

ALGORITMI CARE LUCREAZA CU TABLOURI BIDIMENSIONALE - PARTEA I

Probleme grilă: <https://forms.gle/Ng3PSZpFp921wTKX9>

Pb. 3 Partea 2 Probleme Grila - argumentare

N = nr. coloane

- Linia 1: Nr. valori = 1 și $N-1$ valori de 0
- Linia 2: Nr. valori = 2 și $N-2$ valori de 0
- Linia 3: Nr. valori = 4 și $N-2^2$ valori de 0
- Linia 4: Nr. valori = 8 și $N-2^3$ valori de 0
- Linia 5: Nr. valori = 16 și $N-2^4$ valori de 0
- Linia 6: Nr. valori = 32 și $N-2^5$ valori de 0
- Linia 7: Nr. valori = 64 și $N-2^6$ valori de 0

- Linia 8: Nr. Valori: 128 și $N-2^7$ valori de 0
- Ultima linie conține o singură valoare de 0 $\rightarrow N - 2^7 = 1 \rightarrow N = 129$

Calculăm numărul de valori de 0 prin scăderea din numărul total de elemente ale matricii ($8 \cdot N$) numărul de valori de 1 ($1+2+\dots+2^7$)

$$\begin{aligned} S &= 8 \cdot N - (1+2+\dots+2^7) \\ &= 8 \cdot 129 - (2^8-1) \\ &= 1032-255 \\ &= 777 \end{aligned}$$

Alternativ, putem calcula direct numărul de valori de 0:

- $(N-1) + (N-2) + (N-2^2) + \dots + (N-2^6) + 1 = 7 \cdot N - (1+2+\dots+2^6) + 1 = 7 \cdot 129 - (2^7-1) + 1 = \dots$

1. Se consideră matricea definită mai jos, reprezentând punctajele a 4 concurenți la 5 probe ale unui concurs.

30	16	10	45	29
17	54	25	11	42
21	12	33	38	15
18	18	12	19	11

Ex. Concurentul #1 a obținut 45 de puncte la Proba #4.

Concurentul #3 a obținut 12 de puncte la Proba #2.

Punctajul minim pentru oricare dintre probe este 10, iar cel maxim 60. Nu pot exista 2 punctaje egale într-o probă.

Se cere:

a) Afișarea punctajului total al fiecărui concurent.

b) Afișarea concurentului câștigător în cadrul fiecărei probe și punctajul său.

Soluție C++

```
#include <iostream>
using namespace std;
const int N = 4;
const int M = 5;

int main() {
    int A[N][M] = {
        {30, 16, 10, 45, 29},
        {17, 54, 25, 11, 42},
        {21, 12, 33, 38, 15},
        {18, 18, 12, 19, 11}
    };

    //linii: concurenti
    //linia 1 - punctajele obtinute de concurentul 1 la fiecare dintre cele 5 probe,
    // linia 2 - punctajele obtinute de concurentul 2 la probe etc

    //coloane: probe
    //coloana 1 - punctajele obtinute de toti concurentii la proba 1
    //coloana 2 - punctajele obtinute de toti concurentii la proba 2 etc

    //punctaj total concurent -> suma linie

    for (int i = 0; i < N; i++) {
        int punctaj_total = 0;
        for (int j = 0; j < M; j++)
            punctaj_total += A[i][j];

        cout << "Concurentul #" << i+1 << " a obtinut un punctaj total de " <<
            punctaj_total << "." << endl;
    }

    //castigator proba X -> linia de pe coloana X cu valoare maxima

    for (int j = 0; j < M; j++) {
        int max_punctaj = 0;
        int winner = -1;
        for (int i = 0; i < N; i++)
            if (A[i][j] > max) {
                max_punctaj = A[i][j];
                winner = i;
            }

        cout << "La proba #" << j+1 << " castigatorul este concurentul #" << winner+1 <<
            " cu punctajul " << max_punctaj << "." << endl;
    }

    return 0;
}
```

Soluție Pascal

```
program matrice_ex1;
const
  a: array[0..3, 0..4] of integer = ((30,16,10,45,29),
                                     (17,54,25,11,42),
                                     (21,12,33,38,15),
                                     (18,18,12,19,11));

  N: integer = 4;
  M: integer = 5;

var
  i,j,punctaj_total,max_punctaj,winner: integer;

begin
  writeln('Solutia in Pascal');

  {Afisarea punctajului total al fiecarui concurent}
  for i:=0 to N-1 do
    begin
      punctaj_total:=0;
      for j:=0 to M-1 do
        punctaj_total := punctaj_total + a[i,j];
      writeln('Punctajul concurentului ', i+1, ' este ', punctaj_total);
    end;

    {Afisarea concurentului si a punctajului maxim de la fiecare proba}
    for j:=0 to M-1 do
      begin
        max_punctaj := 0;
        winner := -1;

        for i:=0 to N-1 do
          if a[i,j]>max_punctaj then
            begin
              max_punctaj := a[i,j];
              winner := i;
            end;
        writeln('Concurent castigator ', winner+1, ' cu punctaj ',max_punctaj);
      end;

      readln;
    end.
end.
```

2. Un număr natural se numește *magic* dacă este divizibil cu numărul de divizori ai săi. De exemplu, 9 este un număr magic deoarece divizorii săi sunt $\{1, 3, 9\}$, iar 9 este divizibil cu 3, în timp ce 10 nu este un număr magic deoarece 10 nu se divide cu 4, divizorii lui 10 fiind $\{1, 2, 5, 10\}$.

Să se scrie un program care citește de la tastatură dimensiunea unei matrici ($N \times N$, $N < 10$). Programul va umple toate pozițiile din această matrice cu numere magice consecutive, începând de la 1 și va afișa matricea formată.

Ex. $N=3$

1	2	8
9	12	18
24	36	40

Ex. $N=4$

1	2	8	9
12	18	24	36
40	56	60	72
80	84	88	96

În rezolvarea problemei, avem nevoie de:

- Subalgoritm care calculează numărul de divizori al unui număr dat
- subalgoritm care verifică dacă un număr dat este magic
- Subalgoritm `next_magic` care primește ca parametru nr. magic curent și returnează următorul număr magic
- Subalgoritm pentru completarea matricii
- Subalgoritm pentru afișarea matricii

Pasii algoritmului principal:

Algoritm matriceMagica

`@citeste dimensiunile matricii`

`@completeaza matricea cu numere magice`

`@afiseaza matricea`

Sf. Algoritm

Soluție C++

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

void afiseaza_matrice(int a[10][10], int n) {
    /// <summary>
    /// Afiseaza matricea data
    /// </summary>
    /// <param name="a"></param> matricea de afisat
    /// <param name="n"></param> numar de linii/coloane

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

int numar_divizori(int n) {
    /// <summary>
    /// Calculeaza numar de divizori pentru numarul dat
    /// </summary>
    /// <param name="n"></param> numarul dat
    /// <returns></returns> numarul de divizori ai lui n (se numara si 1, n)

    int k = 0;
    for (int d = 1; d <= n; d++) {
        if (n % d == 0)
            k++;
    }
    return k;
}

bool este_magic(int n) {
    /// <summary>
    /// Verifica daca un numar dat este magic
    /// </summary>
    /// <param name="n"></param> numarul de verificat
    /// <returns></returns> true daca numarul este magic, false daca numarul nu este
    /// magic

    int k = numar_divizori(n);
    if (n % k == 0)
        return true;
    return false;
}

int next_magic(int n) {
    /// <summary>
    /// Calculeaza urmatorul numar magic dupa n
    /// </summary>
    /// <param name="n"></param> numarul magic curent
    /// <returns></returns> urmatorul numar magic (i.e. primul numar magic mai mare
    /// decat n)

    n++;
    while (!este_magic(n))
        n++;
    return n;
}
```

```
void completeaza_matrice(int matrice[10][10], int n) {
    /// <summary>
    /// Completeaza matricea cu numere magice
    /// </summary>
    /// <param name="A"></param> matricea data
    /// <param name="n"></param> numarul de linii/coloane

    int k = 1;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            matrice[i][j] = k;
            k = next_magic(k);
        }
}

int main() {
    int n;
    int A[10][10];
    cout << "n=";
    cin >> n;

    completeaza_matrice(A, n);
    afiseaza_matrice(A, n);
    return 0;
}
```

Soluție Pascal

```
program matrice_ex2;
type
    matrice = array[0..9, 0..9] of integer;
var
    a: matrice;
    n: integer;

procedure afiseaza_matrice(a:matrice; n:integer);
var
    i,j:integer;
begin
    for i:=0 to n-1 do
        begin
            for j:=0 to n-1 do
                write(a[i,j], ' ');
            writeln;
        end;
    end;

function numar_divizori(numar:integer):integer;
var
    k,d:integer;
begin
    k:=0;
    for d:=1 to numar do
        if (numar mod d=0) then
            k:=k+1;
    numar_divizori:=k;
end;
```

```
function este_magic(numar:integer):integer;
var
  k:integer;
begin
  k := numar_divizori(numar);
  if (numar mod k=0) then
    este_magic := 1
  else
    este_magic:=0;
  end;

function next_magic(numar:integer):integer;
begin
  numar:=numar+1;
  while (este_magic(numar)=0) do
    numar:=numar+1;
  next_magic:=numar;
end;

procedure completeaza_matrice(n:integer;var a:matrice);
var
  k,i,j: integer;
begin
  k:=1;
  for i:=0 to n-1 do
    for j:=0 to n-1 do
      begin
        a[i,j]:=k;
        k:=next_magic(k);
      end;
    end;
  end;
begin
  read(n);
  completeaza_matrice(n,a);
  afiseaza_matrice(a,n);
  readln;
end.
```

3. Se citește un număr natural n ($n \leq 30$). Construiți și afișați o matrice pătratică cu n linii și n coloane după modelul de mai jos, obținut pentru $n=6$.

252	126	56	21	6	1
126	70	35 → 15	5	1	1
56	35	↓ 20	10	4	1
21	15	10	6 → 3	1	1
6	5	4	↓ 3	2 → 1	1
1	1	1	1	↓ 1	1

Regula:

- $a[n-1][j]=1$ pentru $j=0, n-1$
- $a[i][n-1]=1$ pentru $i=0, n-1$
- $a[i][j] = a[i+1][j] + a[i][j+1]$

Observații: se începe completarea matricii din colțul dreapta jos

Secvență cod:

C++

```
for (int i = n-1; i >=0; i--)
    for (int j = n-1; j >=0; j--)
        if (i == n-1 || j == n-1) A[i][j] = 1;
        else A[i][j] = A[i + 1][j] + A[i][j + 1];
```

PASCAL

```
i:=n-1;
j:=n-1;
while (i>=0) do
begin
    j:=n-1;
    while (j>=0) do
    begin
        if (i=n-1) or (j=n-1) then
            a[i,j]:=1
        else
            a[i,j] := a[i+1,j] + a[i,j+1];

        j:=j-1;
    end;
    i:=i-1;
end;
```


4.¹ Se dă o matrice **A** cu **N** linii și **M** coloane, cu valori cuprinse între **1** și **N·M** inclusiv, nu neapărat distincte.

O **operație** constă în selectarea a două linii sau două coloane consecutive și interschimbarea acestora (swap).

O matrice **yin-yang** este o matrice în care $A[i][j] \geq A[i][j-1]$, pentru orice pereche (i, j) cu $1 \leq i \leq N$ și $2 \leq j \leq M$ și $A[i][j] \geq A[i-1][j]$, pentru orice pereche (i, j) cu $2 \leq i \leq N$ și $1 \leq j \leq M$.

Cerințe

- 1) Formulați condiția pentru care nu există soluție (matricea dată nu se poate transforma în matrice yin-yang).
- 2) Determinați numărul minim de **operații** necesare pentru a transforma matricea dată într-o matrice **yin-yang**.

Date de intrare

Se citesc **N**, **M** și elementele matricii.

Date de ieșire

Se afișează numărul minim de operații necesare pentru a transforma matricea dată într-o matrice yin-yang.

Restricții și precizări

- $1 \leq N, M \leq 100$

Exemple

yinyang.in	yinyang.out	Explicații
<div>2 3</div> <div>1 2 4</div> <div>3 5 6</div>	0	Matricea dată este matrice yin-yang
<div>2 3</div> <div>6 6 5</div> <div>4 6 2</div>	3	Operațiile pot fi următoarele: swap(linia 1 , linia 2), swap(coloana 2, coloana 3), swap(coloana 1, coloana 2). Matricea dată va ajunge la final în forma yin-yang: <div> <div>6 6 5</div> <div>4 6 2</div> <div>4 2 6</div> <div>2 4 6</div> <div>4 6 2</div> <div>6 6 5</div> <div>6 5 6</div> <div>5 6 6</div> </div>

Observăm că dacă avem 2 elemente x și y aflate pe aceeași linie sau pe aceeași coloană, indiferent ce operații aplicăm asupra matricii, cele 2 elemente vor rămâne pe aceeași linie sau coloană.

Ex. Dacă avem matricea din exemplu, observăm că prin efectuarea operației *swap (coloana 2, coloana 3)* și apoi *swap (linia 1, linia 2)*, mulțimile de elemente care formează liniile, respectiv coloanele, nu se schimbă: elementele cu valoarea 1,2,3 rămân pe aceeași linie, de asemenea 4,5,6.

4	6	5
1	3	2

 \longrightarrow

4	5	6
1	2	3

 \longrightarrow

1	2	3
4	5	6

În mod similar, valorile 1,4 sau 2,5 sau 3, 6 rămân pe aceeași coloană, indiferent de operațiile pe care le-am efectuat.

¹ Enunț adaptat OJI 2019.

Astfel, putem deduce că pentru a obține o matrice yin-yang, i.e. o matrice în care, informal spus, atât elementele de pe linii cât și elementele de pe coloane trebuie să fie în ordine crescătoare, trebuie să definim o **relație de ordine** între liniile și coloanele matricei.

Mai concret:

Pentru 2 linii L_1, L_2 din matrice, spunem că L_1 este “mai mică” decât L_2 dacă oricare ar fi coloana C , $C = 1, \dots, m$ (sau $0, \dots, m-1$), $A[L_1][C] \leq A[L_2][C]$

Pentru 2 coloane C_1, C_2 din matrice, spunem că C_1 este “mai mică” decât C_2 dacă oricare ar fi linia L , $L = 1, \dots, n$ (sau $0, \dots, n-1$), $A[L][C_1] \leq A[L][C_2]$

	Coloana A	Coloana B	Coloana C
Linia X	4	6	5
Linia Y	1	3	2

Ex. În matricea de mai sus:

- Coloana A este mai mică decât coloana B ($4 \leq 6$, $1 \leq 3$) și decât coloana C ($4 \leq 5$, $1 \leq 2$)
- Coloana C este mai mică decât coloana B (și mai mare decât coloana A)
- Linia X este mai mare decât linia Y ($4 > 1$, $6 > 3$, $5 > 2$)

Putem observa că dacă am ordona aceste linii și coloane în ordine „crescătoare”, de la stânga la dreapta (coloane) și de sus în jos (linii), am obține o matrice yin-yang.

	Coloana A	Coloana C	Colona B
Linia Y	1	2	3
Linia X	4	5	6

Mentținând notațiile de mai sus pentru linii și coloane, și având în vedere prima observație (componenta unei linii sau a unei coloane nu se schimbă indiferent de operațiile *swap* aplicate), obținem matricea yin-yang prin „ordonarea” liniilor: Y, X (linia Y mai mică decât linia X), și a coloanelor: A, C, B (coloana A mai mică decât coloanele B, C, iar coloana C mai mică decât coloana B).

În consecință, revenind la cerințele problemei:

1) Formulați condiția pentru care nu există soluție (matricea dată nu se poate transforma în matrice yin-yang).

Problema nu are soluție în momentul în care există două linii (sau coloane) între care nu putem defini o relație de ordine („mai mic” sau „mai mare”), i.e.

Dacă există 2 linii diferite L_1, L_2 și două coloane diferite C_1, C_2 pentru care avem simultan:

$$A[L_1][C_1] < A[L_2][C_1]$$

și

$$A[L_1][C_2] > A[L_2][C_2],$$

problema nu are soluție (nicio secvență de operații *swap* nu va duce la obținerea unei matrici yin-yang).

Analog definim și pentru coloane.

Ex.

	Coloana A	Coloana B	Coloana C
Linia X	1	2	3
Linia Y	4	6	5

Pentru linia X și linia Y, și coloana B și coloana C avem $2 < 3$ și $6 > 5$. Astfel, dacă am muta coloana C în stânga coloanei B, am avea $5 \leq 6$, ceea ce îndeplinește condiția dată pentru a avea o matrice yin-yang, însă am modifica și ordinea între 2, și 3 (3 va fi în stânga lui 2), și, după cum reiese din prima observație, prin aceste interschimbări nu putem modifica componența liniilor/coloanelor (2,6 vor fi mereu pe aceeași coloană, după cum vor fi 3,5).

2) Determinați numărul minim de **operații** necesare pentru a transforma matricea dată într-o matrice **yin-yang**.

Conform observațiilor anterioare, numărul de operații necesare pentru a transforma matricea dată într-o matrice yin-yang este numărul de operații necesare pentru a obține o matrice „sortată” (după definiția relației de ordine între linii/coloane de mai sus).

Trebuie să numărăm câte inversiuni trebuie să facem în cazul liniilor, și câte în cazul coloanelor.

Ex. 1

4	6	5
1	3	2

→

4	5	6
1	2	3

4	5	6
1	2	3

→

1	2	3
4	5	6

- 1 inversiune a coloanelor (5,2 cu 6,3)
- 1 inversiune a liniilor

Răspuns: 2 operații

Ex. 2

6	6	5
4	6	2

→

4	6	2
6	6	5

4	6	2
6	6	5

→

4	2	6
6	5	6

→

2	4	6
5	6	6

- 1 inversiune a liniilor
- 2 inversiuni ale coloanelor

Răspuns: 3 operații

Soluție C++

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

void citeste_matrice(int a[101][101], int n, int m) {
    /// <summary>
    /// Citeste elementele unei matrici cu n linii si m coloane
    /// </summary>
    /// <param name="a"></param> matricea
    /// <param name="n"></param> numar de linii
    /// <param name="m"></param> numar de coloane
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> a[i][j];
        }
    }
}

bool exista_solutie(int a[101][101], int n, int m) {
    /// <summary>
    /// Verifica daca pentru matricea data exista solutie (daca se poate
    /// transforma in matrice yin-yang)
    /// </summary>
    /// <param name="a"></param> matricea pentru care verificam
    /// <param name="n"></param> numar de linii
    /// <param name="m"></param> numar de coloane
    /// <returns></returns> false daca nu exista solutie, true daca exista
    for (int l1 = 0; l1 < n; l1++) {
        for (int l2 = l1 + 1; l2 < n; l2++) {
            for (int c1 = 0; c1 < m; c1++)
                for (int c2 = 0; c2 < m; c2++) {
                    if (a[l1][c1] < a[l2][c1] && a[l1][c2] > a[l2][c2])
                        return false;
                }
        }
    }
    for (int c1 = 0; c1 < m; c1++) {
        for (int c2 = c1 + 1; c2 < m; c2++) {
            for (int l1 = 0; l1 < n; l1++)
                for (int l2 = 0; l2 < n; l2++) {
                    if (a[l1][c1] < a[l1][c2] && a[l2][c1] > a[l2][c2])
                        return false;
                }
        }
    }
    return true;
}

int operatii_linii(int a[101][101], int n, int m) {
    /// <summary>
    /// Calculeaza numarul de inversiuni de linii care trebuie realizate
    /// 2 linii Li, Lj, i < j trebuie schimbate daca Li > Lj
    /// </summary>
    /// <param name="a"></param> matricea data
    /// <param name="n"></param> numar linii matrice
    /// <param name="m"></param> numar coloane matrice
    /// <returns></returns> numarul de inversiuni de linii necesare
    int numar_operatii = 0;
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++) {
            int valori_inversate = 0;
            for (int k = 0; k < m; k++)
                if (a[i][k] > a[j][k])
                    valori_inversate++;
            if (valori_inversate > 0)
                numar_operatii++;
        }
    return numar_operatii;
}
```

```
int operatii_coloane(int a[101][101], int n, int m) {
    /// <summary>
    /// Calculeaza numarul de inversiuni de coloane care trebuie realizate
    /// 2 coloane Ci, Cj, i<j trebuie schimbate daca Ci > Cj
    /// </summary>
    /// <param name="a"></param> matricea data
    /// <param name="n"></param> numar linii matrice
    /// <param name="m"></param> numar coloane matrice
    /// <returns></returns> numarul de inversiuni de coloane necesare
    int numar_operatii = 0;
    for (int i = 0; i < m; i++)
        for (int j = i + 1; j < m; j++) {
            int valori_inversate = 0;

            for (int k = 0; k < n; k++)
                if (a[k][i] > a[k][j])
                    valori_inversate++;

            if (valori_inversate > 0)
                numar_operatii++;
        }
    return numar_operatii;
}

int main() {
    int n, m;
    cin >> n >> m;
    int A[101][101];
    citeste_matrice(A, n, m);

    if (!exista_solutie(A, n, m)) {
        cout << "Nu exista solutie pentru matricea data." << endl;
        return 0;
    }

    int nr_operatii = operatii_linii(A, n, m) + operatii_coloane(A, n, m);

    cout << "Numar minim operatii:" << nr_operatii<<endl;

    return 0;
}
```

Soluţie Pascal

```
program matrice_yinyang;

type
  matrice = array[0..10, 0..10] of integer;
var
  a:matrice;
  n,m, total_operatii:integer;

procedure citeste_matrice(n,m:integer;var a:matrice);
var
  i,j,crt_number:integer;
begin
  for i:=0 to n-1 do
    for j:=0 to m-1 do
      begin
        read(crt_number);
        a[i,j]:=crt_number;
      end;
    end;
  end;

procedure afiseaza_matrice(a:matrice;n,m:integer);
var
  i,j:integer;
begin
  for i:=0 to n-1 do
    begin
      for j:=0 to m-1 do
        write(a[i,j], ' ');
      writeln;
    end;
  end;

function exista_solutie(a:matrice;n:integer;m:integer):integer;
var
  l1,l2,c1,c2:integer;
  ok:integer;
begin
  ok:=1;
  for l1:=0 to n-1 do
    for l2:=0 to n-1 do
      for c1:=0 to m-1 do
        for c2:=0 to m-1 do
          if (a[l1,c1]<a[l2,c1]) and (a[l1,c2]>a[l2,c2]) then
            ok:=0;

        for c1:=0 to m-1 do
          for c2:=0 to m-1 do
            for l1:=0 to n-1 do
              for l2:=0 to n-1 do
                if (a[l1,c1]<a[l1,c2]) and (a[l2,c1]>a[l2,c2]) then
                  ok:=0;

        exista_solutie:=ok;
  end;

function operatii_linii(a:matrice;n,m:integer):integer;
var
  i, j, k, valori_inversate, nr_operatii:integer;
begin
  nr_operatii:=0;
  for i:=0 to n-1 do
    for j:=i+1 to n-1 do
      begin
        valori_inversate:=0;
        for k:=0 to m-1 do
          if (a[i,k] > a[j,k]) then
            valori_inversate:=valori_inversate+1;
        if (valori_inversate>0) then
          nr_operatii:=nr_operatii+1;
      end;
    end;
  end;
```

```
function operatii_coloane(a:matrice;n,m:integer):integer;
var
  i, j,k, valori_inversate, nr_operatii:integer;
begin
  nr_operatii:=0;
  for i:=0 to m-1 do
    for j:=i+1 to m-1 do
      begin
        valori_inversate:=0;
        for k:=0 to n-1 do
          if (a[k,i] > a[k,j]) then
            valori_inversate:=valori_inversate+1;
          if (valori_inversate>0) then
            nr_operatii:=nr_operatii+1;
        end;
      end;
    end;
  end;
  operatii_coloane:=nr_operatii;

begin
  readln(n);
  readln(m);
  citeste_matrice(n,m,a);
  afiseaza_matrice(a,n,m);
  if (exista_solutie(a,n,m)=0) then
    writeln('Nu exista solutie.')
  else
    begin
      total_operatii := operatii_linii(a,n,m) + operatii_coloane(a,n,m);
      writeln('Numarul de operatii:', total_operatii);
    end;
  readln;
end.
```

TO THINK ABOUT

- Care e complexitatea algoritmului de verificare existență soluție?
- Care e complexitatea algoritmului de determinare a numărului minim de operații?
- Putem implementa acești 2 algoritmi mai eficient?
- Cum am proceda dacă ar trebui să efectuăm operațiile și să afișăm tablourile intermediare, respectiv matricea finală?

Mai multe explicații, alte implementări și cazuri de test predefinite: în arhiva de [aici](#)

*Rezolvările prezentate nu sunt optimizate pentru viteza de execuție.