Investigation

You are going to travel from Syrjälä to Lehmälä by plane. You would like to find answers to the following questions:

-> what is the minimum price of such a route?
-> how many minimum-price routes are there? (modulo 10^9+7)
-> what is the minimum number of flights in a minimum-price route?
-> what is the maximum number of flights in a minimum-price route?

Input

The first input line contains two integers n and m: the number of cities and the number of flights.
The cities are numbered 1,2,....,n. City 1 is Syrjälä, and city n is Lehmälä.

After this, there are m lines describing the flights.
Each line has three integers a, b, and c: there is a flight from city a to city b with price c. All flights are one-way flights.

You may assume that there is a route from Syrjälä to Lehmälä.

Output

Print four integers according to the problem statement.

Constraints

1 <= n <= 10^5
1 <= m <= 2 . 10^5
1 <= a,b <= n
1 <= c <= 10^9

Example

Input:
```
4 5
1 4 5
1 2 4
2 4 5
1 3 2
3 4 3
```

Output:
```
5 2 1 2
```

Problem Breakdown

Lets understand the problem first

We're given a weighted directed graph with n nodes and m edges.
The task is to go from node 1 to node n with the following objectives:

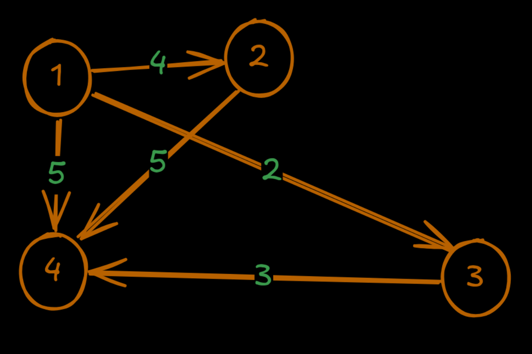
📌 You need to find:

The minimum total cost to reach node n.

The number of such minimum-cost paths.

The minimum number of edges used among such paths.

The maximum number of edges used among such paths.



Algorithm Choice

To find the shortest paths in a graph with non-negative weights, we use Dijkstra's Algorithm.
But here, we enhance it by tracking more information during the process.

We'll maintain a DP table for every node:

```
dp[node] = {
    minimum cost to reach node,
    number of shortest paths to node,
    min number of edges in those paths,
    max number of edges in those paths
}
```

Initialize DP Table

Create a dp[n+1][4] where each row represents:

dp[i][0] → min cost to reach node i.

dp[i][1] → number of min-cost paths to i.

dp[i][2] → min flights to i.

dp[i][3] → max flights to i.

```
dp[1] = {0, 1, 0, 0} // Starting at node 1 with cost 0, one path, and 0 flights
All other dp[i] = {INF, 0, INF, 0}
```

Modified Dijkstra's Algorithm

Use a min-priority queue pq (min heap) where we store {cost, node}.

For each node u popped from the queue:

Visit its neighbors v:

If a better cost is found → update everything from u to v.

If the same cost is found → update number of paths, min/max flights accordingly.

This is the heart of the algorithm.

Algorithm

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
const ll INF = 1e18;
const int MOD = 1e9 + 7;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n, m;
    cin >> n >> m;

    vector<vector<pair<int, int>>> graph(n + 1);

    for (int i = 0; i < m; ++i) {
        int u, v, wt;
        cin >> u >> v >> wt;
        graph[u].push_back({v, wt});
    }
```

Read input, construct an adjacency list
for the directed weighted graph.

```cpp
    vector<vector<ll>> dp(n + 1, vector<ll>(4, INF));

    dp[1][0] = 0;
    dp[1][1] = 1;
    dp[1][2] = 0;
    dp[1][3] = 0;
```

dp[i][0] = min cost
dp[i][1] = number of shortest paths
dp[i][2] = min flights
dp[i][3] = max flights

```cpp
    priority_queue<pair<ll, int>, vector<pair<ll, int>>, greater<>> pq;
    pq.push({0, 1});

    while (!pq.empty()) {
        auto [currDist, node] = pq.top();
        pq.pop();

        if (currDist > dp[node][0]) continue;

        for (auto [nbr, weight] : graph[node]) {
            ll newDist = currDist + weight;
            if (newDist < dp[nbr][0]) {
                dp[nbr][0] = newDist;
                dp[nbr][1] = dp[node][1]; // copy number of ways
                dp[nbr][2] = dp[node][2] + 1;
                dp[nbr][3] = dp[node][3] + 1;
                pq.push({newDist, nbr});
            } else if (newDist == dp[nbr][0]) {
                dp[nbr][1] = (dp[nbr][1] + dp[node][1]) % MOD;
                dp[nbr][2] = min(dp[nbr][2], dp[node][2] + 1);
                dp[nbr][3] = max(dp[nbr][3], dp[node][3] + 1);
            }
        }
    }
```

Modified Dijkstra's Algorithm

```cpp
    cout << dp[n][0] << " " << dp[n][1] << " " << dp[n][2] << " " << dp[n][3] << "\n";

    return 0;
}
```

Time Complexity

Dijkstra $O((n + m) * \log n)$
DP Updates $O(m)$

Space Complexity

$O(n + m)$ for graph + $O(n)$ for dp.