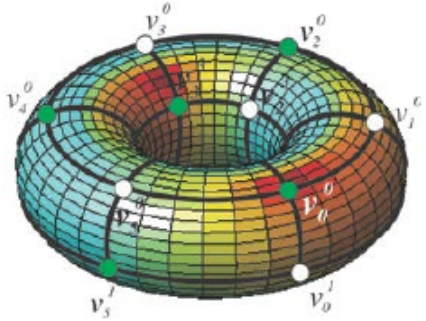
6903 Torus

As is well known, a *regular polygon* is a polygon having all sides of the same length and all equal angles. A *regular tiling* of the Euclidean plane is a covering of the entire plane with non-overlapping, identical regular polygons, in which the polygons are placed vertex-to-vertex. There exist only three types of polygons that admit a regular tiling: square, equilateral triangle and regular hexagon. A *lattice* is an infinite graph whose drawing on the plane forms a regular tiling. In Figure 1, three lattices are illustrated: square lattice, triangular lattice, and hexagonal lattice (from left to right).



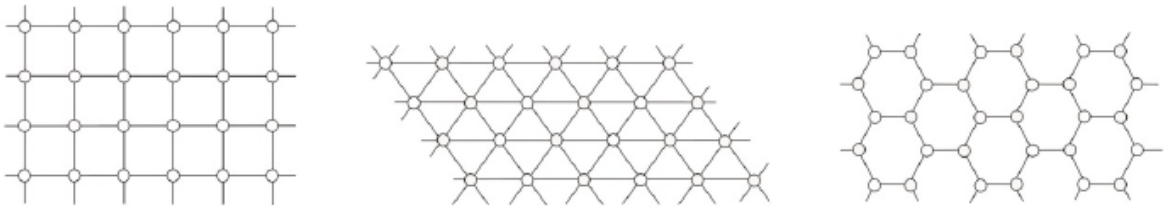


Figure 1. The square, triangular and hexagonal lattices.

One day, Heechul, an eminent researcher in the community of graph theory, discovered the interesting fact that such lattices can be converted into finite graphs (graphs with finite numbers of vertices and edges), conveying their inherent characteristics, e.g. the beautiful symmetrical structures. In particular, the finite graphs allow effective drawings on the surface of a torus without edge crossings, also resulting in the tiles (i.e., the regions bounded by the minimum-length cycles) that are very similar in their shapes and sizes. So, he delightedly called such drawings *toroidal lattices*, which are now defined as follows (refer to Figure 2):

Definition 1. For two integers $m, n \geq 3$, the $m \times n$ *toroidal square lattice* is a graph whose vertex set is $\{v_j^i : 0 \leq i \leq m-1, 0 \leq j \leq n-1\}$ and edge set is $\{(v_j^i, v_{j'}^{i'}) : (i' = i \text{ and } j' \equiv j+1 (\text{mod } n)) \text{ or } (j' = j \text{ and } i' \equiv i+1 (\text{mod } m))\}$.

Definition 2. For two integers $m, n \geq 3$ with m even, the $m \times n$ *toroidal triangular lattice* is the graph constructed from the $m \times n$ toroidal square lattice by adding edges of $E_0 \cup E_1 \cup \cdots \cup E_{m-1}$, where

$$E_i = \begin{cases} \{(v_j^i, v_{j'}^{i'}) : i' \equiv i+1 (\text{mod } m) \text{ and } j' \equiv j-1 (\text{mod } n)\} & \text{if } i \text{ is even,} \\ \{(v_j^i, v_{j'}^{i'}) : i' \equiv i+1 (\text{mod } m) \text{ and } j' \equiv j+1 (\text{mod } n)\} & \text{if } i \text{ is odd.} \end{cases}$$

Definition 3. For two integers $m, n \geq 4$ with both even, the $m \times n$ *toroidal hexagonal lattice* is the graph whose vertex and edge sets are $\{v_j^i : 0 \leq i \leq m-1, 0 \leq j \leq n-1\}$ and $\{(v_j^i, v_{j'}^{i'}) : (j' = j \text{ and } i' \equiv i+1 (\text{mod } m)) \text{ or } (i' = i \text{ and } j' \equiv j+1 (\text{mod } n)) \text{ and } i+j \equiv 0 (\text{mod } 2))\}$, respectively.

To celebrate the 60-th birthday of Heechul, a project was initiated by his colleagues to build a C/C++ library for lattice-related graph algorithms. In order to help them, you are going to write a program that, given an $m \times n$ toroidal lattice, finds a cycle that visits every vertex exactly once. (Note that a cycle of a graph is represented as a sequence $(u_1, u_2, \ldots, u_{mn})$ of mn distinct vertices such that u_k and u_{k+1} are adjacent in the graph for all $k \in \{1, \ldots, mn-1\}$ and moreover, u_{mn} and u_1 are also adjacent.) Since the toroidal square lattice is rather simple, we will only consider the toroidal triangular/hexagonal lattices in this coding.

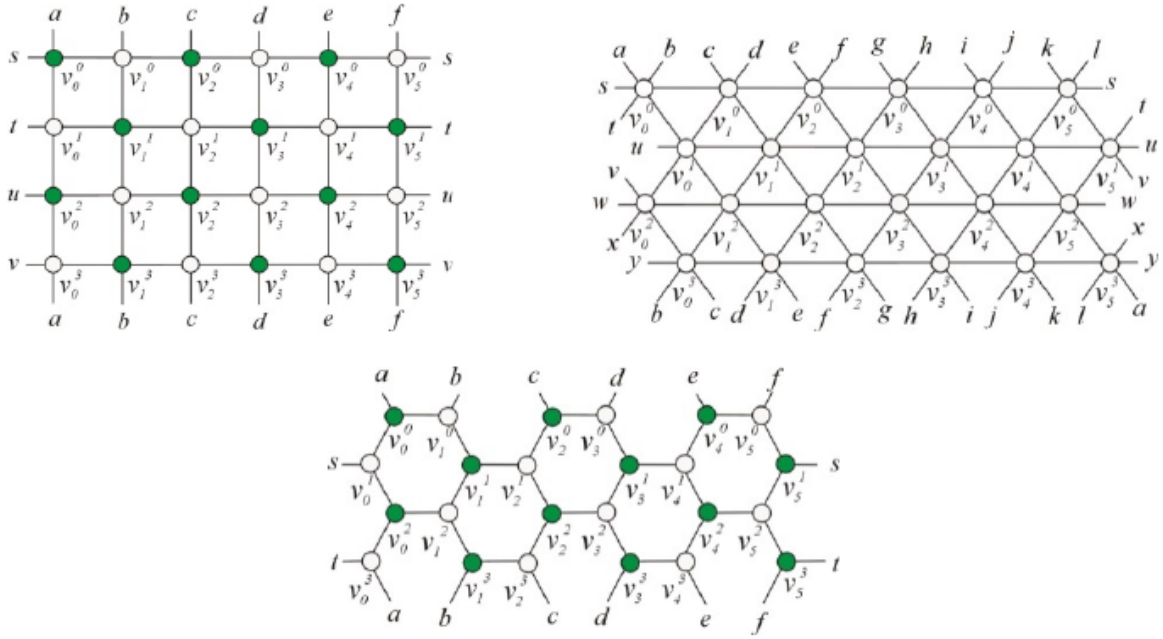


Figure 2. The 4×6 toroidal square, triangular and hexagonal lattices.

Input

Your program is to read from standard input. The input consists of T test cases. The number of test cases T is given in the first line of the input. Then, T lines are followed, where each line contains three integers, m, n and p, where $3 \leq m, n \leq 100$ and $p \in \{3, 6\}$, indicating that the corresponding input graph is an $m \times n$ toroidal lattice of type triangular (if $p = 3$) or hexagonal (if $p = 6$). You are always given a well-defined toroidal lattice.

Output

Your program is to write to standard output. The output consists of the results for the T test cases in the given order. For each test case, the first line must contain an integer indicating whether there exists a feasible solution. If yes, the integer must be '`1`'; otherwise '`-1`'. When and only when the first line is '`1`', it must be followed by mn lines, describing the sequence of vertices of the found cycle, where the index of vertex v_j^i is to be output as '`(i,j)`'. In case multiple solutions are possible, just output any one of them. No whitespace characters (blanks and/or tabs) are allowed inside a line.

The following shows a sample output for an input with one test case.

Sample Input

```
1
4 3 3
```

Sample Output

```
1
(0,0)
(1,0)
(2,0)
(3,0)
(0,1)
(1,1)
```

```
(2,1)
(3,1)
(3,2)
(2,2)
(1,2)
(0,2)
```