# Not The Best

This problem involves finding the second shortest path from node 1 to node N in an undirected graph, where paths are allowed to revisit nodes or edges. Unlike simple pathfinding, the second shortest path is not necessarily a simple path—it may include cycles or repeated segments that make it slightly longer than the shortest path. To handle this, we can use a modified Dijkstra's algorithm that maintains both the shortest and second shortest distances to each node. When relaxing edges, we update both distances appropriately, allowing the algorithm to consider paths that deviate slightly from the optimal route while still being efficient. This approach systematically explores all paths that exceed the shortest distance but are minimal among such alternatives, correctly identifying the second shortest path even if it involves backtracking or reusing edges, with a complexity of $O((V + E)\log\ V)$.

## Steps

1. **Initialize Arrays**
    - Create two arrays: dist1 for the shortest distances and dist2 for the second shortest distances.
    - Set all values in both arrays to a very large number (infinity).
    - Create a min-priority queue to store pairs of (distance, node).
2. **Setup Starting Node**
    - Set the shortest distance to node 1 as 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 distance 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:
        - Calculate new_distance = current_distance + w.
        - **If** new_distance < dist1[neighbor]:
            - Set dist2[neighbor] = dist1[neighbor].
            - Set dist1[neighbor] = new_distance.
            - Push (new_distance, neighbor) into the queue.
        - **Else if** dist1[neighbor] < new_distance < dist2[neighbor]:
            - Set dist2[neighbor] = new_distance.
            - Push (new_distance, neighbor) into the queue.
5. **Get the Answer**
    - After processing all nodes, dist2[N] contains the second shortest distance from node 1 to node N.
    - This value is the length of the second best path.

## Input:

2

3 3

1 2 100

2 3 200

1 3 50

---

## Execution:

### Step 1: Initialize

- dist1 = [0, ∞, ∞], dist2 = [∞, ∞, ∞]

- Priority Queue: [(0, 1)]

---

### Step 2: Process (0, 1)

- Edge 1→2: new = 0 + 100 = 100 < ∞ → dist1[2] = 100, push (100, 2)
- Edge 1→3: new = 0 + 50 = 50 < ∞ → dist1[3] = 50, push (50, 3)
- Queue: [(50, 3), (100, 2)]

---

### Step 3: Process (50, 3)

- Edge 3→1: new = 50 + 50 = 100 > dist1[1] = 0 → dist2[1] = 100, push (100, 1)
- Edge 3→2: new = 50 + 200 = 250 > dist1[2] = 100 → dist2[2] = 250, push (250, 2)
- Queue: [(100, 2), (100, 1), (250, 2)]

---

### Step 4: Process (100, 2)

- Edge 2→1: new = 100 + 100 = 200 > dist2[1] = 100 → skip
- Edge 2→3: new = 100 + 200 = 300 > dist1[3] = 50 → dist2[3] = 300, push (300, 3)
- Queue: [(100, 1), (250, 2), (300, 3)]

---

### Step 5: Process (100, 1)

- Edge 1→2: new = 100 + 100 = 200 > dist1[2] = 100 but < dist2[2] = 250 → dist2[2] = 200, push (200, 2)
- Edge 1→3: new = 100 + 50 = 150 > dist1[3] = 50 but < dist2[3] = 300 → dist2[3] = 150, push (150, 3)
- Queue: [(150, 3), (200, 2), (250, 2), (300, 3)]

---

### Step 6: Process (150, 3)

- Target node reached; dist2[3] = 150 → second shortest distance

## Output

150

## Pseudocode:

```
FOR each test case:
    READ graph
    INIT dist1[1..N] = ∞
    INIT dist2[1..N] = ∞
    dist1[1] = 0
    PUSH (0, 1) into min-heap

    WHILE heap is not empty:
        POP (d, u) from heap
        IF d > dist2[u]:
            CONTINUE  // skip if larger than second shortest

        FOR each neighbor v of u with edge weight w:
            new_dist = d + w

            IF new_dist < dist1[v]:
                dist2[v] = dist1[v]
                dist1[v] = new_dist
                PUSH (new_dist, v) into heap

            ELSE IF dist1[v] < new_dist < dist2[v]:
```

```
            dist2[v] = new_dist
            PUSH (new_dist, v) into heap

    OUTPUT dist2[N]  // second shortest distance to node N
```

## Here is the Not the Best solution code:

https://github.com/Hazra32/Algorithm-Problem/blob/main/Dijkstra/Not%20The%20Best/notthebest.cpp