LOJ 1108

LOJ 1108 is a shortest-path problem with a special cost function: traveling from junction **u** to junction **v** costs $(\text{busyness}[v] - \text{busyness}[u])^3$. Because this is a cubic function, some roads can have **negative costs**, which means the graph can contain **negative cycles**. If a path from the source to a junction passes through a negative cycle, the total cost can become arbitrarily small, making the minimum cost undefined. Additionally, any path with a total cost less than 3 must be reported as ?. To solve this, the **Bellman-Ford algorithm** is used, as it handles negative weights and can detect negative cycles. The algorithm computes shortest paths from junction 1, propagates the influence of negative cycles to affected junctions, and ensures that any query junction with a cost less than 3 or affected by a negative cycle is correctly marked with ?.

Steps

1. Read Input

- Read the number of junctions.
- Read the busyness values for each junction.
- Read the number of roads and store each road as a **directed edge** with weight:

$$\text{weight} = (\text{busyness[destination]} - \text{busyness[source]})^3$$

- Read the number of queries and the query junctions.

2. Initialize Arrays

- Create a distance array with large values (e.g., ∞).
- Set distance[1] = 0 (starting junction).
- Create an array to mark nodes affected by negative cycles.

3. Run Bellman-Ford Algorithm

- Repeat $n - 1$times:
    - For each edge $u \to v$:
        - If distance[u] + weight(u, v) < distance[v], 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 part of a negative cycle.
- Use **BFS/DFS** to mark all nodes reachable from these cycle nodes.

5. Answer Queries

- For each query junction:
    - If unreachable, in a negative cycle, or distance < 3 → print ?.
    - Otherwise → print distance.

6. Output Results

- Print "Set #" with the 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] (1-indexed)
- inCycle = [false, false, false, false, false, false]

Step 3: Run Bellman-Ford (n-1 = 4 iterations)

Iteration 1:

- 1→2: 0 + 1 < inf → dist[2] = 1
- 2→3: 1 + 1 < inf → dist[3] = 2
- 3→4: 2 + 1 < inf → dist[4] = 3
- 1→5: 0 + 64 < inf → dist[5] = 64
- 5→4: 64 + (-1) = 63 → no update (3 < 63)
- 4→5: 3 + 1 = 4 < 64 → dist[5] = 4

Iterations 2–4:

- No further updates.

Current distances:

dist[1]=0, dist[2]=1, dist[3]=2, dist[4]=3, dist[5]=4

Step 4: Detect Negative Cycles

- Run one more pass through all edges:
- No distance improves → no negative cycles.
- inCycle remains all false.

Step 5: Process Queries

- Query 4: dist[4] = 3 → ≥3, not in cycle → output **3**
- Query 5: dist[5] = 4 → ≥3, not in cycle → output **4**

Step 6: Output

Set #1

3

4

Pseudocode:

```
function LOJ_1108():
    read n  // number of junctions
    busyness[1..n]  // array of busyness values

    read r  // number of roads
    edges = empty list
    for i = 1 to r:
```

```
    read u, v  // directed edge u->v
    weight = (busyness[v] - busyness[u])^3
    edges.append((u, v, weight))

read q  // number of queries
queries = read q junctions

// Step 1: Initialize
dist[1..n] = INF
dist[1] = 0  // start from junction 1
inCycle[1..n] = false

// Step 2: Bellman-Ford algorithm
for i = 1 to n-1:
    for each (u, v, weight) in edges:
        if dist[u] + weight < dist[v]:
            dist[v] = dist[u] + weight

// Step 3: Detect negative cycles
for each (u, v, weight) in edges:
    if dist[u] + weight < dist[v]:
        inCycle[v] = true

// Propagate negative cycle effect
queue = all nodes v where inCycle[v] = true
while queue not empty:
```

```
    node = queue.pop()
    for each neighbor of node:
        if not inCycle[neighbor]:
            inCycle[neighbor] = true
            queue.push(neighbor)

// Step 4: Answer queries
print "Set #1"
for junction in queries:
    if dist[junction] < 3 or dist[junction] == INF or inCycle[junction]:
        print "?"
    else:
        print dist[junction]
```

Here is the code for LOj 1108:

https://github.com/Hazra32/Algorithm-Problem/blob/main/Bellmam%20Ford/LOj%201108/loj1108.cpp