Problem 2: : Not The Best.

Problem Link: [Not the Best | LightOJ]

Problem Statement

- Given a weighted, undirected graph with n nodes and m edges.
- Find both the **shortest path** and the **second-shortest path** from node 1 to node n.
- The second-shortest path must be **strictly longer** than the shortest path (not equal).
- Paths may revisit nodes or edges (i.e., paths are **not necessarily simple**).

Hint

- Use a **modified Dijkstra's algorithm** to track two best distances for each node:
  - dist[u][0] = shortest distance to u
  - dist[u][1] = second-shortest distance to u
- Use a **priority queue** that stores (u, state, d) where:
  - u = node
  - state = 0 (shortest) or 1 (second-shortest)
  - d = current distance
- When relaxing edges u → v with weight w:
  - Compute alt = dist[u][k] + w for k = 0 and 1.
  - If alt < dist[v][0]:
    - dist[v][1] = dist[v][0] (downgrade old shortest)
    - dist[v][0] = alt
    - Push both (v, 0, dist[v][0]) and (v, 1, dist[v][1]) into the queue
  - Else if dist[v][0] < alt < dist[v][1]:
    - dist[v][1] = alt
    - Push (v, 1, alt) into the queue

Solution Approach (Step-by-Step)

1. Graph Representation

- Use an **adjacency list**: for each node u, store (v, w) for its neighbors.

2. Distance Arrays

- dist[nodes][2]
  - dist[u][0] = shortest distance
  - dist[u][1] = second-shortest distance
- Initialize both to **infinite**
- Set dist[1][0] = 0

3. Visited / Process Arrays

- vis[u][0] and vis[u][1] to track whether a state has been finalized

4. Priority Queue

- Store entries (u, state, d)
- Min-heap sorted by d

5. Modified Dijkstra Loop

1. While queue is not empty:
   - Pop (u, state, d)
   - Skip if vis[u][state] is true
   - Set vis[u][state] = true
   - For each neighbor (v, w) of u:
     - alt = d + w
     - **Case A – New shortest for v:**
       - If alt < dist[v][0]
         - dist[v][1] = dist[v][0]
         - dist[v][0] = alt
         - Push (v, 0, dist[v][0]) and (v, 1, dist[v][1]) into queue
     - **Case B – Between shortest and second:**
       - Else if dist[v][0] < alt < dist[v][1]
         - dist[v][1] = alt
         - Push (v, 1, alt) into queue

6. Answer

- After completion, dist[n][1] is the **second-shortest distance** to node n
- Print dist[n][1]

Complexity

- **Time Complexity:** O((n + m) log(n + m)) (two states per node, edges relaxed possibly twice)
- **Memory Complexity:** O(n + m) for adjacency list + O(n) for distance and visited arrays

Pseudocode

```
function SecondShortestPath(n, adjacency, source=1, target=n):
    for each node u in 1..n:
        dist[u][0] = INF
        dist[u][1] = INF
        vis[u][0] = vis[u][1] = false

    dist[source][0] = 0
    priority_queue Q
    push (source, 0, 0) into Q   // (node, state, distance)

    while Q not empty:
        (u, state, d) = pop Q
        if vis[u][state]:
            continue
        vis[u][state] = true

        for each neighbor (v, w) in adjacency[u]:
            alt = d + w

            if alt < dist[v][0]:
                dist[v][1] = dist[v][0]
                dist[v][0] = alt
                push (v, 0, dist[v][0]) into Q
                push (v, 1, dist[v][1]) into Q

            else if dist[v][0] < alt < dist[v][1]:
                dist[v][1] = alt
                push (v, 1, alt) into Q

    return dist[target][1]
```

Implementation Link:

[algorithm-/Dijkstra/Not the best/not_the_best.cpp at main · Jannat651/algorithm-]()