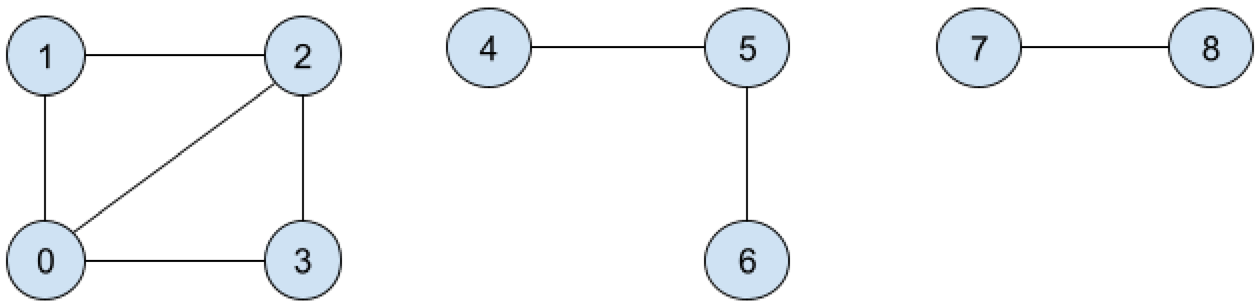
MEDIUM

No. of operations to make network connected

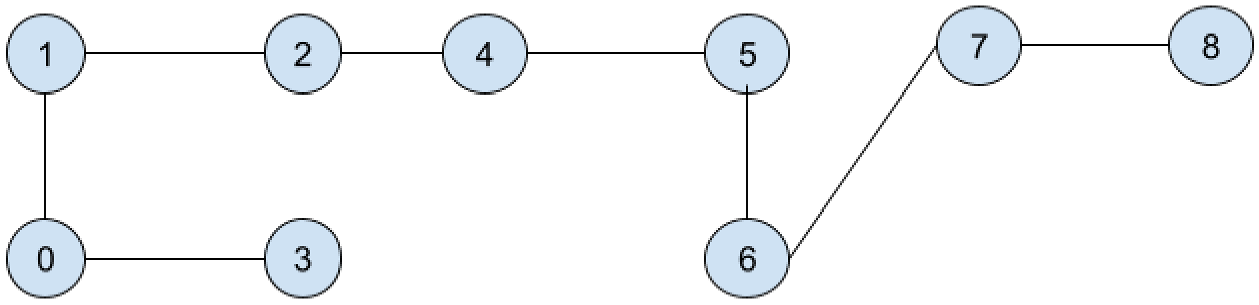
Intuition

Given : n vertices and m edges, and we can remove edge between any one component and add it anywhere to make the graph connected

Eg.



We can do the following operation :
Remove edge between 0-2 and add 2-4
Remove edge between 2-3 and add 6-7



Now all components are connected

We cannot draw imaginary edges, we need to use the already present edges.

What are components and how can we end up connecting all components

If there are 5 components, then we would require 4 edges to connect 5 components into a single component graph

Ie. N components we require N-1 edges to make it into a single component

If we can figure the number of connected components the number of connected components will be n-1

Imaging Components as above :

- Any node connected to any other node of different component will connect them
- We require n-1 edges to connect all components as minimum

We will count the redundant edges, and count the number of connected components.

Approach

- Create a disjoint set for n nodes
- Create a variable to count the redundant edges
- Traverse for all edges :
    - Check if its a redundant edge :
        - Increment redundantedges counter
    - Insert edge into the disjoint set
- Create a variable parent to count the number of parents
- Traverse for all nodes :
    - Check if node is already a parent :
        - Increment parents counter
- Check if parents counter - 1 is greater than redundant edges ,ie. We cannot connect it :
    - Return -1 // because we cannot connect
- Return parents counter - 1 // this is the minimum value of edges that will be required to connect all components

Function Code

```
class DisjointSet
{
   public:
    vector<int> parent;
    vector<int> rank;

    DisjointSet(int n)
    {
        rank.resize(n,0);
        parent.resize(n);
        for(int i=0;i<n;i++)
        {
            parent[i] = i;
        }
    }
    int findParent(int node)
    {
        if(parent[node]==node)
        {
            return node;
        }
        return parent[node] = findParent(parent[node]);
    }
    void unionByRank(int u,int v)
    {
        int pu = findParent(u);
        int pv = findParent(v);
        if(pu==pv)return;
        if(rank[pu]<rank[pv])
        {
            parent[pu] = pv;
            rank[pv]+=1;
        }
        else if(rank[pu]==rank[pv])
        {
            parent[pv] = pu;
        }
        else
        {
            parent[pv] = pu;
            rank[pu]+=1;
        }
    }
};
class Solution {
  public:
    int Solve(int n, vector<vector<int>>& edge) {
        // creating a disjoint set
        DisjointSet ds(n);
        // making a variable to count the redundant edge
        int redundant = 0;
        // traverse for all edges and check if they are connected already
        for(auto it:edge)
        {
            // checking if its a redundant edge
            if(ds.findParent(it[0])==ds.findParent(it[1]))
            {
                redundant+=1;
            }
            // constructing disjoint set
            ds.unionByRank(it[0],it[1]);
        }
        // creating a variable to count the number of parents
        int parents = 0;
        // counting the component bosses
        for(int i=0;i<n;i++)
        {
            if(ds.parent[i]==i)
            {
                parents+=1;
            }
        }
        // checking if parents are more than redundant edges, ie. can't
connect
        if(parents-1>redundant)
        {
            return -1;
        }
        return parents-1;

    }
};
```

Time Complexity

O(N)