



Problema Sniper

Fișier de intrare **stdin**
 Fișier de ieșire **stdout**

Într-o matrice $N \times N$ avem toate numerele de la 1 la N^2 . Se consideră următorul procedeu prin care se scot pe rând elementele din matrice și se așează într-o permutare: un sniper lovește unul dintre cele 4 colțuri ale matricei. Se parcurge diagonală/semidiagonală matricei care pleacă din acel colț. Elementele se adaugă la finalul permutării în ordinea parcurgerii. Elementele parcurse se elimină din matrice, iar cele rămase se deplasează vertical sau orizontal, reformând o nouă matrice care "a pierdut" o linie sau o coloană. O deplasare se poate face la alegere orizontal sau vertical. **După lovitura de sniper și deplasarea pe orizontală sau verticală, elementele rămase trebuie să formeze în continuare o matrice!**

Cerință

Să se determine a P -a permutare de dimensiune N^2 în ordine lexicografică care se poate obține folosind procesul descris mai sus.

Atenție!

Datorită dimensiunilor mari ale matricei, datele vor fi generate conform programului pe care concurenții îl vor primi în workspace, în funcția main aflându-se un exemplu de input/output.

Date de intrare

Pe prima linie se află numerele N , P și $Seed$. Dacă $Seed = 0$, fiecare din următoarele N linii conține N numere naturale, altfel numerele vor fi generate conform algoritmului din anexă, și se pot accesa folosind funcția `ReadPerm(N)`.

Date de ieșire

Dacă $seed = 0$, atunci se vor afișa pe o singură linie N^2 numere reprezentând a P -a permutare în ordine lexicografică ce se poate obține, altfel se va afișa o singură valoare ce se calculează conform codului de mai jos. Vă recomandăm să folosiți funcția `WritePerm(value)` pentru toate cele N^2 numere în ordine.

Restricții

- $1 \leq N \leq 2500$
- $1 \leq P \leq 10^{18}$
- $1 \leq A_{i,j} \leq N^2$
- Valorile din matricea A sunt distincte.
- Se garantează că există cel puțin P permutări care se pot obține folosind procedeul descris în cerință.
- Semidiagonală unei matrice se consideră ca fiind diagonală ce începe din colțul stânga-jos și parcurge elementele înspre dreapta-sus sau viceversa, iar diagonală unei matrice se consideră ca fiind diagonală ce începe din colțul stânga-sus și parcurge elementele înspre dreapta-jos.
- Corectare: numerele nu formează neapărat o permutare, dar sunt distincte și nu depășesc un milion.

#	Punctaj	Restricții
1	9	$N \leq 3$
2	21	$P = 1$
3	43	$1 \leq N \leq 500$
4	27	Fără restricții suplimentare

Exemple

Fișier de intrare	Fișier de ieșire
3 5 0 6 4 3 1 5 9 2 8 7	2 5 3 1 4 8 9 6 7
6 9 1729	5976389439715066411

Pentru primul exemplu se alege colțul din stânga jos, deci se trage în numerele 2 5 3, care se adaugă la permutare. Matricea

devine astfel:

$$\begin{array}{cc} 6 & 4 \\ 1 & 9 \\ & 8 \ 7 \end{array}$$

Se alege apoi deplasarea pe verticală și matricea devine:

$$\begin{array}{ccc} 6 & 4 & 9 \\ 1 & 8 & 7 \end{array}$$

Alegem colțul stânga jos (deci se adaugă 1 4 la permutare), iar matricea devine:

$$\begin{array}{cc} 6 & 9 \\ & 8 \ 7 \end{array}$$

iar după deplasarea pe orizontală, matricea devine:

$$\begin{array}{cc} 6 & 9 \\ 8 & 7 \end{array}$$

Alegem din nou colțul stânga jos (deci se adaugă 8 9 la permutare), iar matricea devine

$$\begin{array}{cc} 6 & 7 \end{array}$$

Alegem pe 6, iar mai apoi pe 7 și obținem în final permutarea 2 5 3 1 4 8 9 6 7.

Code template

```
#include <iostream>
#include <vector>
#include <cstring>
#include <algorithm>

using namespace std;

struct RandGen {
    uint64_t state;

    ~RandGen() {
        // Called automatically when the program ends
        this->finish();
    }

    uint64_t nextInt() {
        state += 0x9e3779b97f4a7c15;
        uint64_t z = state;
        z = (z ^ (z >> 30)) * 0xbf58476d1ce4e5b9;
        z = (z ^ (z >> 27)) * 0x94d049bb133111eb;
        return z ^ (z >> 31);
    }

    void write(int value) {
        state ^= value;
        state = (state << 32) ^ (state >> 32);
        nextInt();
    }

    int random(int N) { // Dragi concurenti, stim ca nu facem debias bine
        return nextInt() % N;
    }

    void finish() {
        if (state) {
            cout << state;
        }
    }
} rng;
```

```
vector<int> perm;
int permIndex = 0;

void generatePermutation(int N) {
    perm.resize(N * N);
    if (rng.state == 0) {
        for (int i = 0; i < N * N; i++) {
            cin >> perm[i];
        }
        return;
    }
    for (int i = 0; i < N * N; i++) {
        perm[i] = i + 1;
    }
    for (int i = 0; i < N * N; i++) {
        const int poz = i + rng.random(N * N - i);
        swap(perm[i], perm[poz]);
    }
}

int ReadPerm(int N) {
    if (perm.size() == 0) {
        generatePermutation(N);
    }
    return perm[permIndex++];
}

void WritePerm(int value) {
    if (rng.state == 0) {
        cout << value << " ";
    } else {
        rng.write(value);
    }
}

//// Start Implementation here
#define MaxN 3020

int N;
long P;
int a[MaxN][MaxN];

int main() {
    cin >> N >> P >> rng.state;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            a[i][j] = ReadPerm(N);
            WritePerm(a[i][j]);
        }
    }

    return 0;
}
```