul a fost compilat cu Visual Studio Community 2015
#include <iostream>
#include "Matrice.h"

using namespace std;

struct Punct {
	int x, y;
};
```

```
struct Elicopter {
	Punct p1, p2;
	int varf;
};

// aria pe care o ocupa un elicopter o calculam pe baza numarului de patratele al ipotenuzei
int arie(Elicopter e) {
	int diff = abs(e.p1.y - e.p2.y) + 1;
	return diff*(diff + 1) / 2;
}

// calculam cate patrate sunt neafectate de acest elicopter
int patrateNeafectate(Matrice& m, Elicopter e) {
	int suma = 0;
	int aux = 0;
	// elicopterul are varful indreptat in sus si 'stanga'
	if (e.varf == 1 && (e.p1.y > e.p2.y)) {
		for (int linie = e.p1.x; linie <= e.p2.x; linie++) {
			for (int coloana = e.p1.y - aux; coloana >= e.p2.y; coloana--) {
				suma += m.elem[linie - 1][coloana - 1];
			}
			aux += 1;
		}
	}

	// elicopterul are varful indreptat in sus si 'dreapta'
	if (e.varf == 1 && (e.p1.y < e.p2.y)) {
		for (int linie = e.p1.x; linie <= e.p2.x; linie++) {
			for (int coloana = e.p1.y + aux; coloana <= e.p2.y; coloana++) {
				suma += m.elem[linie - 1][coloana - 1];
			}
			aux += 1;
		}
	}

	// elicopterul are varful indreptat in jos si 'stanga'
	if (e.varf == -1 && (e.p1.y < e.p2.y)) {
		for (int linie = e.p1.x; linie <= e.p2.x; linie++) {
			for (int coloana = e.p1.y; coloana <= e.p1.y + aux; coloana++) {
				suma += m.elem[linie - 1][coloana - 1];
			}
			aux += 1;
		}
	}

	// elicopterul are varful indreptat in jos si 'dreapta'
	if (e.varf == -1 && (e.p1.y > e.p2.y)) {
		for (int linie = e.p1.x; linie <= e.p2.x; linie++) {
			for (int coloana = e.p1.y; coloana >= e.p1.y - aux; coloana--) {
				suma += m.elem[linie - 1][coloana - 1];
				std::cout << linie << "|" << coloana << "|" << suma << std::endl;
			}
			aux += 1;
		}
	}
	return suma;
```

```
}

// citirea datelor matricii si a elicopterelor
void citireDate(char* fisier, Matrice& m, Elicopter eli[], int& nrEli) {
	FILE *fin;

	// citim matricea
	fopen_s(&fin, fisier, "r");
	fscanf_s(fin, "%d", &m.n);
	fscanf_s(fin, "%d", &m.m);

	int v;
	for (int i = 0; i < m.n; i++) {
		for (int j = 0; j < m.m; j++) {
			fscanf_s(fin, "%d ", &m.elem[i][j]);
		}
	}

	//citim elicopterele
	fscanf_s(fin, "%d", &nrEli);
	for (int i = 0; i < nrEli; i++) {
		Elicopter e;
		fscanf_s(fin, "%d %d %d %d %d ", &e.p1.x, &e.p1.y, &e.p2.x, &e.p2.y, &e.varf);

		//ordonam varfurile ipotenuzei dupa axa Ox
		if (e.p1.x > e.p2.x) {
			Punct aux = e.p1;
			e.p1 = e.p2;
			e.p2 = aux;
		}
		eli[i] = e;
	}
}

// aterizarea este corecta daca numarul de patrate neafectate este mai mult de
// jumatate din umbra elicopterului
bool aterizatOk(Matrice& m, Elicopter e) {
	return patrateNeafectate(m, e) * 2 > arie(e);
}

// tiparim rezultatul
void tiparire(Elicopter eli[], int nrEli) {
	cout << "Elicopterele aterizate gresit sunt:" << endl;
	for (int i = 0; i < nrEli; i++) {
		cout << "(" << eli[i].p1.x << "," << eli[i].p1.y << "),(" << eli[i].p2.x << ","
<< eli[i].p2.y << "), varful in " << (eli[i].varf == 1 ? "sus" : "jos") << endl;
	}
}

// verificam aterizarea fiecarui elicopter
void rezolva(Matrice& m, Elicopter eli[], int nrEli, Elicopter eliGresit[], int&
eliGresitIndex) {
	for (int i = 0; i < nrEli; i++) {
		if (aterizatOk(m, eli[i]) == false)
			eliGresit[eliGresitIndex++] = eli[i];
	}
```

```
}

void main() {
	Matrice m;
	Elicopter eli[100];
	int nrEli = 0;
	citireDate("6.in", m, eli, nrEli);

	Elicopter eliGresit[100];
	int eliGresitIndex = 0;

	rezolva(m, eli, nrEli, eliGresit, eliGresitIndex);
	tiparire(eliGresit, eliGresitIndex);
}
```

Matricea Energetică

Enunț

Neo trebuie să traverseze matricea energetică. El pornește de pe ultima linie, unde poate intra în matrice pe oricare din coloane. El poate părăsi matricea de pe oricare coloană a primei linii. Pentru a traversa matricea, el se poate mișca de pe căsuța curentă pe una din căsuțele adiacente de pe linia imediat următoare. Astfel, fiecare pas constă în traversarea unei linii, vertical sau pe orizontală. Fiecare căsuță are asociat un cost energetic, exprimat sub forma unui număr întreg pozitiv. Scrieți un program care găsește costul energetic minim al traversării matricii.

Exemplu

6	7	4	7	8
7	6	1	1	4
3	5	7	8	2
9	6	7	0	9
9	9	5	1	9

Figure 4 - Exemplu matricea energetică

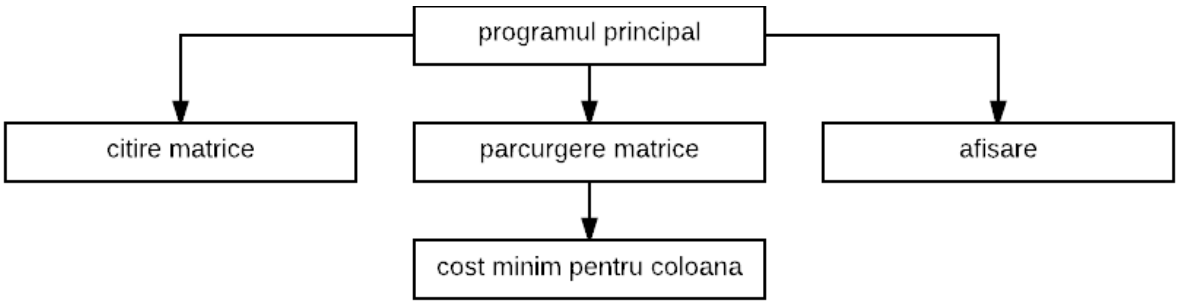
Pentru matricea de mai sus, costul energetic minim este de 8. Se intră pe coloana 4 (valoarea 1) și se iese pe coloana 3 (valoarea 4).

Pasii algorimului principal

Algoritm Matricea Energetică

@ citire matrice
@ parcurgerea coloanelor
@ costul minim aferent ieșirii pe o anumită coloană
Sf.Algoritm

Identificarea subalgoritmilor



Programul

-- matrice.h --

```
const int MAX = 200;

struct Matrice {
	int m;
	int n;
	int elem[MAX][MAX];
};

void afisare(Matrice m);
Matrice citire(char*);
```

-- matrice.cpp –

```
#include "Matrice.h"
#include <iostream>
#include <iomanip>
using namespace std;

void afisare(Matrice m) {
	cout << "linii=" << m.n << ", coloane=" << m.m << endl;
	for (int i = 0; i < m.n; i++) {
		for (int j = 0; j < m.m; j++) {
			cout << setw(4) << m.elem[i][j];
		}
		cout << endl;
	}
}

Matrice citire(char* fisier) {
	FILE *fin;
	Matrice m;

	fopen_s(&fin, fisier, "r");
```

```
	fscanf_s(fin, "%d", &m.n);
	fscanf_s(fin, "%d", &m.m);

	int v;
	for (int i = 0; i < m.n; i++) {
		for (int j = 0; j < m.m; j++) {
			fscanf_s(fin, "%d ", &m.elem[i][j]);
		}
	}
	return m;
}
```

-- checkerboard.cpp –

```
// Rezolvarea nu este optimizata pentru viteza de executie
// Rezolvarea exemplifica o abordare a problemei bazata pe descompunerea in subprobleme
// Programul a fost compilat cu Visual Studio Community 2015
#include <iostream>
#include "Matrice.h"
using namespace std;

// minimul a trei intregi
int minim(int a, int b, int c) {
	if (a < b) {
		if (a < c) return a;
		return c;
	}
	else {
		if (b < c) return b;
		return c;
	}
}

// calculeaza costul minim pentru a ajunge pe casuta (linie, coloana) pornind de pe ultima
linie
int costMinim(Matrice& m, int linie, int coloana) {
	if (coloana<0 || coloana > m.m - 1) {
		// daca iesim din matrice costul este +inf
		return INT_MAX;
	}

	if (linie == m.n - 1) {
		// costul este calculat direct pentru ultima linie
		return m.elem[linie][coloana];
	}

	// costul minim este costul casutei (linie,coloana) + costul minim de a ajunge la ea
	return m.elem[linie][coloana] + minim(costMinim(m, linie + 1, coloana - 1),
costMinim(m, linie + 1, coloana), costMinim(m, linie + 1, coloana + 1));
}

// calculam costul traversarii matricii
void traversareMinim(Matrice& m, int& min, int& colMin) {
	min = INT_MAX;
	// calculam pentru fiecare punct de pornire posibil
	for (int index = 0; index < m.m; index++) {
		int costMinimStartX = costMinim(m, 0, index);
```

```
			if (costMinimStartX < min) {
				min = costMinimStartX;
				colMin = index;
			}
		}
}

void main() {
		Matrice m;
		m = citire("7.in");
		afisare(m);

		int min = 0;
		int colMin = 0;
		traversareMinim(m, min, colMin);

		cout << "Costul minim al traversarii este " << min << " cand se ajunge pe coloana " <<
colMin << endl;
}
```