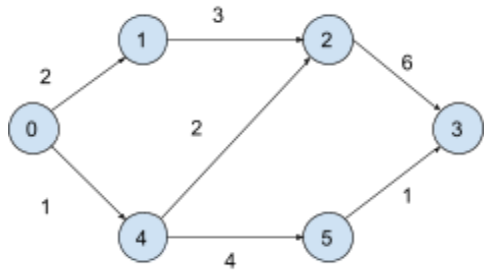
MEDIUM

Shortest path in Directed Acyclic Graph

Intuition

Given : Directed Acyclic Graph [Graph with no cycles]

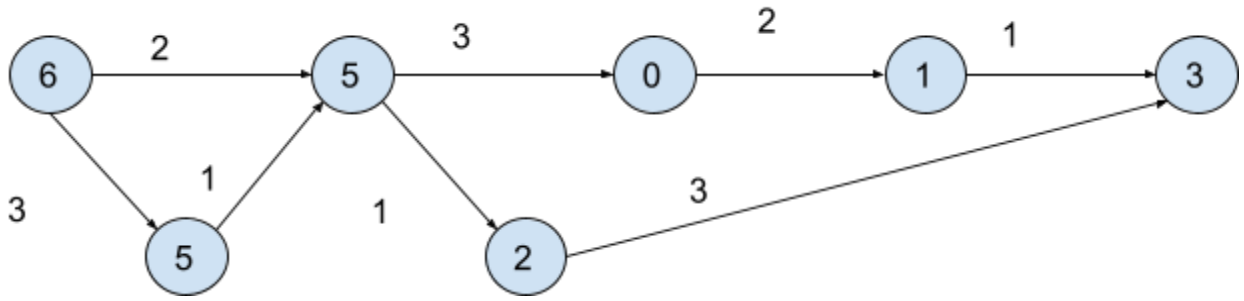
Eg.



Distance : 0,2,3…..

We will be using toposort with slight modification

Eg.



Adjacency List :
0 - [1, 2]
1 - [3, 1]
2 - [3, 3]
3 - [ ]
4 - [0, 3] [2, 1]
5 - [4, 1]
6 - [4, 2] [5, 3]

Instead of storing an integer this time we will store pairs vector<pair<int, int>>

Source will always be 0 as per question

Let's assume we can have any source

Step - 1

Do a toposort on graph

DFS toposort implementation

Stack = [3 1 0 2 4 5 6]

Visited = [0 1 2 3 4 5 6]

// Toposort = [6 5 4 2 0 1 3]

We will do take the nodes out of the stack and relax the edges

Distance = [INF, INF, INF, INF, INF, INF, INF] // INITIAL

DIST = [5 7 3 6 2 3 0]

Approach

dfs()
- Visit the node
- Traverse for adjacent elements :
    - Check if the adjacent node is unvisited
        - Call dfs for the adjacent node
- Add node to stack

shortestPath()
- Create an adjacency list having pairs of {v : weight}
- Create an empty stack
- Declare a visited vector initialized with 0
- Perform the topological sort to add elements in the stack in topological order
- Create a distance vector having n element where all elements are initially assigned 1e9
- Assign the start element as 0 distance
- Traverse until the stack becomes empty :
    - Extract the top of stack
    - Pop the top of stack
    - Traverse for adjacent elements :

        - Check if the distance of node to adjacent element is shorter than the distance in the distance vector as distance [node] + weight < distance [adjacentNode]
            - Update the distance in the distance vector if the distance from given node is shorter
- Those nodes that have distance == 1e9 convert them to -1
- Return the distance vector

Function Code

```
void topoSort(int node, vector <pair<int, int>> adj[],vector<int>& visited, stack
< int > & s) {
      //visit the node
      visited[node] = 1;
      // traverse through the adjacent elements
      for (auto it: adj[node]) {
          // check if the node is visited or not
        int v = it.first;
        if (!visited[v]) {
            // recursive call for the adjacent node
          topoSort(v, adj, visited, s);
        }
      }
      // add the last element to the stack
      s.push(node);
    }
    vector < int > shortestPath(int n, int m, vector < vector < int >> & edges) {
      //We create a graph first in the form of an adjacency list.
      vector < pair < int, int >> adj[n];
      for (int i = 0; i < m; i++) {
        int u = edges[i][0];
        int v = edges[i][1];
        int wt = edges[i][2];
        adj[u].push_back({v, wt});
      }
      // A visited array is created with initially
      // all the nodes marked as unvisited (0).
      vector<int> visited(n,0);
      //Now, we perform topo sort using DFS technique
      //and store the result in the stack st.
      stack < int > s;
      for (int i = 0; i < n; i++) {
        if (!visited[i]) {
```

```
            topoSort(i, adj, visited, s);
          }
        }
      // create a distance vector having n no of elements
      // initially assign 1e9 highest value to the distance for every node
        vector < int > distance(n,1e9);
              // mark the source node having distance 0
        distance[0] = 0;
        // traverse until the stack becomes empty
        while (!s.empty()) {
            // extract the top of stack
          int node = s.top();
          // pop the element from top of stack
          s.pop();
          // traverse for adjacent elements of the node
          for (auto it: adj[node]) {

            int v = it.first;
            int wt = it.second;
          // check if path through the node is having a shorter value than the
distance value of adjacent node
            if (distance[node] + wt < distance[v]) {
                // update value if its shorter
              distance[v] = wt + distance[node];
            }
          }
        }
      // the distance of those element who are not reachable change it to -1 as per
the question
        for (int i = 0; i < n; i++) {
          if (distance[i] == 1e9) distance[i] = -1;
        }
        // returning the distance vector
        return distance;
      }
```

Time Complexity

O(V+E)