Sam's Puzzle (Approximate)

by nikasvanidze

Problem Submissions Leaderboard Discussions

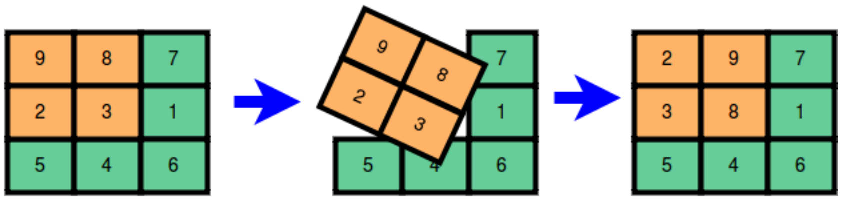
Sam invented a new puzzle game played on an $n \times n$ matrix named $puzzle$, where each cell contains a unique integer in the inclusive range between 1 and n^2. The coordinate of the top-left cell is $(1, 1)$.

The Moves

A move consists of two steps:

1. Choose a sub-square of $puzzle$.
2. Rotate the sub-square in the *clockwise* direction.

For example:



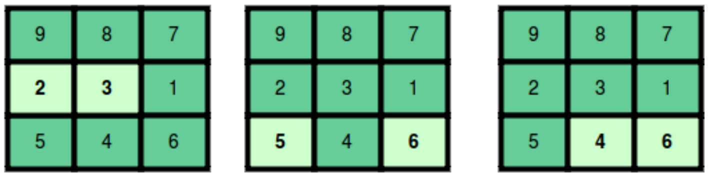
We describe a move as the clockwise rotation of a $k \times k$ sub-square whose top-left corner is located at coordinate (i, j). In the example above, $i = 1$, $j = 1$, and $k = 2$.

Good Pairs of Cells

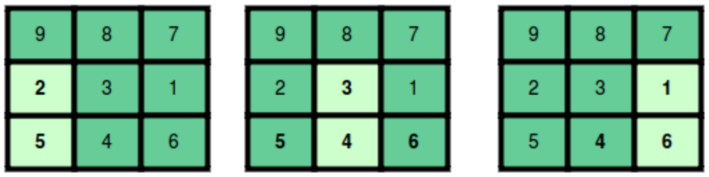
A pair of cell is *good* if one of the following is true:

- They're located in the same row and the number in the left cell is less than the number in the right cell.
- They're located in the same column and the number in the upper cell is less than the number in the lower cell.

The diagram below depicts all the *good* pairs of cells located in the same row:



The diagram below depicts all the *good* pairs of cells located in the same column:



Goodness of a Square

We define the *goodness* of a sub-square to be the total number of good pairs of cells in the sub-square.

The Goal

Given the initial value of $puzzle$, maximize its goodness as much as is possible by performing a sequence of *at most* 500 moves. Then print the necessary moves according to the *Output Format* specified below.

Input Format

The first line contains an integer denoting n. Each of the n subsequent lines contains n space-separated integers. The j^{th} integer in the i^{th} line denotes the cell located in coordinate (i, j).

Constraints

- $1 \leq n \leq 30$
- Each cell contains a unique number in the inclusive range between 1 and n^2.

Scoring

- We define g_b as the goodness in the beginning, g_a as the goodness after your moves, and g_{max} as the maximum possible goodness.
- A valid answer earns $max(0, \frac{g_a - g_b}{g_{max} - g_b}) \times 100\%$ of a test case's available points (it's guaranteed that $g_{max} > g_b$). The total score will be rounded up to the next 1%.

Test Case Generation

- Consider all the cells in $puzzle$ to be initially empty. Sam sorts the n^2 numbers in ascending order and then picks them one by one and places them in some random cell which has no empty cell to its left and no empty cell above it. This generates a square with goodness $n^2 \times (n-1)$.
- After generating $puzzle$, Sam makes some random rotations. During each step, he chooses three random numbers, i, j, and k, and rotates a $k \times k$ sub-square with the top-left corner at coordinate (i, j) in the *counterclockwise* direction. Here $1 \leq i, j, k \leq n$ and $max(i, j) + k \leq n + 1$.
- Sam makes *at most* 100 such random counterclockwise rotations. This means that it's possible to achieve maximum goodness in as little as 100 moves.

Output Format

Print the following lines of output:

- On the first line, print an integer, m, denoting the number of moves necessary to maximize the goodness of $puzzle$. Recall that this number must be ≤ 500.
- For each move, print three space-separated integers describing its respective i, j, and k values on a new line. Recall that a move is described as the clockwise rotation of a $k \times k$ sub-square whose top-left corner is located at coordinate (i, j).

Sample Input 0

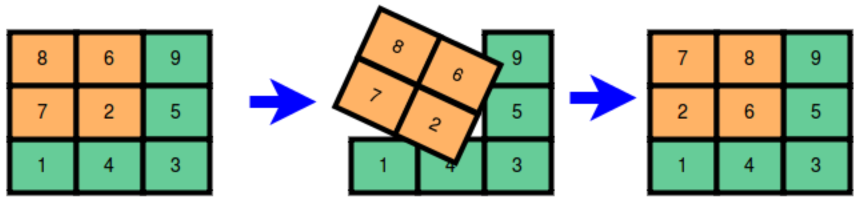
```
3
8 6 9
7 2 5
1 4 3
```

Sample Output 0

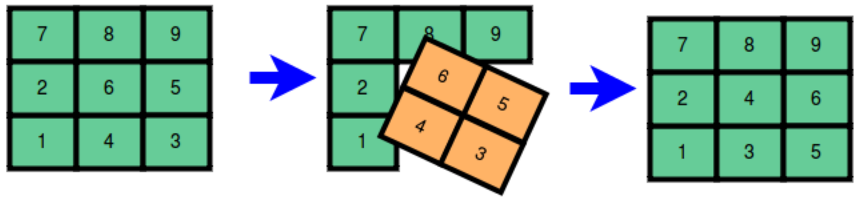
```
3
1 1 2
2 2 2
1 1 3
```

Explanation 0

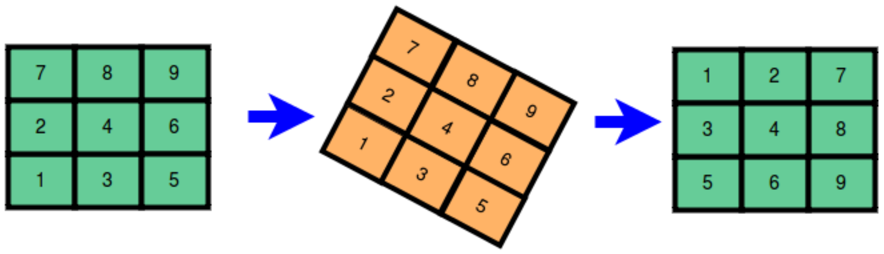
- After the first move:



- After the second move:



- After the third move:



The goodness after this sequence of moves is $g_a = 18$, and the maximum possible goodness is $g_{max} = n^2 \cdot (n-1) = 3^2 \cdot (3-1) = 18$.

Because the initial goodness was $g_b = 6$, this solution will get $\frac{g_a - g_b}{g_{max} - g_b} \cdot 100\% = \frac{18-6}{18-6} \cdot 100\% = 100\%$ of the test case's available points.

Solved score: 25.50pts

Submissions:76

Max Score:85
Difficulty: Advanced

Rate This Challenge:
☆☆☆☆☆

More

Current Buffer (saved locally, editable)

Java 7

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        int[][] puzzle = new int[n][n];
        for(int puzzle_i=0; puzzle_i < n; puzzle_i++){
            for(int puzzle_j=0; puzzle_j < n; puzzle_j++){
                puzzle[puzzle_i][puzzle_j] = in.nextInt();
            }
        }
        // your code goes here
    }
}

```

Line: 1 Col: 1

Upload Code as File

☐ Test against custom input

Run Code

Submit Code