PROBLEM 1: Dijkstra?

Problem Link:

https://codeforces.com/contest/20/problem/C

Problem Statement:

A weighted and undirected graph is provided with n vertices and m edges.
Each edge connects two vertices and has a positive cost.
The objective is to determine one valid **shortest path** from vertex **1** to vertex **n**.
If no such route exists, the correct output is -1.
If a route exists, the output must list all vertices on that shortest route in correct order.

Hint:

Since all edge weights are positive, **Dijkstra's algorithm** is the correct technique for computing minimum distances.
To recover the actual route, a **predecessor array** is required so that the final path can be traced backward from vertex n.

Solution Approach :

1. Construction of the Graph

Store the graph using an adjacency list, which is suitable for up to 10^5 vertices and edges.
For every input edge (a, b, w):

- Insert (b, w) into the list of neighbors for a
- Insert (a, w) into the list of neighbors for b

This ensures accurate representation of the undirected structure.

2. Initialization

Three main components are needed:

- **Distance array dist[]**
    - Holds the shortest known distance to each vertex
    - Initialize with a very large number
    - Set dist[1] = 0 for the starting vertex
- **Predecessor array prev[]**

    - Stores the previous vertex on the best path discovered so far
    - Initially all entries are -1
- **Priority queue (min-heap)**
    - Contains pairs (distance, vertex)
    - Always extracts the vertex with the smallest tentative distance

3. Dijkstra's Algorithm Logic

The algorithm proceeds by repeatedly selecting the vertex with the minimal recorded distance.
For each extracted vertex u, examine all edges going from u to its neighbors.

For every neighbor (v, w):

- Compute a candidate distance: dist[u] + w
- If this candidate is less than the current dist[v], perform:
    - dist[v] = dist[u] + w
    - prev[v] = u
    - Insert (dist[v], v) into the priority queue

Outdated queue entries are ignored by comparing them with the current dist[] value.

4. Verification of Reachability

After all reachable vertices have been processed:

- If dist[n] remains at the initial infinite value, vertex n cannot be reached
    - Output should be -1
- Otherwise, a shortest route has been found and can be reconstructed

5. Path Reconstruction Steps

To build the final route:

1. Begin at vertex n
2. Follow prev[] repeatedly:
    - n → prev[n] → prev[prev[n]] → ...
3. Stop at vertex 1

4. Reverse the collected list to obtain the correct order
5. Output all vertices of the path

This produces one valid shortest path.

6. Complexity and Suitability

Time complexity of the solution is:

O((n + m) log n)

This complexity is efficient for the given limits.
Memory usage is also efficient due to adjacency lists and simple auxiliary arrays.

Pseudocode :

```
input n, m
adj = list of lists size n+1

for each edge:
   read a, b, w
   adj[a].add( (b, w) )
   adj[b].add( (a, w) )

INF = large number
dist[1..n] = INF
prev[1..n] = -1
dist[1] = 0

min-heap pq
pq.push(0, 1)

while pq not empty:
   (d, u) = pq.pop()
   if d != dist[u]: continue

   for each (v, w) in adj[u]:
      if dist[u] + w < dist[v]:
         dist[v] = dist[u] + w
         prev[v] = u
         pq.push(dist[v], v)

if dist[n] == INF:
   print -1
   stop

path = empty list
x = n
while x != -1:
   path.add(x)
   x = prev[x]

reverse path
print path
```