Dijkstra? — Dijkstra? Problem from Codeforces

সমস্যাটি কী?

এই সমস্যাটির লক্ষ্য হলো একটি **weighted undirected graph** -এ **vertex 1** থেকে **vertex n** পর্যন্ত সর্বনিম্ন (minimum) cost বা দূরত্বের পথ বের করা।

অর্থাৎ, আমরা জানতে চাই —"vertex 1 থেকে vertex n পর্যন্ত যেতে কত কম edge weight sum লাগে, এবং কোন পথ দিয়ে যেতে হবে।"

ইনপুট বর্ণনা

```
n m
a1 b1 w1
a2 b2 w2
...
am bm wm
```

প্রতীক	অর্থ
`n`	মোট vertex সংখ্যা ($2 \le n \le 10^5$)
`m`	মোট edge সংখ্যা ($0 \le m \le 10^5$)
`a`, `b`	দুটি vertex যা একে অপরের সাথে যুক্ত
`w`	ঐ edge-এর ওজন ($1 \le w \le 10^6$)

Graph-এ **multiple edges** ও **loops** থাকতে পারে।

আউটপুট বর্ণনা

- যদি 1 থেকে n পর্যন্ত কোনো পথ না থাকে, প্রিন্ট করতে হবে `-1`
- অন্যথায়, **shortest path** টা প্রিন্ট করতে হবে (vertex গুলো ক্রমানুসারে)।

সমস্যার প্রকৃতি

এটি একটি **Single Source Shortest Path (SSSP)** সমস্যা, যেখানে edge weight গুলো **positive**।

- যদি সব edge এর weight সমান হতো (যেমন = 1), তবে BFS দিয়ে সমাধান করা যেত।
- কিন্তু এখানে weight গুলো ভিন্ন (1 থেকে 10^6 পর্যন্ত), তাই **Dijkstra's Algorithm** সবচেয়ে কার্যকর সমাধান।

Dijkstra's Algorithm সংক্ষেপে

Dijkstra's Algorithm মূলত **Greedy approach** অনুসরণ করে।

1. **Start node (vertex 1)** থেকে শুরু করা হয়।
2. প্রতিবার এমন vertex বেছে নেওয়া হয় যার দূরত্ব সবচেয়ে ছোট (minimum distance)।
3. তার সমস্ত neighbour আপডেট করা হয় — যদি নতুন দূরত্বটি কম হয়।
4. এই প্রক্রিয়া চলতে থাকে যতক্ষণ না আমরা গন্তব্য vertex n-এ পৌঁছাই।

গ্রাফের উপস্থাপন **(Representation)**

গ্রাফটি **Adjacency List** দিয়ে রাখা সবচেয়ে উপযুক্ত।
প্রতি vertex একটি list রাখবে যেখানে তার প্রতিবেশী এবং ঐ edge-এর ওজন থাকবে।

উদাহরণ:

```
1 → (2, 2), (4, 1)
2 → (1, 2), (3, 4), (5, 5)
4 → (1, 1), (3, 3)
3 → (2, 4), (4, 3), (5, 1)
5 → (2, 5), (3, 1)
```

ভেরিয়েবল সমূহ

ভেরিয়েবল	বর্ণনা
`dist[i]`	vertex 1 থেকে vertex i পর্যন্ত সর্বনিম্ন দূরত্ব
`orig[i]`	vertex i এর parent (কোন vertex থেকে এসেছে)
`pq`	একটি min-heap (priority queue), যা সবসময় সবচেয়ে ছোট dist যুক্ত vertex দেয়
`INF`	খুব বড় একটি মান, যেমন 1e18

PseudoCode

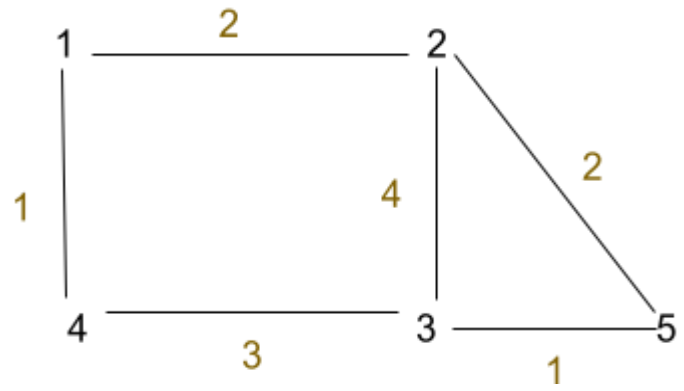
```
FUNCTION Dijkstra(n, m, edges):
    Initialize graph as adjacency list
    For each edge (a, b, w):
        Add (b, w) to graph[a]
        Add (a, w) to graph[b]

    For i = 1 to n:
        dist[i] = INF
        orig[i] = -1

    dist[1] = 0
    pq = priority_queue()
    pq.push((0, 1))   // (distance, vertex)

    WHILE pq is not empty:
        (d, u) = pq.top()
        pq.pop()

        IF d > dist[u]:
            CONTINUE

        FOR each (v, w) in graph[u]:
            IF dist[u] + w < dist[v]:
                dist[v] = dist[u] + w
                orig[v] = u
                pq.push((dist[v], v))

    IF dist[n] == INF:
        RETURN -1
```

```
    ELSE:
        BUILD path from n to 1 using orig[]
        REVERSE path
        PRINT path
```

উদাহরণ ব্যাখ্যা

Input:

```
5 6
1 2 2
2 5 5
2 3 4
1 4 1
4 3 3
3 5 1
```



Step-by-Step Visualization

Step	Current Node	Neighbors (v, w)	Relaxation	Updated dist[ ]	PQ Content
Init	—	—	—	dist[1]=0, others=∞	(0,1)
1	1	(2,2), (4,1)	dist[2]=2, dist[4]=1	[0,2,∞,1,∞]	(1,4), (2,2)
2	4	(1,1), (3,3)	dist[3]=4	[0,2,4,1,∞]	(2,2), (4,3)
3	2	(1,2), (5,5), (3,4)	dist[5]=7 (temporary), dist[3]=4 (same)	[0,2,4,1,7]	(4,3), (7,5)

4	3	(2,4), (4,3), (5,1)	dist[5]=5 (better!)	[0,2,4,1,5]	(5,5)
5	5	done	—	—	—

Path Reconstruction

`orig[]` ট্রেস করলে পাওয়া যায়:

`5 ← 3 ← 4 ← 1`

অর্থাৎ path:
1 → 4 → 3 → 5

Output:

```
1 4 3 5
```

Time Complexity Analysis

জটিলতা	বিশ্লেষণ
Time Complexity	**O((n + m) log n)** — কারণ প্রতিটি edge সর্বাধিক একবার relax হয় এবং priority queue ব্যবহার হয়
Space Complexity	**O(n + m)** — adjacency list এবং dist/orig array রাখার জন্য

সারসংক্ষেপ

বিষয়	ব্যাখ্যা
সমস্যা	Weighted Graph এ vertex 1 থেকে vertex n পর্যন্ত সংক্ষিপ্ততম পথ
অ্যালগরিদম	Dijkstra’s Algorithm
গ্রাফ টাইপ	Undirected, Weighted
Negative weight edge?	সমর্থিত নয়
ডেটা স্ট্রাকচার	Priority Queue (Min-Heap)
ফলাফল	Shortest Path অথবা -1