

Tablouri bidimensionale

Capitolul 12

- ❖ Definiție
- ❖ Operații cu tablouri bidimensionale
- ❖ Careuri magice
- ❖ Implementări sugerate
- ❖ Probleme propuse
- ❖ Soluțiile problemelor

Am văzut că într-un tablou unidimensional se pot păstra mai multe valori de același tip. Fie, de exemplu, mediile generale ale tuturor elevilor unei clase. Toate mediile (numere reale) se denumesc cu același identificator, de exemplu *MedGen*, iar un element în această grupare se găsește pe baza poziției ocupate în grupul de valori, aranjate liniar. Dacă privim șirul numelor, acestea, la rândul lor formează un alt șir, în care fiecare element este un șir de caractere.

Exemplu

<i>MedGen</i>	Poziție	1	2	...	30
	Valoare	9.33	8.20	...	9.63

<i>Nume</i>	Poziție	1	2	...	30
	Valoare	Albu	Ban	...	Zeciu

Corespunzător acestui exemplu ne puteam referi la nota unui elev prin *MedGen[nr]*, unde *nr* este numărul de ordine din catalog al elevului respectiv. Această valoare este indicele elementului în șir.

Ne propunem să referim toate mediile pe discipline ale fiecărui elev printr-o singură variabilă. Conform catalogului (cunoscut din școală) avem următoarele date:

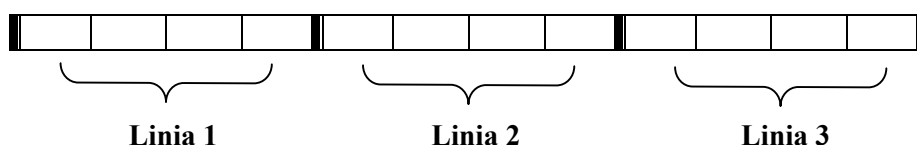
		1 (Limba română)	2 (Matematică)	...	12 (Educație fizică)
1	Albu	7.33	9.20	...	10
2	Ban	6.67	7.33	...	10
...
30	Zeciu	9.10	10

Mediile tuturor elevilor unei clase la toate disciplinele studiate pot fi păstrate într-o singură structură de date în care valorile sunt aranjate liniar pe linii și coloane (o linie corespunde unui elev, o coloană corespunde unei discipline studiate). În exemplul de mai sus, linia 2 corespunde elevului Ban, în coloana 12 sunt scrise mediile la educație fizică.

Tabloul bidimensional este o structură de date în care fiecărui element îi este asociată o pereche de indici, primul precizând numărul de ordine al liniei în care se află elementul, iar cel de-al doilea – numărul de ordine al coloanei. Media aflată pe linia i și coloana j se va nota cu $Medie[i, j]$ și reprezintă media elevului i la disciplina j . Oricărui tablou bidimensional i se alocă spațiu de memorie într-o zonă continuă de memorie, elementele aflându-se în locații succesive de memorie așezate linie după linie.

Exemplu

Un tablou bidimensional având trei linii și patru coloane se va reprezenta în memorie astfel:



Vom da definiția tabloului bidimensional^{*)}, în caz general:

12.1. Definiție

Fie $M = \{1, 2, \dots, m\}$ și $N = \{1, 2, \dots, n\}$ mulțimea primelor m , respectiv n numere naturale nenule. Considerăm o mulțime de elemente E ale căror tip de bază este T .

Se numește tablou bidimensional de dimensiune $m \times n$, având elemente de tipul T , o funcție $A: M \times N \rightarrow E$.

Notăm elementele cu a_{ij} , unde $i \in M, j \in N$. Aceste elemente sunt aranjate pe m linii și n coloane, astfel încât acele elemente care au indicele de linie i , sunt plasate pe aceeași linie i , iar cele având același indice de coloană j , pe aceeași coloană j :

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

^{*)} Noțiunea de tablou bidimensional derivă din noțiunea de matrice (cunoscută din matematică).

Cazuri particulare

1) Dacă $n = 1$, avem un tablou de dimensiuni $m \times 1$ care se numește tablou *coloană* și este de forma:

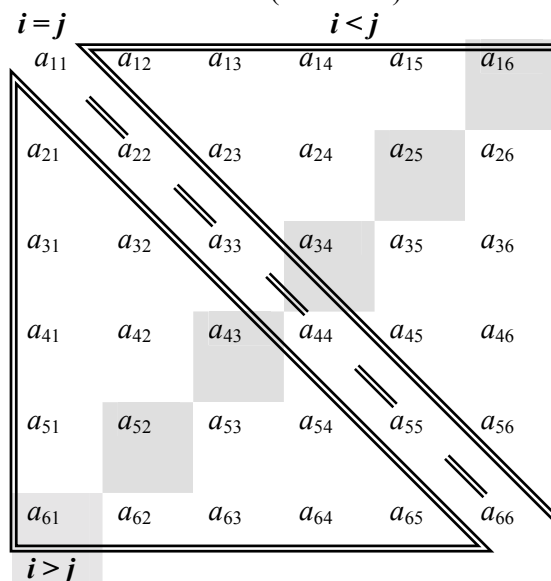
$$A = \begin{pmatrix} a_{11} \\ a_{21} \\ \dots \\ a_{m1} \end{pmatrix}$$

2) Dacă $m = 1$, tabloul de dimensiune $1 \times n$ se numește tablou *linie* și este de forma:
 $A = (a_{11}, a_{12}, \dots, a_{1n})$

3) Dacă $m = n$, avem un tablou de dimensiune $n \times n$, numit tablou *pătratic*.

Exemplu

Fie un tablou pătratic de dimensiune 6 ($m = n = 6$).



- Elementele în cazul cărora indicele de linie coincide cu indicele de coloană ($a_{11}, a_{22}, \dots, a_{nn}$) formează *diagonala principală* a tabloului.
- Paralelele la diagonala principală a tabloului vor atinge acele elemente în cazul cărora $i = j + k$, unde $-n + 1 < k < n - 1$. Pentru fiecare valoare a lui k , variind j , obținem o *paralelă la diagonala principală*.
- Elementele de *deasupra diagonalei principale* satisfac proprietatea $i < j$, iar cele care se situează *sub diagonala principală* au proprietatea că indicii lor satisfac relația $i > j$.

- În mod similar, șirul de elemente $a_{1,n}, a_{2,n-1}, \dots, a_{n,1}$ formează *diagonala secundară* a tabloului. În cazul acestora, indicii elementelor respectă relația $i + j = n + 1$.
- În cazul elementelor aflate *deasupra diagonalei secundare* indicii au proprietatea $i + j < n + 1$, iar cele care se află *sub diagonala principală* au indicii cu proprietatea $i + j > n + 1$.

Observăm că pentru a parcurge doar elementele de sub diagonala principală, indicii de linie și de coloană variază astfel încât să acopere un „triunghi dreptunghic” din tablou. Acest triunghi are ipotenuza sub diagonala principală și cele două catete sunt coloana 1 și linia n . În concluzie, la acest triunghi de pe linia i vor participa elementele de pe coloanele 1, 2, ..., $i - 1$.

12.2. Operații cu tablouri bidimensionale

În algoritmi vom prelucra tablouri bidimensionale în cele mai variate moduri. Vom face distincție între operații care accesează tabloul ca pe un întreg și operații care accesează elementele tabloului. Dar înainte de toate un tablou trebuie declarat.

12.2.1. Declararea tablourilor bidimensionale

Declarația are următoarea formă generală:

```
type TipTab=array[tip_indice1, tip_indice2] of tip_de_bază;  
var tablou:TipTab;
```

unde prin *tip_indice*₁, *tip_indice*₂ se înțelege tipul valorilor din care se alimentează indicii (este obligatoriu un tip ordinal), iar *tip_de_bază* este tipul elementelor tabloului. Acest tip de bază poate fi orice: putem avea elemente de orice tip numeric, de caractere sau șiruri de caractere, valori booleene, înregistrări (vezi tipul **record**), tablouri de cele mai diverse tipuri etc.

*tip_indice*₁, *tip_indice*₂ pot fi identificatori de tip predefinit sau definit de utilizator și, de asemenea, pot fi expresii, cu mențiunea că în acestea pot interveni doar constante și constante simbolice:

Exemplu

```
const n=5;  
var TipTablou=array[1..2*n,1..n] of Byte;  
    Numere=array[Boolean,Byte] of Integer;
```

Dacă structura unui tablou este descrisă în secțiunea **var**, atunci el va avea un tip *anonim*. Asociind tipului respectiv un identificator de tip într-o declarație **type**, acesta va putea fi folosit în program, oriunde vrem să referim acest tip.

Reamintim că două tablouri declarate în două declarații anonime diferite vor fi de tipuri diferite, chiar dacă în ele s-a descris aceeași structură.

Exemplu

```

type TipTablou=array[1..10,1..5] of Byte;
var x,y:TipTablou;
    a:array[1..10,1..5] of Byte;
    b,c:array[1..10,1..5] of Byte;

```

În programul care conține aceste declarații, tablourile x și y vor fi considerate de același tip (`TipTablou`), b și c , de asemenea vor avea un tip comun (anonim), iar a va fi de tip anonim diferit de `TipTablou`, respectiv de tipul anonim al lui b și c .

12.2.2. Citirea tablourilor bidimensionale

În programe, de regulă, prima operație va fi citirea unui astfel de tablou. În limbajele Pascal și C nu se poate citi o variabilă de tip tablou. Această operație se va realiza element cu element.

```

Subalgoritm Citire(n,x) :
    citește m,n
    pentru i=1,m execută:
        pentru j=1,n execută:
            citește x[i,j]
        sfârșit pentru
    sfârșit pentru
sfârșit subalgoritm

```

{ tabloul x are m linii și n coloane }

12.2.3. Afişarea tablourilor bidimensionale

Afișarea tablourilor se realizează asemănător. Dacă dorim să realizăm o afișare linie după linie a tabloului `x` de tipul `tablou`, având `m` linii și `n` coloane, în Pascal vom scrie procedura în felul următor:

```

procedure Afişare(m,n:Byte; x:tablou);
begin
    for i:=1 to m do begin
        for j:=1 to n do
            Write(x[i,j], ' ');
        WriteLn                                     { trecem la linie nouă }
    end
end;

```

12.2.4. Atribuirea

Atribuirea la nivel de tablou este permisă doar dacă identificatorii menționați în partea stângă și cea dreaptă a operației de atribuire au același tip. De exemplu, conform declarațiilor din exemplul menționat, x poate primi valoarea lui y , iar b poate lua valoarea lui c . Putem scrie în program: $x := y$; și $b := c$; dar nu putem scrie $a := b$;

Ținând cont de faptul că în multe aplicații va fi necesară efectuarea unor operații cu elementele tablourilor bidimensionale, asemănător operațiilor din teoria matricelor în matematică, vom prezenta câteva din aceste operații.

12.2.5. Adunarea matricelor

Fie A și B două matrice de dimensiune $m \times n$:

$$A = (a_{ij}), 1 \leq i \leq m, 1 \leq j \leq n,$$

$$B = (b_{ij}), 1 \leq i \leq m, 1 \leq j \leq n.$$

Definim matricea

$$C = (c_{ij}), 1 \leq i \leq m, 1 \leq j \leq n,$$

ale cărei elemente sunt date de egalitățile:

$$c_{ij} = a_{ij} + b_{ij} \quad (*)$$

oricare ar fi $i = 1, 2, \dots, m, j = 1, 2, \dots, n$.

Matricea C se numește **suma** dintre matricele A și B și se notează

$$C = A + B.$$

În program vom realiza adunarea element cu element, conform formulei (*).

Menționăm că adunarea a două matrice este comutativă: $A + B = B + A$ și este asociativă: $(A + B) + C = A + (B + C)$. Matricea *element neutru* față de adunare se numește matrice *nulă* și are toate elementele egale cu 0. Se va nota: 0_{mn} .

O matrice formată din elementul 0, mai puțin pe diagonala principală, unde are numai valori egale cu 1, se numește matrice *unitate* și se notează cu I_n .

12.2.6. Matricea opusă

Oricare ar fi matricea A , există o matrice notată cu $-A$ astfel încât:

$$A + (-A) = (-A) + A = 0_{mn}$$

12.3. Careuri magice

Există o bogată tradiție a pătratelor magice. Referirile la primele pătrate magice se pierd în negura timpurilor.

Definiție

Un careu de dimensiune n care conține numere de la 1 la n^2 se numește magic, dacă suma numerelor de pe oricare linie a careului este egală cu suma numerelor de pe ori-

care coloană și sunt egale cu suma numerelor aflate pe oricare dintre cele două diagonale. Această valoare se numește *sumă magică* și se calculează cu formula:

$$\text{sumă_magică}(n) = n \cdot (n^2 + 1)/2.$$

Prezentăm trei algoritmi diferiți pentru generarea pătratelor magice, în funcție de proprietatea lui n :

- n este impar;
- n este multiplu de 4;
- n este par dar nu este multiplu de 4.

12.3.1. Pătratul magic de ordin impar

În cartea lui *Feng Shui* (*Arta vieții în armonie cu natura*), precum și în cartea lui *Yi Jing* (*Cartea schimbărilor*) autorii consideră că pătratul magic descoperit de *Wu Hsia* a fost imaginat pe carapacea unei broaște țestoase. Acest pătrat magic este atribuit civilizației chineze dintre anii 2858 – 2738 înainte de Christos:

6	1	8
7	5	3
2	9	4

Traseul obținut prin parcurgerea în ordine a numerelor din acest careu magic este cunoscut sub numele de „*drumul împăratului Ven*”. Urmărind acest traseu, putem deduce modul de generare al oricărui careu magic de ordin impar. Se observă următoarele reguli:

- Pornim din poziția având indicii 1 și $\lceil n/2 \rceil$ (parte întreagă superioară).
- Dacă suntem pe prima linie, drumul continuă pe ultima linie și pe coloana precedentă, dacă există o astfel de coloană.
- Dacă suntem pe prima coloană, drumul continuă pe ultima coloană și pe linia precedentă, dacă există o astfel de linie.
- În rest încercăm să avansăm pe diagonală (paralel cu diagonala principală) spre stânga-sus, dacă avem poziție liberă în careu în acea direcție. Când nu mai putem avansa astfel, coborâm cu o linie pe aceeași coloană.

Exemplu $n = 5$,

pașii 1, 2, 3, 4:

		1		
				4
3				
	2			

pașii până la pasul 17:

15	8	1		17
16	14	7	5	
		13	6	4
3			12	10
9	2			11

12.3.2. Pătratul magic de ordin multiplu de 4Pentru a genera un pătrat magic de ordin n (n multiplu de 4), se aplică următorii pași:

- Se generează pătratul de dimensiune n în care așezăm numerele de la 1 la n^2 în ordine pe linii.
- Elementele aflate pe linii având număr de ordine pentru care restul împărțirii la 4 este 0 sau 1 și coloane care nu au această proprietate, sau pe coloane având număr de ordine pentru care restul împărțirii la 4 este 0 sau 1 și linii care nu au această proprietate, vor fi schimbate cu elementele aflate în poziții simetrice față de mijlocul careului, așa cum se poate observa din figura următoare:

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

1	63	62	4	5	59	58	8
56	10	11	53	52	14	15	49
48	18	19	45	44	22	23	41
25	39	38	28	29	35	34	32
33	31	30	36	37	27	26	40
24	42	43	21	20	46	47	17
16	50	51	13	12	54	55	9
57	7	6	60	61	3	2	64

12.3.3. Pătrat magic de ordin n , unde $n = 2 \cdot m$ (m număr impar)Algoritmul de generare a unui pătrat magic de ordin n , unde $n = 2 \cdot m$ și m este număr impar, este o metodă elegantă, ajunsă prima dată în Europa prin *Siam*.

Acest algoritm se aplică conform următorilor pași:

- Se calculează valoarea lui m ;
- Se generează o matrice de litere (construcție ajutătoare) astfel:
 - primele $[m / 2] + 1$ linii se completează cu litera „L”;
 - linia următoare se completează cu litera „U”;
 - restul liniilor se completează cu litera „X”;
 - litera „L” aflată în mijlocul careului se interschimbă cu litera „U” aflată sub ea.

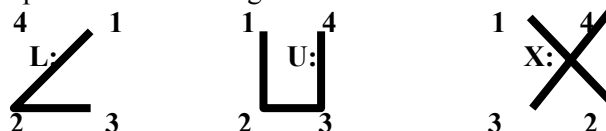
Exemplu

Dacă $n = 10$, avem $m = 5$.

L	L	L	L	L
L	L	L	L	L
L	L	L	L	L
U	U	U	U	U
X	X	X	X	X

L	L	L	L	L
L	L	L	L	L
L	L	U	L	L
U	U	L	U	U
X	X	X	X	X

- pătratul magic de dimensiune $2 \cdot m$ se construiește urmărind algoritmul de generare a pătratelor magice impare, considerând pătratul de dimensiune m (m este impar), format din pătrate de dimensiune 2.
- Fiecare pătrat de dimensiune 2×2 va avea ca și corespondent o literă „L”, „U”, sau „X” și va fi completat în ordinea sugerată de această literă:



Aplicăm pașii algoritmului Ven prezentat anterior:

		1		
				4
3				
	2			

Obținem:

60	57	32	29	4	1	96	93	68	65
58	59	30	31	2	3	94	95	66	67
64	61	56	53	28	25	20	17	92	89
62	63	54	55	26	27	18	19	90	91
88	85	80	77	49	52	24	21	16	13
86	87	78	79	50	51	22	23	14	15
9	12	81	84	76	73	45	48	37	40
10	11	82	83	74	75	46	47	38	39
33	36	5	8	97	100	69	72	41	44
35	34	7	6	99	98	71	70	43	42

12.4. Implementări sugerate

Pentru a vă familiariza cu modul în care trebuie rezolvate problemele în cadrul cărora intervin operații cu fișiere text, vă sugerăm să încercați să implementați algoritmi pentru:

1. suma elementelor unui tablou bidimensional;
2. identificarea liniei și coloanei pe care se află minimul/maximul unui tablou bidimensional;
3. suma elementelor de pe diagonala principală;
4. suma elementelor de pe diagonala secundară;
5. suma elementelor de deasupra (de dedesubtul) diagonalei principale;
6. suma elementelor de deasupra (de dedesubtul) diagonalei secundare;

7. determinarea sumei celor opt vecini ai unui element;
8. determinarea sumelor elementelor aflate în triunghiul de N, S, E și V delimitat de diagonalele matricei;
9. rotirea unui tablou bidimensional cu 90 de grade;
10. determinarea matricei simetrice față de diagonala principală (secundară);
11. înlocuirea elementelor din triunghiul de nord cu cele din triunghiul de vest (și variante);
12. parcurgerea pe diagonale a unei matrice;
13. parcurgerea în spirală a unei matrice;
14. calcularea sumelor de-a lungul laturilor matricelor pătratice;
15. verificarea unei proprietăți globale a unui tablou bidimensional;
16. produsul a două matrice de numere;
17. produsul a n matrice;
18. inversarea a două linii/coloane într-o matrice;
19. ordonarea liniilor astfel încât elementele de pe diagonala principală/secundară să fie în ordine crescătoare/descrescătoare;
20. verificarea triunghiularității unei matrice (inferior, superior triunghiulară);
21. suma elementelor unui tablou n -dimensional;
22. identificarea coordonatelor unui element având o valoare dată (toate aparițiile) într-un tablou n -dimensional;
23. verificarea proprietății de pătrat magic.

12.5. Probleme propuse

12.5.1. Peștera

O peșteră are n încăperi numerotate de la 1 la n . Între anumite încăperi s-au amenajat coridoare de acces, altele au rămas izolate. Administratorul complexului turistic căruia i s-a dat în grijă peștera ar vrea să știe răspunsul la următoarele întrebări:

1. Care sunt încăperile în care intră cele mai multe coridoare?
2. Care sunt încăperile unde peștera se înfundă?
3. Care sunt încăperile izolate?

Date de intrare

Prima linie a fișierului de intrare **PEȘTERA.IN** conține numărul natural nenul n al încăperilor peșterii. Pe următoarele n linii sunt scrise câte n valori 0 și 1, valorile elementelor unui tablou bidimensional p . Valorile 0 și 1 au următoarea semnificație:

- $p_{ij} = 1$, dacă între încăperea i și încăperea j există cale de acces amenajată.
 - $p_{ij} = 0$, în caz contrar.
- Valorile scrise pe o aceeași linie din fișier sunt despărțite prin câte un spațiu.

Date de ieșire

Pe prima linie a fișierului de ieșire **PESTERA.OUT** se vor scrie numerele de ordine ale încăperilor în care intră sau din care ies un același număr maxim de coridoare.

Pe a doua linie se vor afla numerele de ordine ale încăperilor în care peștera se înfundă. În cazul în care nu există astfel de încăperi, în fișier se va scrie mesajul 'Nu exista'.

A treia linie a fișierului (și ultima) va conține numerele de ordine ale încăperilor izolate. Dacă asemenea încăperi nu există, se va afișa mesajul: 'Nu exista.'

Restricții și precizări

- $1 \leq n \leq 100$.

Exemple**PESTERA1.IN**

```
5
0 0 1 0 0
0 0 0 0 0
1 0 0 0 1
0 0 0 0 1
0 0 1 1 0
```

PESTERA1.OUT

```
3 5
1 4
2
```

PESTERA2.IN

```
5
0 1 1 0 0
1 0 0 0 0
1 0 0 0 1
0 0 0 0 1
0 0 1 1 0
```

PESTERA2.OUT

```
1 3 5
2 4
Nu exista.
```

12.5.2. Acvariul cu pești

Într-un acvariu sunt introduși n pești de diferite specii, printre care și unele carnivore. Pentru fiecare pește se cunosc peștii pe care acesta i-ar mânca (în cazul în care mănâncă alți pești). Deoarece nu toți peștii sunt foarte înfomețați, nu se cunoaște ordinea în care aceștia se vor ataca. Stabiliți speciile de pești care, după un interval de timp, vor rămâne în viață în mod sigur!

Date de intrare

Pe prima linie a fișierului de intrare **PESTI.IN** se găsește numărul natural nenul n , reprezentând numărul de pești. Pe următoarele n linii se află elementele tabloului bidimensional p , având valori 0 și 1 (despărțite prin spații) cu următoarea semnificație:

- $p_{ij} = 1$, dacă specia i mănâncă specia j .
- $p_{ij} = 0$, în caz contrar.

Date de ieșire

Pe prima linie a fișierului de ieșire **PESTI.OUT** se va scrie un șir de numere, separate prin câte un spațiu, reprezentând numerele de ordine a speciilor de pești care supraviețuiesc în mod sigur în acvariu.

Restricții și precizări

- $1 \leq n \leq 100$;
- Dacă un pește de specia i mănâncă pești de specia j și specia j mănâncă pești de specia i , nici unul nu rămâne în mod sigur în viață, deoarece nu știm nimic referitor la ordinea în care ei se vor mânca între ei.

Exemplu**PESTI.IN**

```
6
0 0 1 0 0 0
0 0 0 0 0 0
0 1 0 0 0 0
1 0 0 0 0 1
0 0 0 0 0 0
0 0 0 0 0 0
```

PESTI.OUT

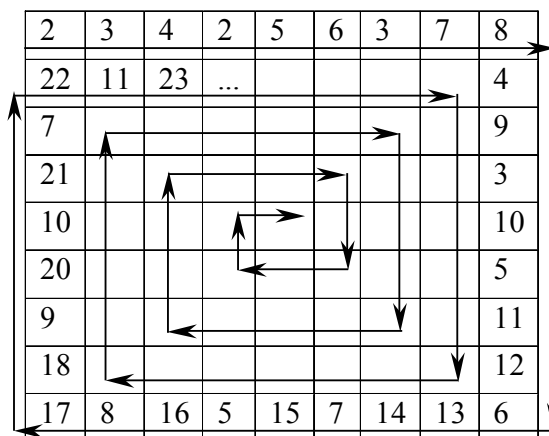
4 5

Explicație

Peștele 1 mănâncă peștele 3.
 Peștele 2 nu este carnivor.
 Peștele 3 mănâncă peștele 2.
 Peștele 4 mănâncă peștele 1 și 6.
 Peștele 5 și peștele 6 nu sunt carnivori.
 Deci, deoarece peștii 1, 2, 3 și 6 pot fi mâncați de alți pești, rămân în viață cu siguranță peștii 4 și 5.

12.5.3. Tir sportiv

Un matematician, iubitor de tir sportiv, și-a pus problema construirii unei ținte pătrate de latură n în care punctajele să fie distribuite după reguli matematice proprii. El mai întâi a generat șirul punctajelor pe baza unei reguli, obținând: 2, 3, 4, 2, 5, 6, 3, 7, 8, 4, 9, 3, 10, 5, 11, 12, 6, ...



Se observă că fiecare număr natural neprim este urmat de cel mai mare divizor (prim sau nu) al său. Pentru o bună distribuție a acestor numere, matematicianul s-a gândit să le așeze pe ținta pătratică de-a lungul unei spirale, începând din colțul stânga-sus, continuând spre dreapta, apoi în jos, urmând apoi marginea de jos de la dreapta la stânga, apoi în sus și continuând spirala cât timp este posibil.

Scrieți un program care „construiește” ținta pentru un pătrat de latură n , respectând regulile matematicianului.

Date de intrare

În fișierul de intrare **TIR.IN** se găsește numărul natural nenul n , reprezentând lungimea laturii țintei.

Date de ieșire

Ținta generată se va scrie sub forma unui tablou bidimensional în fișierul de ieșire **TIR.OUT**, unde pe linia i se vor scrie elementele situate pe cea de-a i -a linie în țintă. Numerele scrise pe linii vor fi separate prin câte un spațiu.

Restricții și precizări

- $1 \leq n \leq 150$.

Exemplu

TIR.IN	TIR.OUT				
5	2	3	4	2	5
	12	6	13	14	6
	11	8	17	7	3
	5	16	5	15	7
	10	3	9	4	8

12.5.4. Biliard

O bilă este lovită cu tacul pe o masă de biliard dreptunghiulară, de dimensiune $m \times n$ într-o direcție oblică (dreapta-sus, dreapta-jos, stânga-jos sau stânga-sus). Atunci când bila se lovește de o margine a mesei, cu excepția colțurilor, ea ricoșează, noul traseu al ei formând un unghi de 90° cu vechiul traseu. Dacă bila ajunge într-un colț, ea iese de pe masă. Cunoscând dimensiunile mesei, poziția de plecare a bilei și direcția de deplasare inițială, să se simuleze mișcarea bilei până la ieșirea de pe masă, sau până când se observă că traseul acesteia intră în ciclu.

Date de intrare

Prima linie a fișierului de intrare **BILIARD.IN** conține numerele m și n , reprezentând dimensiunea mesei. Pe linia a doua se găsesc coordonatele poziției inițiale a bilei, iar pe a treia linie se află o cifră din mulțimea $\{1, 2, 3, 4\}$, reprezentând direcția de pornire a bilei: 1 corespunde direcției dreapta-sus, 2 direcției dreapta-jos, 3 direcției stânga-jos, iar 4 direcției stânga-sus.

Date de ieșire

Fișierul de ieșire **BILIARD.OUT** va conține traseul bilei sub forma unui tablou bidimensional de dimensiuni corespunzătoare mesei. Fiecare linie a tabloului se va scrie pe linie nouă în fișier. Elementul corespunzător poziției inițiale a bilei se marchează cu 1, următoarea poziție atinsă de bilă conform direcției de mișcare cu 2 etc. În cazul în care bila trece a doua oară printr-o poziție, se păstrează primul marcaj. Pozițiile nestrăbătute vor conține valoarea 0.

Dacă bila nu poate ieși de pe masă, deoarece traseul ei formează un ciclu, în fișier se va scrie mesajul 'Bila nu poate iesi de pe masa.', apoi se va scrie în fișier tabloul bidimensional corespunzător mișcărilor bilei.

Restricții și precizări

- $1 \leq m, n \leq 100$.

Exemple**BILIARD1.IN**

```
3 3
2 1
1
```

BILIARD2.IN

```
5 7
4 2
1
```

BILIARD1.OUT

```
Bila nu poate iesi de pe masa.
0 2 0
1 0 3
0 4 0
```

BILIARD2.OUT

```
12 0 0 0 4 0 0
0 11 0 3 0 5 0
0 0 2 0 0 0 6
0 1 0 9 0 7 0
0 0 0 0 8 0 0
```

12.5.5. Puncte șa

Se numește punct *șa* într-un tablou bidimensional, poziția în care elementul este minim pe coloana lui și maxim pe linie, sau invers. Scrieți un program care determină toate punctele *șa* într-o matrice bidimensională dreptunghiulară.

Date de intrare

Prima linie a fișierului de intrare **SA.IN** conține numerele m și n , reprezentând dimensiunile tabloului. Pe fiecare dintre următoarele m linii se află câte n numere întregi, despărțite prin câte un spațiu, reprezentând elementele matricei.

Date de ieșire

Fișierul de ieșire **SA.OUT** va conține atâtea linii câte puncte *șa* s-au găsit. Corespunzător unui astfel de punct în fișier se vor scrie două numere naturale, despărțite printr-un spațiu, reprezentând indicii punctului *șa*. Dacă tabloul dat nu are nici un punct *șa*, în fișier se va scrie mesajul 'Nu exista punct sa.'.

Restricții și precizări

- $1 \leq n \leq 100$;
- $1 \leq m \leq 100$;
- Numerele tabloului sunt distincte.

Exemple**SA . IN**

```
3 4
2 -1 3 0
1 0 4 0
5 -8 9 1
```

SA . OUT

```
1 3
2 2
```

12.5.6. Rame de tablouri care trebuie suprapuse

Într-un spațiu dreptunghiular de dimensiune $m \times n$ trebuie așezate una după alta p rame de tablouri de dimensiune dată. Ramele se pot așeza unele peste altele, astfel încât laturile lor vor fi paralele cu marginile spațiului de depozitare.

Să se afișeze, sub forma precizată la date de ieșire ceea ce va vedea din ramele suprapuse o persoană care se va uita la teancul de rame dintr-o poziție situată deasupra spațiului în care acestea s-au așezat.

Date de intrare

De pe prima linie a fișierului de intrare **RAME . IN** se vor citi numerele m și n , reprezentând dimensiunile spațiului în care se așează ramele. Pe următoarea linie se află numărul natural p , reprezentând numărul ramelor care trebuie suprapuse. Următoarele p linii conțin fiecare coordonatele colțului stânga-sus (indicele de linie și de coloană) al unei rame, lungimea (în număr de linii) și lățimea (în număr de coloane) a ramei.

Date de ieșire

În fișierul de ieșire **RAME . OUT** se va scrie un tablou bidimensional de caractere, reprezentând imaginea cu ramele suprapuse. Pentru o ramă se va specifica doar chenarul ei. Acestea se vor codifica cu literele mari ale alfabetului englez în ordinea în care sunt descrise în fișierul de intrare, începând cu litera 'A'. Atunci când o ramă se suprapune peste una așezată deja îi va acoperi caracterele din chenar în pozițiile în care cele două rame se intersectează pe lățimea unui singur caracter (coloană).

Restricții și precizări

- $1 \leq m \leq 100$;
- $1 \leq n \leq 100$;
- $1 \leq p \leq 25$;
- Dacă dimensiunile unei rame depășesc spațiul de depozitare, rama respectivă nu va fi depozitată, iar litera care îi corespunde nu va fi utilizată.

Exemplu**RAME . IN**

```

12 9
3
3 1 8 7
2 3 4 3
1 4 12 6

```

RAME . OUT

```

          C  C  C  C  C  C
          B  C  B
A  A  B  C  B  A  A  C
A      B  C  B      A  C
A      B  C  B      A  C
A      C      A  C
A      C      A  C
A      C      A  C
A      C      A  C
A  A  A  C  A  A  A  C
          C
          C  C  C  C  C  C

```

12.5.7. Rame de tablouri suprapuse

Într-un spațiu dreptunghiular de dimensiune $m \times n$ au fost așezate una după alta p rame de tablouri. Ramele pot fi așezate unele peste altele, astfel încât laturile lor vor fi paralele cu marginile spațiului de depozitare.

Cunoscând imaginea pe care o vede din ramele suprapuse o persoană care se uită la teancul de rame dintr-o poziție situată deasupra spațiului în care acestea s-au așezat, să se determine ordinea în care este posibilă ridicarea tuturor ramelor. Se știe că fiecare ramă este descrisă în mod unic folosind o literă mare a alfabetului englez și că atunci când au fost suprapuse două rame se va vedea complet numai rama de deasupra.

Date de intrare

De pe prima linie a fișierului de intrare **RAME . IN** se citesc dimensiunile m și n a spațiului în care așezăm ramele. De pe următoarele m linii se citește imaginea pe care o vede persoana. Codificarea imaginii este realizată într-un tablou bidimensional corespunzător spațiului de depozitare în care pentru o ramă se va specifica doar chenarul ei. Acestea sunt codificate cu literele mari ale alfabetului englez în ordinea în care au fost depozitate. Atunci când o ramă s-a suprapus peste una așezată deja ea acoperă caracterele din chenar în pozițiile în care cele două rame se intersectează pe lățimea unui singur caracter (coloană).

Date de ieșire

Fișierul **RAME . OUT** va conține un șir de litere mari ale alfabetului englez, despărțite printr-un spațiu, care corespund ramelor în ordinea în care acestea se pot ridica.

Restricții și precizări

- $1 \leq m \leq 100$;
- $1 \leq n \leq 100$;

- $1 \leq p \leq 25$;
- Din fiecare latură a fiecărei rame se vede cel puțin un punct.

Exemplu**RAME . IN**

```

12  9
      O O O O O O
      V O V      O
K K V O V K K O
K   V O V   K O
K   V O V   K O
K     O     K O
K     O     K O
K     O     K O
K     O     K O
K K K O K K K O
      O
      O O O O O O

```

RAME . OUT

```

O V K

```

12.5.8. Prelucrare de imagine

O imagine alb-negru este preluată codificat într-un tablou bidimensional de valori 0 sau 1 de dimensiune $m \times m$. Asupra acestei imagini se pot efectua următoarele transformări:

- Transformarea „video Invers”: valorile 0 se transformă în 1 iar valorile 1 în 0.
- „Rotire cu 90°”: se creează un tablou rotit în sensul acelor de ceasornic.
- „Zoom”: imaginea se mărește la dimensiunea $2m \times 2m$.

Se definește o secvență de transformări ca fiind o succesiune de litere I, R și Z.

Să se afișeze tabloul bidimensional la care s-a ajuns în urma unui șir de transformări cerute.

Date de intrare

De pe prima linie a fișierului de intrare **IMAG . IN** se citește un număr natural m , reprezentând dimensiunea tabloului. Pe următoarele m linii se află elementele tabloului bidimensional corespunzător unei imagini, linie după linie. Elementele sunt separate prin câte un spațiu.

Din fișierul de intrare **TRANS . IN** se citește succesiunea de litere I, R și Z corespunzătoare unor transformări dorite. Aceste caractere nu sunt despărțite de spațiu.

Date de ieșire

Pe prima linie a fișierului **IMAG . OUT** se vor scrie dimensiunile actuale ale tabloului. Pe următoarele linii se vor afla elementele tabloului bidimensional obținute ca urmare a transformărilor efectuate.

Restricții și precizări

- $1 \leq m \leq 100$ (indiferent de transformările aplicate);
- literele sunt majuscule din alfabetul englez.

Exemple**IMAG . IN**

```

8
0 0 0 0 0 0 0 0
0 1 0 0 0 0 1 0
1 1 1 0 0 1 1 1
0 0 0 0 0 0 0 0
0 1 0 0 0 0 1 0
0 0 1 0 0 1 0 0
0 0 0 1 1 0 0 0
0 0 0 0 0 0 0 0

```

TRANS . IN

RIR

IMAG . OUT

```

8
1 1 1 1 1 1 1 1
1 1 1 0 0 1 1 1
1 1 0 1 1 0 1 1
1 0 1 1 1 1 0 1
1 1 1 1 1 1 1 1
0 0 0 1 1 0 0 0
1 0 1 1 1 1 0 1
1 1 1 1 1 1 1 1

```

IMAG . IN

```

4
1 1 1 1
1 0 0 1
1 0 0 1
1 1 1 1

```

TRANS . IN

Z

IMAG . OUT

```

8
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 0 0 0 0 1 1
1 1 0 0 0 0 1 1
1 1 0 0 0 0 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1

```

12.5.9. Careu magic

Să se genereze un pătrat magic de dimensiune n , unde n este un număr întreg pozitiv.

Date de intrare

În fișierul de intrare **MAG . IN** se găsește numărul n , reprezentând dimensiunea careului care trebuie generat.

Date de ieșire

Pe prima linie a fișierului de ieșire **MAG . OUT** se va afla numărul n . Pe următoarele n linii se vor scrie numerele de pe cele n linii ale careului, două numere fiind separate prin câte un spațiu.

Restricții și precizări

- $1 \leq n \leq 40$.

Exemple**MAG. IN MAG. OUT**

```

6
6  32  3 34 35  1
7  11 27 28  8 30
19 14 16 15 23 24
18 20 22 21 17 13
25 29 10  9 26 12
36  5 33  4  2 31

```

Explicație

n este număr par fără a fi
multiplu de 4.

MAG. IN MAG. OUT

```

4
4  14 15  1
9  7  6 12
5 11 10  8
16  2  3 13

```

Explicație

n este multiplu de 4.

MAG. IN MAG. OUT

```

3
3
4 9 2
3 5 7
8 1 6

```

Explicație

n este număr impar.

12.6. Soluțiile problemelor propuse

12.5.1. Peștera

Vom descompune problema în subprobleme și vom proiecta pentru fiecare subproblemă rezolvări pe care le vom implementa folosind subprograme.

1. Tabloul bidimensional care „conține” particularitățile peșterii îl citim din fișier.

```

Subalgoritm Citire(n,p):
    citește n
    pentru i=1,n execută:
        pentru j=1,n execută:
            citește p[i,j]
        sfârșit pentru
    sfârșit pentru
sfârșit subalgoritm

```

2. Numărăm coridoarele care intră în încăperi. Cunoaștem semnificația valorilor p_{ij} , deci, pentru a determina numărul coridoarelor care se întâlnesc în încăperea i (pentru i fixat și valori posibile $i = 1, 2, \dots, n$), vom aduna valorile p_{ij} , unde $j = 1, 2, \dots, n$, adică vom aduna elementele pe linii.

```

Subalgoritm Numărare(n,p,nr_coridoare):
  pentru i=1,n execută:
    nr_coridoare[i] ← 0
    pentru j=1,n execută:
      nr_coridoare[i] ← nr_coridoare[i] + p[i,j]
    sfârșit pentru
  sfârșit pentru
sfârșit subalgoritm

```

3. Determinăm valoarea maximă a numerelor care reprezintă numărul coridoarelor care se întâlnesc într-o încăpere.

```

Subalgoritm Maxim(n,nr_coridoare,max):
  max ← nr_coridoare[1]
  pentru i=2,n execută:
    dacă nr_coridoare[i] > max atunci
      max ← nr_coridoare[i]
    sfârșit dacă
  sfârșit pentru
sfârșit subalgoritm

```

4. Selectarea numerelor de ordine ale încăperilor în care intră sau ies un același număr maxim de coridoare se realizează cu un algoritm de selectare:

```

Subalgoritm CareMax(n,nr_coridoare,max):
  pentru i=1,n execută:
    dacă nr_coridoare[i] = max atunci
      scrie i
    sfârșit dacă
  sfârșit pentru
sfârșit subalgoritm

```

5. Determinăm încăperile în care peștera se înfundă, folosind șirul construit anterior. Dacă valoarea elementului având indicele i este egal cu 1 (am putut veni de undeva pe un coridor până aici), atunci peștera se înfundă în încăperea i .

```

Subalgoritm Se_înfundă(n,nr_coridoare):
  găsit ← fals
  pentru i=1,n execută:
    dacă nr_coridoare[i] = 1 atunci
      scrie i
      găsit ← adevărat
    sfârșit dacă
  sfârșit pentru

```

```

    dacă nu găsit atunci
        scrie 'Nu exista.'
    sfârșit dacă
sfârșit subalgoritm

```

6. Depistarea încăperilor izolate o realizăm cu ajutorul aceluiași șir, căutând elemente egale cu 0 (nu intră, nu iese nici un coridor).

Subalgoritm Izolate($n, nr_coridoare$):

```

    găsit  $\leftarrow fals$ 
    pentru  $i=1, n$  execută:
        dacă  $nr\_coridoare[i] = 0$  atunci
            scrie  $i$ 
            găsit  $\leftarrow adevărat$ 
        sfârșit dacă
    sfârșit pentru
    dacă nu găsit atunci
        scrie 'Nu exista.'
    sfârșit dacă
sfârșit subalgoritm

```

12.6.2. Acvariul cu pești

Este evident că, din cauza că nu știm nimic despre ordinea în care se mănâncă peștii, pentru a depista speciile care rămân sigur în viață, este suficient să căutăm în tabloul dat coloanele care nu conțin nici o valoare 1. Indicii acestor coloane reprezintă numerele de ordine ale speciilor căutate.

Vom parcurge tabloul dat pe linii și vom marca într-o variabilă booleană (*victimă*) cu valoarea *adevărat*, dacă pe linia respectivă găsim cel puțin o valoare 1. Cu alte cuvinte, peștii care pot fi mâncați de un alt pește, indiferent care, sunt marcați ca victime. Această prelucrare o putem realiza în paralel cu citirea datelor. De asemenea, înainte să trecem la altă linie, în cazul în care peștele de indice i nu este victimă, îl scriem în fișier.

Algoritm Citire_cu_Prelucrare($n, p, victimă$):

```

    citește  $n$ 
    pentru  $i=1, n$  execută:
        victimă  $\leftarrow fals$ 
        pentru  $j=1, n$  execută:
            citește  $p[i, j]$ 
            dacă  $p[i, j] = 1$  atunci
                victimă  $\leftarrow adevărat$ 
        sfârșit dacă
    sfârșit pentru

```

```

    dacă nu victimă atunci
        scrie i                                { afișarea speciilor supraviețuitoare }
    sfârșit dacă
sfârșit pentru
sfârșit subalgoritm

```

12.6.3. Tir sportiv

Pentru rezolvarea acestei probleme este util să creăm în prealabil șirul de numere după regula enunțată. Pentru valori mici ale lui n șirul va fi creat într-un tablou unidimensional, pentru valori mari ale lui n putem crea „șirul” respectiv într-un fișier de tip text. De asemenea, lăsăm pe seama cititorilor să realizeze o implementare în care numerele generate se scriu direct „pe țintă”, evitând astfel utilizarea unui șir (sau fișier) de lucru.

În pseudocodul descris în continuare nu verificăm separat dacă numărul este prim sau nu, ci în primul rând, după ce orice număr curent k s-a scris în șir, vom căuta primul său cel mai mic divizor, urmând să folosim în șir câtușul împărțirii întregi al lui k la acesta (adică pe cel mai mare divizor). Dacă nu găsim un astfel de divizor, înseamnă că numărul nu este prim, iar divizorul găsit îl scriem în fișier.

```

Subalgoritm GenerareȘir(nn):                { nn = n*n, numărul numerelor pe țintă }
i ← 0                                         { i = numără numere generate }
k ← 2                                         { k = numerele pe care le scriem în fișier, primul număr este 2 }
cât timp i < nn execută:
    i ← i + 1                                { crește contorul pătrățelelor ocupate }
    scrie k                                  { îl scriem pe k în fișier }
    rad ← parte întreagă din rădăcina pătrată a lui k
    dacă k este număr par atunci
        divizor ← 2
    altfel
        divizor ← 3                          { cel mai MIC divizor al lui k }
    cât timp (k mod divizor ≠ 0) și (divizor ≤ rad) execută:
        divizor ← divizor + 2                { divizori impari }
    sfârșit cât timp
    sfârșit dacă
    dacă divizor ≤ rad atunci                { avem un divizor }
        i ← i + 1                            { ne pregătim să scriem un număr în fișier }
        dacă i ≤ nn atunci                  { îl scriem numai dacă avem nevoie de el }
            scrie k div divizor              { scriem cel mai MARE divizor }
        sfârșit dacă
    sfârșit dacă
    k ← k + 1                                { următorul număr }
    sfârșit cât timp
sfârșit subalgoritm

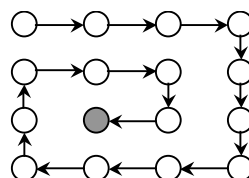
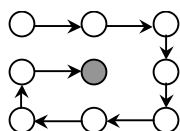
```

Având numerele necesare țintei generate în fișier, le vom citi unul câte unul și le vom așeza în tabloul bidimensional pe poziția corespunzătoare parcurgerii acestuia în spirală.

Vom reține câteva proprietăți pe care le vom fructifica în algoritm:

1. Mai întâi observăm că această spirală se termină, fie după o parcurgere a ultimelor poziții *din stânga spre dreapta* (dimensiunea tabloului este număr *impar*), fie după o parcurgere a ultimelor poziții *din dreapta spre stânga* (dimensiunea tabloului este număr *par*).

În figurile de mai jos sunt prezentate cele două cazuri, pentru $n = 3$, respectiv $n = 4$.



2. O altă observație se referă la faptul că, atunci când suntem pe „cercul” exterior al spiralei, avem de parcurs linia 1 de la primul său element la ultimul, a n -a coloană de la al doilea element al său (primul a fost parcurs atunci când am lucrat pe linia 1) la ultimul, apoi a n -a linie de la penultimul element la primul. În final, vom parcurge prima coloană de la penultimul său element până la al doilea (ultimul a fost parcurs atunci când am prelucrat ultima linie, iar primul atunci când am prelucrat linia 1). Ceea ce este și mai important de observat se referă la faptul că atunci când vom parcurge un „cerc” interior, imediat următor exteriorului, toate *limitele de început cresc cu 1*, iar *limitele de sfârșit scad cu 1*. În consecință, vom inițializa două variabile (primul cu 1 și ultimul cu n) care își vor schimba valorile după închiderea unui „cerc” a spiralei ($\text{primul} \leftarrow \text{primul} + 1, \text{ultimul} \leftarrow \text{ultimul} - 1$).

Din observația 1. rezultă că vom începe așezarea numerelor din fișier, indiferent de paritatea lui n parcurgând elementele tabloului pe linia de indice *primul* spre dreapta. Ar urma parcurgerea de sus în jos a coloanei de indice *ultimul*, dar această parcurgere nu întotdeauna este posibilă (necesară) și anume, dacă n este impar, după o asemenea parcurgere se termină prelucrarea. Subalgoritmul care creează ținta este următorul:

Subalgoritm CreareȚintă($n, nn, \text{ținta}$):

$\text{primul} \leftarrow 1$

$\text{ultimul} \leftarrow n$

$\text{folosite} \leftarrow 0$

parcurem prima linie de la stânga la dreapta

dacă n este număr impar **atunci**

cât timp $\text{folosite} < nn$ **execută:**

 { în variabila folosite numărăm pozițiile prelucrate pe țintă }

parcurem coloana de indice ultimul de sus în jos


```

    parcurgem linia de indice ultimul de la dreapta la stânga
    parcurgem coloana de indice primul de jos în sus
    primul ← primul + 1
    ultimul ← ultimul - 1
    parcurgem linia de indice primul de la stânga la dreapta
sfârșit cât timp
altfel n este număr impar atunci
    parcurgem ultima coloană de sus în jos
    parcurgem ultima linie de la dreapta la stânga
    cât timp folosite < nn execută:
        { în variabila folosite numărăm pozițiile prelucrate pe țintă }
    parcurgem coloana de indice primul de jos în sus
    primul ← primul + 1
    ultimul ← ultimul - 1
    parcurgem linia de indice primul de la stânga la dreapta
    parcurgem coloana de indice ultimul de sus în jos
    parcurgem linia de indice ultimul de la dreapta la stânga
    sfârșit cât timp
sfârșit dacă
sfârșit subalgoritm

```

Parcurgerile menționate mai sus nu sunt problematice. Trebuie doar să avem grijă să nu prelucrăm un același element de două ori, deoarece astfel am suprascrie anumite valori și în final nu ne-ar ajunge numerele generate în fișier. Pentru exemplificare, să vedem parcurgerea de jos în sus:

```





Subalgoritm JosSus(primul, ultimul, folosite):
    pentru i=ultimul-1, primul+1, -1 execută: { pasul este -1! }
        folosite ← folosite + 1
        citește nr { se citește numărul curent din fișierul de manevră }
        țintă[i, primul] ← nr
        { elementul de pe linia i și coloana primul în tabloul corespunzător țintei }
    sfârșit pentru
sfârșit subalgoritm

```

12.6.4. Biliard

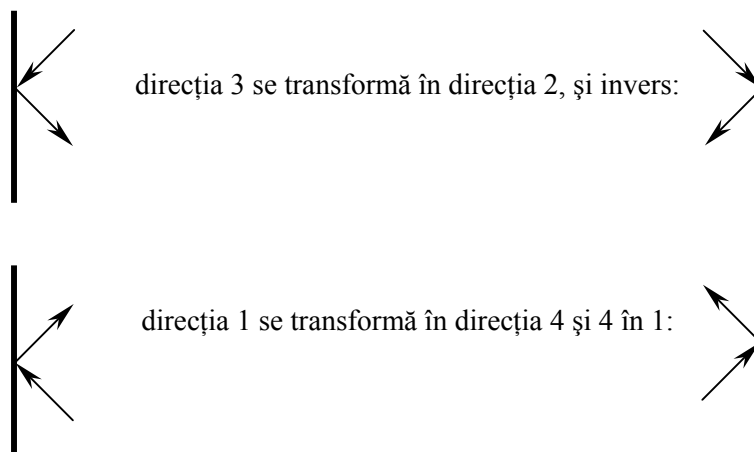
Rezolvarea problemei necesită simularea anumitor deplasări de-a lungul unor linii care sunt paralele cu diagonalele pătratului „decupat” de mișcarea bilei din tabloul bidimensional dreptunghiular, corespunzător mesei și o stăpânire corectă a schimbărilor de direcție atunci când bila se lovește de marginea mesei.

Vom defini două șiruri de câte patru elemente în care vom păstra valorile care trebuie adunate coordonatei x , respectiv coordonatei y , pentru a ne deplasa pe o direcție dată d . Deplasarea în cele patru direcții le putem codifica în felul următor:

d	1 	2 	3 	4 
poziția curentă	i, j	i, j	i, j	i, j
noua poziție	$i - 1, j + 1$ scade linia crește coloana	$i + 1, j + 1$ crește linia crește coloana	$i + 1, j - 1$ crește linia scade coloana	$i - 1, j - 1$ scade linia scade coloana
Dx	-1	1	1	-1
Dy	1	1	-1	-1

La fiecare pas, coordonatele i și j se modifică în $i + Dx[d]$, respectiv $j + Dy[d]$. Acest mod de reprezentare ne va ușura gestionarea schimbărilor de direcție.

Observăm că la întâlnirea unei margini a mesei, corespunzătoare unei margini verticale în tablou, direcția de deplasare se modifică astfel:



La întâlnirea unei margini a mesei, corespunzătoare unei margini orizontale în tablou, direcția 1 se schimbă în direcția 2 și 2 în 1, iar direcția 4 se transformă în 3 și 3 în 4. Vom asigura schimbarea direcției cu o structură de tip **repetă**, care se va executa până când bila iese printr-un colț sau intră în ciclu.

În subalgoritmul următor, în variabila *pas* generăm valoarea care marchează momentul în care bila traversează un anumit punct de pe masă. Dacă mingea ar urma să-și reia traseul și astfel să intre în ciclu infinit, variabila booleană *infinit* primește valoarea *adevărat*. Pentru a avea control asupra momentului în care bila iese de pe masă (ajunge într-un colț) ne folosim de o funcție booleană:

Subalgoritm Afară:

```
Afară ← ((i = 1) sau (i = m)) și ((j = 1) sau (j = n))
sfârșit subalgoritm
```

Rezolvarea problemei constă în simularea mișcării bilei. Anterior apelului subalgoritmului Mișcare s-au citit datele de intrare și s-a inițializat tabloul *a* cu elemente nule.

Subalgoritm Mișcare(*m, n, i, j, a, infinit*):

```
x ← dx[dir]
y ← dy[dir]
pas ← 1 { contorul pentru marcarea pozițiilor străbătute }
a[i,j] ← pas
se_repetă ← fals
infinit ← fals
cât timp mingea nu a ieșit afară și nu infinit execută:
    pas ← pas + 1 { crește contorul de poziții }
    i ← i + x { deplasare pe direcția dir, până la întâlnirea unei margini }
    j ← j + y
    dacă (i = 0) sau (i > m) atunci { bila s-a lovit de o margine orizontală }
        x ← -x
        i ← i + 2*x
    sfârșit dacă
    dacă (j = 0) sau (j > n) atunci { bila s-a lovit de o margine verticală }
        y ← -y
        j ← j + 2*y
    sfârșit dacă
    dacă a[i,j]=0 atunci { dacă această poziție nu a fost străbătută încă }
        se_repetă ← fals
        a[i,j] ← pas { o marcăm }
    altfel { dacă această poziție a mai fost marcată și este pe o margine }
        dacă se_repetă și (a[i,j] = 2) atunci
            infinit ← adevărat { bila a intrat în mișcare ciclică deoarece }
            { din poziția marcată cu 1 a ajuns în poziția marcată cu 2 }
        altfel
            dacă a[i,j] = 1 atunci
                se_repetă ← adevărat
            altfel se_repetă ← fals
            sfârșit dacă
        sfârșit dacă
    sfârșit dacă
sfârșit cât timp
sfârșit subalgoritm
```

Afișarea tabloului a , corespunzător mesei de biliard, este precedată de verificarea variabilei *infin*it pentru a stabili dacă se va scrie mesajul referitor la mișcarea bilei:

```
Subalgoritm Afișare:
    dacă infinit atunci
        scrie 'Mingea nu poate iesi de pe masa.'
    sfârșit dacă
    pentru i=1,m execută:
        pentru j=1,n execută:
            scrie a[i,j]
        sfârșit pentru
    sfârșit pentru
sfârșit subalgoritm
```

12.6.5. Puncte șa

Subproblemele în care descompunem această problemă sunt:

1. citirea tabloului;
2. determinarea minimului pe o linie a tabloului;
3. determinarea maximului pe o coloană a tabloului;
4. determinarea maximului pe o linie a tabloului;
5. determinarea minimului pe o coloană a tabloului;
6. compararea rezultatelor obținute la punctul 2. și 3;
7. compararea rezultatelor obținute la punctul 4. și 5.

Observăm că trebuie să căutăm un minim (sau un maxim) într-o linie (respectiv într-o coloană) din matrice. Această observație ne sugerează să „transformăm” tabloul bidimensional, într-un șir de linii, folosindu-ne de declarațiile de tip și de variabile:

```
type linie=array[1..4] of Integer;
    TipTab=array[1..3] of linie;
var t:TipTab;
    sir:linie;
```

Astfel linia i din tabloul t se va putea transmite ca parametru de tip `linie` subprogramelor care determină minimul (respectiv maximul) din linia respectivă. Din păcate nu putem proceda la fel în cazul coloanelor decât dacă le transformăm temporar, cu un subprogram simplu, în șir de tip `linie`:

```
Subalgoritm Transformă(m,k,t,sir):
    { transformă coloana k din tablou în șir de tip linie }
    pentru i=1,m execută:
        sir[i] ← t[i,k]
    sfârșit pentru
sfârșit subalgoritm
```

Funcțiile $\max(n, \text{șir}, \text{poz})$ și $\min(n, \text{șir}, \text{poz})$ determină valoarea maximă, respectiv minimă dintr-un șir, returnând totodată și poziția unde s-a găsit maximum, respectiv minimum.

În algoritm vom traversa tabloul linie după linie. Pentru fiecare linie i , vom determina minimum (*minim*) și poziția lui pe linie (*pozmin*), apoi vom transforma coloana de indice *pozmin* în șir și vom căuta maximumul lui (*maxim*). Dacă cele două valori sunt egale (*minim = maxim*), afișăm indicele liniei (i) și indicele coloanei (*pozmax*). Procedăm la fel, căutând maximumul pe linie și minimumul pe coloana pe care am găsit maximumul.

Algoritm Puncte_Șa:

Citirea datelor de intrare (m, n, t)

găsit \leftarrow *fals*

pentru $i=1, m$ **execută:**

minim \leftarrow $\min(n, t[i], \text{pozmin})$

{ minimul pe linia i se află pe coloana pozmin }

transformă ($m, \text{pozmin}, t, \text{șir}$)

{ transformăm coloana pozmin în șir }

maxim \leftarrow $\max(m, \text{șir}, \text{pozmax})$

{ maximumul de pe coloana pozmin }

dacă *minim = maxim* **atunci**

scrie $i, ' ', \text{pozmax}$

găsit \leftarrow *adevărat*

sfârșit dacă

maxim \leftarrow $\max(n, t[i], \text{pozmax})$

{ maximumul pe linia i se află pe coloana pozmax }

transformă ($m, \text{pozmax}, t, \text{șir}$)

{ transformăm coloana pozmax în șir }

minim \leftarrow $\min(m, \text{șir}, \text{pozmin})$

{ minimumul de pe coloana pozmax }

dacă *maxim = minim* **atunci**

scrie $i, ' ', \text{pozmax}$

găsit \leftarrow *adevărat*

sfârșit dacă

sfârșit pentru

dacă nu *găsit* **atunci**

scrie 'Nu exista puncte sa.'

sfârșit dacă

sfârșit algoritm

Variabila *găsit* primește valoarea *adevărat* dacă am găsit un punct șa. A fost nevoie de reținerea faptului că am reușit să determinăm un rezultat pentru a scrie mesajul solicitat în cazul în care nu există punct șa în tablou.

12.6.6. Rame de suprapus

Algoritmul următor trasează ramele pe rând cu literele mari ale alfabetului englez, începând cu litera 'A' și le depune într-un tablou bidimensional de caractere. Tabloul va fi în prealabil inițializat cu caracterul spațiu (' ').

În implementarea propusă desenăm prima ramă în spațiul de depozitare, apoi în același tablou desenăm a doua ramă. Nu este nevoie de verificări, caracterul c curent fie suprascris un caracter spațiu, fie un caracter aparținând unei rame așezate deja.

Subalgoritm Rame_de_suprapus:

```

c ← 'A'
pentru k=1,p execută:
    citește y1,x1          { indicele de linie și de coloană a colțului stânga-sus }
    citește Ly,Lx          { lungimile laturilor ramei curente }
    y2 ← y1 + Ly - 1      { indicele de linie a colțului dreapta-jos }
    x2 ← x1 + Lx - 1      { indicele de coloană a colțului dreapta-jos }
    dacă (x2 ≤ n) și (y2 ≤ m) atunci
        pentru i=y1,y2 execută:          { desenăm rama }
            a[i,x1] ← c                    { verticala din stânga }
            a[i,x2] ← c                    { verticala din dreapta }
        sfârșit pentru
        pentru j=x1+1,x2-1 execută:
            a[y1,j] ← c                    { orizontala de sus }
            a[y2,j] ← c                    { orizontala de jos }
        sfârșit pentru
        sfârșit dacă                      { caracterul cu care se va trasa următoarea ramă }
    c ← succesorul caracterului c
sfârșit pentru
sfârșit subalgoritm

```

12.6.7. Rame suprapuse

Problema este descompusă în subprobleme, după cum urmează:

- Citirea datelor de intrare (dimensiunile spațiului de depozitare și tabloul de caractere;
- Determinarea coordonatelor colțurilor ramelor;
- Determinarea pozițiilor relative a ramelor (suprapunerile);
- Afișarea caracterelor cu care s-au desenat ramele, în ordinea în care acestea se pot ridica.

Notăm cu $minx_i$, $miny_i$, $maxx_i$, $maxy_i$ coordonatele colțului stânga-sus și a celui din dreapta-jos a celei de a i -a rame. Aceste valori se pot inițializa cu coordonatele spațiului de depozitare, deoarece sigur vom găsi cel puțin o ramă mai mică decât spațiul.

```

Subalgoritm Dimensiuni (m,n,minx,miny,maxx,maxy):
  pentru c='A','Z' execută:
    minx[c] ← n           { inițializarea coordonatelor colțurilor }
    miny[c] ← m
    maxx[c] ← 0
    maxy[c] ← 0
  sfârșit pentru
  pentru i=1,m execută:           { determinarea coordonatelor colțurilor }
    pentru j=1,n execută:
      { determinăm coordonatele colțurilor ramei din care face parte poziția curentă }
      c ← a[i,j]
      dacă minx[c] > j atunci
        minx[c] ← j
      sfârșit dacă
      dacă miny[c] > i atunci
        miny[c] ← i
      sfârșit dacă
      dacă maxx[c] < j atunci
        maxx[c] ← j
      sfârșit dacă
      dacă maxy[c] < i atunci
        maxy[c] ← i
      sfârșit dacă
    sfârșit pentru
  sfârșit pentru
sfârșit subalgoritm

```

Pentru a determina pozițiile relative ale ramelor, vom construi un tablou de constante logice *peste*, în care valoarea elementului *peste_{xy}* va fi *adevărat* dacă rama desenată cu caracterul *y* este deasupra ramei desenate cu caracterul *x*. Acest tablou se inițializează cu *fals* pentru toate ramele (toate caracterele cu care este posibilă desenarea unei rame). Vom realiza căutări pentru fiecare caracter posibil între limitele dimensiunilor ramei desenate cu caracterul curent. Căutările trebuie efectuate în ambele direcții (orizontală și verticală).

```

Subalgoritm Poziții_Relative (peste,minx,miny,maxy,maxy):
  pentru c='A','Z' execută:
    pentru ch='A','Z' execută:
      peste[c,ch] ← fals { deocamdată nu am descoperit nici o suprapunere }
    sfârșit pentru
  sfârșit pentru
  pentru c='A','Z' execută:           { determinăm suprapunerile }
    pentru i=miny[c],maxy[c] execută: { rama desenată cu caracterul c }
      ch ← a[i,minx[c]]           { caracter din desenul ramei desenate cu c }

```

```

dacă  $ch \neq c$  atunci { dacă întâlnim ch în desenul ramei desenate cu c }
                        { rama desenată cu ch este deasupra ramei desenate cu c }
    peste[c,ch]  $\leftarrow$  adevărat
sfârșit dacă
    ch  $\leftarrow$  a[i,maxx[c]]
dacă  $ch \neq c$  atunci
    peste[c,ch]  $\leftarrow$  adevărat
sfârșit dacă
sfârșit pentru
pentru j=minx[c]+1,maxx[c]-1 execută:
    ch  $\leftarrow$  a[miny[c],j]
dacă  $ch \neq c$  atunci
    peste[c,ch]  $\leftarrow$  adevărat
sfârșit dacă
    ch  $\leftarrow$  a[maxy[c],j]
dacă  $ch \neq c$  atunci
    peste[c,ch]  $\leftarrow$  adevărat
sfârșit dacă
sfârșit pentru
sfârșit pentru
sfârșit subalgoritm

```

Afișarea în această problemă necesită câteva verificări în plus. De asemenea, va trebui să ținem evidența ramelor afișate deja (în șirul *de afișat*, având valori booleene), pentru a evita afișarea lor repetată. Acest șir se inițializează cu *fals*. Dacă o ramă nu are nici una deasupra ei, poate fi ridicată și, deci, afișată. Vom continua afișările cât timp mai există ramă neafișată. După afișarea ramei desenate cu caracterul y , ridicarea ei va însemna modificarea valorii lui $peste_{xy}$, care va deveni *fals*.

```

Subalgoritm Afișare(peste,minx,miny,maxx,maxy):
pentru c='A','Z' execută:
    { dacă maxx > 0, înseamnă că există ramă desenată cu c }
dacă maxx[c] > 0 atunci
    deafișat[c]  $\leftarrow$  adevărat { deci trebuie afișată }
altfel
    deafișat[c]  $\leftarrow$  false
sfârșit dacă
sfârșit pentru
    avemdeafișat  $\leftarrow$  adevărat
cât timp avemdeafișat execută: { cât timp avem rame de afișat, le afișăm }
    avemdeafișat  $\leftarrow$  fals

```



```

pentru c:='A','Z' execută:
    dacă deafișat[c] atunci
        ok ← adevărat
        ch ← 'A'
        cât timp ok și (ch ≤ 'Z') execută:
            { verificăm dacă rama desenată cu c poate fi ridicată }
            dacă peste[c,ch] atunci
                ok ← fals
                sfârșit dacă
                ch ← succesorul lui ch
            sfârșit cât timp
            dacă ok atunci { dacă da, o afișăm și o ridicăm }
                scrie c
                pentru ch='A','Z' execută:
                    peste[ch,c] ← fals
                    sfârșit dacă
                    deafișat[c] ← fals { nu mai trebuie afișată }
                    avemdeafișat ← adevărat
                sfârșit dacă
            sfârșit dacă
        sfârșit pentru
    sfârșit cât timp
sfârșit subalgoritm

```

12.6.8. Prelucrare imagine

Întrucât cerințele acestei probleme sunt formulate pe subpuncte separate, se descriu în continuare algoritmi separați pentru fiecare transformare în parte. Am notat cu a tabloul bidimensional pătratic conținând imaginea, iar cu m dimensiunea acestuia.

A. Inversarea valorilor 0 cu valori 1 și viceversa

În mod normal, inversarea valorilor se face parcurgând tabloul, iar dacă elementul are valoarea 0 aceasta se schimbă în 1 și invers:

```

dacă a[i,j] = 1 atunci a[i,j] ← 0
    altfel a[i,j] ← 1

```

Această schimbare de valoare se poate exprima sintetic aplicând operatorii pe biți astfel: $a[i,j] \leftarrow a[i,j] \text{ xor } 1$. Exprimările sunt echivalente deoarece:

$0 \text{ xor } 1 = 1$ și $1 \text{ xor } 1 = 0$.

În aceste condiții, pseudocodul algoritmului de inversare este următorul:

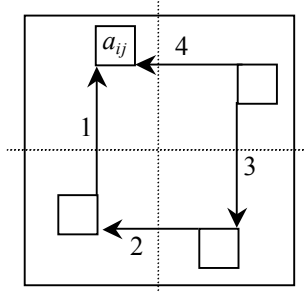
```

Subalgoritm Invers(m,a)
  pentru i=1,m execută:
    pentru j=1,m execută:
       $a[i,j] \leftarrow a[i,j] \text{ xor } 1$ 
    sfârșit pentru
  sfârșit pentru
sfârșit subalgoritm

```

B. Rotirea tabloului cu 90° în sensul acelor de ceasornic

Pentru a roti toate elementele tabloului trebuie să luăm pe rând fiecare element și să-l așezăm într-un alt tablou bidimensional pe poziția potrivită. Dezavantajul acestei variante de algoritm este consumul excesiv de spațiu de memorie. Se prezintă în continuare o variantă mai economicoasă care se bazează pe rotirea succesivă a celor patru puncte.



Dacă pentru un element se aranjează patru elemente la locul lor (folosind o variabilă auxiliară), înseamnă că trebuie repetată această operație doar pentru o pătrime din elementele tabloului. Se parcurge, de exemplu, numai triunghiul superior dintre diagonale, inclusiv diagonala principală. Elementul a_{ij} se salvează în variabila auxiliară după care se efectuează cele trei atribuiri evidențiate pe figură.

Exemple

$m = 4$:

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

$m = 5$:

a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
a_{51}	a_{52}	a_{53}	a_{54}	a_{55}

- $m = 4$: a_{21} se va înlocui cu a_{42} care se va înlocui cu a_{34} care se va înlocui cu a_{13} .
- $m = 5$: a_{21} se va înlocui cu a_{52} care se va înlocui cu a_{45} care se va înlocui cu a_{14} .

În concluzie, indicele de linie devine indice de coloană, iar indicele de coloană ne ajută la calcularea noului indice de linie: $m - \text{fostul indice coloană} + 1$.

În ceea ce privește parcurgerea triunghiului dintre diagonalele tabloului (fără elementele de pe diagonala secundară): indicele de linie variază între prima linie și linia de mijloc (sau deasupra mijlocului atunci când dimensiunea tabloului este pară), iar indicele de coloană pornește de la elementul de pe diagonala principală i și crește până la $m - i$.

$$m = 4: \begin{array}{|c|c|c|c|} \hline a_{11} & a_{12} & a_{13} & a_{14} \\ \hline a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ \hline a_{41} & a_{42} & a_{43} & a_{44} \\ \hline \end{array} \quad m = 5: \begin{array}{|c|c|c|c|c|} \hline a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ \hline a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ \hline a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ \hline a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ \hline \end{array}$$

Prezentăm în continuare pseudocodul corespunzător algoritmului descris.

Subalgoritm Rotește (m, a) :

pentru $i=1, [m/2] + \text{rest}[m/2]$ **execută:**

pentru $j=i, M-i$ **execută:** $\{ \text{pentru o pătrime din elementele tabloului} \}$

$$i1 \leftarrow i$$
$$j_1 \leftarrow j$$

$\text{aux} \leftarrow a[i,j]$ { se păstrează elementul de pornire }

pentru $k=1, 3$ execută:

$$i2 \leftarrow m - j1 + 1 \quad \{ \text{se inițializează indicii de unde se ia valoarea} \}$$
$$i_2 \leftarrow i_1$$
$$a[i1, j1] \leftarrow a[i2, j2] \quad \{ mutare \}$$

i1 ← i2 { se inițializează indicii unde se va pune valoarea }

$j_1 \leftarrow j_2$ { la următoarea iterație }

sfârșit pentru

```
a[i2, j2] ← aux
```

sfârșit pentru

sfârșit pentru

sfârșit subalgoritm

C. Dublarea imaginii (*Zoom*)

Pentru a extinde tabloul pe un tablou de două ori mai mare se vor construi în noul tablou câte patru elemente cu aceea valoare, așezate în forma unei matrice pătratică de dimensiune 2.

De exemplu, un element având valoarea 1 în matricea inițială se înlocuiește cu:

1	1
0	1

Ne propunem în continuare să determinăm modalitatea în care se obțin indicii elementelor matricei mărite din matricea inițială.

indici de linie:

indici de coloană:

	1	2	3	4
1	1	1	0	0
2	1	1	0	0
3	0	0	1	1
4	0	0	1	1

Elementului a_{11} îi corespund elementele $a_{11}, a_{12}, a_{21}, a_{22}$. Elementului a_{12} îi corespund elementele $a_{13}, a_{14}, a_{23}, a_{24}$. Generalizând cele observate, elementului a_{ij} îi corespund elementele $a_{2i-1, 2j-1}, a_{2i-1, 2j}, a_{2i, 2j-1}, a_{2i, 2j}$.

```

Subalgoritm Zoom(m,a) :
    pentru i=1,m execută:
        pentru j=1,m execută:
            b[2*i-1,2*j-1] ← a[i,j]                { b – tablou auxiliar }
            b[2*i-1,2*j]   ← a[i,j]
            b[2*i,2*j-1]   ← a[i,j]
            b[2*i,2*j]     ← [i,j]
        sfârșit pentru
    sfârșit pentru
    m ← 2*m
    a ← b                                           { atribuire la nivel de tablou, suprascriem vechiul a cu b }
sfârșit subalgoritm

```

În algoritmul care va apela subalgoritmii prezentați am notat cu *transformare* caracterul care conține codul transformării curente.

```

Algoritm Prelucrare:
    ...                                           { se citesc dimensiunea și elementele tabloului dat }
    cât timp nu s-a ajuns la sfârșitul fișierului execută:
        citește transformare
        dacă transformare = 'I' atunci
            Invers(m,a)
        altfel
            dacă transformare = 'R' atunci
                Rotește(m,a)
            altfel
                dacă transformare = 'Z' atunci
                    Zoom(m,a)
                sfârșit dacă
            sfârșit dacă
        sfârșit dacă
    sfârșit cât timp
    ...                                           { se afișează tabloul bidimensional obținut }
sfârșit algoritm

```

12.6.9. Careu Magic

Algoritmul de rezolvare a acestei probleme începe prin a testa dimensiunea careului de generat. În funcție de proprietatea pe care acest număr o are: impar, par multiplu de 4 sau par fără a fi multiplu de 4, aplicăm algoritmul prezentat la teorie.

Să urmărim descrierea implementărilor pentru fiecare dintre cei trei algoritmi.

a. Drumul Împăratului *Ven* sau generarea pătratelor magice de ordin impar

Descriem în continuare o variantă de generare a unui pătrat magic de ordin impar care respectă drumul Împăratului *Ven*. Careul magic îl generăm în tabloul *a*. Algoritmul

atribuie pe rând valorile cuprinse în intervalul $[1, n^2]$ variabilei nr , urmând ca această valoare să se atribuie elementelor careului pe baza regulilor precizate.

Subalgoritm Ven:

```

y ← n
x ← [(n+1)/2]
nr ← 1
pentru i=1,n execută:
  a[y,x] ← nr
  pentru j=1,n-1 execută:
    dacă x = 1 atunci
      x ← n
    altfel
      x ← x - 1
    sfârșit dacă
    dacă y = 1 atunci
      y ← n
    altfel
      y ← y - 1
    sfârșit dacă
    nr ← nr + 1
    a[y,x] ← nr
  sfârșit pentru
  dacă y = n atunci
    y ← 1
  altfel
    y ← y + 1
  sfârșit dacă
  nr ← nr + 1
sfârșit pentru
sfârșit subalgoritm

```

b. Careul magic de dimensiune pară ($n = 4 \cdot m$)

Pentru a genera careul magic de dimensiune pară, pornim de la matricea de ordin n formată cu numerele $1, 2, \dots, n^2$ așezate în ordine pe linii.

Din descrierea algoritmului se desprinde că elementele aflate pe linii având număr de ordine pentru care restul împărțirii la 4 este 0 sau 1 și coloane care nu au această proprietate sau pe coloane având număr de ordine pentru care restul împărțirii la 4 este 0 sau 1 și linii care nu au această proprietate trebuie schimbate cu elementele aflate în poziții simetrice față de mijlocul careului. Această proprietate privind poziția elementului o vom exprima folosind operatorul logic **xor**. Schimbarea cu elementul simetric se asigură folosind valoarea variabilei mn .

Subalgoritm Magic4M:

```

nr ← 1
nn ← n*n + 1
pentru i=1,n execută:
  pentru j=1,n execută:
    dacă (rest[i/4] ≤ 1) xor (rest[j/4] ≤ 1) atunci
      a[i,j] ← nn - nr
    altfel
      a[i,j] ← nr
    sfârșit dacă
  nr ← nr + 1
sfârșit pentru
sfârșit pentru
sfârșit subalgoritm

```

c. Pătrat magic de ordin n , unde $n = 2 \cdot m$ și m este număr impar

Să urmărim pașii algoritmului de generare a unui pătrat magic de ordin n , unde $n = 2 \cdot m$ și m este număr impar. Trebuie să împărțim careul în careuri mici de dimensiune $2 \cdot 2$, asupra cărora decidem în ce formă se vor completa. Această formă se stabilește în subalgoritmul $\text{Forma}(x, y, nr)$, apelat de subalgoritmul următor:

Subalgoritm CareuMagic2M:

```

n ← [n/2]
y ← 1
x ← [(n+1)/2]
nr ← 1
pentru i=1,n execută:
  Forma(x,y,nr)
  pentru j=1,n-1 execută:
    dacă x = 1 atunci
      x ← n
    altfel
      x ← x - 1
    sfârșit dacă
  dacă y = 1 atunci
    y ← n
  altfel
    y ← y - 1
  sfârșit dacă
  Forma(x,y,nr)
sfârșit pentru
dacă y = n atunci
  y ← 1

```

```

    altfel
        y ← y + 1
    sfârșit dacă
sfârșit pentru
n ← n*2
sfârșit subalgoritm

Subalgoritm Forma(x,y,nr):
    dacă y > [n/2] + 2 atunci
        FormaX(x,y,nr) { XXXXX }
    altfel
        dacă y = [n/2] + 2 atunci { UULUU }
            dacă x = [(n+1)/2] atunci
                FormaL(x,y,nr)
            altfel
                FormaU(x,y,nr)
            sfârșit dacă
        altfel
            dacă y = [n/2] + 1 atunci { LLULL }
                dacă x = [n/2] + 1 atunci
                    FormaU(x,y,nr)
                altfel
                    FormaL(x,y,nr)
                sfârșit dacă
            altfel
                FormaL(x,y,nr) { LLLLL }
            sfârșit dacă
        sfârșit dacă
    nr ← nr + 4
sfârșit subalgoritm

Subalgoritm FormaL(x,y,nr):
    a[2*y-1,2*x-1] ← nr + 2
    a[2*y-1,2*x ] ← nr + 1
    a[2*y ,2*x-1] ← nr
    a[2*y ,2*x ] ← nr + 3
sfârșit subalgoritm

Subalgoritm FormaU(x,y,nr):
    a[2*y-1,2*x-1] ← nr + 2
    a[2*y-1,2*x ] ← nr + 1
    a[2*y ,2*x-1] ← nr + 3
    a[2*y ,2*x ] ← nr
sfârșit subalgoritm

```

```
Subalgorithm FormaX(x,y,nr):  
    a[2*y-1,2*x-1] ← nr + 1  
    a[2*y-1,2*x  ] ← nr + 2  
    a[2*y  ,2*x-1] ← nr + 3  
    a[2*y  ,2*x  ] ← nr  
sfârșit subalgorithm
```