Tiling

A few weeks ago, Lea moved into her new apartment. When she moved in, all the rooms were painted in a very sad greyish color, so she immediately decided to remodel the entire flat. Just now, she is thinking about the bathroom. She spent hours in the hardware store comparing different shades of delightful colored tiles that she could use to pave the wall.

In the end, she found several nice hues. Now the only problem left is how to arrange the tiles. Every tile she picked out is a square shape made up of 4 individual triangles. Of course, Lea wants to make a nice repeating pattern out of them.

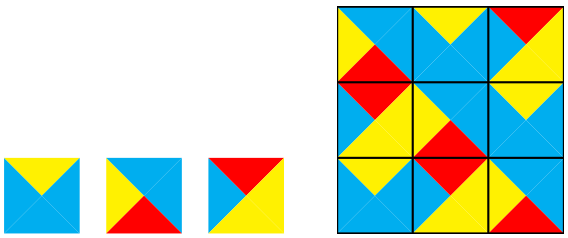

Figure 1: The first sample input - 3 different tile layouts and a nice repeatable pattern

A $k \times k$ pattern is *nice* if the color of every two adjacent triangles matches and it is *repeatable* iff it can be repeated in all directions and still stays nice. Can you help her find some nice repeatable patterns?

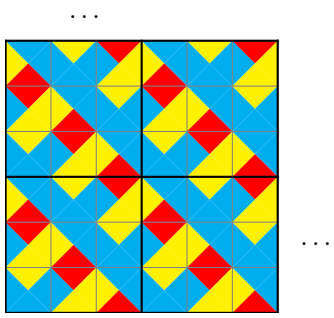

Figure 2: The first sample input - repeating the pattern from above

Input

The first line of the input contains an integer t. t test cases follow, each of them separated by a blank line.

Each test case starts with a line containing an integer n the number of different tile layouts Lea has available. n lines follow, each containing 4 numbers u_i r_i d_i l_i describing tile layout i - it has color u on the upper part, color r on the right part, color d on the lower part and color l on the left part.

Output

For each test case, print a line containing "Case #i: k" where i is its number, starting at 1, and k is the dimension of a periodic tiling layout. Output k more lines, each containing k tile layout indices that together form a nice, repeatable pattern. In case there is no periodic tiling, print "Case #i: impossible".

Constraints

- $1 \leq t \leq 20$

- $1 \leq n \leq 20$
- $0 \leq u_i, r_i, d_i, l_i \leq 10$ for $1 \leq i \leq n$

Sample Input 1

```
4
3
2 1 1 1
1 1 3 2
3 2 2 1

4
1 3 5 2
4 2 3 3
5 4 1 2
3 2 4 4

6
1 2 3 4
6 7 1 2
3 4 6 7
3 4 6 5
6 5 1 2
3 4 6 5

2
0 1 2 3
2 1 0 3
```

Sample Output 1

```
Case #1: 3
1 0 2
2 1 0
0 2 1
Case #2: 2
3 2
1 0
Case #3: 3
0 1 2
2 0 1
1 2 0
Case #4: impossible
```