Not the best — [Not the best Problem in Lightoj]

সমস্যাটি কী?

Robin একটি গ্রামের ভেতর বসবাস করে। গ্রামে মোট **N**টি **intersection (**নোড**)** এবং **R**টি দ্বিমুখী রাস্তা **(bidirectional edges)** রয়েছে।

Robin সাধারণত সুন্দর দৃশ্য দেখার জন্য সবচেয়ে ছোট পথটি নেয় না— বরং সে দ্বিতীয় সর্বোত্তম পথ (second-best / second shortest path) নিতে চায়।

সমস্যা:

Robin যাবে **intersection 1** → **intersection N** কিন্তু নেবে **second-shortest path**, যেখানে—

- পথটি shortest path অপেক্ষা বড়
- কিন্তু অন্য সব বড় পথের চেয়ে ছোট
- এমনকি চাইলে edge পুনরায় ব্যবহার করতে পারে

কাজ:

প্রতিটি টেস্ট কেসের জন্য
`Case X: second_shortest_path_cost` প্রিন্ট করতে হবে।

সমাধান ধারণা

এই সমস্যার মূল টেকনিক হলো:

Modified Dijkstra Algorithm

যেখানে প্রতি নোডের জন্য আমরা রাখব দুইটি দূরত্ব:

1. dist1[v] — shortest distance
2. dist2[v] — second shortest distance

গ্রাফে edge relax করার সময়:

- যদি নতুন পথ shortest থেকে ছোট হয় → shortest আপডেট হবে, পুরনো shortest → second shortest
- যদি shortest < নতুন পথ < second shortest হয় → শুধু second shortest আপডেট হবে

এভাবে কাজ করার কারণ Dijkstra naturally কম দূরত্বের পথ আগে process করে। তাই dist1 সবসময় সর্বনিম্ন হবে এবং dist2 নিশ্চিতভাবে **second best** হবে।

মূল উপায়

- priority_queue থেকে state (দূরত্ব + নোড) বের করি
- adjacency list দিয়ে সব edges relax করি
- শেষ উত্তর: `dist2[N]`

ইনপুট-আউটপুট বর্ণনা

ইনপুট:

```
T
N R
u v w
u v w
...
```

- T = টেস্ট কেস
- N = নোড সংখ্যা (1–5000)
- R = রাস্তার সংখ্যা (1–100,000)
- u–v = সংযুক্ত intersection
- w = রাস্তার ওজন (1–5000)

আউটপুট:

```
Case X: result
```

উদাহরণ:

ইনপুট

```
2
3 3
1 2 100
2 3 200
1 3 50
4 4
1 2 100
2 4 200
2 3 250
3 4 100
```

আউটপুট

```
Case 1: 150
Case 2: 450
```

গ্রাফের উপস্থাপন

আমরা গ্রাফটি **adjacency list** দিয়ে উপস্থাপন করি:

```
vector<vector<Edge>> adj(N+1);
struct Edge {
    int to;
    int weight;
};
```

adjacency list ব্যবহারের কারণ

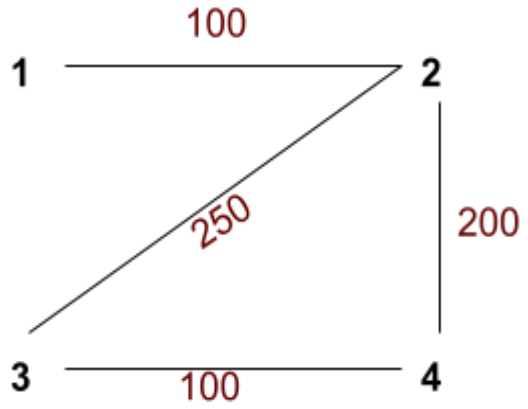
- R খুব বড় ($১০^৫$ পর্যন্ত)
- adjacency matrix হলে memory = N^2 = 25 মিলিয়ন যা অকার্যকর
- list কম জায়গায় দ্রুত traversal করতে দেয়

Step-by-Step Visualization

দেখা যাক, কীভাবে "second shortest path" খুঁজে বের হয়।

1
4 4
1 2 100
2 4 200
2 3 250
3 4 100

এখানে Robin যাবে **1** → **4**, এবং আমাদের লক্ষ্য **second-shortest path** বের করা।



Initial

Node	dist1	dist2
1	0	∞
2	∞	∞
3	∞	∞
4	∞	∞

Priority Queue = `{ (0,1) }`

Step 1: নোড **1** প্রসেস করা

Neighbour:

1 → 2 = 100

- dist1[2] = 100

Node	dist1	dist2
1	0	∞
2	100	∞
3	∞	∞
4	∞	∞

Priority Queue = `{ (100,2) }`

Step 2: নোড **2** প্রসেস করা

Neighbour:

(A) 2 → 4 = 100 + 200 = 300

- dist1[4] = 300

(B) 2 → 3 = 100 + 250 = 350

- dist1[3] = 350

Node	dist1	dist2
1	0	∞
2	100	∞
3	350	∞
4	300	∞

Priority Queue = `{ (300,4), (350,3) }`

Step 3: নোড **4** প্রসেস করা **(dist1 = 300)**

Neighbour:

(A) 4 → 2 = 300 + 200 = 500

এটি dist1[2] = 100 এর চেয়ে বড় এবং dist2[2] = ∞ এর চেয়ে ছোট → dist2[2] = 500

Node	dist1	dist2
1	0	∞
2	100	500
3	350	∞
4	300	∞

Priority Queue = `{ (350,3), (500,2) }`

Step 4: নোড **3** প্রসেস করা **(dist1 = 350)**

Neighbour:

(A) 3 → 2 = 350 + 250 = 600

600 > dist1[2] (= 100) কিন্তু < dist2[2] (= 500)
=> কোনো আপডেট নয়।

(B) 3 → 4 = 350 + 100 = 450

450 > dist1[4] (= 300) এবং < dist2[4] (= ∞)
=> dist2[4] = 450

Node	dist1	dist2
1	0	∞
2	100	500
3	350	∞
4	300	450

Priority Queue = `{ (450,4), (500,2) }`

Step 5: নোড **4** প্রসেস করা **(dist2 = 450)**

450 হচ্ছে second-shortest entry
Neighbors check করলেও আর কোনো ভালো পথ পাওয়া যাবে না, কারণ সবই 450 + positive weight → dist2 থেকে বড় হয়ে যাবে।

No more updates.

Final Answer

Shortest path = dist1[4] = 300

(1 → 2 → 4)

Second shortest path = dist2[4] = 450

(1 → 2 → 3 → 4)

Final Output:

```
Case 1: 450
```

Pseudocode

```
FUNCTION solve():

    READ N, R
    IF input invalid:
        RETURN -1

    CREATE adjacency list adj[1..N]

    FOR i = 1 to R:
        READ u, v, w
        ADD (v, w) to adj[u]
        ADD (u, w) to adj[v]

    INITIALIZE dist1[1..N] = INF
    INITIALIZE dist2[1..N] = INF

    CREATE min-heap priority queue pq

    dist1[1] = 0
    PUSH (0, 1) INTO pq          // (distance, node)

    WHILE pq is NOT empty:

        (d, u) = pq.pop()

        IF d > dist2[u]:
            CONTINUE

        FOR each edge (u → v) with weight w in adj[u]:

            d_new = d + w

            // Case 1: Found a better shortest path
            IF d_new < dist1[v]:
```

```
            dist2[v] = dist1[v]
            dist1[v] = d_new

            PUSH (dist1[v], v) INTO pq

            IF dist2[v] != INF:
               PUSH (dist2[v], v) INTO pq

        // Case 2: Not shortest, but candidate for second shortest
        ELSE IF d_new > dist1[v] AND d_new < dist2[v]:

            dist2[v] = d_new
            PUSH (dist2[v], v) INTO pq

    RETURN dist2[N]

END FUNCTION



MAIN:

    READ T

    FOR i = 1 to T:
        result = solve()
        IF result != -1:
            PRINT "Case i: result"
```

Time Complexity Analysis

অপারেশন	জটিলতা
প্রতিটি এজ relax	O(log N)
মোট এজ	R
dist1 + dist2 এর জন্য অতিরিক্ত কিউ অপারেশন	সর্বোচ্চ 2R

সুতরাং মোট কমপ্লেক্সিটি:

$$O(R \log N)$$

যা N = 5000 এবং R = 100000 এর জন্য একদম পারফেক্ট।