**Longest Path In Directed Graph**

**Difficulty: MEDIUM**

**Success Rate**

**70%**

**Problem Statement**

**You are given a Weighted Directed Acyclic Graph (DAG) consisting of ‘N’ nodes and ‘E’ directed edges. Nodes are numbered from 0 to ’N’-1. You are also given a source node ‘Src’ in it. Your task is to find the longest distances from ‘Src’ to all the nodes in the given graph.**

**Return an array of ‘N’ integers where ‘ith’ integer gives the maximum distance of the node ‘i’ from the source node ‘Src’.**

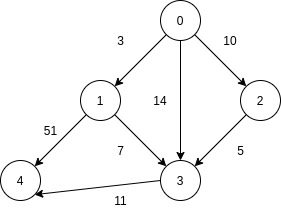
**A Directed Acyclic Graph (DAG) is a directed graph that contains no cycles.**

**Note:**

Print -1 if a node is not reachable from the given source node.

**Example:**

Consider the following DAG consists of 5 nodes and 7 edges, Let the source node ‘Src’ be 0.



Then the maximum distance of node 0 from the source node 0 is 0. (the distance of a node from itself is always 0).

The maximum distance of node 1 from the source node 0 is 3. The path that gives this maximum distance is 0 -> 1.

The maximum distance of node 2 from the source node 0 is 10. The path that gives this maximum distance is 0 -> 2.

The maximum distance of node 3 from the source node 0 is 15. The path that gives this maximum distance is 0 -> 2 -> 3.

The maximum distance of node 4 from the source node 0 is 54. The path that gives this maximum distance is 0 -> 1 -> 4.

Thus we should print 0 3 10 15 54

**Input Format:**

The first line of input contains an integer ‘T’ denoting the number of test cases. then ‘T’ test cases follow.

The first line of each test case consists of three space-separated integers ‘N’, ‘E’, and ‘Src’ representing the number of nodes, number of edges, and the given source node in the given DAG respectively.

The next ‘E’ lines consist of three space-separated integers ‘U’, ‘V’, ‘W’ representing that there is a directed edge from node U to V having weight ‘W’.

**Output Format :**

For each test case, print ‘N’ space-separated integers where ’ith’ integer gives the maximum distance of node ‘i’ from the source node ‘Src’.

The output of each test case will be printed in a new line.

**Note:**

You do not need to print anything, it has already been taken care of. Just implement the given function.

**Constraints:**

1 <= T <= 50

1 <= N <= 10^4

0 <= E <= 10^4

0 <= Src <= N-1

0 <= U, V <= N-1

1 <= W <= 10^5

Where ‘T’ is the total number of test cases, ‘N’, ‘E’, and ‘Src’ representing the number of nodes, number of edges, and the given source node in the given DAG respectively. ‘U’, ‘V’, ‘W’ represents that there is a directed edge from node U to V having weight ‘W’.

Time limit: 1 sec

**Sample Input 1:**

2

1 0 0

5 7 0

0 1 3

0 2 10

0 3 14

1 3 7

1 4 51

2 3 5

3 4 11

**Sample Output 1:**

0

0 3 10 15 54

**Explanation Of Sample Input 1:**

Test case 1:

There is only one node, which is also the source node since the distance from the source node to itself is 0, so we will print only a single integer 0.

Test case 2:

See the problem statement for an explanation.

**Sample Input 2:**

2

5 4 2

0 1 1

0 2 1

1 2 1

3 4 1

5 4 0

0 1 1

0 2 1

1 2 1

3 4 1

**Sample Output 2:**

-1 -1 0 -1 -1

0 1 2 -1 -1

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

    visited[i]=true;

    for (auto x : graph[i]) {

        if (!visited[x.first]) dfs(x.first, graph, visited, ans);

    }

    ans.push\_back(i);

}

vector<int> findMaxDistances(int n, int src, vector<vector<int>> &edges) {

    /\*

        Parameters of this function are -:

        'n': Number of nodes in given directed graph.

        'src': Source node.

        'edges': list of all edges, such that the 'ith edge is a

                 directed edge from 'edges[i][0]' to 'edges[i][1]' and have weight 'edges[i][2]'.

    \*/

    // Write your code here.

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

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

        vector<int> v=edges[i];

        graph[v[0]].push\_back({v[1], v[2]});

    }

    vector<bool> visited(n, false);

    vector<int> topoSort, distance(n, -1);

    for (int i=0; i<n; i++) {

        if (!visited[i]) dfs(i, graph, visited, topoSort);

    }

    for (int i=0; i<=(n-1)/2; i++) swap(topoSort[i], topoSort[n-1-i]);

    distance[src]=0;

    for (int i=0; i<n; i++) {

        if (distance[topoSort[i]]!=-1) {

            for (auto x : graph[topoSort[i]]) {

                distance[x.first]=max(distance[x.first], distance[topoSort[i]]+x.second);

            }

        }

    }

    return distance;

}