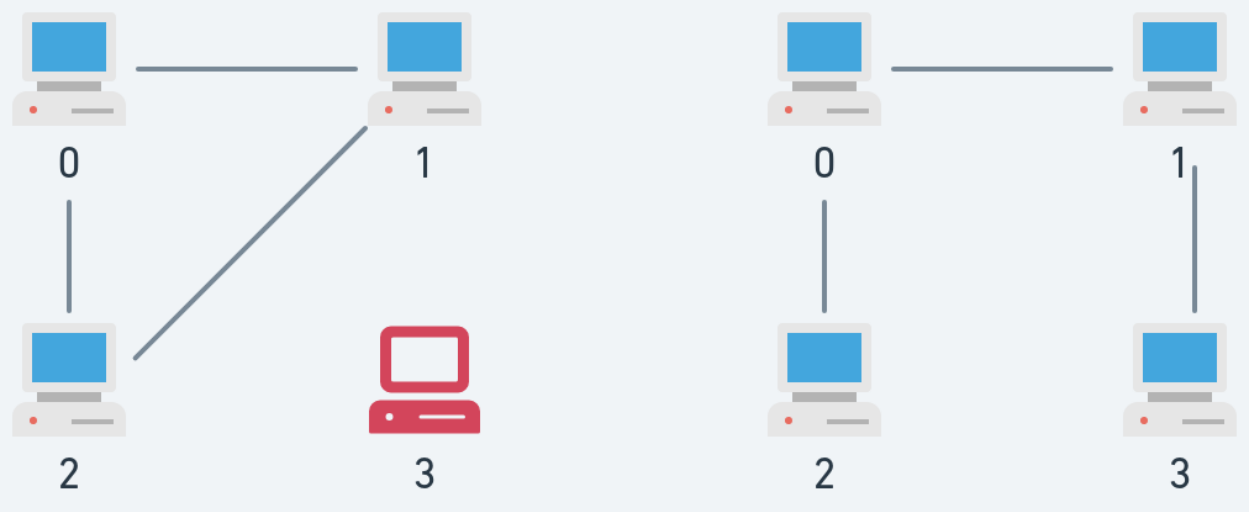
Wavelabs' Lab setup consists of n workstations numbered from 0 to n - 1 connected by ethernet cable connections forming a network where connections[i] = [a_i, b_i] represents a connection between workstations a_i and b_i. Any workstation can reach any other workstation directly or indirectly through the network.

As a Network Engineer, you are given initial computer network connections. You can extract certain cables between two directly connected workstations, and place them between any pair of disconnected workstations to make them directly connected.

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

Example:



Input: n = 4, connections = [[0,1],[0,2],[1,2]]
Output: 1
Explanation: Remove the cable between workstations 1 and 2 and place it between workstations 1 and 3.

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 workstations are connected by more than one cable.

Solve the questions in C/C++/ Go lang only No other Language

Solution:

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

class Solution
{
public:
   int find(vector<int>& parent, int i)
   {
      if (parent[i] == -1)
      {
         return i;
      }
      return find(parent, parent[i]);
   }

   void unionSet(vector<int>& parent, int i, int j)
   {
      parent[i] = j;
   }

   int minimumCostToConnectAllWorkstations(int n, vector<vector<int>>& connections)
   {
      vector<int> parent(n, -1);
      sort(connections.begin(), connections.end(), [](const vector<int>&a,  const vector<int>&
b)
      {
         return a[2] < b[2];
      });

      int edges = 0;
      int i = 0;
      while (i < connections.size() && edges < n - 1)
      {
         int x = find(parent, connections[i][0]);
         int y = find(parent, connections[i][1]);
         if (x != y)
         {
            unionSet(parent, x, y);
```

```
            edges++;
        }
        i++;
    }

    if (edges < n - 1)
    {
        return -1;
    }

    int cost = 0;
    for (const auto& c : connections)
    {
        cost += c[2];
    }
    return cost - (n - 1);

  }
};

int main()
{
  int n = 4;
  vector<vector<int>> connections = {{0,1},{0,2},{1,2}};
  Solution solution;
  int result = solution.minimumCostToConnectAllWorkstations(n, connections);
  cout << result << endl;
  return 0;
}
```

Output: 1