ALGORITMUL LUI LEE

Berengea Cristian-Traian

Cuprins:

Introducere

1.Descrierea temei şi motivaţia alegerii

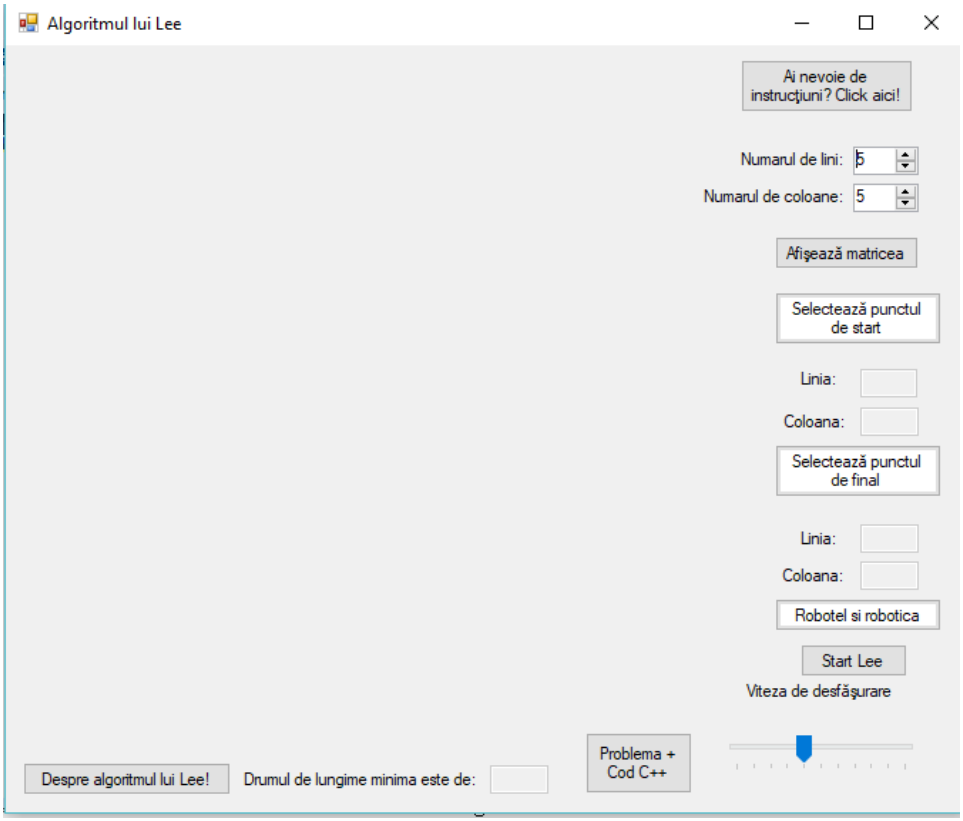
Lucrarea este o aplicaţie care ilustrează modul de funcţionare a algoritmului lui Lee în rezolvarea problemelor.Aplicaţia are caracter educativ deoarece utilizatorii pot înţelege cu uşurinţă paşii făcuţi de algoritm. Abordarea reprezintă de asemenea un model de bună practică pentru rezolvarea de probleme la informatică/

2.Ce este algoritmul lui Lee?

Algoritmul lui Lee rezolvă o problemă frecvent întâlnită în practică: determinarea unui drum cu număr minim de paşi, în anumite condiţii, spaţiul de căutare fiind un tablou (cel mai frecvent, bidimensional). Reprezintă de asemenea una dintre cele mai remarcabile aplicaţii ale unei structuri de date abstracte, fundamentală în informatică, coada. Din punct de vedere didactic, algoritmul lui Lee este un subiect dificil, care solicită extrem de multă energie: trebuie creată imaginea vizuală, reprezentarea în memorie, explicat modul de funcţionare şi în plus toate acestea urmărite în paralel cu implementarea.

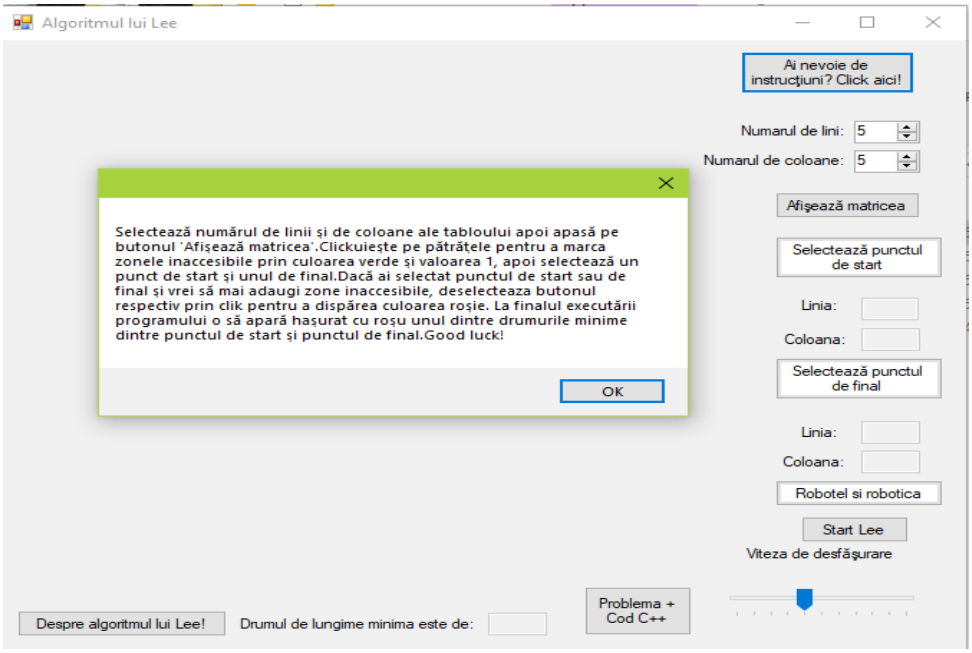
Descriere aplicație

La deschiderea aplicației utilizatorul va fi întâmpinat cu următoarea fereastră:

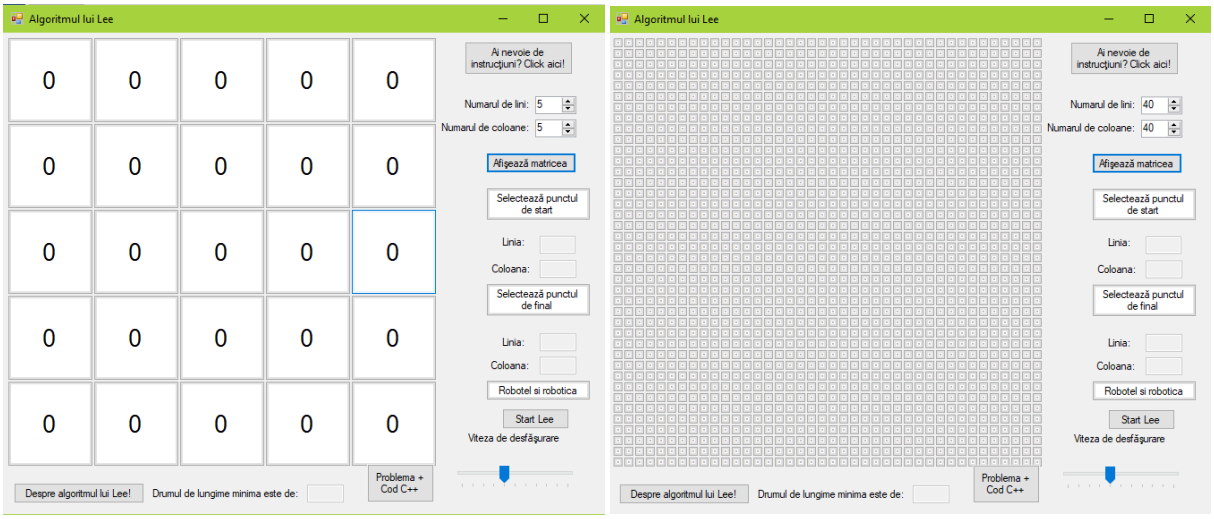


În cazul în care utilizatorul aplicaţiei nu ştie cum să o folosească, acesta are la dispoziţie instrucţiunile.Pentru acestea, el trebuie doar să apese pe butonul “Ai nevoie de instructiuni?Click aici!”.Va apărea o fereastră ca în imaginea de pe pagina urmatoare.

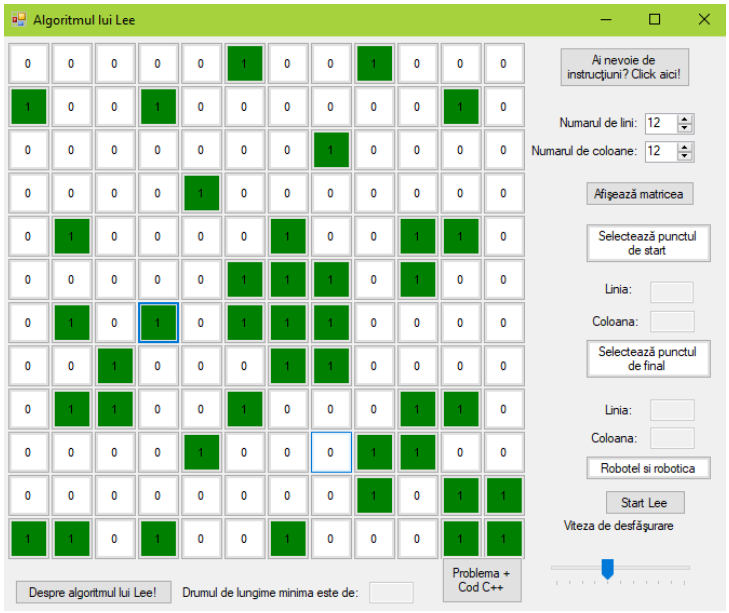
După ce a înţeles modul de utilizare al programului, utilizatorul poate închide fereastra şi reveni la aplicaţie apăsând “X” din partea dreaptă sus.



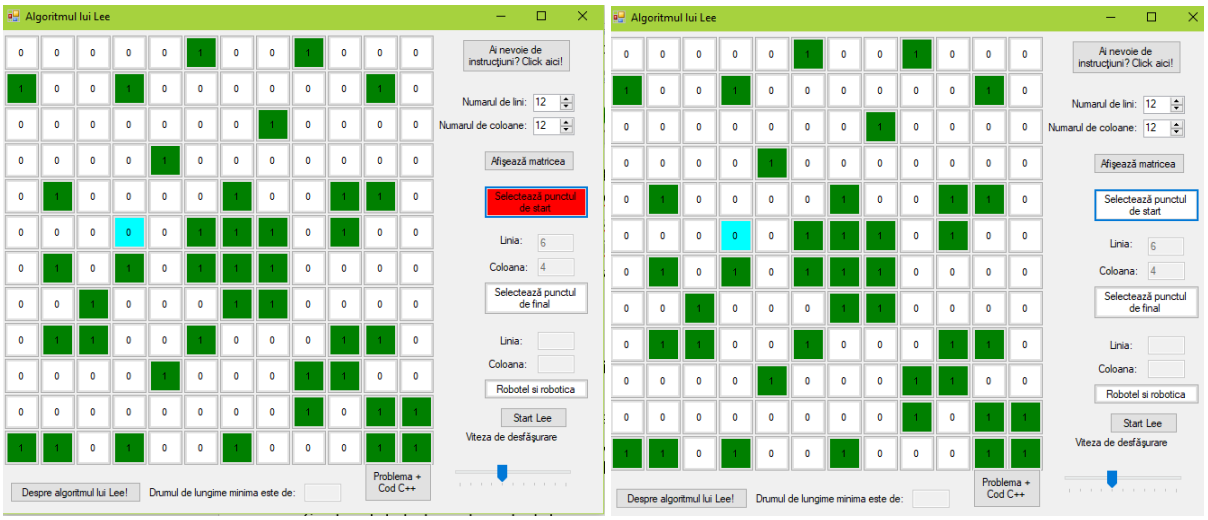
După cum se observă şi în imaginile anterioare, tabloul bidimensional are iniţial 5 linii şi 5 coloane. În cazul în care utilizatorul doreşte să redimensioneze matricea, acesta poate alege o dimensiune până la 40 de linii şi coloane.Pentru afişarea matricei alese, acesta trebuie să apese pe butonul „Afiseaza matricea”.Se va încărca o matrice de butoane, valoarea iniţiala a butoanelor fiind 0.**Important: mărimea butoanelor de pe linii şi coloane va fi invers proporţională cu mărimea matricei.**



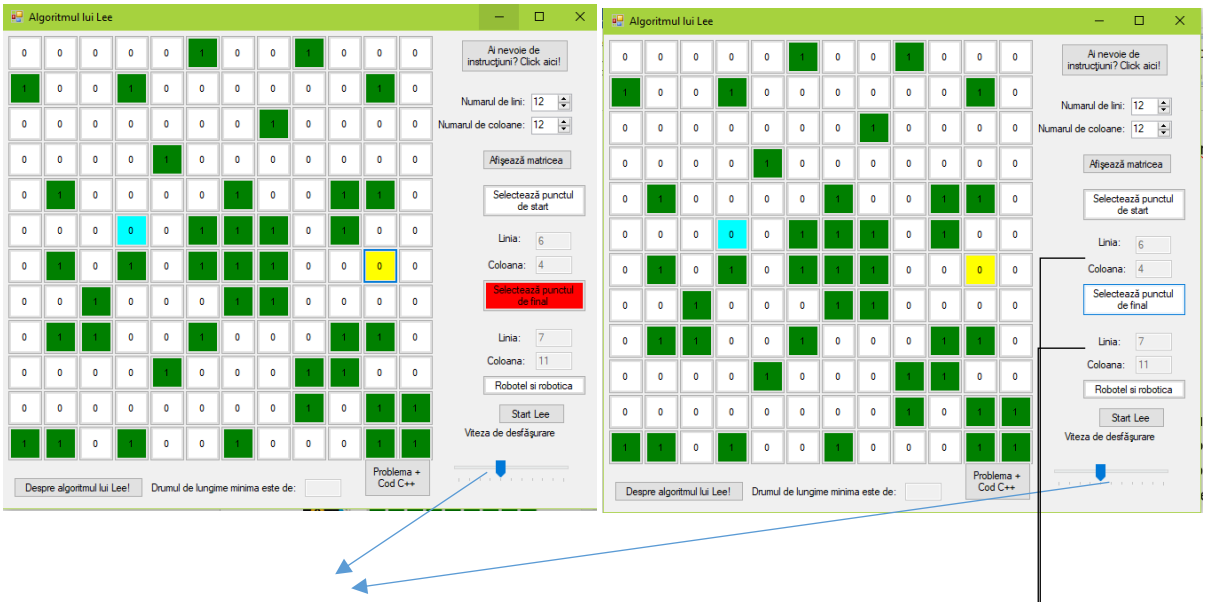
Pentru a marca zonele inaccesibile, utilizatorul va trebui să apese pe butoanele respective, valoarea acestora va deveni 1 şi vor primi culoarea verde.



Pentru a alege un punctul de start al algoritmului este necesar ca utilizatorul să dea click pe butonul “Selectează punctul de start”.În momentul clickuirii butonului, acesta va primi culoarea roşie pentru a arăta că este selectat.Pe matrice, utilizatorul trebuie să selecteze butonul care să primească punctul de start, acesta va primi culoarea albastră.Pentru a ieşi din opţiunea de alegere a punctului initial, utilizatorul va deselecta butonul roşu.



Utilizatorul va proceda la fel pentru alegerea punctului de final.



De aici utilizatorul poate modifica viteza de desfăşurare a aplicaţiei, pentru a înţelege mai bine.

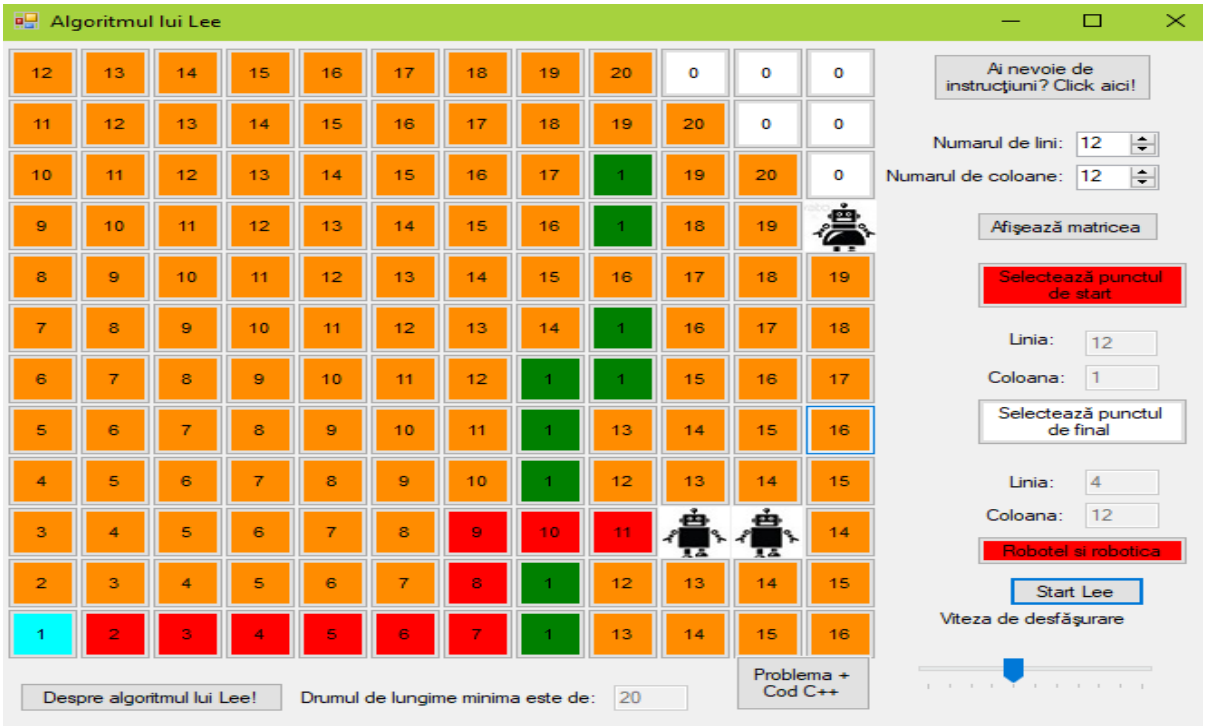
Coordonatele punctului de start, respectiv punctului de final.

Important: Daca nici unul dintre cele doua butoane („Selecteaza punctul de start”, „Selecteaza punctul de final”) nu este selectat (nu are culoarea roşie) se pot marca din nou zone inaccesibile.

Pentru a porni aplicaţia, apăsaţi butonul “Start Lee”:

Lungimea drumului minim(cel colorat cu roşu)

Pentru a vedea animaţia algoritmului, va trebui selectat butonul “Robotel si robotica”, iar apoi urmati pasii din paginile anterioare.Punctul de start va fi reprezentat de un robotel, iar punctul de final de o robotica.La rularea aplicaţiei robotelul va sări pe butoanele care formează drumul cel mai scurt pentru a ajunge la roboţică.



Problemă C++

Cerinţă:

Se da o matrice cu n linii si m coloane si elemente 0 sau 1, reprezentând planul unui teren în care 0 reprezinta o zona accesibila, iar 1 reprezinta o zona inaccesibila.

O zona a terenului are ca si coordonate linia si coloana corespunzatoare din matrice. Într-o zona cunoscuta a matricei se afla un robot, iar în alta zona, de asemenea cunoscuta, se afla o robotica.

Determinati numarul minim de pasi prin care robotul va ajunge la robotica.

Daca nu este posibil ca robotul sa ajunga la robotica, rezultatul va fi -1.

Date de intrare

Fisierul de intrare roboti.in contine pe prima linie numerele n m.

Urmatoarele n linii contin câte m valori, 0 sau 1.

Urmatoarele doua linii contin câte doua valori, reprezentând coordonatele robotului, respectiv ale roboticii.

Date de ieşire

Fisierul de iesire roboti.out va contine pe prima linie valoarea ceruta.

Restricţii şi precizări:

1 = n , m = 1000

zonele pe care se afla initial cei doi roboti sunt libere si sunt diferite

un pas reprezinta trecerea robotului din zona curenta într-o zona vecina cu aceasta pe linie sau pe coloana, fara a parasi matricea.

EXEMPLU

Intrare :	***Ieşire:***
4 5 1 0 0 0 1 0 0 1 0 0 0 0 0 0 1 1 1 0 0 1 1 2 2 5	4

Explicaţie:

Un traseu al robotului format din 4 pasi este evidentiat mai jos:

1	**0**	**0**	**0**	1
0	0	1	**0**	**0**
0	0	0	0	1
1	1	0	0	1

Exista si alte trasee posibile, dar lungimea lor este mai mare.

Rezolvare:

Metoda I:

```
#include <fstream>
using namespace std;
ifstream fin("roboti.in");
ofstream fout("roboti.out");
int A[1001][1001],X[1000001],Y[1000001],n,m;
int inside (int i,int j)
{
   return i>=1 && i<=n && j>=1 && j<=m;
}
int main()
{
   fin>>n>>m;
```

```
for(int i=1;i<=n;i++)
{
   for(int j=1;j<=m;j++)
      fin>>A[i][j];
}
int ib,jb,ir,jr;
fin>>ib>>jb>>ir>>jr;
int s=1,d=1;
A[ib][jb]=0;
A[ir][jr]=0;
X[1]=ib;
Y[1]=jb;
while(s<=d &&  A[ir][jr]==0)
 {
    int i=X[s],j=Y[s];
    if(inside(i-1,j) && A[i-1][j]==0)
    {
       A[i-1][j]=A[i][j]+1;
       d++;
       X[d]=i-1; Y[d]=j;
    }
    if(inside(i+1,j) && A[i+1][j]==0)
    {
       A[i+1][j]=A[i][j]+1;
       d++;
       X[d]=i+1; Y[d]=j;
    }
    if(inside(i,j-1) && A[i][j-1]==0)
    {
       A[i][j-1]=A[i][j]+1;
       d++;
```

```
            X[d]=i; Y[d]=j-1;
        }
        if(inside(i,j+1) && A[i][j+1]==0)
        {
            A[i][j+1]=A[i][j]+1;
            d++;
            X[d]=i; Y[d]=j+1;
        }
        s++;
    }
   if(A[ir][jr]!=0) fout<<A[ir][jr];
   else fout<<-1;
   return 0;
}
```

Metoda II:

```
#include <iostream>
#include <fstream>
#include <cassert>
using namespace std;

ifstream fin("roboti.in");
ofstream fout("roboti.out");

int n , m;
int a[1002][1002];
short x[1000005], y[1000005];
int x1 , y1 , x2 , y2;

const int dx[]={0 , 0 , 1 , -1}, dy[]={1 , -1 , 0 , 0};

int main(){
```

```
fin >> n >> m;
for(int i = 1 ; i <= n ; i ++)
	for(int j = 1 ; j <= m ; j ++)
		fin >> a[i][j];
assert(fin >> x1 >> y1 >> x2 >> y2);
assert(a[x1][y1] == 0);
assert(a[x2][y2] == 0);
fin.close();
//facem obstacolele -1
for(int i = 1 ; i <= n ; i ++)
	for(int j = 1 ; j <= m ; j ++)
		a[i][j] = - a[i][j];
//bordare cu -1, ca sa nu iesim din matrice
for(int i = 0 ; i <= n + 1 ; i ++)
	a[i][0] = a[i][m+1] = -1;
for(int j = 0 ; j <= m + 1; j ++)
	a[0][j] = a[n+1][j] = -1;


int st , dr;
st = dr = 1;
x[dr] = x1, y[dr] = y1;
a[x1][y1] = 1;
while(st <= dr)
{
	int i = x[st], j = y[st];
	//cerr << i << " " << j << endl;
	for(int k = 0 ; k < 4 ; k ++)
	{
		int ii = i + dx[k], jj = j + dy[k];
		if(a[ii][jj] == 0)
		{
```

```
                                a[ii][jj] = a[i][j] + 1;
                                dr ++;
                                x[dr] = ii, y[dr] = jj;
                        }
                }
                st ++;
        }
        /*
        for(int i = 1 ; i <= n ; i ++)
        {
                for(int j = 1 ; j <= m ; j ++)
                        cerr << a[i][j] << " " ;
                cerr << endl;
        }
        */
        if(a[x2][y2] == 0)
                fout << -1;
        else
                fout << a[x2][y2] - 1;
        fout.close();
        return 0;
}
```