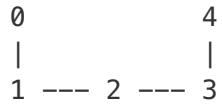
Number of Connected Components in an Undirected Graph

Given `n` nodes labeled from `0` to `n - 1` and a list of undirected edges (each edge is a pair of nodes), write a function to find the number of connected components in an undirected graph.

Example 1:

Given `n = 5` and `edges = [[0, 1], [1, 2], [3, 4]]`, return `2`.

Example 2:



Given `n = 5` and `edges = [[0, 1], [1, 2], [2, 3], [3, 4]]`, return `1`.

Note:
You can assume that no duplicate edges will appear in `edges`. Since all edges are undirected, `[0, 1]` is the same as `[1, 0]` and thus will not appear together in `edges`.

Solution 1

```
    private int[] father;
public int countComponents(int n, int[][] edges) {

    Set<Integer> set = new HashSet<Integer>();
    father = new int[n];
    for (int i = 0; i < n; i++) {
        father[i] = i;
    }
    for (int i = 0; i < edges.length; i++) {
         union(edges[i][0], edges[i][1]);
    }

    for (int i = 0; i < n; i++){
        set.add(find(i));
    }
    return set.size();
}

int find(int node) {
    if (father[node] == node) {
        return node;
    }
    father[node] = find(father[node]);
    return father[node];
}

void union(int node1, int node2) {
    father[find(node1)] = find(node2);
}
```

written by jmnjmnjmn original link here

Solution 2

DFS:

```
def countComponents(n, edges):
        def dfs(n, g, visited):
            if visited[n]:
                return
            visited[n] = 1
            for x in g[n]:
                dfs(x, g, visited)

        visited = [0] * n
        g = {x:[] for x in xrange(n)}
        for x, y in edges:
            g[x].append(y)
            g[y].append(x)

        ret = 0
        for i in xrange(n):
            if not visited[i]:
                dfs(i, g, visited)
                ret += 1

        return ret
```

BFS:

```
def countComponents(n, edges):
        g = {x:[] for x in xrange(n)}
        for x, y in edges:
            g[x].append(y)
            g[y].append(x)

        ret = 0
        for i in xrange(n):
            queue = [i]
            ret += 1 if i in g else 0
            for j in queue:
                if j in g:
                    queue += g[j]
                    del g[j]

        return ret
```

Union Find:

```
def countComponents(n, edges):
        def find(x):
            if parent[x] != x:
                parent[x] = find(parent[x])
            return parent[x]

        def union(xy):
            x, y = map(find, xy)
            if rank[x] < rank[y]:
                parent[x] = y
            else:
                parent[y] = x
                if rank[x] == rank[y]:
                    rank[x] += 1

        parent, rank = range(n), [0] * n
        map(union, edges)
        return len({find(x) for x in parent})
```

written by chris.zhang.336 original link here

Solution 3

Use the similar method as Number of Islands II. Use a findRoot function. See more details in https://leetcode.com/discuss/69572/easiest-java-solution-with-explanations

```
public int countComponents(int n, int[][] edges) {
    int res = n;

    int[] root = new int[n];
    for (int i = 0; i < n; i++) {
        root[i] = i;
    }
    for (int[] pair : edges) {
        int rootX = findRoot(root, pair[0]);
        int rootY = findRoot(root, pair[1]);
        if (rootX != rootY) {
            root[rootY] = rootX;
            res--;
        }
    }
    return res;
}
public int findRoot(int[] root, int i) {
    while (root[i] != i) i = root[i];
    return i;
}
```

written by beckychiu1988 original link here