UVa Online Judge

12717 Fiasco

Natasha always mixes up things in her algorithm classes. Last week Prof. Chhaya Murthy gave the class a classical problem - finding shortest path to all nodes from a specific node of a weighted graph. Before that in another class, the professor taught them Prim's and Dijkstra's algorithms which are similar looking but does very different things. Always confused, poor Natasha submitted something very similar to the Prim's algorithm (actually we should call it Natasha's algorithm), she didn't test it properly. Pseudocode of her algorithm looks as follows:

```
function Shortest(Graph, source):
  for each vertex v in Graph:            /* Initializations */
      visited[v] := false;
      dist[v]  := infinity ;             /* Distance function from source */
      previous[v]  := undefined ;        /* Previous node in optimal path */
      ans[v]  := undefined ;             /* Answer array */
  end for
   dist[source]  := 0 ;
  Q := {}                                /* A priority queue which keeps (node, distance)
pairs. When pop is
called always gives the node with smallest distance. In case of a tie, returns the node with
smallest id. */
   Push (source, dist[source]) to Q
  while Q is not empty:                  /* The main loop */
    Pop node u from Q ;
    visited[u] := true;
    if dist[u] = infinity:
      break ;
    end if
    for each neighbor v of u:
      if visited[v] = true:
        continue;
      end if
        alt := edge_cost(u, v) ;         /* edge_cost(u, v) = cost of the edge between
vertices u and v */
      if alt < dist[v]:
        dist[v]  := alt ;
        previous[v]  := u ;
        Push (v, dist[v]) to Q;
      end if
    end for
  end while
  for each node node in the graph
    answer := 0
    u := node
    while previous[u] is defined:        /* Traverse the shortest path */
      answer := answer + edge_cost(u, previous[u]);
                                         /* edge_cost function as defined earlier */
      u := previous[u]
    end while ;
    ans[node] := answer;
  end for
  return ans;
endfunction
```

After submission, she showed the code to a friend who is good at algorithms. That friend immediately found the flaw. Natasha was really terrified and started to cry. Then there came the guys, the

guys who liked her. They jumped at the opportunity and got hold of the dataset to be used to test the solutions. Your friend Rehan is one of those guys. He really wanted to impress Natasha. He asked you to change the dataset in such a way that Natasha's algorithm gives the right result. After the change, Rehan will somehow be able to put the modified data with the old timestamp. Rehan told you that:

1. The dataset has T test cases.
2. Each case contains a connected weighted undirected graph with n nodes and m edges.
3. Each case will contain a source node *source*.
4. Edge weights are positive and distinct.

To avoid suspicion, Rehan asked you not to change which vertices the edges connect, or the overall set of edge weights. You may only reorder which weights are assigned to which edges.

Input

The first line of the input denotes the number of datasets T ($1 \leq T \leq 15$). T sets of case will follow. Each case will start with a triplet of numbers n ($2 \leq n \leq 2500$), m ($1 \leq m \leq 25000$) and *source* ($1 \leq source \leq n$) — the number of nodes, the number of edges and the starting node respectively. Each of the next m lines will contain a triplet of numbers (u, v, w) meaning that there is an edge between node u and node v with weight w ($1 \leq w \leq m$). Nodes are numbered from 1 to n. It is guaranteed that there is no duplicate or self-edges in the input and the graph is connected. In each given graph, all edge weights will be distinct.

Output

For each set of input, print one set of output. First line of a set should be of the format, '`Case` X`:`' where X is the case number. Then print each edge triplet — one triplet (u, v and w) in each line separated by a single space. The edges should be printed in the order given in the input. If there is more than one solution to a dataset, any one of them will do.

Sample Input

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

Sample Output

```
Case 1:
2 4 7
1 4 9
7 2 3
```

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