MICROSOFT PREPARATION

DAY : 14

Network Delay Time :

Problem Link : https://leetcode.com/problems/network-delay-time/

Test Cases Passed : 53 / 53

Time Used : 10.15

Difficulty Level : MEDIUM

Approach Used :

- Create an adjacency list
- Create a distance vector initialized with all elements initially assigned infinity
- Mark the source element to have a 0 distance
- Create a set having the pair of node and time
- Insert the source with 0 time elapsed
- Traverse until the set becomes empty :
    - Extract the first element from the set
    - Erase the first element from the set
    - Traverse for the adjacent elements :
        - Check if the adjacent node can be reached using better time than already reachable time :
            - Check if already the node exists in the set :
                - Remove node from set
            - Update the time
            - Insert node with updated time into the set
- Create a variable to store the maximum elapsed time
- Traverse for the time vector :
    - If the time is infinity for a node :
        - Return -1
    - If we get a greater time :
        - Update the maximum elapsed time
- Return maximum elapsed time

Solution :

```
int networkDelayTime(vector<vector<int>>& times, int n, int k) {
       // Creating a graph
       vector<vector<pair<int,int>>> adj(n+1);
       for(auto it : times) {
           int from = it[0];
           int to = it[1];
           int time = it[2];
           // Given graph is a directed graph
           adj[from].push_back({to, time});
       }

       // Creating a set to store the source and distance
       set<pair<int, int>> s;
       // Inserting the source element into the set with a distance of 0
       s.insert({k, 0});

       // Creating a distance vector to store the distance of every element from
the source and assigning all elements to infinity
       vector<int> time(n+1, INT_MAX);
       // Marking the source to have a time 0
      time[k] = 0;

       // Traversing until the set becomes empty
       while(!s.empty()) {
           // Extracting the first element from the set
           auto it = *(s.begin());
           int node = it.first;
           int t = it.second;
           // Erasing element from the set
           s.erase(it);

           // Traversing for the adjacent elements
           for(auto i : adj[node]) {
               // Getting time to the nearest node
               int adjnode = i.first;
               int adjweight = i.second;
               // Checking if we can get to the node with a better time
               if(t + adjweight < time[adjnode]) {
                   // Checking if already exists in set
                   if(time[adjnode] != INT_MAX) {
                       s.erase({adjnode, time[adjnode]});
                   }
                   // Updating the time
                   time[adjnode] = t + adjweight;
                   // Inserting into set
```

```
                s.insert({adjnode, time[adjnode]});
            }
        }
    }
    // declaring variable to store max elapsed time
    int maxtime = INT_MIN;
    for(int i = 1; i <= n; i++) {
        // if we can't reach
        if(time[i] == INT_MAX) {
            return -1;
        }
        // if we get greater time
        if(maxtime < time[i]) {
            maxtime = time[i];
        }
    }
    // returning max elapsed time
    return maxtime;
}
```