ALGORITMI ELEMENTARI FOLOSIND MATRICE

Declararea unei matrice

a) Variantă **statică**

```
int / float a[20][20];
   //spatiul disponibil: a[0][0],….,a[0][19],….,a[19][19]
int m,n; //m=numarul de linii, n = numarul de coloane
   //deci elementele sunt (a[i][j]), i=0,m-1 si j=0,n-1
   // =>  a[0][0],…,a[0][n-1]
   //      …
   //     a[m-1][0],…, a[m-1][n-1]
```

Citirea unei matrice declarată static se poate face astfel:

```
//intai dimensiunile
cout<<"Dati numarul de linii: "; cin>>m;
cout<<"Dati numarul de coloane: "; cin>>n;
cout<<"Dati elementele matricei: "<<endl;
int i,j;
for (i=0;i<m;i++)
  for (j=0;j<n;j++) cin>>a[i][j];
```

Afișarea unei matrice declarată static se poate face astfel:

```
//intai dimensiunile
cout<<"Elementele matricei: "<<endl;
int i,j;
for (i=0;i<m;i++)
{
  for (j=0;j<n;j++) cout<<setw(6)<<a[i][j]; //de inclus <iomanip>
  cout<<endl;//dupa ce s-a afisat linia i se trece pe randul urmator
}
```

b) Variantă *dinamică*

```
float **a; //pointeri catre pointeri catre elemente de tip float
int m,n; //numarul de linii si de coloane
cout<<"Dati numarul de linii: "; cin>>m;
cout<<"Dati numarul de coloane: "; cin>>n;
//alocare memorie pentru matrice
a = new float* [m];
//se rezerva spatiu pentru m elemente de tip pointer catre float
//(capetele de linii)
int i,j;
for (i=0; i<m; i++) //pentru fiecare cap de linie
   a[i] = new float [n];
//fiecare cap de linie este pointer catre n elemente de tip float
//(linia i)
//acum avem spatiu rezervat pentru elem.(a[i][j]), i=0,m-1, j=0,n-1
//de exemplu, citirea elementelor se poate face astfel
for (i=0;i<m;i++)
  for (j=0;j<n;j++) cin>>a[i][j];
//dupa ce s-a incheiat utilizarea matricei, trebuie dealocat
//spatiul alocat la inceput pentru liniile matricei,
```

```
//respectiv capetele de linii (in ordinea inversa alocarii)
for (i=0; i<m; i++) //pentru fiecare cap de linie
   delete [] a[i];//se elibereaza spatiul ocupat de liniile matricei
delete [] a; //se elibereaza spatiul ocupat de capetele de linii
```

R1 (3*).

Enunțul problemei: Să se descrie un algoritm pentru adunarea a două matrice date.

Metoda de rezolvare: Ținând cont ca pot fi însumare doar două matrice de aceleași dimensiuni, există două abordări:

1. fie se consideră $A \in M_{m1\times n1}(R)$ și $B \in M_{m2\times n2}(R)$ și doar dacă $m_1 = m_2$ și $n_1 = n_2$ se poate calcula suma celor două matrice;
2. fie se impune ca cele două matrice să fie de aceleași dimensiuni $A,B \in M_{m\times n}(R)$ și se realizează suma fără probleme.

Pentru prima modalitate, o variantă de implementare ar fi următoarea:

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

void Citire(float a[10][10], int &m, int &n, char c) {
//citirea dimensiunilor unei matrice si a elementelor sale
 int i,j;
 cout<<"Dati numarul de linii: "; cin>>m;
 cout<<"Dati numarul de coloane: "; cin>>n;
 cout<<"Dati elementele matricei "<<c<<endl;
 for (i=0;i<m;i++)
  for (j=0;j<n;j++) cin>>a[i][j];
}
void Afisare(float a[10][10], int m, int n) {
//afisarea elementelor unei matrice cu m linii si n coloane
 int i,j;
 for (i=0;i<m;i++) {
   for (j=0;j<n;j++) cout<<a[i][j]<<" ";
   cout<<endl;
  }
}
void SumaMatrice(float a[10][10],float b[10][10],int m,int n,float
c[10][10]) {
// determinarea matricei C=A+B toate de dim. m*n
 int i,j;
 for (i=0;i<m;i++)
  for (j=0;j<n;j++) c[i][j]=a[i][j]+b[i][j];
}
int main() {
 int m1,n1,m2,n2;
 float a[10][10],b[10][10],c[10][10];
 cout<<"Matricea A:"<<endl;  Citire(a,m1,n1,'A');
 cout<<"Matricea B:"<<endl;  Citire(b,m2,n2,'B');
 if (m1==m2 && n1==n2)
 {
    SumaMatrice(a,b,m1,n1,c);
    cout<<endl<<"A+B="<<endl;
    Afisare(c,m1,n1);
 }
 else
```

```
   cout<<"Nu se poate calcula A+B"<<endl;
 return 0;
}
```

În ce-a de-a doua variantă, se pot citi in `main` dimensiunile comune celor două matrice, apoi prin două apeluri de funcții, se pot citi elementele celor două matrice și apoi se vor însuma cele două matrice. O posibilă implementare ar fi următoarea:

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

void Citire(float a[10][10], int m, int n, char c)
{
//citirea elementelor unei matrice
 int i,j;
 cout<<"Dati elementele matricei "<<c<<endl;
 for (i=0;i<m;i++)
  for (j=0;j<n;j++) cin>>a[i][j];
}

void Afisare(float a[10][10], int m, int n)
{
//afisarea elementelor unei matrice cu m linii si n coloane
 int i,j;
 for (i=0;i<m;i++) {
   for (j=0;j<n;j++) cout<<a[i][j]<<" ";
   cout<<endl;
  }
}

void SumaMatrice(float a[10][10],float b[10][10],int m,int n,float
c[10][10])
{
// determinarea matricei C=A+B toate de dim. m*n
 int i,j;
 for (i=0;i<m;i++)
  for (j=0;j<n;j++) c[i][j]=a[i][j]+b[i][j];
}

int main()
{
 int m,n;
 float a[10][10],b[10][10],c[10][10];
 //citim dimensiunile comune celor doua matrice
 cout<<"Dati numarul de linii: "; cin>>m;
 cout<<"Dati numarul de coloane: "; cin>>n;
 Citire(a,m,n,'A');
 Citire(b,m,n,'B');

 SumaMatrice(a,b,m,n,c);

 cout<<endl<<"A+B="<<endl;
 Afisare(c,m,n);
 return 0;
}
```

R2 (3*).

Enunţul problemei: Pentru o matrice pătrată dată să se determine suma elementelor de pe diagonala principală şi suma elementelor de pe diagonala secundară.

Metoda de rezolvare: O matrice pătratică are forma

$$A=\begin{pmatrix} a_{11} & \dots & \dots & \dots & a_{1n} \\ \dots & a_{22} & \dots & a_{2,n-1} & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & a_{n-1,2} & \dots & a_{n-1,n-1} & \dots \\ a_{n1} & \dots & \dots & \dots & a_{nn} \end{pmatrix}$$

În pseudocod, elementele de pe diagonala principală sunt: a_{11}, a_{22}, …, $a_{n-1,n-1}$, a_{nn}, iar elementele de pe diagonala secundară sunt: a_{n1}, $a_{n-1,2}$, $a_{n-2,3}$,… $a_{2,n-1}$, a_{1n}. Atunci suma elementelor de pe diagonala principală este $S_1 = a_{11}+a_{22}+\dots+a_{n-1,n-1}+a_{nn} = \sum_{i=1}^{n} a_{ii}$, iar suma elementelor de pe diagonala principală (cele care au suma indicilor constantă, n+1) este $S_2 = a_{n1} + a_{n-1,2} + a_{n-2,3} +\dots + a_{2,n-1} + a_{1n} = \sum_{i=1}^{n} a_{i,n+1-i}$. Aşadar, avem de calculat sume.

Descrierea algoritmului în pseudocod:

```
citeşte n,a11,…,ann
S1 ← 0     *suma elem. de pe diag. principala, initial 0
S2 ← 0     *suma elem. de pe diag. secundara, initial 0
pentru i=1,n repeta
    S1 ← S1 + aii
    S2 ← S2 + ai,n+1-i
scrie S1, S2
```

Descrierea algoritmului în pseudocod C++(CodeBlocks):

În C/C++, elementele ar fi bine să fie memorate începând cu linia şi coloana 0:

$$A=\begin{pmatrix} a_{00} & \dots & \dots & \dots & a_{0,n-1} \\ \dots & a_{11} & \dots & a_{1,n-2} & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & a_{n-2,1} & \dots & a_{n-2,n-2} & \dots \\ a_{n-1,0} & \dots & \dots & \dots & a_{n-1,n-1} \end{pmatrix}$$

elementele de pe **diagonala principală sunt: a_{00}, a_{11}, …, $a_{n-2,n-2}$, $a_{n-1,n-1}$** (adică a_{ii}, i=0,…,n-1) iar elementele de pe **diagonala secundară sunt: $a_{n-1,0}$, $a_{n-2,1}$, $a_{n-3,2}$,… $a_{1,n-2}$, $a_{0,n-1}$** - suma indicilor este constanta n-1(adică $a_{i,n-i-1}$, i=0,…, n-1).

În acest caz sumele pe diagonala principală, respectiv secundară devin:

$S_1 = a_{00}+a_{11}+\dots+a_{n-2,n-2}+a_{n-1,n-1}=\sum_{i=0}^{n-1} a_{ii}$, respectiv

$S_2=a_{n-1,0}+a_{n-2,1}\dots+a_{1,n-2}+a_{0,n-1}=\sum_{i=0}^{n-1} a_{i,n-1-i}$.

```
#include <iostream>
using namespace std;
int main()
{
```

```
 int n,i,j;
 float a[10][10], S1, S2;

 cout<<"Dati dimeniunea matricei: "; cin>>n;
 //matrice patrata
 /* for (i=0;i<n;i++)
  for (j=0;j<n;j++)
  {cout<<"a["<<i+1<<"]["<<j+1<<"]=";
   cin>>a[i][j]; } */
 cout<<"Dati elementele matricei:"<<endl;
 for (i=0;i<n;i++)
  for (j=0;j<n;j++) cin>>a[i][j];

 S1 = S2 = 0; //sumele se initializeaza cu 0
 for (i=0;i<n;i++) //parcurgem ambele diag. simultan
 {
   S1 += a[i][i];
   S2 += a[i][n-1-i];
 }
 cout<<"Suma elem de pe diag principala: "<<S1;
 cout<<endl<<"Suma elem de pe diag secundara: "<<S2<<endl;
 return 0;
}
```

$$S_1 = \sum_{i=0}^{n-1} a_{ii}$$

$$S_2 = \sum_{i=0}^{n-1} a_{i,n-1-i}$$

Rulare:

```
Dati dimeniunea matricei: 4 <Enter>
Dati elementele matricei:
1 1 1 1
2 2 1 1
0 1 2 1
0 1 1 2
Suma elem de pe diag principala: 7
Suma elem de pe diag secundara: 3
```

R3 (3*-suplimentar).

Enunţul problemei: Să se determine transpusa unei matrice dată.

Metoda de rezolvare: Pentru matricea $\boldsymbol{A} = (a_{ij})_{i=1,2,\ldots,\boldsymbol{m},\ j=1,2,\ldots,\boldsymbol{n}}$, ($m$ linii şi n coloane), matricea transpusă este $B \overset{not}{=} \boldsymbol{A^t} = (b_{ij})_{i=1,2,\ldots,n,\ j=1,2,\ldots,m}$, ($n$ linii şi m coloane) cu $b_{ji} = a_{ij}$, pentru i=1, 2, …, m, j = 1, 2,…, n. Aşadar, se pot parcurge elementele matricei A (i =1, …, m, j = 1, …, n) şi elementul a_{ij} se copiază în B pe poziţia ji. Altfel, pentru calculul fiecărui element din B, b_{ij} , i=1, 2,…, n, j = 1, 2,…, m, se calculează ca fiind a_{ji}. În continuare se merge pe prima variantă.

Descrierea algoritmului în pseudocod:

```
citeşte m,n,a11,…,amn   *citim intai dim. matricei, apoi elem
pentru i = 1,m,1 repeta
   pentru j = 1,n,1 repeta  *parcurgem elem matriei a
      bji ← aij
      *elem de pe lin i, col j din a vine pe linia j col i in b
scrie (bij) pentru i=1,2,…,n, j=1,2,…,m
```

Descrierea algoritmului în C++ (CodeBlocks):

```
#include <iostream>
using namespace std;
int main()
{int m,n,i,j;
 float a[10][10],b[10][10];
 cout<<"Dati numarul de linii: "; cin>>m;
```

```
 cout<<"Dati numarul de coloane: "; cin>>n;
 cout<<"Dati elementele matricei:"<<endl;
 for (i=0;i<m;i++)
  for (j=0;j<n;j++) cin>>a[i][j];

 //determinarea transpusei
 for (i=0;i<m;i++) //parcurg elementele matricei a
  for (j=0;j<n;j++)  b[j][i]=a[i][j];

 //afisarea transpusei
 cout<<"Matricea transpusa este:"<<endl;
 for (i=0;i<n;i++) //pentru fiecare linie
 {
   for (j=0;j<m;j++) cout<<b[i][j]<<" "; //elem liniei i
   cout<<endl; //se trece pe randul urmator
  }
 return 0;
}
```

sau, folosind funcţii:

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

void Citire(float a[10][10], int &m, int &n) {
//citirea dimensiunilor unei matrice si a elementelor sale
 int i,j;
 cout<<"Dati numarul de linii: "; cin>>m;
 cout<<"Dati numarul de coloane: "; cin>>n;
 cout<<"Dati elementele matricei:"<<endl;
 for (i=0;i<m;i++)
  for (j=0;j<n;j++) cin>>a[i][j];
}
void Afisare(float a[10][10], int m, int n) {
//afisarea elementelor unei matrice cu m linii si n coloane
 int i,j;
 for (i=0;i<m;i++) {
   for (j=0;j<n;j++) cout<<a[i][j]<<" ";
   cout<<endl;
  }
}
void Transpusa(float a[10][10],int m,int n,float b[10][10]) {
// determinarea transpusei unei matrice de dim. m*n
 int i,j;
 for (i=0;i<m;i++)
  for (j=0;j<n;j++) b[j][i]=a[i][j];
}
int main()
{
 int m,n,i,j;
 float a[10][10],b[10][10];
  Citire(a,m,n);
  Transpusa(a,m,n,b);
  // afisarea transpusei
  cout<<endl<<"Matricea transpusa este:"<<endl;
  Afisare(b,n,m); //matr b are n linii si m col
 return 0;
}
```

Rulare:

```
Dati numarul de linii: 3
```

```
Dati numarul de coloane: 2
Dati elementele matricei:
1 2 <Enter>
3 4 <Enter>
5 6 <Enter>
Matricea transpusa este:
1 3 5
2 4 6
```

R4 (5*-suplimentar).

Enunțul problemei: Să se produsul a două matrice: $A \in M_{m\times n}$ și $B \in M_{n\times p}$.

Metoda de rezolvare: Pentru o matrice $A = (a_{ij})_{i=1,2,\ldots,m,\ j=1,2,\ldots,n}$ și $B = (b_{ij})_{i=1,2,\ldots,n,\ j=1,2,\ldots,p}$, produsul acestora este matricea $C = A\cdot B$, cu $c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}$, pentru i=1,2,…, m, j=1,2,…, p.

De exemplu, pentru $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \in M_{3\times 2}$, $B = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \in M_{2\times 1}$, matricea $C = A\cdot B = \begin{pmatrix} 5 \\ 5 \\ 6 \end{pmatrix} \in M_{3\times 1}$.

Descrierea algoritmului în C++ (CodeBlocks):

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

void Citire(float a[10][10], int m, int n)
{
//citirea elementelor unei matrice
 cout<<"Dati elementele matricei:"<<endl;
 for (int i=0;i<m;i++)
  for (int j=0;j<n;j++) cin>>a[i][j];
}
void Afisare(float a[10][10], int m, int n) {
 for (int i=0;i<m;i++) {
   for (int j=0;j<n;j++) cout<<a[i][j]<<" ";
   cout<<endl;
  }
}
void Inmultire(float a[10][10],int m, int n, float b[10][10],int p,
               float c[10][10])
// C = A*B, matrice patratice m*p
{
 int i,j,k;
 for (i=0;i<m;i++)
  for (j=0;j<p;j++)
   {
    c[i][j]=0;
    for (k=0;k<n;k++) c[i][j] += a[i][k]*b[k][j];
   }
}
int main() {
 int m,n,p;
 float a[10][10], b[10][10], c[10][10];
 cout<<"m="; cin>>m;
 cout<<"n="; cin>>n;
```

$c_{ij} = \sum_{k=0}^{n-1} a_{ik}b_{kj}$, i=0,1,…, m-1, j=0,1,…, p-1

```
 cout<<"p="; cin>>p;
 cout<<"Matricea A: "<<endl; Citire(a,m,n);
 cout<<"Matricea B: "<<endl; Citire(b,n,p);
 Inmultire(a,m,n,b,p,c);
 cout<<endl<<"Produsul matricelor este: "<<endl;
 Afisare(c,m,p);
 return 0;
}
```

Rulare:

```
      m = 3
      n = 2
      p = 1
      Matricea A:
      Dati elementele matricei:
      1 2 <Enter>
      3 4 <Enter>
      5 6 <Enter>
      Matricea B:
      Dati elementele matricei:
      2 <Enter>
      3 <Enter>
      Produsul matricelor este:
      8
      18
      28
sau
      m = 2
      n = 2
      p = 2
      Matricea A:
      Dati elementele matricei:
      1 2 <Enter>
      3 4 <Enter>
      Matricea B:
      Dati elementele matricei:
      2 0 <Enter>
      3 -1 <Enter>
      Produsul matricelor este:
      8   -2
      18  -4
```

R5 (5*-suplimentar).

Enunţul problemei: Să se determine puterea p a matricei pătratice A.

Metoda de rezolvare: Considerăm matricea pătratică $A = (a_{ij})_{i=1,2,\ldots,n,\ j=1,2,\ldots,n}$.

$A^2 = A \cdot A$

$A^3 = A^2 \cdot A$

….

$A^p = A^{p-1} \cdot A$ (şi este tot matrice pătratică $n \times n$).

Atunci vom proceda astfel:

$B \leftarrow A$

$k = 2,\ldots,p$

  $C \leftarrow B \cdot A$

  $B \leftarrow C$

Aşadar la bază stă înmulţirea dintre două matrice. În general, pentru o matrice $A=(a_{ij})_{i=1,2,\ldots,m,\ j=1,2,\ldots,n}$ şi $B=(b_{ij})_{i=1,2,\ldots,n,\ j=1,2,\ldots,p}$, produsul acestora este matricea $C=A\cdot B$, cu $c_{ij}=\sum_{k=1}^{n} a_{ik}b_{kj}$, pentru $i=1,2,\ldots,m,\ j=1,2,\ldots,p$)

Descrierea algoritmului în C++ (CodeBlocks):

```
#include <iostream>
using namespace std;
void Citire(float a[10][10], int &n) {
//citirea dimensiunea unei matrice si a elementelor sale
 cout<<"Dati dimensiunea matricei: "; cin>>n;
 cout<<"Dati elementele matricei:"<<endl;
 for (int i=0;i<n;i++)
  for (int j=0;j<n;j++) cin>>a[i][j];
}
void Afisare(float a[10][10], int n) {
 for (int i=0;i<n;i++){
   for (int j=0;j<n;j++) cout<<a[i][j]<<" ";
   cout<<endl;
  }
}
void Inmultire(float a[10][10], float b[10][10], int n, float
c[10][10]) { // C = A*B, matrice patratice n*n
 int i,j,k;
 for (i=0;i<n;i++)
  for (j=0;j<n;j++)
   {
    c[i][j]=0;
    for (k=0;k<n;k++) c[i][j] += a[i][k]*b[k][j];
   }
}
void RidicarePutere(float a[10][10],int n,int p, float b[10][10]) {
 //A^p
 float c[10][10];
 int i,j,k;
 for (i=0;i<n;i++)
   for (j=0;j<n;j++) b[i][j] = a[i][j];
 for (k=2;k<=p;k++) {
   Inmultire(b,a,n,c);
   for (i=0;i<n;i++)
     for (j=0;j<n;j++) b[i][j] = c[i][j];
  }
}
int main() {
 int n,p;
 float a[10][10],b[10][10];
 Citire(a,n);
 cout<<"Dati puterea p: "; cin>>p;
 RidicarePutere(a,n,p,b);
 cout<<"Matricea initiala este: "<<endl;
 Afisare(a,n);
 cout<<"Matricea initiala ridicata la puterea "<<p<<" este: "<<endl;
 Afisare(b,n);
 return 0;
}
```

Rulare:

```
Dati dimensiunea matricei: 2
Dati elementele matricei:
1 0 <Enter>
2 1 <Enter>
Dati puterea p: 3
Matricei initiale ridicata la puterea 3 este:
1 0
6 1
```

R6 (5* - suplimentar).

Enunţul problemei: Să se determine algoritmul pentru afişarea triunghiul lui Pascal.

Metoda de rezolvare:

De exemplu, triunghiul lui Pascal până la n=5:

				1						
			1		1					
		1		2		1				
	1		3		3		1			
1		4		6		4		1		
	5		10		10		5		1	

Aceste numere reprezintă

				C_0^0						
			C_1^0		C_1^1					
		C_2^0		C_2^1		C_2^2				
	C_3^0		C_3^1		C_3^2		C_3^3			
C_4^0		C_4^1		C_4^2		C_4^3		C_4^4		
C_5^0		C_5^1		C_5^2		C_5^3		C_5^4		C_5^5

În cazul general,

		C_0^0		
	C_1^0		C_1^1	
	.....			
C_n^0		....		C_n^n

sau

		C_0^0		
	C_1^0		C_1^1	
	.....			
C_i^0	....	C_i^j	....	C_i^i
	.....			
C_n^0		....		C_n^n

Aşadar, poate fi privit ca o matrice cu elementele C_i^j, j= 0,1,…, i (indici coloane) şi liniile cu indicii i = 0, 1, …, n, unde n este cunoscut. Pentru calculul combinărilor, reamintim:

$$C_n^k = \frac{n!}{k!(n-k)!} = \frac{(n-k+1)(n-k+2)...n}{k!} = \prod_{i=n-k+1}^{n} i \Big/ \prod_{i=1}^{k} i$$

Descrierea algoritmului în pseudocod:

```
functie Combinari(n,k)
returneaza n!/(k!(n-k)!)
sfarsit
citeşte n
pentru i = 0,n,1 repeta
   pentru j = 0,i,1 repeta
      scrie Combinari(i,j)
```

O descrierea algoritmului în C++ *(CodeBlocks):*

```
#include <iostream>
#include <iomanip>
using namespace std;

int Combinari(int n, int k) {
 int i,p1=1,p2=1;
 for (i=n-k+1;i<=n;i++) p1*=i;
 for (i=2;i<=k;i++) p2*=i;
 return p1/p2;
}
int main() {
 int n;
 cout<<"Dati ordinul ultimului nivel: ";  cin>>n;
 for (int i=0;i<=n;i++) {
   for (int j=0;j<=i;j++) cout<<setw(8)<<Combinari(i,j);
   cout<<endl;
  }
 return 0;
}
```

Rulare:

```
Dati ordinul ultimului nivel: 5 <Enter>
1
1     1
1     2     1
1     3     3     1
1     4     6     4     1
1     5     10    10    5     1
```

R7 (4-5*).

Enunţul problemei: Să se descrie un algoritm pentru a determina suma elementelor unei matrice dreptunghiulare pe fiecare linie.

Metoda de rezolvare: Se parcurge fiecare linie a matricei şi se calculează suma pe linia respectivă. De exemplu:

$$A=\begin{pmatrix}1 & 2\\ 3 & 4\\ 1 & 5\end{pmatrix}\begin{matrix}\rightarrow S_1=1+2=3\\ \rightarrow S_2=3+4=7\\ \rightarrow S_3=1+5=6\end{matrix}$$

În general:

$$A=\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{i1} & a_{i2} & \dots & a_{in} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{matrix} \\ \to S_1 \\ \to S_2 \\ \dots \\ \to S_i = a_{i1}+a_{i2}+\dots+a_{in}=\sum_{j=1}^{n} a_{ij} \\ \dots \\ \to S_m \end{matrix}$$

Aşadar, pentru fiecare linie, trebuie calculată o sumă. Dacă imediat după calculare se afişează valoarea sumei, atunci se poate folosi o singură variabilă, *S*, şi nu un vector al sumelor, mai ales că nu mai folosim sumele ulterior.

Descrierea algoritmului în pseudocod:

```
citeşte m,n,a11,…,amn
pentru i = 1,m  repeta * pentru fiecare linie
      S ← 0       *initializarea sumei elementelor de pe linia i
   pentru j = 1,n  repeta *parcurgem linia
      S ← S + aij *adunam elementul curent de pe linia i
scrie S
```

În C/C++:

$$A=\begin{pmatrix} a[0][0] & a[0][1] & \dots & a[0][n-1] \\ a[1][0] & a[1][1] & \dots & a[1][n-1] \\ \dots & \dots & \dots & \dots \\ a[i][0] & a[i][1] & \dots & a[i][n-1] \\ \dots & \dots & \dots & \dots \\ a[m-1][0] & a[m-1][1] & \dots & a[m-1][n-1] \end{pmatrix} \begin{matrix} \\ \to S[0] \\ \to S[1] \\ \dots \\ \to S[i] \\ \dots \\ \to S[m-1] \end{matrix}$$

unde $S[i]=a[i][0]+a[i][1]+\dots+a[i][n-1]=\sum_{j=0}^{n-1} a[i][j]$, pentru i = 0, 1,…, m-1.

Descrierea algoritmului în C++ (CodeBlocks):

```
#include <iostream>
using namespace std;
int main()
{
 int m,n,i,j;
 float a[10][10],S;

 //citim dimensiunile si elementele matricei
 cout<<"Dati numarul de linii: "; cin>>m;
 cout<<"Dati numarul de coloane: "; cin>>n;
 cout<<"Dati elementele matricei: "<<endl;
 for (i=0;i<m;i++)
   for (j=0;j<n;j++) cin>>a[i][j];

 for (i=0;i<m;i++)  //pentru fiecare linie
 {
  S = 0;          //suma de pe linia i se initializeaza cu 0
  for (j=0;j<n;j++)   //se parcurge linia i
    S += a[i][j];     //se adauga elementul curent
  cout <<"Suma elem. de pe linia "<<i+1<<" este: " <<S<<endl;
 }
 return 0;
}
```

Rulare:

```
Dati numarul de linii: 3 <Enter>
Dati numarul de coloane: 2 <Enter>
Dati elementele matricei:
1 2 <Enter>
3 4 <Enter>
5 6 <Enter>
Suma elem. de pe linia 1 este: 3
Suma elem. de pe linia 2 este: 7
Suma elem. de pe linia 3 este: 11
```

R8 (5*).

Enunţul problemei: Să se descrie un algoritm pentru a determina valoarea maximă dintre suma elementelor unei matrice dreptunghiulare pe fiecare coloană.

Metoda de rezolvare: De exemplu, pentru $A = \begin{pmatrix} 1 & 5 & 2 \\ 3 & 3 & 3 \end{pmatrix}$, suma maximă pe coloane este 8 (obţinută pe a doua coloană). În general:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1j} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2j} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mj} & \dots & a_{mn} \end{pmatrix}$$

$$\downarrow \quad \downarrow \quad \dots \quad \downarrow \quad \dots \quad \downarrow$$

$$S_1 \quad S_2 \quad \dots \quad S_j \quad \dots \quad S_n$$

unde $S_j = a_{1j} + a_{2j} + \dots + a_{mj} = \sum_{i=1}^{m} a_{ij}$, pentru $j = 1, 2, \dots, n$.

Algoritmul va consta în calculul sumelor elementelor pe fiecare coloană şi determinarea valorii maxime. Varianta cea mai simplă, constă în folosirea un vector pentru calculul sumelor elementelor pe fiecare coloană şi apoi determinareea valorii maxime dintre elemenele vectorului sumelor pe coloane.

O descriere a algoritmului în pseudocod:

```
citeşte m,n,a11,…,amn
pentru j = 1,n repeta * pentru fiecare linie
      Sj ← 0      *initializarea sumei elementelor de pe linia i
  pentru i = 1,m repeta *parcurgem linia
      Sj ← Sj + aij *adunam elementul curent de pe linia i
scrie Sj

max ← S1    *determinam valoarea maxima dintre S1,…,Sn
pentru j = 2,n repeta
  daca Sj > max atunci
     max ← Sj
scrie max
```

În C/C++:

$$A=\begin{pmatrix} a[0][0] & a[0][1] & \dots & a[0][j] & \dots & a[0][n-1] \\ a[1][0] & a[1][1] & \dots & a[1][j] & \dots & a[1][n-1] \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a[m-1][0] & a[m-1][1] & \dots & a[m-1][j] & \dots & a[m-1][n-1] \\ \downarrow & \downarrow & \dots & \downarrow & \dots & \downarrow \\ S[0] & S[1] & \dots & S[j] & \dots & S[n-1] \end{pmatrix}$$

unde $S[j]=a[0][j]+a[1][j]+\dots+a[m-1][j]=\sum_{i=0}^{m-1}a[i][j]$, pentru j = 0, 1,..., n-1.

O descriere a algoritmului în pseudocod C++ (CodeBlocks)*:*

```
#include <iostream>
using namespace std;
int main() {
 int m,n,i,j;
 float a[10][10],S[10],max;

 //citim dimensiunile si elementele matricei
 cout<<"Dati numarul de linii: "; cin>>m;
 cout<<"Dati numarul de coloane: "; cin>>n;
 cout<<"Dati elementele matricei: "<<endl;
 for (i=0;i<m;i++)
   for (j=0;j<n;j++) cin>>a[i][j];
 for (j=0;j<n;j++)  //pentru fiecare coloana
 {
  S[j] = 0;      //suma de pe coloana j se initializeaza cu 0
  for (i=0;i<m;i++)   //se parcurge coloana j
    S[j] += a[i][j];  //se adauga elementul curent
  cout<<"Suma elem.de pe coloana "<<j+1<<" este: "<<S[j]<<endl;
 }
 max = S[0]; //presupunem ca valoarea maxima este pe poz. 0
 for (j=1;i<n;i++) //parcurgem si restul pozitiilor
  if(S[j]>max) //daca S[j] depaseste maximul de pana acum
   max = S[j]; //actualizam valoarea maxima
 cout<<"Suma maxima pe coloane: "<<max<<endl;
 return 0;
}
```

sau, folosind o singură variabilă pentru calculul sumelor pe coloane:

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
 int m,n,i,j;
 float a[10][10],S,max;

 //citim dimensiunile si elementele matricei
 cout<<"Dati numarul de linii: "; cin>>m;
 cout<<"Dati numarul de coloane: "; cin>>n;
 cout<<"Dati elementele matricei: "<<endl;
 for (i=0;i<m;i++)
   for (j=0;j<n;j++) cin>>a[i][j];
```

```
 max = -INFINITY;
 for (j=0;j<n;j++) {  //pentru fiecare coloana
 {
  S = 0;   //suma de pe coloana j se initializeaza cu 0
  for (i=0;i<m;i++)   //se parcurge coloana j
    S += a[i][j];     //se adauga elementul curent
  cout<<"Suma elem.de pe coloana "<<j+1<<" este: "<<S<<endl;
  if(S > max) //daca S depaseste maximul de pana acum
   max = S; //actualizam valoarea maxima
 }
 cout<<"Suma maxima pe coloane: "<<max<<endl;  return 0;
}
```

Rulare:

```
Dati numarul de linii: 2 <Enter>
Dati numarul de coloane: 3 <Enter>
Dati elementele matricei:
1 2 3<Enter>
1 4 2 <Enter>
Suma elem. de pe coloana 1 este: 2
Suma elem. de pe coloana 2 este: 6
Suma elem. de pe coloana 3 este: 5
Suma maxima pe coloane: 6
```

Observație: Se poate evita folosirea vectorului *S*, prin folosirea sumei pe coloană imediat după calcul.

R9 (4-5*- suplimentar).

Enunţul problemei: Să se descrie un algoritm pentru a determina media armonică a elementelor strict pozitive de pe coloana *k* a unei matrice dreptunghiulare.

Metoda de rezolvare: De exemplu, pentru $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 0 & 5 \\ 6 & -1 & 2 \\ 1 & 1 & 1 \end{pmatrix}$, media armonică a elementelor de pe coloana k=2 este $m_{arm} = \frac{2}{\frac{1}{2}+\frac{1}{1}} = 1,(3)$. În general, $m_{arm} = \frac{n}{\frac{1}{x_1}+\frac{1}{x_2}+...+\frac{1}{x_n}}$, adică media armonică a anumitor elemente reprezintă raportul dintre numărul elementelor şi suma inverselor elementelor. Pentru o matrice de forma:

$$A = \begin{pmatrix} a_{11} & a_{12} & ... & a_{1k} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2k} & ... & a_{2n} \\ ... & ... & ... & ... & ... & ... \\ a_{m1} & a_{m2} & ... & a_{mk} & ... & a_{mn} \end{pmatrix}$$

algoritmul constă în parcurgerea coloanei *k* pentru numărarea elementelor strict pozitive şi însumarea inverselor elementelor nenule.

O descriere a algoritmului în pseudocod:

```
citeşte m,n,a11,…,amn,k
contor ← 0 *initial, numarul elem. nenule de pe col.k este 0
S ← 0 *initial, suma elementelor nenule de pe col.k este 0
pentru i = 1,m repeta        *se parcurg elementele coloanei k
 daca aik ≠ 0 atunci         *daca elementul curent este nenul
    contor ← contor + 1      *contorizam elementul curent
    S ← S + 1/aik            *insumam inversul elem. curent
```

```
scrie contor/S                  *se afiseaza valoarea raportului ce
                                *reprezinta media armonica
```

O descriere a algoritmului în pseudocod C++ (CodeBlocks):

```
#include <iostream>
#include <iomanip>
using namespace std;
int main() {
 int m,n,i,j,k,contor;
 float a[10][10],S;

 //citim dimensiunile si elementele matricei
 cout<<"Dati numarul de linii: "; cin>>m;
 cout<<"Dati numarul de coloane: "; cin>>n;
 cout<<"Dati elementele matricei: "<<endl;
 for (i=0;i<m;i++)
   for (j=0;j<n;j++) cin>>a[i][j];
 cout<<"Dati numarul coloanei: "; cin>>k;
 k--; //diferenta de indici intre numarul logic si cel din C/C++

 S = contor = 0; //initializarile contorului si sumei
 //elementelor nenule de pe coloana k
 for (i=0;i<m;i++)                    //se parcurge coloana k
  if (a[i][k]>0)            //daca elementul este pozitiv
   {
    contor++;          //se contorizeaza
    S += 1 / a[i][k]; //se insumeaza inversul elementului curent
   }
 cout<<"Media armonica a elementelor nenule de pe coloana k = "
<<fixed<<setprecision(2) << contor/S<<endl;
 return 0;
}
```

R10 (suplimentar, 3*).

Enunţul problemei: Să se scrie un algoritm pentru afişarea următoarea piramidă de numere
1
1 2
1 2 3
……
1 2 3 … *n*
pentru un *n*≥1 dat.

Metoda de rezolvare: Piramida poate fi văzută ca o matrice, cu *n* linii şi pe linia *i* fiind *i* elemente şi anume: 1, 2,..., *j* , ...*i* (*i* = 1, 2,..., *n*).

Descrierea algoritmului în pseudocod:

```
citeste n
pentru i = 1,n repeta *pentru linia i in general
  pentru j = 1,i repeta *se afiseaza componentele liniei i
     scrie j
  *se trece pe randul urmator (la urmatoarea linie)
```

Descrierea algoritmului în C++ (CodeBlocks):

```
#include <iomanip> //pt setw
#include <iostream>
using namespace std;
int main()
{
```

```
 int n,i,j;
 cout<<"n="; cin>>n;
 for (i=1;i<=n;i++)
 {
  for (j=1;j<=i;j++) cout<<setw(3)<<j;
  cout<<endl;
 }
 return 0;
}
```

Câteva explicaţii cod C++:
Afişarea valorii *j* se face pe 3 câmpuri (caractere), valoarea afişată ocupându-le de la dreapta la stânga (astfel vor apare coloanele frumos aliniate) şi eventualele spaţii rămânând libere la stânga valorii lui *j*. De aceea se foloseşte `setw` care este declarat în fişierul header `iomanip`.
Dacă am fi afişat valoarea lui *i* în loc de valoarea lui *j*, piramida ar fi fost:

1
2 2
3 3 3
….
n n n … n

R11 (suplimentar, 3*).

Enunţul problemei: Să se scrie un algoritm pentru afişarea următoarea piramidă de numere
n n-1 *n*-2 … 1
…..
i i-1 *i*-2 … 1
….
3 2 1
2 1
1 (pentru un *n*≥1 dat).

Metoda de rezolvare: Algoritmul este similar cu cel anterior doar că indicele liniei se poate considera de *n* înapoi la 1, iar pentru o linie *i* în general, *j* se ia de la *i* înapoi la 1.

Descrierea algoritmului în pseudocod:

```
citeste n
pentru i = n,1,-1 repeta
   pentru j = i,1,-1 repeta
      scrie j
   *se trece pe randul urmator (la urmatoarea linie)
```

Descrierea algoritmului în C++ (CodeBlocks):

```
#include <iomanip.h> //pt setw
#include <iostream>
using namespace std;
int main() {
 int n,i,j;
 cout<<"n="; cin>>n;
 for (i=n;i>=1;i--)
 {
  for (j=i;j>=1;j--) cout<<setw(3)<<j;
  cout<<endl;
 }
 return 0;
}
```

R12 (4-5*).

Enunţul problemei: Să se descrie un algoritm pentru a determina câte linii au cel puţin un element negativ într-o matrice dată?

Metoda de rezolvare: Se parcurge fiecare linie a matricei şi în cazul în care există un element negativ se contorizează linia. De exemplu, pentru matricea:

$$A=\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & -1 & -2 & 0 & 1 \\ -1 & 0 & 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 & -1 & 5 \\ -1 & -2 & -3 & -4 & -5 & -6 \end{pmatrix}$$

sunt 4 linii (liniile 2, 3, 4, 5) care au cel puţin un element negativ.

Descrierea algoritmului în pseudocod:

```
citeşte m,n,a11,…,amn
contor ← 0                    *initializarea contorului de linii
pentru i = 1,m repeta         *pentru fiecare linie i
  pentru j = 1,n repeta       *parcurgem linia
    daca aij < 0 atunci       *elementul curent este negativ
      contor ← contor + 1     *contorizam linia curenta
      break                   *se trece la urmatoarea linie
scrie contor
```

Descrierea algoritmului în C++:

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

int main() {
 int m,n,i,j,contor;
 float a[10][10];

 //citim dimensiunile si elementele matricei
 cout<<"Dati numarul de linii: "; cin>>m;
 cout<<"Dati numarul de coloane: "; cin>>n;
 cout<<"Dati elementele matricei: "<<endl;
 for (i=0;i<m;i++)
   for (j=0;j<n;j++) cin>>a[i][j];

 contor = 0;          //se initializeaza contorul liniilor
 for (i=0;i<m;i++)    //pentru fiecare linie i
  for (j=0;j<n;j++)   //se parcurge linia
   if (a[i][j]<0)
   {  //daca elementul curent este negativ
      contor++;       //se contorizeaza linia curenta
      break;     //iesire din "for j" => se trece la urm. linie
     }
 cout<<"Numarul liniilor cu cel putin un nr negativ: "<<contor;
 return 0;
}
```

Rulare:

```
Dati numarul de linii: 3 <Enter>
Dati numarul de coloane: 4 <Enter>
Dati elementele matricei:
1   2  3  4 <Enter>
1  -1  2  2 <Enter>
-1 -1 -1 -1<Enter>
Numarul liniilor cu cel putin un nr negativ: 2
```

R13 (4-5*).

Enunţul problemei: Să se descrie un algoritm pentru a afişa indicele liniilor ce conţin cel puţin *k* elemente negative. În cazul în care nu există astfel de linii se va afişa un mesaj.

Metoda de rezolvare: De exemplu, pentru matricea

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & -1 & 2 & -2 \\ -1 & -2 & -3 & 0 \\ 0 & -1 & 2 & 3 \\ -1 & -2 & -3 & -4 \end{pmatrix}$$

şi *k*=2 sunt 3 linii cu cel puţin *k* elemente negative. O variantă simplă a algorimului ar fi să se parcurgă liniile matricei şi să se contorizeze elementele negative, iar în cazul în care acest număr depăşeşte valoarea *k* să se afişeze indicele liniei respective. Pentru a verifica dacă nu există astfel de linii, se poate lua o variabila booleană care-şi schimbă valoarea în cazul în care s-a afişat indicele unei linii corespunzătoare. La final, după parcurgerea tuturor liniilor, dacă această variabilă nu şi-a schimbat valoarea iniţială, atunci se va afişa un mesaj.

O descriere a algoritmului în pseudocod:

```
citeşte m,n,a11,…,amn,k
există ← fals    *presupunem ca nu exista linii corespunzatoare
pentru i = 1,m repeta       *pentru fiecare linie
  contor_neg ← 0            *initializare contor pentru linia i
  pentru j = 1,n repeta     *parcurgem linia
    daca aij < 0 atunci     *elementul curent este negativ
      contor_neg ← contor_neg + 1 *contorizam elem. neg. curent
  daca contor_neg >= k atunci    *sunt cel putin k elem. neg.
    exista ← adevarat       *schimbam valoarea var. exista
    scrie i                 *afisam indicele liniei
daca exista = fals atunci
  scrie "Nu exista linii cu cel putin k elem. negative"
```

O descriere a algoritmului în C++:

```
#include <iostream>
using namespace std;
int main() {
 int m,n,i,j,k,exista,contor_neg;
 float a[10][10];

 cout<<"Dati numarul de linii: "; cin>>m;
 cout<<"Dati numarul de coloane: "; cin>>n;
 cout<<"Dati elementele matricei: "<<endl;
 for (i=0;i<m;i++)
   for (j=0;j<n;j++) cin>>a[i][j];
 cout<<"k = "; cin>>k;

 cout<<"Liniile care au cel putin k elem. neg.: ";
 exista = 0;     //presupunem ca nu exista linii corespunzatoare
 for (i=0;i<m;i++) {  //pentru fiecare linie i
   contor_neg = 0;    //initializam numarul elem. negative
   for (j=0;j<n;j++)  //se parcurge linia
     if (a[i][j]<0)contor_neg++; //contorizam elementele neg.
   if (contor_neg >= k){//avem cel putin k elem. neg. pe lin. i
     exista = 1;         //exista linii corespunzatoare
     cout<<i+1<<" ";     //afisam indicele liniei
   }
 }
 if (exista==0) cout<<"nu exista";
 return 0; }
```

O optimizare a acestui algoritm poate fi aceea în care parcurgerea liniei se continuă doar dacă nu s-au găsit deja *k* elemente negative, iar linia este corespunzătoare dacă numărul de elemente negative este egal cu *k*:

```
exista = 0;    //presupunem ca nu exista linii corespunzatoare
for (i=0;i<m;i++) {  //pentru fiecare linie i
  contor_neg = 0;    //initializam numarul elem. negative
  for (j=0; j<n && contor_neg<k; j++)      //se parcurge linia
   if (a[i][j]<0)contor_neg++;  //contorizam elementele neg.
  if (contor_neg == k) {        //avem k elem. neg. pe lin. i
   { exista = 1; cout<<i+1<<" "; }
}
```

Rulare:

```
Dati numarul de linii: 5 <Enter>
Dati numarul de coloane: 4 <Enter>
Dati elementele matricei:
 1  2  3  4 <Enter>
 0 -1  2 -2 <Enter>
-1 -2 -3  0 <Enter>
 0 -1  2  3 <Enter>
-1 -1 -1 -1 <Enter>
k = 2 <Enter>
Liniile care au cel putin k elem. neg.: 2, 3, 5
```

sau

```
Dati numarul de linii: 3 <Enter>
Dati numarul de coloane: 3 <Enter>
Dati elementele matricei:
 1  2  3 <Enter>
 0 -1  2 <Enter>
-1  2  2 <Enter>
k = 2 <Enter>
Liniile care au cel putin k elem. neg.: nu exista
```

R14 (4-5*).

Enunţul problemei: Să se descrie un algoritm pentru a determina câte linii au toate elementele egale într-o matrice dreptunghiulară.

Metoda de rezolvare: De exemplu, pentru $A = \begin{pmatrix} 1 & 1 & 2 \\ 0 & 0 & 0 \\ 1 & -1 & 1 \\ -1 & -1 & -1 \end{pmatrix}$, sunt 2 linii cu elemente egale.

În general, pentru o matrice $A \in M_{m \times n}$, algoritmul constă în parcurgerea liniilor şi determinarea faptului că linia respectivă are sau nu elementele egale (adică $a_{ij} = a_{i,j+1}$, pentru $j = 1,2,...,n-1$).

O descriere a algoritmului în pseudocod:

```
citeşte m,n,a11,…,amn
contor ← 0                      *initializare contor linii
pentru i = 1,m repeta           *pentru fiecare linie i
  OK ← adevarat                 *presupunem ca linia are elem.egale
  pentru j = 1,n-1 repeta       *parcurgem linia
    daca aij ≠ ai,j+1 atunci    *elem.curent este diferit de urmatorul
      OK ← fals                 *schimbam valoarea variabilei
      break                     *ieşim de pe linia curenta
  daca OK = adevarat atunci     *sunt cel putin k elem. neg.
    contor ← contor + 1         *contorizam linia curenta
scrie contor
```

O descriere a algoritmului în C++:

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

int main() {
 int m,n,i,j,contor,OK;
 float a[10][10];
 //citim dimensiunile si elementele matricei
 cout<<"Dati numarul de linii: "; cin>>m;
 cout<<"Dati numarul de coloane: "; cin>>n;
 cout<<"Dati elementele matricei: "<<endl;
 for (i=0;i<m;i++)
   for (j=0;j<n;j++) cin>>a[i][j];
contor = 0; //initializarea nr. de linii cu elementele egale
for (i=0;i<m;i++) {        //pentru fiecare linie i
  OK = 1;  //presupunem ca elem. liniei i sunt egale
  for (j=0;j<=n-2;j++)     //parcurgem linia i
   if (a[i][j] != a[i][j+1]) {//doua elemente vecine diferite
     OK = 0;          //linia curenta nu are elem. egale
     break;           //se iese din "for j" (de pe linia i)
    }                 //=> se trece la linia urmatoare
  if (OK==1) contor++;
  //la finalul liniei testam valoarea var. OK
 }
cout<<"Numarul liniilor cu elementele egale: "<<contor;
return 0;
}
```

sau folosind funcție

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

int m,n;
float a[10][10];

int LinieCuElemEgale(int i)
{
  for (int j=0;j<=n-2;j++) //parcurgem linia i
   if (a[i][j] != a[i][j+1]) //doua elemente vecine diferite
     return 0;        //linia curenta nu are elem. egale
  return 1;
}
int main() {
 int i,j,contor,OK;
 //citim dimensiunile si elementele matricei
 cout<<"Dati numarul de linii: "; cin>>m;
 cout<<"Dati numarul de coloane: "; cin>>n;
 cout<<"Dati elementele matricei: "<<endl;
 for (i=0;i<m;i++)
   for (j=0;j<n;j++) cin>>a[i][j];
 contor = 0; //initializarea nr. de linii cu elementele egale
 for (i=0;i<m;i++)          //pentru fiecare linie i
   if (LinieCuElemEgale(i)==1) contor++;
cout<<"Numarul liniilor cu elementele egale: "<<contor<<endl;
return 0;
}
```

Rulare:

```
Dati numarul de linii: 4 <Enter>
Dati numarul de coloane: 3 <Enter>
Dati elementele matricei:
 1  1 2 <Enter>
 0  0 0 <Enter>
 1 -1 1 <Enter>
-1 -1 -1 <Enter>
Numarul liniilor cu elementele egale: 2
```

R15 (4-5*).

Enunţul problemei: Să se descrie un algoritm pentru a determina suma elementelor prime mai mici sau egale decât k într-o matrice dreptunghiulară.

Metoda de rezolvare: De exemplu, pentru $A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 6 \\ 7 & -1 & 8 \\ -2 & 0 & 11 \end{pmatrix}$ şi $k = 7$, suma elementelor prime mai mici sau egale decât k este: 2 + 3 + 7 = 12. Algoritmul constă în parcurgerea elementelor matricei şi dacă elementul curent este prim şi este mai mic sau egal cu k atunci se însumează. Ar fi de preferat ca în cadrul unei expresii logice compuse, prima condiţie testată să fie cea cu valoarea cel mult k căci este mai puţin complexă decât testarea primalităţii.

O descriere a algoritmului în C++:

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

int Prim(int n)
{
 if (n<=1) return 0; //testam inclusiv pentru numere negative
 //else
 for (int i=2;i*i<=n;(i==2)?i=3:(i+=2))  //sau i++
   if (n%i==0) return 0;
 return 1;
}
int main()
{
 int m,n,i,j,k,a[10][10],S;
 cout<<"Dati numarul de linii: "; cin>>m;
 cout<<"Dati numarul de coloane: "; cin>>n;
 cout<<"Dati elementele matricei: "<<endl;
 for (i=0;i<m;i++)
   for (j=0;j<n;j++) cin>>a[i][j];
 cout<<"k = "; cin>>k;
 S=0;
 for (i=0;i<m;i++)
   for (j=0;j<n;j++)
     if (a[i][j]<=k && Prim(a[i][j])==1) S += a[i][j];
 cout<<"suma elementelor prime <=k: "<<S;
 return 0;
}
```

R16 (4-5*).

Enunţul problemei: Să se descrie un algoritm pentru a interschimba două linii *p* şi *q* specificate dintr-o matrice dreptunghiulară.

Metoda de rezolvare: Algoritmul constă în parcurgerea simultană a celor două linii (linia *p*: a_{p1}, a_{p2}, ..., a_{pj}, ..., a_{pn} şi linia *q*: a_{q1}, a_{q2}, ..., a_{qj}, ..., a_{qn}) dintr-o matrice cu *m* linii şi *n* coloane şi interschimbarea a câte două componente din cele două linii ($a_{pj} \leftrightarrow a_{qj}$).

O descriere a algoritmului în C++:

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

void InterschimbareLinii(int p,int q,float a[10][10],int n) {
//interschimba liniile p si q intr-o matrice cu n coloane
 float aux;
 for (int j=0;j<n;j++) {   //parcurgem simultan liniile p si q
  aux = a[p][j]; //interschimbam elementele a[p][j] cu a[q][j]
  a[p][j] = a[q][j];
  a[q][j] = aux;
 }
}
void Afisare(float a[10][10],int m,int n) {
 for (int i=0;i<m;i++) {//pentru fiecare linie
   for (int j=0;j<n;j++) cout<<setw(3)<<a[i][j]; //afisam linia
   cout<<endl;          //trecem pe randul urmator
  }
}
int main() {
 int m,n,i,j,p,q;
 float a[10][10];
 cout<<"Dati numarul de linii: "; cin>>m;
 cout<<"Dati numarul de coloane: "; cin>>n;
 cout<<"Dati elementele matricei: "<<endl;
 for (i=0;i<m;i++)
   for (j=0;j<n;j++) cin>>a[i][j];
 cout<<"p = "; cin>>p;  p--; //diferenta de indici in C/C++
 cout<<"q = "; cin>>q;  q--;
 InterschimbareLinii(p,q,a,n);
 cout<<"Matricea dupa interschimbarea liniilor p si q:"<<endl;
 Afisare(a,m,n);
 return 0;
}
```

Rulare:

```
Dati numarul de linii: 4 <Enter>
Dati numarul de coloane: 3 <Enter>
Dati elementele matricei:
1 1 1 <Enter>
2 2 2 <Enter>
3 3 3 <Enter>
4 4 4 <Enter>
p=2
q=4
Matricea dupa interschimbarea liniilor p si q:
1 1 1
4 4 4
3 3 3
2 2 2
```

R17 (4-5*).

Enunţul problemei: Să se descrie un algoritm pentru a stabili dacă o matrice pătrată este pătrat magic sau nu, adică suma pe fiecare linie, coloană au diagonale este aceeaşi.

Metoda de rezolvare: De exemplu, matricea:

$$A = \begin{pmatrix} 2 & 9 & 4 \\ 7 & 5 & 3 \\ 6 & 1 & 8 \end{pmatrix}$$

este pătrat magic, 15 fiind fiecare sumă a elementelelor de pe diagonala principală, secundară, orice linie şi orice coloană.

O descriere a algoritmului în C++:

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

float SumaDiag1(float a[10][10],int n)
{
 //calculeaza suma pe diagonala principala
 float S=0;
 for (int i=0;i<n;i++) S += a[i][i];
 return S;
}
float SumaDiag2(float a[10][10],int n)
{
 //calculeaza suma pe diagonala secundara
 float S=0;
 for (int i=0;i<n;i++) S += a[i][n-1-i];
 return S;
}


float SumaLinie(int p, float a[10][10],int n)
{
 //calculeaza suma pe linia p intr-o matrice patrata de dim.n
 float S=0;
 for (int j=0;j<n;j++) S += a[p][j];
 return S;
}
float SumaColoana(int q, float a[10][10],int n)
{
//calculeaza suma pe coloana q intr-o matrice patrata de dim.n
 float S=0;
 for (int i=0;i<n;i++) S += a[i][q];
 return S;
}
int main() {
 int n,i,j;
 float a[10][10],S;
 cout<<"Dati dimensiunea matricei: "; cin>>n;
 cout<<"Dati elementele matricei: "<<endl;
 for (i=0;i<n;i++)
   for (j=0;j<n;j++) cin>>a[i][j];
 S = SumaDiag1(a,n); //retinem in S suma elem de pe diag.princ.
 if (SumaDiag2(a,n)!=S)
 //daca suma elem. de pe diagonala secundara nu este egala cu S
      cout<<"Nu este patrat magic";
 else
```

```
       { for (int i=0;i<n;i++) //parcurgem simultan liniile si col.
         if (SumaLinie(i,a,n)!=S || SumaColoana(i,a,n)!=S)
         //suma elem. de pe linia i sau coloana i nu este egala cu S
          { cout<<"Nu este patrat magic";     break;    }
        if (i==n) //daca inainte n-au fost iesiri fortate din for
                  //atunci din for se iese cu valoarea n
         cout<<"Este patrat magic";
        }
       return 0;
      }
```

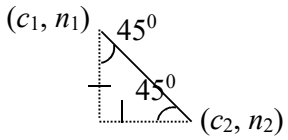
Rulare:

```
      Dati numarul de linii: 3 <Enter>
      Dati elementele matricei:
      2 9 4 <Enter>
      7 5 3 <Enter>
      6 1 8 <Enter>
      Este patrat magic
sau
      Dati numarul de linii: 3 <Enter>
      Dati elementele matricei:
      1 2 3 <Enter>
      1 2 3 <Enter>
      1 2 3 <Enter>
      Nu este patrat magic
```

R18 (5*).

Enunţul problemei: Pe o tablă de şah (cu liniile numerotate de la 1 la 8 şi coloanele de la *A* la *H*) se dau coordonatele a doi nebuni, unul alb şi unul negru. Să se descrie un algoritm pentru a stabili dacă cei doi nebuni se atacă sau nu.

Metoda de rezolvare: De exemplu, dacă un nebun este pe poziţia D3 şi al doilea în poziţia G6, aceştia se atacă, însă dacă aceştia sunt în poziţiile D3 şi G5 nu se atacă. Algoritmul constă în a stabili dacă cele două piese sunt pe o paralelă cu diagonala principală, adică formează un triunghi dreptunghic isoscel cu paralele la axele orizontală şi verticală ($|c_1-c_2| = |n_1-n_2|$).



O descriere a algoritmului în C++:

```
      #include <iostream>
      #include <math.h>
      using namespace std;
      int main()
      {
       char c1,c2;
       int n1,n2;
       cout<<"Dati pozitia primului nebun: ";
       cin>>c1>>n1;
       cout<<"Dati pozitia celui de-al doilea nebun: ";
       cin>>c2>>n2;
       if (abs(int(c1)-int(c2)) == abs(n1-n2))
        //coordonatele coloanelor se convertesc in
        //codurile ASCII corespunzatoare ('A'->65, ..., 'H'->72)
        //apoi se calculeaza diferenta dintre ele => lungime latura
```

```
         cout<<"Cei doi nebuni se pot ataca";
       else
        cout<<"Cei doi nebuni nu se pot ataca";
       return 0;
      }
```

```
Rulare:
      Dati pozitia primului nebun: D 3 <Enter>
      Dati pozitia celui de-al doilea nebun: G 6 <Enter>
      Cei doi nebuni se pot ataca
```