LOJ 1108

This problem presents a shortest path challenge with a unique cost function based on the **cube of the difference in busyness values between junctions**. The core complexity arises from the mathematical nature of the cost calculation: when traveling from junction **u** to junction **v**, the cost is defined as:

$$cost=(busyness[v]-busyness[u])^3$$

This cubic function can produce **negative edge weights** if the destination junction is less busy than the source junction, potentially forming **negative cycles**.

The goal is to find the **minimum total cost** (earning from the city authority's perspective) from junction 1 to various query junctions. Additionally, any total cost **less than 3 units** should be reported as `'?'`.

The solution requires careful handling of **negative cycles** and **unreachable nodes**. If a negative cycle exists along a path to a junction, the total cost can become arbitrarily small, effectively undefined. Similarly, if the computed minimum cost is less than 3, it should also be reported as `'?'`.

To solve this, the **Bellman-Ford algorithm** is used to compute shortest paths and detect negative cycles reachable from the source. All nodes affected by negative cycles must also be marked to ensure correct query results.

Steps

1. **Read Input**
    - Read number of junctions.
    - Read busyness values for each junction.
    - Read number of roads.
    - Store each road as a directed edge with weight = ((busyness[destination] - busyness[source])^3).
    - Read number of queries and query junctions.

2. **Initialize Arrays**
    - Create a distance array with large values.
    - Set distance of junction 1 to 0.
    - Create an array to mark nodes affected by negative cycles.
3. **Run Bellman-Ford Algorithm**
    - Repeat **n-1 times**:
        - For each edge (u → v):
            - If we can get a shorter path to v through u, update `distance[v]`.
4. **Check for Negative Cycles**
    - Do **one more pass** through all edges.
    - If any distance can still be improved, mark that node as in a negative cycle.
    - Use **BFS/DFS** to mark all nodes reachable from cycle nodes.
5. **Answer Queries**
    - For each query junction:
        - If unreachable **OR** in negative cycle **OR** distance < 3 → print `'?'`.
        - Else → print the distance.
6. **Output Results**
    - Print `"Set #"` with case number.
    - Print each query result on a separate line.

Input:

5 6 7 8 9 10

6

1 2

2 3

3 4

1 5

5 4

4 5

2

4

5

Execution:

Step 1: Read Input

n = 5 busyness array: index 1=6, index 2=7, index 3=8, index 4=9, index

5=10 r = 6 roads

Compute edge weights:

1→2: (7-6)^3 = 1

2→3: (8-7)^3 = 1

3→4: (9-8)^3 = 1

1→5: (10-6)^3 = 64

5→4: (9-10)^3 = -1

4→5: (10-9)^3 = 1 q = 2

queries: 4 and 5

Step 2: Initialize Arrays

dist = [inf, 0, inf, inf, inf, inf] for nodes 1-5

inCycle = all false visited = all false

Step 3: Run Bellman-Ford (4 iterations)

Iteration 1:

Process edge 1→2: 0+1=1 < inf → update dist[2]=1

Process edge 2→3: 1+1=2 < inf → update dist[3]=2

Process edge 3→4: 2+1=3 < inf → update dist[4]=3

Process edge 1→5: 0+64=64 < inf → update dist[5]=64

Process edge 5→4: 64+(-1)=63 > 3 → no update Process

edge 4→5: 3+1=4 < 64 → update dist[5]=4

Iteration 2:

All edges checked, no improvements found

Iterations 3-4: No

changes

Current distances: dist[1]=0, dist[2]=1, dist[3]=2, dist[4]=3, dist[5]=4

Step 4: Detect Negative Cycles

Run Bellman-Ford again to check for cycles All

edges processed, no distance improvements

inCycle array remains all false

No BFS propagation needed since no cycles detected

Step 5: Process Queries

Query for node 4: dist[4]=3, not inf, not in cycle, ≥3 → output 3

Query for node 5: dist[5]=4, not inf, not in cycle, ≥3 → output 4

Output:

Set #1

3

4

Pseudocode:

READ n

READ busyness[1..n]

READ r roads, store edges with weight = (busyness[v]-busyness[u])3

READ q queries

dist[1..n] = INF, dist[1] = 0

REPEAT n-1 times:

FOR each edge (u,v,w):

IF dist[u] + w < dist[v]:

dist[v] = dist[u] + w

inCycle[1..n] = false

REPEAT n-1 times:

FOR each edge (u,v,w):

IF dist[u] + w < dist[v]:

inCycle[v] = true

Propagate to all reachable nodes

FOR each query:

IF dist[query] < 3 OR unreachable OR inCycle: PRINT "?"

ELSE: PRINT dist[query]

Code:

https://github.com/shanto470/algorithm_plm/blob/main/BellmanFord/LOj%201108/LOj1108.cpp