https://course.acciojob.com/idle?question=8ece7d1c-895e-4584-b084-64f1032d0611

- MEDIUM
- Max Score: 40 Points

Number of Operations to Make Network Connected

There are `n` computers numbered from `0` to `n-1` connected by ethernet cables connections forming a network where `connections[i] = [ai, bi]` represents a connection between computers `ai` and `bi`. Any computer can reach any other computer directly or indirectly through the network.

You are given an initial computer network connections. You can extract certain cables between two directly connected computers, and place them between any pair of disconnected computers to make them directly connected.

Return the minimum number of times you need to do this in order to make all the computers connected. If it is not possible, return -1.

Note: You just need to complete makeconnected() function and return the minimum number of operations required.

Input Format

The first line contains two space-separated integers `n` and `m` where `n` denotes the number of nodes in the graph and `m` denotes the number of edges in the graph. Next `m` lines contains two integers `u` and `v` representing undirected edge between u and v.

Output Format

Print minimum number of operations needed to make network connected.

Example 1

Input

```
4 3
0 1
0 2
1 2
```

Output

```
1
```

Explanation

```
Remove cable between computer 1 and 2 and place between computers 1 and 3.
```

Example 2

Input

```
6 4
0 1
0 2
0 3
1 2
```

Output

```
-1
```

Explanation

`There are not enough cables.`

Constraints

1 <= n <= 100000

1 <= m <= min(n * (n - 1) / 2, 10^5)

0<= a_i, b_i < n

No two computers are connected by more than one cable.

There are no repeated connections.

Topic Tags

- Graphs

My code

```
// n java
import java.util.*;
class Solution{
    static int[] parent, rank;

        public static int find(int x) {
                if(parent[x] == x)
                        return x;

                int res = find(parent[x]);
```

```
        parent[x] = res;

        return res;
    }

    public static void union(int x, int y) {
        int px = find(x);
        int py = find(y);

        if(rank[px] < rank[py]) {
            parent[px] = py;
        } else if(rank[px] > rank[py]) {
            parent[py] = px;
        } else {
            parent[py] = px;
            rank[px]++;
        }
    }

public static int makeConnected(int n , int[][] connections){
    //write your code here

        parent = new int[n];
        rank = new int[n];

        for(int i = 0; i < n; i++) {
            parent[i] = i;
            rank[i] = 1;
        }
```

```
            int comps = n;

            if(connections.length < n-1) return -1;

            for(int i = 0; i < connections.length; i++) {
            int x = connections[i][0], y = connections[i][1];

            int px = find(x);
            int py = find(y);

            if(px != py) {
                union(px, py);
                        comps--;
            }
        }

            return comps-1;
    }
}

public class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        int[][] edges = new int[m][2] ;
        for(int i = 0 ; i < m ; ++i){
            edges[i][0] = sc.nextInt();
            edges[i][1] = sc.nextInt();
        }
```

```
        System.out.print(Solution.makeConnected(n,edges));
    }
}
```