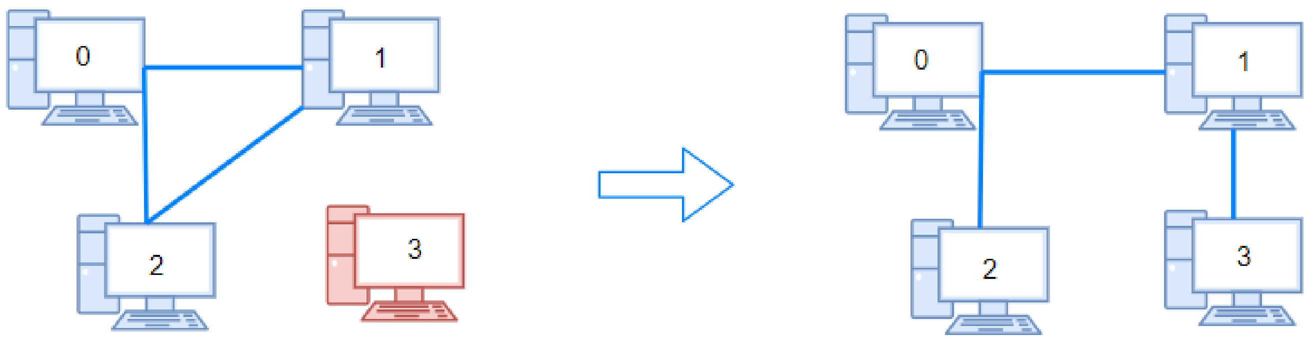
1319. Number of Operations to Make Network Connected

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

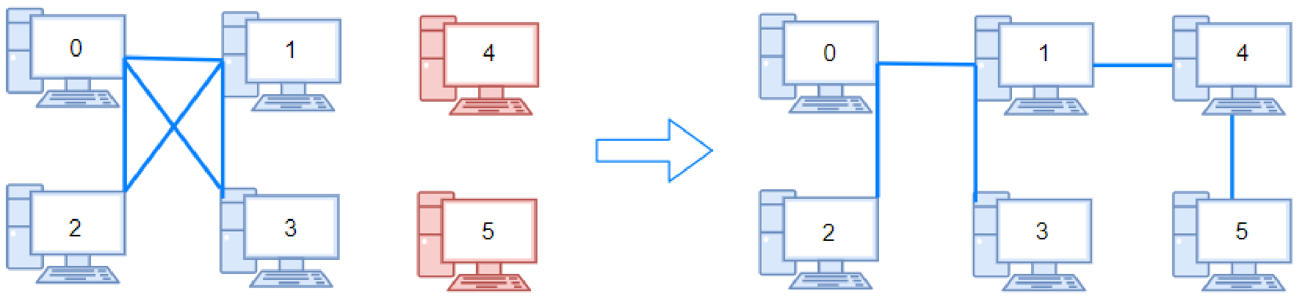
Given an initial computer network `connections`. You can extract certain cables between two directly connected computers, and place them between any pair of disconnected computers to make them directly connected. Return the *minimum number of times* you need to do this in order to make all the computers connected. If it's not possible, return -1.

Example 1:



```
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.
```

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:
    def makeConnected(self, n: int, connections: List[List[int]]) -> int:
        comps = []
        visited = [False]*n
        if n-1>len(connections):
            return -1
        graph = defaultdict(list)
        for ele in connections:
            fr,to = ele
            graph[fr].append(to)
            graph[to].append(fr)
        count = 0
        for i in range(n):
            if visited[i]==False:
                self.dfs(visited,graph,i)
                count = count+1
        return count-1



    def dfs(self,visited,graph,src):

        visited[src] = True
        for neigh in graph[src]:
            if visited[neigh] == False:
                self.dfs(visited,graph,neigh)
```