1976. Number of Ways to Arrive at Destination

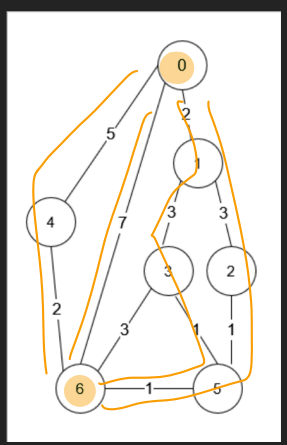
Medium | Topics | Companies | Hint

You are in a city that consists of `n` intersections numbered from `0` to `n - 1` with **bi-directional** roads between some intersections. The inputs are generated such that you can reach any intersection from any other intersection and that there is at most one road between any two intersections.

You are given an integer `n` and a 2D integer array `roads` where `roads[i] = [u_i, v_i, time_i]` means that there is a road between intersections `u_i` and `v_i` that takes `time_i` minutes to travel. You want to know in how many ways you can travel from intersection `0` to intersection `n - 1` in the **shortest amount of time**.

Return *the **number of ways** you can arrive at your destination in the **shortest amount of time***. Since the answer may be large, return it **modulo** `10^9 + 7`.

Example 1:

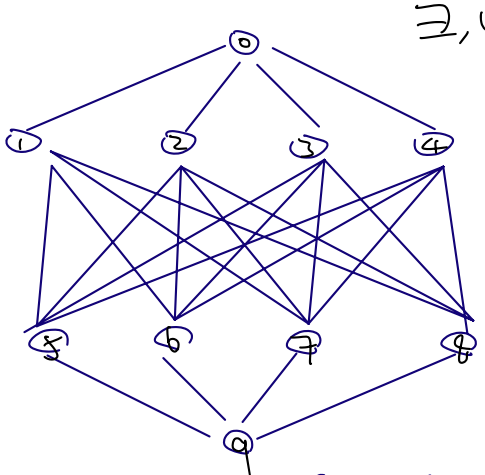



```
Input: n = 7, roads = [[0,6,7],[0,1,2],[1,2,3],[1,3,3],[6,3,3],[3,5,1],[6,5,1],[2,5,1],
[0,4,5],[4,6,2]]
Output: 4
Explanation: The shortest amount of time it takes to go from intersection 0 to intersection
6 is 7 minutes.
The four ways to get there in 7 minutes are:
- 0 → 6
- 0 → 4 → 6
- 0 → 1 → 2 → 5 → 6
- 0 → 1 → 3 → 5 → 6
```

shortest of time:

1. no cycle
2. dijkstra
3. Brute force not working

intuition



$\exists, w=1$

min cost

0 → 0

(1,2,3,4) → 1

(5,6,7,8) → 2

9 → 3

count

(1,2,3,4) → 1

(5,6,7,8) → 1+1+1+1

9 → 4+4+4+4

- if $w_{4,8}=0$,

(0,4,5) is the optimal path, so for (0,4,6) (0,4,7) (0,4,8)

min cost

0 → 0

(1,2,3,4) → 1

(5,6,7,8) → 1 ← (5,6,7,8) → 1

9 → 2

count

(1,2,3,4) → 1

(5,6,7,8) → 1

9 → 1+1+1+1

↳ do the min(,) everywhere if weight is not unified.

```python
class Solution:
    def countPaths(self, n: int, roads: list[list[int]]) -> int:
        MOD = 10**9 + 7
        adj = defaultdict(list)

        for u, v, w in roads:
            adj[u].append((v, w))
            adj[v].append((u, w))

        min_cost = [float("inf")] * n
        path_count = [0] * n

        min_cost[0] = 0
        path_count[0] = 1

        heap = [(0, 0)]  # (cost, node)

        while heap:
            cost, node = heappop(heap)

            if cost > min_cost[node]:
                continue

            for neighbor, weight in adj[node]:
                new_cost = cost + weight
                if new_cost < min_cost[neighbor]:
                    min_cost[neighbor] = new_cost
                    path_count[neighbor] = path_count[node]
                    heappush(heap, (new_cost, neighbor))
                elif new_cost == min_cost[neighbor]:
                    path_count[neighbor] = (path_count[neighbor] + path_count[node]) % MOD

        return path_count[n - 1]
```

⚠️ **Critical Issue in Your Code:**

python

```python
heapq.heappush(mh, (nei, new_c))  # ❌ wrong order!
```

Here, you're pushing `(node, cost)` to the heap — but Python's `heapq` **always sorts by the first element**, so:

- It will incorrectly treat `nei` (a node index) as the priority (instead of cost).
- This breaks **Dijkstra's core rule**, which relies on popping nodes in **order of increasing cost**.

所以在最开始时就应该是

mh = [(0, 0)]
↓ cost ↓ node

✅ **Correct Version:**

python

```python
heapq.heappush(mh, (new_c, nei))  # ✅ cost first
```