Problem D. Shortest Path Fast Algorithm

Recently, BaoBao has learned the Shortest Path Fast Algorithm (SPFA, or more formally, Bellman-Ford-Moore Algorithm) to solve the shortest path problem efficiently. He realizes that the algorithm looks so similar to the Dijkstra's algorithm after replacing the FIFO queue with priority queue, and shows you the below pseudo code.

Algorithm 1 The Shortest Path Fast Algorithm

```
1: function SPFA(G, s)                          ▷ G is the given graph and s is the source vertex
2:     Create vertex priority queue Q
3:     for each vertex v in G do
4:         dist[v] ← +∞                          ▷ Unknown distance from s to v
5:         vis[v] ← false                        ▷ No vertex is in Q at the beginning
6:     end for
7:     dist[s] ← 0                               ▷ Initialize distance from s to s to be 0
8:     Add s into Q with priority value 0
9:     vis[s] ← true
10:    cnt ← 0                                   ▷ Number of times we poll the priority queue
11:    while Q is not empty do
12:        Pick and remove best vertex u from Q  ▷ Explained below
13:        vis[u] ← false
14:        cnt ← cnt + 1
15:        for each neighbor v of u in G do
16:            d ← dist[u] + w_{u,v}             ▷ w_{u,v} is the weight of edge connecting vertices u and v in G
17:            if dist[v] > d then
18:                dist[v] ← d                   ▷ Update dist[v] if d is better
19:                if vis[v] is false then
20:                    Add v into Q with priority value d
21:                    vis[v] ← true
22:                end if
23:            end if
24:        end for
25:    end while
26: end function
```

By picking the best vertex from Q we mean picking the vertex with the smallest priority value (in case that multiple vertices have the smallest priority value, pick the vertex with the largest index among them).

You, the future computer scientist, find the BaoBao-modified SPFA algorithm works so slow in some carefully construted graph. However, BaoBao is sure that his algorithm works well, unless you show him a simple undirected graph that makes the variable `cnt` in the SPFA function no less than a certain k **at some time**.

Just teach him a lesson!

Input

There is only one test case in each test file.

The first and only line of the input contains a single integer k where $k = 1$ for the sample test case and $k = 10^5$ for the only secret test case.

Output

Output several lines in the following format to describe the input data of a simple undirected graph that makes the variable `cnt` in the SPFA function no less than k **at some time**.

The first line contains two integers n ($1 \le n \le 100$) and m ($0 \le m \le 10^3$), indicating the number of vertices and edges in the graph.

Then m lines follow, the i-th of which contains three integers u_i, v_i ($1 \le u_i, v_i \le n$) and w_i ($1 \le w_i \le 10^6$), indicating that the i-th edge in the graph has a weight of w_i and connects the u_i-th and the v_i-th vertices.

Note that a simple graph contains no self-loops and no multiple edges.

Example

standard input	standard output
1	4 6 1 2 1 2 3 2 3 4 3 4 1 4 1 3 5 2 4 6

Note

For your convenience, you can copy the `C++` code, which corresponds to the given pseudo code, from the contest website. Save the code as `spfa.cpp`, use `g++ spfa.cpp -O2 -o spfa` to compile it and you will get an executable file named `spfa`. Run `spfa`, feed your output to its standard input and it will print out the **final** value of `cnt`. Given the sample output it will print out 4, which means the sample output is not sufficient to pass the secret test case.

Note that the given code does not check the validity of your output (for example it does not check if your output is really a simple graph). You might still fail the test if your output is invalid, even if the executable prints out a large value.