Facultatea de Matematica si Informatica
Universitatea Bucuresti

Tablouri

Operatii pe tablouri bidimensionale

Lectii de pregatire pentru Admitere

11 / 03 / 2017

Operatii pe tablouri bidimensionale

Cuprins

0. Tablouri unidimensionale – scurta recapitulare

C / C++	Pascal
Declarare	
int a[20]; double b[30]; char c[23];	var a : array [1..20] of integer; var b : array [1..30] of double; var c : array [1..23] of char;

double tab [100] ;

0.3	-1.2	10	5.7	...	0.2	-1.5	1
0	1	2	3		97	98	99

tab [3] = 5.7;

int a[5];

3	-12	10	7	1
0	1	2	3	4

a [1] = 3;

char b1 [34];

A	&	*	+	...	c	M	#
0	1	2	3	…			

b1 [1] = '&';

0. Tablouri unidimensionale – scurta recapitulare

Varianta C / C++

Cantitatea de memorie necesara pentru inregistrarea unui tablou este direct proportionala cu tipul si marimea sa.

tip nume [dimensiune] → **sizeof(nume) = sizeof (tip) * dimensiune ;**

```
double tab [100] ;

int a[5];

char b1 [34];

printf("Stocarea unui tablou de elemente double = %d octeti\n",sizeof(tab));
printf("Stocarea unui tablou de elemente int = %d octeti\n",sizeof(a));
printf("Stocarea unui tablou de elemente char = %d octeti\n",sizeof(b1));
```

```
C:\Users\Ank\Desktop\testSizeof\bin\Debug\testSizeof.exe
Stocarea unui tablou de elemente double = 800 octeti
Stocarea unui tablou de elemente int = 20 octeti
Stocarea unui tablou de elemente char = 34 octeti
```

0. Tablouri unidimensionale – scurta recapitulare

C / C++ | **Pascal**

Traversare (complexitate **O(n)**)

```
for (i = 0; i<n; i++)
     // viziteaza a[i];
```

```
for i:= 1 to n do
 { viziteaza a[i];}
```

Cautare (**liniara** – complexitate **O(n)**)

```
int t = 0;
for (i = 0; i<n; i++)
     if (a[i]==x) t = 1;
if (t==0) // cautare fara succes
```

```
var t : boolean;
t := false;
for i:= 1 to n do
    if (a[i] = x) then t := true;
if (t = true) then
      write('cautare fara succes');
```

0. Tablouri unidimensionale – scurta recapitulare

C / C++

Inserare (valoare **val** pe pozitia **poz**)

```
n++;
for (i = n-1; i >= poz; i--)
      a[i+1] = a[i];
a[poz] = val;
```

Pascal

```
n := n+1;
for i:= n downto poz do
     a[i+1] := a[i];
a[poz]:=val;
```

Stergere (valoare de pe pozitia **poz**)

```
for (i = poz; i<n-1; i++)
      a[i] = a[i+1];
n--;
```

```
for i := poz to n-1 do
     a[i] := a[i+1];
n:=n-1;
```

1. Tablouri bidimensionale – notiuni teoretice

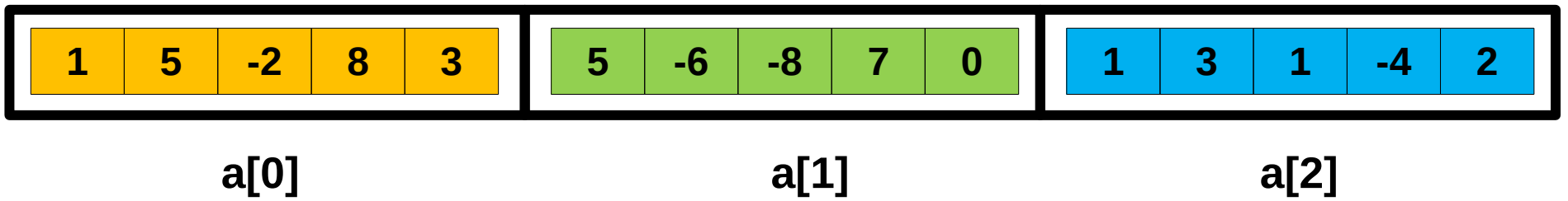
Reprezentarea matricelor in memorie = tablouri de tablouri

int a[3][5]

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

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

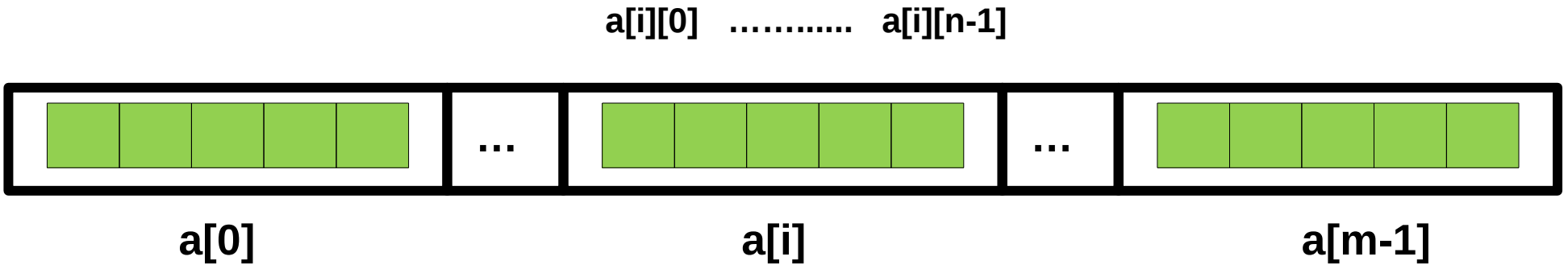
a[0][0] ……. a[0][4] a[1][0] …… a[2][4]



1. Tablouri bidimensionale – notiuni teoretice

Reprezentarea matricelor in memorie = tablouri de tablouri

Generalizare: int a[m][n]



Numele unui tablou este un pointer constant catre primul sau element

```
int v[100];  v = &v[0];
float a[4][6];  a = &a[0][0];
```

1. Tablouri bidimensionale – notiuni teoretice

Varianta C / C++

Cantitatea de memorie necesara pentru inregistrarea unui tablou este direct proportionala cu tipul si marimea sa.

tip nume [dimensiune1][dimensiune2] →

sizeof(nume) = sizeof (tip) * dimensiune1 * dimensiune2 ;

```
double tab [5][10] ;

int a[5][10];

char b1[5][10];

printf("Stocarea unui tablou de elemente double = %d octeti\n",sizeof(tab));
printf("Stocarea unui tablou de elemente int = %d octeti\n",sizeof(a));
printf("Stocarea unui tablou de elemente char = %d octeti\n",sizeof(b1));
```

```
C:\Users\Ank\Desktop\testSizeof\bin\Debug\testSizeof.exe
Stocarea unui tablou de elemente double = 400 octeti
Stocarea unui tablou de elemente int = 200 octeti
Stocarea unui tablou de elemente char = 50 octeti
```

1. Tablouri bidimensionale – notiuni teoretice

Exemplificare citire si afisare tablouri bidimensionale

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

```
cin>>n>>m;
for(i=0; i<n; i++)
   for(j=0; j<m; j++)
      cin>>a[i][j];
```

```
for(i=0; i<n; i++)
{
   for(j=0; j<m; j++)
      cout<<a[i][j]<<" ";
   cout<<endl;
}
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.1 – Puncte “șa”

Se citeste o matrice cu n linii si m coloane, cu elemente distincte. Sa se afisaeze punctele “șa” din matrice. Un punct “șa” este acel element care este minim pe linia sa si maxim pe coloana.

int a[3][5]

	0	1	2	3	4
0	10	5	-2	8	33
1	4	-8	-6	7	0
2	3	13	2	4	9

Punct “**șa**”: a[2][2]

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.1 – Puncte “șa”

```
int poz_min_linie(int lin, int m)
{
    int j,p=0;
    for(j=1; j<m; j++)
        if (a[lin][j]<a[lin][p])
            p = j;
    return p;
}
```

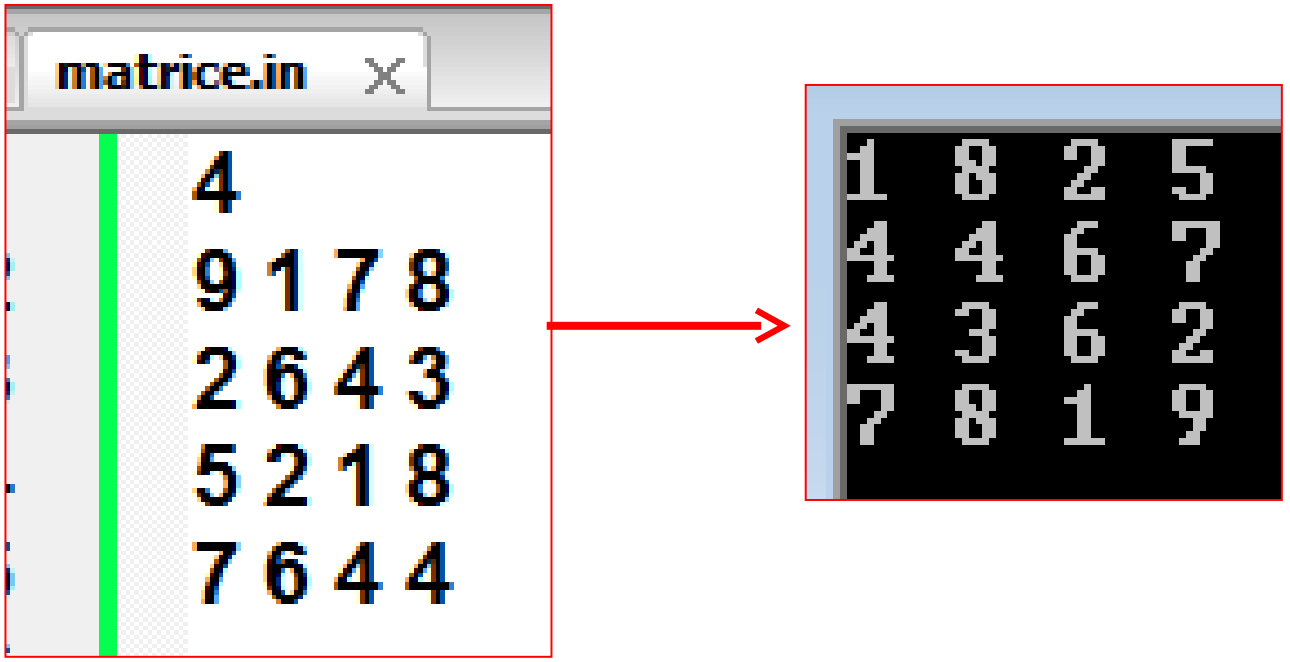
```
int poz_max_coloana(int col, int n)
{
    int i,p=0;
    for(i=1; i<n; i++)
        if (a[i][col]>a[p][col])
            p = i;
    return p;
}
```

```
for(i=0; i<n; i++)
{
    j = poz_min_linie(i,m);
    if (i==poz_max_coloana(j,n))
        cout<<i<<" "<<j<<'\n';
}
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.2 – Ordonare diagonala

Se citeste o matrice patratica de dimensiune n x n. Să se sorteze crescator valorile aflate pe diagonala principală, astfel incat, pe fiecare linie si pe fiecare coloana sa ramana aceleasi valori (intr-o alta ordine).



2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.2 – Ordonare diagonala

```
void swap_linie(int x, int y)
{
    int i, aux;
    for(i=0; i<n; i++)
    {
        aux = a[x][i];
        a[x][i] = a[y][i];
        a[y][i] = aux;
    }
}
```

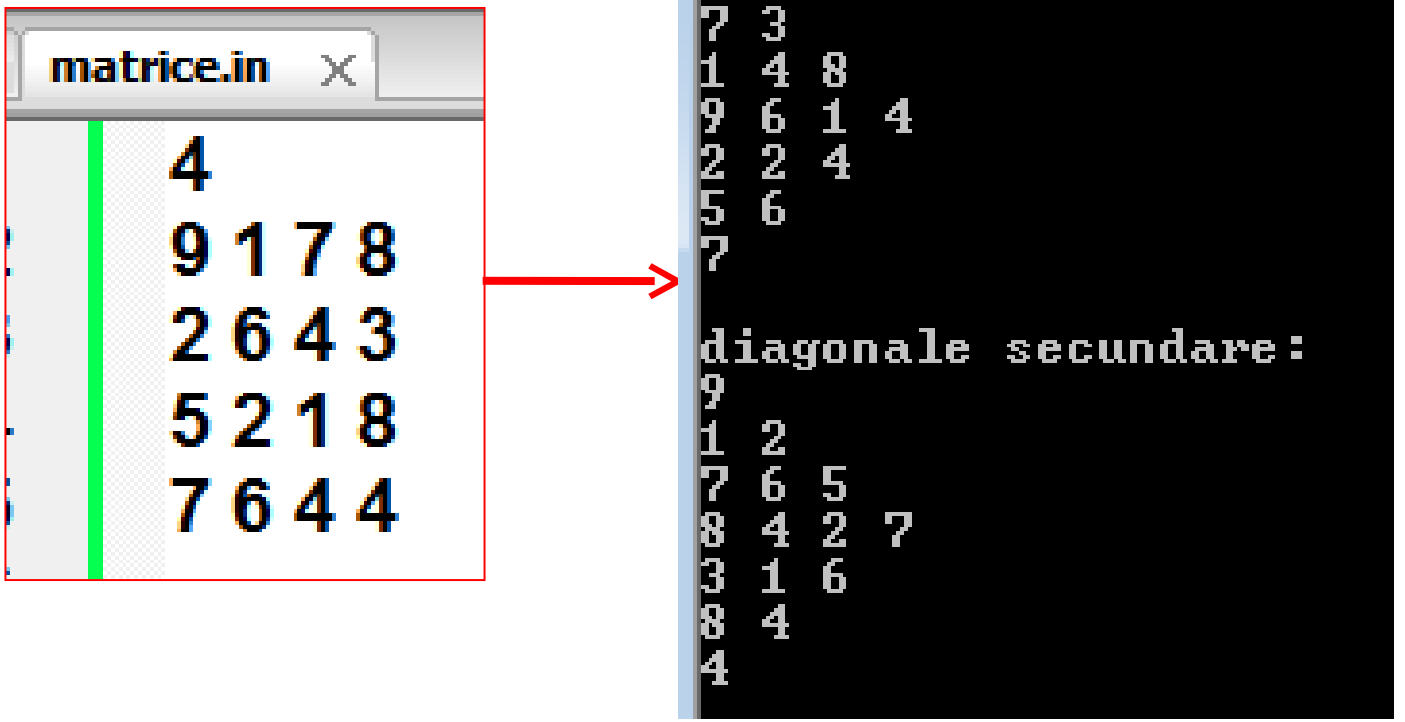
```
void swap_coloana(int x, int y)
{
    int i, aux;
    for(i=0; i<n; i++)
    {
        aux = a[i][x];
        a[i][x] = a[i][y];
        a[i][y] = aux;
    }
}
```

```
for(i=0; i<n-1; i++)
{
    minim = i;
    for(j = i+1; j<n; j++)
        if (a[j][j] < a[minim][minim])
            minim = j;
    swap_linie(i,minim);
    swap_coloana(i,minim);
}
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.3 – Diagonalele si paralelele acestora

Se citeste o matrice patratica de dimensiune n x n. Să se afiseze elementele de pe diagonalele matricei si de pe liniile paralele acestora.



2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.3 – Diagonalele si paralelele acestora

O(n^3)

```
cout<<"diagonale principale:\n";
 /// O(n^3)
for (int k = 0; k<2*n-1; k++)
{
   for(i=0; i<n; i++)
      for(j=0; j<n; j++)
         if (j-i == n - k - 1) cout<<a[i][j]<<" ";
   cout<<'\n';
}
```

```
/// O(n^3)
for (int k = -n; k<n; k++)
{
   for(i=0; i<n; i++)
      for(j=0; j<n; j++)
         if (i+j == n + k) cout<<a[i][n-i+k]<<" ";
   cout<<endl;
}
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.3 – Diagonalele si paralelele acestora

O(n^2)

Diagonala principala

```
for (int k = -n; k<0; k++)
{
   for(i=0; i<n+k+1; i++)
      cout<<a[i][n-i+k]<<" ";
   cout<<'\n';
}

for (int k = 0; k<n; k++)
{
   for(i=k+1; i<n; i++)
      cout<<a[i][n-i+k]<<" ";
   cout<<'\n';
}
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.3 – Diagonalele si paralelele acestora

O(n^2)

Diagonala secundara

```
for (int k = 0; k<n; k++)
{
   for(i=0; i<k+1; i++)
         cout<<a[i][n - k - 1 + i]<<" ";
   cout<<'\n';
}

for (int k = 0; k<n; k++)
{
   for(i=k+1; i<n; i++)
         cout<<a[i][i - k -1]<<" ";
   cout<<'\n';
}
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.4 – Atac regine

Cerința

Pe o tablă de șah de dimensiune n se află m regine. O regină atacă o altă regină dacă cele două se află pe aceeași linie, coloană sau diagonală și între ele nu se află alte regine. Determinați numărul maxim p de regine care sunt atacate de o aceeași regină și numărul q de regine care atacă p alte regine.

Date de intrare

Fișierul de intrare `regine.in` conține pe prima linie numerele n m; următoarele m linii conțin perechi i j reprezentând linia și coloana unde se află poziționată o regină.

https://www.pbinfo.ro/?pagina=probleme&id=602

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.4 – Atac regine

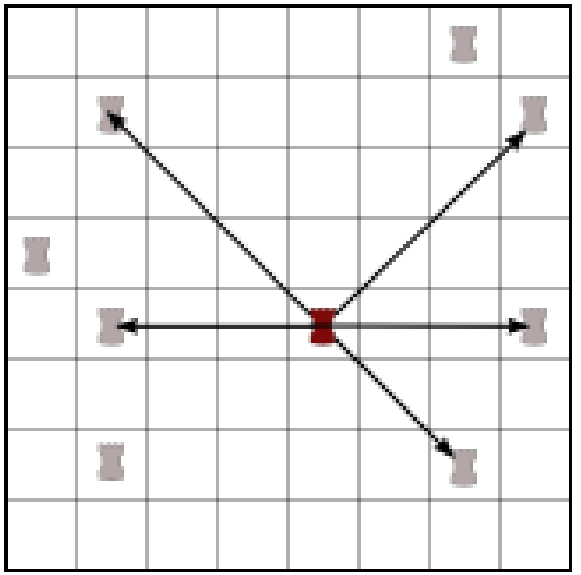
regine.in

```
8 9
1 7
2 2
2 8
4 1
5 2
5 5
5 8
7 2
7 7
```

regine.out

```
5 1
```

Explicație



https://www.pbinfo.ro/?pagina=probleme&id=602

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.4 – Atac regine

O(n^2)

- Daca numarul de regine este mult mai mare decat n

Varianta:

1. Se defineste o matrice atac, asemanatoare unei matrice de adiacenta

2. Pentru fiecare regina, numarul de regine cu care se ataca este dat de suma elementelor pe linie;

3. Se cauta maximul si numarul de aparitie al maximului valorilor calculate la punctul 2.

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.4 – Atac regine

O(n^2)

- Daca numarul de regine este mult mai mare decat n

```
int fault (int x1, int y1, int x2, int y2)
{
   int zerouri;
   if (x1 == x2)
   {
      if ((y1<y2) && (zerouri_coloana(x1,y1,y2)==0)) return 1;
      if ((y2<y1) && (zerouri_coloana(x1,y2,y1)==0)) return 1;
   }
   else
      if (y1 == y2)
      {
         if ((x1<x2) && (zerouri_linie(y1,x1,x2)==0)) return 1;
         if ((x2<x1) && (zerouri_linie(y1,x2,x1)==0)) return 1;
      }
      else
         if ((fabs(x1-x2) == fabs(y1-y2)) &&
            (zerouri_diag(x1,x2,y1,y2) == 0)) return 1;
   return 0;
}
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.4 – Atac regine

O(n^2)

- Daca numarul de regine este mult mai mare decat n

```
int zerouri_coloana(int lin, int a1, int b1)
{
    int i,zerouri = 0;
    for (i = a1+1; i<b1; i++)
        zerouri += a[lin][i];
    return zerouri;
}

int zerouri_linie(int col, int a1, int b1)
{
    int i,zerouri = 0;
    for (i = a1+1; i<b1; i++)
        zerouri += a[i][col];
    return zerouri;
}
```

```
int zerouri_diag(int a1, int b1, int a2, int b2)
{
    int i,zerouri = 0;
    if (a1<a2)
        if (b1<b2)
            for (i = 1; i<a2-a1; i++)
                zerouri += a[a1+i][b1+i];
        else
            for (i = 1; i<a2-a1; i++)
                zerouri += a[a1+i][b1-i];
    else
        if (b1<b2)
            for (i = 1; i<a1-a2; i++)
                zerouri += a[a2+i][b2-i];
        else
            for (i = 1; i<a1-a2; i++)
                zerouri += a[a2+i][b2+i];
    return zerouri;
}
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.4 – Atac regine

O(n^2)

- Daca numarul de regine este mult mai mare decat n

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

dama d[20];
```

```
for(i=1; i<m; i++)
   for(j=i+1; j<=m; j++)
      if (fault(d[i].x,d[i].y,d[j].x,d[j].y)==1) atac[i][j] = atac[j][i] = 1;

for(i=1; i<=n; i++)
{
   s[i] = 0;
   for(j=1; j<=n; j++)
      s[i]+=atac[i][j];
}

maxim_aparitii(s,n,p,q);
cout<<p<<" "<<q;
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.4 – Atac regine

O(m^2)

- Daca numarul de regine este mult mai mic decat n

Varianta:

1. Se defineste o structura “Dama” in care campurile inregistreaza valorile citite din fisierul de intrare;

2. Se ordoneaza crescator, folosind QSORT, vectorul de dame, in functie de coordonata x;

3. Se verifica atacurile fiecarei dame cu cele care-i urmeaza, programul oprindu-se, pentru fiecare mod de atac, la prima dama aflata in situatia descrisa.

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.4 – Atac regine

O(m^2)

- Daca numarul de regine este mult mai mic decat n

```
struct regina
{
    int x,y;
} d[20];;

int cmp (const void *a, const void* b)
{
    regina *r1 = (regina *)a;
    regina *r2 = (regina *)b;
    if(r1->x == r2->x) return r1->y - r2->y;
    return r1->x - r2->x;
}
```

qsort(d,m,sizeof(regina),cmp);

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.4 – Atac regine

O(m^2)

```
for(i=0; i<m-1; i++)
{
   rx = ry = rdiag = 0;
   for(j=i+1; j<m; j++)
   {
      if (d[i].x == d[j].x && rx == 0)
      {
         atac[i]++;
         atac[j]++;
         rx = 1;
      }
      else if (d[i].y == d[j].y && ry == 0)
      {
         atac[i]++;
         atac[j]++;
         ry = 1;
      }
      else if ((fabs(d[j].x - d[i].x) == fabs(d[j].y - d[i].y)) && rdiag == 0)
      {
         atac[i]++;
         atac[j]++;
         rdiag = 1;
      }
   }
}
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.5 – Exista valoare in matrice

Se consideră o matrice în care fiecare linie și fiecare coloană este sortată strict crescător.
Să se determine dacă o valoare dată se găsește în matrice sau nu.

1	2	3	4	5
4	6	8	10	12
5	7	12	24	29

Valoarea 10 se afla in matrice.
Valoarea 9 nu se afla in matrice

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.5 – Exista valoare in matrice

O(n^2)

Cautarea standard a unui element intr-o matrice

```
void rezolvare1(int &gasit, int x)
{
    int i,j;
    gasit = 0;
    for(i=0; i<n; i++)
        for(j=0; j<m; j++)
            if (a[i][j] == x)
                gasit = 1;
}
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.5 – Exista valoare in matrice

O(n * log m)

Cautarea unui element intr-o matrice folosind, pe fiecare linie, cautarea binara, intrucat fiecare linie e ordonata crescator.

```
void rezolvare2(int &gasit, int x)
{
    gasit = 0;
    for(i=0; i<n; i++)
    {
        if (caut_binar(a[i],0,m-1,x)!= -1) gasit = 1;
    }
}
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.5 – Exista valoare in matrice

O(n * log m)

Cautarea unui element intr-o matrice folosind, pe fiecare linie, cautarea binara, intrucat fiecare linie e ordonata crescator.

Optimizare: ignorarea acelor linii care sigur nu contin elementul x (cele care incep cu elemente > x si cele care se termina cu elemente < x)

```
void restrictionare_linii(int &j, int &k)
{
    for (k = n-1; k>=0; k--)
        if (a[k][0] <= x) break;

    for (j = 0; j<=k; j++)
        if (a[j][m-1] >= x) break;
}
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.5 – Exista valoare in matrice

O(m+n)

Parcurgerea alternativa a liniilor si coloanelor pana cand se gaseste numarul sau se atinge limita unei linii sau a unei coloane.

```
void rezolvare4(int gasit&, int x)
{
   int j,k,linie,coloana;

   restrictionare_linii(j,k);
// toate liniile incep cu elem <= x si se termina cu elem >=x

   gasit = 0;
   linie = j;
   coloana = m-1;
   while (gasit == 0 && coloana >=0)
   {
      while (a[linie][coloana]>x) coloana--;
      if (a[linie][coloana] == x){gasit = 1;break;}
      while (linie <= k && a[linie][coloana]<x) linie++;
      if (linie > k) break;
      if (a[linie][coloana] == x){gasit = 1;break;}
   }
}
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.6 – Element comun tuturor liniilor (daca exista)

Se consideră o matrice în care fiecare linie este sortată crescător. Să se determine un element comun tuturor liniilor (dacă există).

Subprograme folosite

```
int max_col(int col)
{
    int i, maxim = a[0][col];
    for(i=1;i<n;i++)
        if (a[i][col] > maxim)
        maxim = a[i][col];
    return maxim;
}
```

```
int min_col(int col)
{
    int i, minim = a[0][col];
    for(i=1;i<n;i++)
        if (a[i][col] < minim)
        minim = a[i][col];
    return minim;
}
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.6 – Element comun tuturor liniilor (daca exista)

O(n^3)

Cautarea fiecarui element de pe prima linie pe toate celelalte.

Obs. Optimizare prin gasire "intervalului de intersectie" a liniilor.

```
void rezolvare1()
{
    int poz_ls[10], poz_ld[10], comun, ls, ld, i, poz, val;
    ls = max_col(0);
    ld = min_col(m-1);
    for (i=0; i<n; i++)
    {
        poz = 0;
        while (poz < m && a[i][poz]<ls) poz++;
        if (poz == m) {cout<<"nu exista valoare comuna"; return;}
        poz_ls[i] = poz;

        poz = m - 1;
        while (poz >= 0 && a[i][poz]>ld) poz--;
        if (poz < 0) {cout<<"nu exista valoare comuna"; return;}
        poz_ld[i] = poz;
    }

        for (poz = poz_ls[0]; poz <= poz_ld[0]; poz++)
        {
        comun = 1; val = a[0][poz];
        for (i = 1; i<n; i++)
            if (gasit(a[i],poz_ls[i],poz_ld[i],val) == 0){comun = 0; break;}
        if (comun == 1) {cout<<"valoare comuna = "<<val; return;}
        }
    cout<<"nu exista valoare comuna"; return;
}
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.6 – Element comun tuturor liniilor (daca exista)

Optimizarea algoritmului anterior prin folosirea cautarii binare: O(n^2 log n)

Puteti gasi o solutie in O(n^2) ? (Tema de gandire)

- “Hint” : urmariti problemele anterioare si incercati sa ajungeti la “iesirea” din matrice. Nu uitati ca elementele pe fiecare linie sunt ordonate!

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.7 – Algoritmul lui Lee

-determina numarul minim de pasi, pentru a ajunge din punctul x in punctul y in anumite conditii (de exemplu: evitand obstacole)

-parcurgere in latime (BFS) a unui graf, aplicat pentru o grila

-complexitatea de O(M*N)

- utilizat in problemele in care apare un labirint

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.7 – alternativa mai ineficienta la algoritmul lui Lee

Varianta 1 - drumul minim de la un nod la toate celelalte

```
void drum(int i)
{
    if(t[i]!=0)
        drum(t[i]);
    cout<<i<<" ";
}
```

```
void lanturi()
{
    if(ic<=sc)
    {
        prim=c[ic];
        for(int k=1; k<=n; k++)
            if(a[prim][k]==1&&viz[k]==0)
            {
                sc++;
                c[sc]=k;
                viz[k]=viz[prim]+1;
                t[k]=prim;
            }
        ic++;
        lanturi();
    }
}
```

```
ic=sc=1;
c[ic]=x;
viz[x]=1;
lanturi();
cout<<"lantul minim are lungimea "<<viz[y]<<" ";
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.7 – Algoritmul lui Lee

– labirintul (explicatie a algoritmului, vizual si implementare – link-ul de mai jos – credit d-na prof. Cerchez si co-autorii).

Într-un labirint se află un şoricel şi o bucată de caşcaval. Şoricelul doreşte să ajungă la caşcaval efectuând un număr minim de paşi.

La un pas şoricelul se poate deplasa în una dintre poziţiile învecinate (sus, jos, stânga, dreapta), evident dacă acolo este culoar de trecere.

Determinaţi numărul minim de poziţii pe care şoricelul trebuie să le parcurgă pentru a ajunge la caşcaval.

http://www.ler.is.edu.ro/~ema/proiecte/soft/2012/lee/index.html

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.7 – Algoritmul lui Lee

```
struct pozitie
{
    int lin, col;
};

pozitie c[100];
int Lab[10][10],n,m;
int dir_lin[4] = {-1, 0, 1, 0}
int dir_col[4] = {0, 1, 0, -1}

prim = ultim = 0;
c[0] = start;
Lab[start.lin][start.col] = 1;
```

```
// bordare matrice

while (prim<=ultim && Lab[curent.lin][curent.col] == 0)
{
    p = c[prim];
    prim ++;
    for(k=0; k<4; k++)
    {
        v.lin = p.lin + dir_lin[k];
        v.col = p.col + dir_col[k];

        if (Lab[v.lin][v.col] == 0)
        {
            Lab[v.lin][v.col] = Lab[p.lin][p.col] + 1;
            ultim ++;
            c[ultim] = v;
        }
    }
}

if (Lab[curent.lin][curent.col] == 0) cout<<"Nu poate iesi";
else cout<<"A iesit";
```

2. Tablouri bidimensionale – Aplicatii

Aplicatie 2.7 – Algoritmul lui Lee

S 1	0 2		0 10	0 11	0 12	0 13
0 2	0 3		0 9		0 13	0 14
0 3	0 4		0 8		0 14	C 15
0 4	0 5	0 6	0 7		0 15	0
0 5	0 6	0 7	0 8		0	0