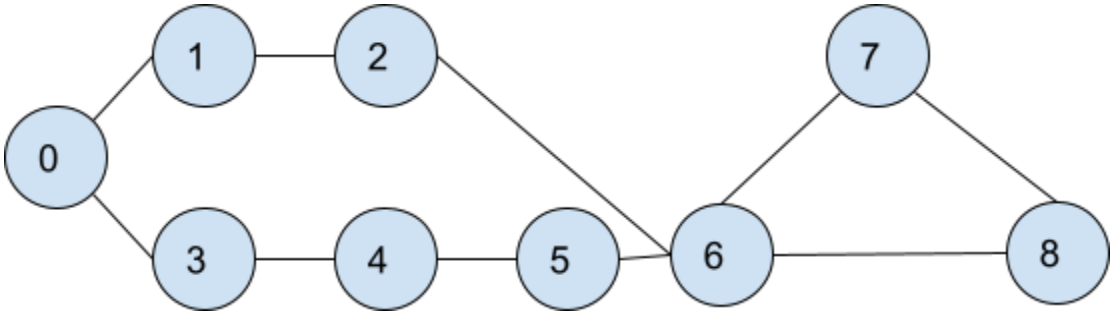
MEDIUM

Shortest path in Undirected Graph having unit distance

Intuition



Adjacency List :

0 - 1, 3
1 - 0, 2, 3
2 - 1, 6
3 - 0, 4
5 - 4, 6
6 - 2, 5, 7, 8
7 - 6, 8
8 - 6, 7

Let source is 0

Possible minimum distance to 2 is 2

This can be solved using plain BFS algorithm

We will be storing pair in the queue data structure

Queue = [0,0] // initially only add source and distance to source as 0
Distance = for 8 nodes fill all of them with infinity and assign the source 0

Eg. q = [0,0] [1,1], [3,1].....

0	1	2	3	4	5	6	7	8
0	1	inf	inf	inf	inf	inf	inf	inf

Node 0. Distance 0 : node1 [distance 1] and node3 [distance3].....

In order to reach 3 and 1 from 0 we have consumed distance 1

We will continue traversal now and update the distance matrix

Q = [3,1],[2,2].....

After the application of algorithm the distance vector will look like

Distance :

0	1	2	3	4	5	6	7	8	
0	1	2	1	2	3	3	4	4	
Now distance	= 0	1	2	1	2	3	3	4	4

Now we can return this distance vector

Approach

- Create an adjacency list
- Create a distance vector of size n and initialize all elements with infinite distance
- Initialize distance of the source as 0
- Create an empty queue
- Insert the source to the queue
- Traverse until the queue becomes empty :
    - Extract the first node from the queue
    - Pop the first node from the queue
    - Traverse for all of the adjacent elements :
        - Check if distance from node + 1 is smaller than the current distance to node :
            - Update distance with distance from node + 1
- Convert all the unreachable elements to have distance -1
- Return distance vector

Function Code

```
vector<int> shortestPath(vector<vector<int>>& edges, int n,int m, int src){
        // creating an adjacency list
        vector<int> adj[n];
        for(int i=0;i<m;i++)
        {
            adj[edges[i][0]].push_back(edges[i][1]);
            adj[edges[i][1]].push_back(edges[i][0]);
```

```
        }

        // creating a distance vector
        vector<int> distance(n,1e9);
        // marking the distance of the source as 0
        distance[src] = 0;
        // creating an empty queue
        queue<int> q;
        // inserting the source into the queue
        q.push(src);
        // traversing until the queue becomes empty
        while(!q.empty())
        {
            // extracting the first element of the queue
            int node = q.front();
            // popping the first element of the queue
            q.pop();
            // traversing for all of the adjacent elements
            for(auto it:adj[node])
            {
                // checking if the distance of the node to the particular
adjacent element is smaller
                if(distance[node]+1<distance[it])
                {
                    // updating the distance
                    distance[it] = distance[node]+1;
                    q.push(it);
                }
            }
        }
        // changing the distances unreachable to -1
        for(int i=0;i<n;i++)
        {
            if(distance[i]==1e9)distance[i]=-1;
        }
        // returning the distance vector
        return distance;
    }
```

Time Complexity

O(V+2*E)