Not The Best

Second Shortest Path – Modified Dijkstra

This problem requires finding the **second shortest path** from **node 1** to **node N** in an **undirected graph** where **backtracking and revisiting nodes/edges are allowed**.

The key insight is:

- Maintain **both the shortest and second shortest distances** to each node.
- Relax edges from **both shortest and second shortest paths**.
- This ensures discovering paths that may slightly deviate from the optimal path, including small detours or reused edges.
- Complexity remains **O((V+E) log V)** due to priority queue operations.

Steps

1. Initialize Arrays

- Create two arrays:
  - `dist1` → shortest distances
  - `dist2` → second shortest distances
- Initialize all values to a **very large number (infinity)**.
- Create a **min-priority queue** storing pairs `(distance, node)`.

2. Setup Starting Node

- `dist1[1] = 0`
- Push `(0, 1)` into the priority queue.

3. Process the Queue

- While the priority queue is not empty:
    - Remove the smallest `(distance, node)` pair from the queue.
    - If this distance is larger than the current **second shortest** for that node, skip it.
    - Otherwise, examine all neighbors connected to this node.

4. Check Each Neighbor

- For each neighbor connected by an edge with weight `w`:
    - `new_distance = current_distance + w`
    - If `new_distance < dist1[neighbor]`:
        - `dist2[neighbor] = dist1[neighbor]`
        - `dist1[neighbor] = new_distance`
        - Push `(new_distance, neighbor)` into the queue.
    - Else if `dist1[neighbor] < new_distance < dist2[neighbor]`:
        - `dist2[neighbor] = new_distance`
        - Push `(new_distance, neighbor)` into the queue.

5. Get the Answer

- After processing all nodes:
    - `dist2[N]` → length of the **second shortest path** from node 1 to node N.

Input:

2

3 3

1 2 100

2 3 200

1 3 50

Execution:

1. Initialize:

o dist1 = [0, ∞, ∞], dist2 = [∞, ∞, ∞] o Queue: [(0,1)]

2. Process (0,1):

o 1→2: new=100 < ∞ → dist1[2]=100, push (100,2) o 1→3:

new=50 < ∞ → dist1[3]=50, push (50,3)

o Queue: [(50,3), (100,2)]

3. Process (50,3):

o 3→1: new=100 > 0 but < ∞ → dist2[1]=100, push (100,1) o

3→2: new=250 > 100 but < ∞ → dist2[2]=250, push

(250,2) o Queue: [(100,2), (100,1), (250,2)]

4. Process (100,2):

o 2→1: new=200 > 100 (dist2[1]) → skip o 2→3: new=300 >

50 but < ∞ → dist2[3]=300, push (300,3) o Queue: [(100,1),

(250,2), (300,3)]

5. Process (100,1):

o 1→2: new=200 > 100 but < 250 → dist2[2]=200, push

(200,2) o 1→3: new=150 > 50 but < 300 → dist2[3]=150,

push (150,3) o Queue: [(150,3), (200,2), (250,2), (300,3)]

6. Process (150,3): o Target reached with second shortest = 150

Answer: 150

Output:

150

Pseudocode:

```
FOR each test case:
    READ graph
    INIT dist1 = INF, dist2 = INF
    dist1[1] = 0
    PUSH (0, 1) to min-heap

    WHILE heap not empty:
        POP (d, u)
        IF d > dist2[u]: SKIP

        FOR each neighbor v with weight w:
            new_dist = d + w
            IF new_dist < dist1[v]:
                dist2[v] = dist1[v]
                dist1[v] = new_dist
                PUSH (new_dist, v)
            ELSE IF new_dist > dist1[v] AND new_dist < dist2[v]:
                dist2[v] = new_dist
                PUSH (new_dist, v)
```

OUTPUT dist2[N]

Code:

https://github.com/shanto470/algorithm_plm/blob/main/Dijkstra/Not%20The%20Best/notTheBest.cpp