MICROSOFT PREPARATION

DAY : 17

Number of Operations to Make Network Connected :

Problem Link :
https://leetcode.com/problems/number-of-operations-to-make-network-connected/

Test Cases Passed : 36 / 36

Time Used : 09.00

Difficulty Level : **MEDIUM**

Approach Used :

DFS() :
- Marking the current node as visited
- Traversing for the adjacent nodes :
    - Checking if adjacent node is unvisited :
        - Making a dfs call to visit it and its adjacent nodes as dfs(adjacentnode, visited, adj)

MakeConnected() :
- Checking if there are not enough edges to cover all elements :
    - Return -1
- Creating an adjacency list from the given edges
- Create a visited vector having an initialization of 0 for all nodes
- Creating a component variable initialized with 0 to count the number of components
- Traversing for all nodes to count the components :
    - Checking if the node is unvisited ie. a new components :
        - Make a dfs call to mark all of the associated nodes as dfs(componentnode, visited, adj)
- Returning the components-1 // it will be the required new connections to be made

Solution :

```
void dfs(int node,vector<int> &visited,vector<vector<int>>&adj)
   {
       // marking the node as visited
       visited[node] = 1;
       // traversing for the adjacent elements to mark them
       for(auto it:adj[node])
       {
           // checking if not visited
           if(!visited[it])
           {
               // calling dfs to mark them
               dfs(it,visited,adj);
           }
       }
   }
   int makeConnected(int n, vector<vector<int>>& connections) {

   // We have found an interesting observation that if the size of the
edges vector is n-1 then we can perform the recabling

   if(connections.size()<n-1)
   {
       return -1;
   }

   // finding the size of the connected components and returning the no of
unconnected components
   // This is an example of question like DSU find

   // creating the adjacency list
   vector<vector<int>> adj(n);
   for(auto it: connections)
   {
       int from = it[0];
       int to = it[1];
       adj[from].push_back(to);
       adj[to].push_back(from);
   }
   // creating a component variable to count the number of components
   int components = 0;
   // creating a visited vector having the identities of all nodes as
unvisited
```

```
   vector<int> visited(n,0);

   // traversing the whole component vector and finding the number of
connected components
   for(int i=0;i<n;i++)
   {
       // checking if its a new component
       if(!visited[i])
       {
           // incrementing the number of components
           components+=1;
           // performing the DFS solution
           dfs(i,visited,adj);
       }
   }
   // returning the connections required to make the network connected
   return components-1;

   }
```