**Number of Ways to Arrive at Destination**

**Medium**

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] = [ui, vi, timei] means that there is a road between intersections ui and vi that takes timei 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** 109 + 7.

**Example 1:**

A screenshot of a computer screen

Description automatically generated with low confidence

**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

**Example 2:**

**Input:** n = 2, roads = [[1,0,10]]

**Output:** 1

**Explanation:** There is only one way to go from intersection 0 to intersection 1, and it takes 10 minutes.

**Constraints:**

* 1 <= n <= 200
* n - 1 <= roads.length <= n \* (n - 1) / 2
* roads[i].length == 3
* 0 <= ui, vi <= n - 1
* 1 <= timei <= 109
* ui!= vi
* There is at most one road connecting any two intersections.
* You can reach any intersection from any other intersection.

class Solution {

public:

    long long int mod=1e9+7;

    void dfs(int i, int weight, vector<vector<pair<int, int>>> &graph, vector<bool> &visited, int &ans, int &shortestTime) {

        if (i==visited.size()-1) {

            if (weight==shortestTime) {

                ans=ans+1;

                ans=(ans%mod+mod)%mod;

            }

            return;

        }

        visited[i]=true;

        for (auto x : graph[i]) {

            if (!visited[x.first]) dfs(x.first, weight+(x.second), graph, visited, ans, shortestTime);

        }

        visited[i]=false;

    }

    int countPaths(int n, vector<vector<int>>& roads) {

        vector<vector<pair<int, int>>> graph(n);

        for (int i=0; i<roads.size(); i++) {

            vector<int> x=roads[i];

            int m1=min(x[0], x[1]), m2=max(x[0], x[1]);

            graph[m1].push\_back({m2, x[2]});

        }

        vector<bool> visited(n, false);

        vector<int> distance(n, INT\_MAX);

        distance[0]=0;

        priority\_queue< pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> > pq;

        pq.push({0, 0});

        while (!pq.empty()) {

            pair<int, int> top=pq.top();

            pq.pop();

            for (auto t : graph[top.second]) {

                if (!visited[t.first] and top.first+t.second<distance[t.first]) {

                    distance[t.first]=top.first+t.second;

                    pq.push({distance[t.first], t.first});

                }

            }

            visited[top.second]=true;

        }

        int shortestTime=distance[n-1];

        for(int i=0; i<visited.size(); i++) visited[i]=false;

        int no\_of\_ways=0;

        dfs(0, 0, graph, visited, no\_of\_ways, shortestTime);

        return no\_of\_ways;

    }

};