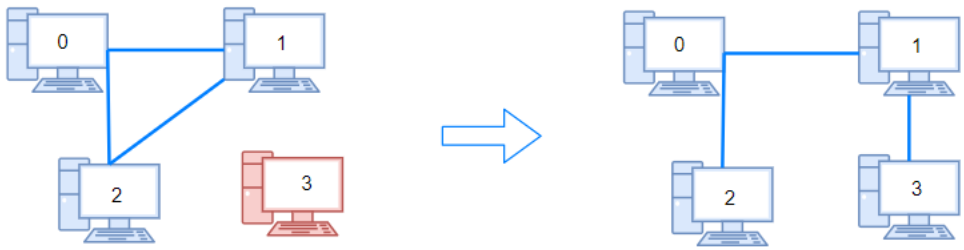
1319. Number of Operations to Make Network Connected

There are n computers numbered from 0 to n - 1 connected by ethernet cables connections forming a network where connections[i] = [a_i, b_i] represents a connection between computers a_i and b_i. 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.

Example 1:

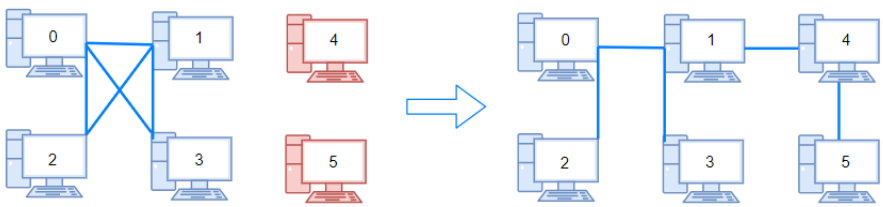


Input: n = 4, connections = [[0,1],[0,2],[1,2]]

Output: 1

Explanation: Remove cable between computer 1 and 2 and place between computers 1 and 3.

Example 2:



Input: n = 6, connections = [[0,1],[0,2],[0,3],[1,2],[1,3]]

Output: 2

Example 3:

Input: n = 6, connections = [[0,1],[0,2],[0,3],[1,2]]

Output: -1

Explanation: There are not enough cables.

Constraints:

- $1 <= n <= 10^5$
- $1 <= connections.length <= min(n * (n - 1) / 2, 10^5)$
- connections[i].length == 2
- $0 <= a_i, b_i < n$
- $a_i != b_i$
- There are no repeated connections.
- No two computers are connected by more than one cable.

```
class Solution {
public:
    void dfs_for_components(vector<vector<int>>& adj, int u,
vector<bool>& vis){
        vis[u] = true;

        for(auto v : adj[u]){
            if(vis[v] == false){
                dfs_for_components(adj, v, vis);
            }
        }
    }

    int makeConnected(int n, vector<vector<int>>& connections)
{
        if(connections.size() < n - 1){
            return -1;
        }

        vector<vector<int>> adj(n);

        for(int i = 0; i < connections.size(); i++){
            int a = connections[i][0];
```

```
            int b = connections[i][1];
            adj[a].push_back(b);
            adj[b].push_back(a);
        }

        vector<bool> vis(n, false);
        int components = 0;

        for(int u = 0; u < n; u++){
            if(vis[u] == false){
                components++;
                dfs_for_components(adj, u, vis);
            }
        }
        return components - 1;
    }
};
```