Text

Description automatically generated with medium confidence

**Number of Operations to Make Network Connected**

**Medium**

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

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's not possible, return -1.

**Example 1:**

**Icon

Description automatically generated with low confidence**

**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:**

**Diagram

Description automatically generated with low confidence**

**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.

**Example 4:**

**Input:** n = 5, connections = [[0,1],[0,2],[3,4],[2,3]]

**Output:** 0

**Constraints:**

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

class Solution {

public:

    void dfs(int i, int &count, vector<int> graph[], vector<bool> &visited) {

        visited[i]=true;

        count++;

        for (int x : graph[i]) {

            if (!visited[x]) dfs(x, count, graph, visited);

        }

    }

    int makeConnected(int n, vector<vector<int>>& connections) {

        vector<int> graph[n];

        int edges=connections.size(), ct=0;

        for (int i=0; i<connections.size(); i++) {

            graph[connections[i][0]].push\_back(connections[i][1]);

            graph[connections[i][1]].push\_back(connections[i][0]);

        }

        vector<bool> visited(n, false);

        int minimum=0;

        for (int i=0; i<n; i++) {

            if (!visited[i]) {

                ct++;

                int count=0;

                dfs(i, count, graph, visited);

                minimum+=(count-1);

            }

        }

        int extraEdges=edges-minimum;

        if (extraEdges>=ct-1) return ct-1;

        else return -1;

    }

};