F. Microcycle

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Given an undirected weighted graph with n vertices and m edges. There is at most one edge between each pair of vertices in the graph, and the graph does not contain loops (edges from a vertex to itself). The graph is not necessarily connected.

A cycle in the graph is called simple if it doesn't pass through the same vertex twice and doesn't contain the same edge twice.

Find any simple cycle in this graph in which the weight of the lightest edge is minimal.

Input

The first line of the input contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. Then follow the descriptions of the test cases.

The first line of each test case contains two integers n and m ($3 \le n \le m \le \min(\frac{n \cdot (n-1)}{2}, 2 \cdot 10^5)$) — the size of the graph and the number of edges.

The next m lines of the test case contain three integers u, v, and w ($1 \le u, v \le n$, $u \ne v$, $1 \le w \le 10^6$) — vertices u and v are connected by an edge of weight w.

It is guaranteed that there is at most one edge between each pair of vertices. Note that under the given constraints, there is always at least one simple cycle in the graph.

It is guaranteed that the sum of the values of m for all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output a pair of numbers b and k, where:

- b — the minimum weight of the edge in the found cycle,
- k — the number of vertices in the found cycle.

On the next line, output k numbers from 1 to n — the vertices of the cycle in traversal order.

Note that the answer always exists, as under the given constraints, there is always at least one simple cycle in the graph.

Example

input Copy

```
5
6 6
1 2 1
2 3 1
3 1 1
4 5 1
5 6 1
6 4 1
6 6
1 2 10
```

```
2 3 8
3 1 5
4 5 100
5 6 40
6 4 3
6 15
1 2 4
5 2 8
6 1 7
6 3 10
6 5 1
3 2 8
4 3 4
5 3 6
2 6 6
5 4 5
4 1 3
6 4 5
4 2 1
3 1 7
1 5 5
4 6
2 3 2
1 3 10
1 4 1
3 4 7
2 4 5
1 2 2
4 5
2 1 10
3 1 3
4 2 6
1 4 7
2 3 3
```

output Copy

```
1 3
1 2 3
3 3
6 4 5
1 5
4 2 1 6 3
1 4
1 4 3 2
3 3
2 3 1
```