6432. Count the Number of Complete Components

My Submissions (/contest/weekly-contest-345/problems/count-the-number-of-complete-components/submissions/)

Back to Contest (/contest/weekly-contest-345/)

You are given an integer `n`. There is an **undirected** graph with `n` vertices, numbered from `0` to `n - 1`. You are given a 2D integer array `edges` where `edges[i] = [a_i, b_i]` denotes that there exists an **undirected** edge connecting vertices a_i and b_i.

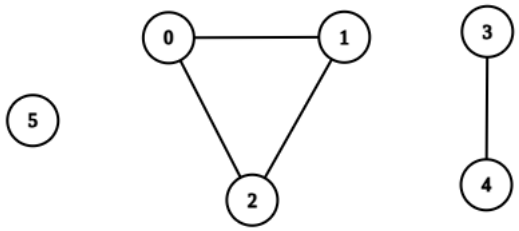
Return *the number of **complete connected components** of the graph*.

A **connected component** is a subgraph of a graph in which there exists a path between any two vertices, and no vertex of the subgraph shares an edge with a vertex outside of the subgraph.

A connected component is said to be **complete** if there exists an edge between every pair of its vertices.

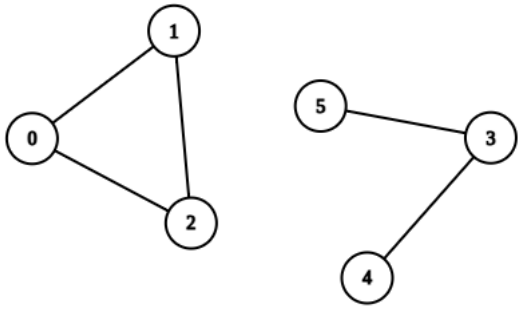
User Accepted:	0
User Tried:	0
Total Accepted:	0
Total Submissions:	0
Difficulty:	Medium

Example 1:




```
Input: n = 6, edges = [[0,1],[0,2],[1,2],[3,4]]
Output: 3
Explanation: From the picture above, one can see that all of the components of this graph are complete.
```

Example 2:




```
Input: n = 6, edges = [[0,1],[0,2],[1,2],[3,4],[3,5]]
Output: 1
Explanation: The component containing vertices 0, 1, and 2 is complete since there is an edge between every pair of two vertice
```

Constraints:

- `1 <= n <= 50`
- `0 <= edges.length <= n * (n - 1) / 2`
- `edges[i].length == 2`
- `0 <= a_i, b_i <= n - 1`
- `a_i != b_i`
- There are no repeated edges.

JavaScript

```javascript
function DJSet(n) {
    // parent[i] < 0, -parent[i] is the group size which root is i. example: (i -> parent[i] -> parent[parent[i]] -> parent[parent[parent[i]]] ...)
    // parent[i] >= 0, i is not the root and parent[i] is i's parent. example: (... parent[parent[parent[i]]] -> parent[parent[i]] -> parent[i] -> i)
    let parent = Array(n).fill(-1);
    return { find, union, count, equiv, par }
    function find(x) {
        return parent[x] < 0 ? x : parent[x] = find(parent[x]);
    }
    function union(x, y) {
        x = find(x);
        y = find(y);
        if (x == y) return false;
        if (parent[x] < parent[y]) [x, y] = [y, x];
        parent[x] += parent[y];
        parent[y] = x;
        return true;
    }
    function count() { // total groups
        return parent.filter(v => v < 0).length;
    }
    function equiv(x, y) { // isConnected
        return find(x) == find(y);
    }
    function par() {
        return parent;
    }
}

const initializeGraphSet = (n) => { let g = []; for (let i = 0; i < n; i++) { g.push(new Set()); } return g; };

const countCompleteComponents = (n, edges) => {
    let ds = new DJSet(n), g = initializeGraphSet(n), groups = new Set(), d = new Set();
    for (const [x, y] of edges) {
        d.add(x + " " + y);
        d.add(y + " " + x);
        ds.union(x, y);
    }
    for (let i = 0; i < n; i++) {
        for (let j = 0; j < n; j++) {
            if (ds.equiv(i, j)) {
                g[i].add(j);
                g[j].add(i);
            }
        }
    }
    // pr(ds.par(), g)
    for (const se of g) {
        let a = [...se].sort((x, y) => x - y);
        groups.add(JSON.stringify(a))
    }
    // pr(groups, d)
    for (const gr of groups) {
        let a = JSON.parse(gr);
        //   pr(a)
        for (let i = 0; i < a.length; i++) {
            for (let j = i + 1; j < a.length; j++) {
                let ke = a[i] + " " + a[j];
                // pr(ke)
                if (!d.has(ke)) groups.delete(gr);
            }
        }
    }
    return groups.size;
};
```

Custom Testcase | Use Example Testcases

Run | Submit

Submission Result: Accepted (/submissions/detail/949990118/) More Details > (/submissions/detail/949990118/)

Share your acceptance!