<https://massivealgorithms.blogspot.com/2015/12/lintcode-431-find-connected-component.html>

<https://www.cnblogs.com/grandyang/p/5166356.html>

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:

0 3

| |

1 --- 2 4

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

Example 2:

0 4

| |

1 --- 2 --- 3

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.

**Attempt 1: 2022-12-16**

**Solution 1:  DFS (10 min)**

class Solution {

public int countComponents(int n, int[][] edges) {

Map<Integer, List<Integer>> adj = new HashMap<Integer, List<Integer>>();

for (int i = 0; i < n; i++) {

adj.put(i, new ArrayList<Integer>());

}

for (int[] edge : edges) {

adj.get(edge[0]).add(edge[1]);

adj.get(edge[1]).add(edge[0]);

}

boolean[] visited = new boolean[n];

int count = 0;

for (int i = 0; i < n; i++) {

if (!visited[i]) {

helper(i, adj, visited);

count++;

}

}

return count;

}

private void helper(int index, Map<Integer, List<Integer>> adj, boolean[] visited) {

for (int j : adj.get(index)) {

if (!visited[j]) {

visited[j] = true;

helper(j, adj, visited);

}

}

}

public static void main(String[] args) {

Solution s = new Solution();

int n = 5;

int[][] edges = {{0, 1}, {1, 2}, {3, 4}};

// int n = 5;

// int[][] edges = {{0,1},{1,2},{2,3},{3,4}};

int result = s.countComponents(n, edges);

System.out.println(result);

}

}

Time Complexity : O(N)

Space Complexity : O(N)

**Refer to**

<https://massivealgorithms.blogspot.com/2015/12/lintcode-431-find-connected-component.html>

<https://www.cnblogs.com/grandyang/p/5166356.html>

这道题让我们求无向图中连通区域的个数，LeetCode中关于图Graph的题屈指可数，解法都有类似的特点，都是要先构建邻接链表Adjacency List来做。这道题的一种解法是利用DFS来做，思路是给每个节点都有个flag标记其是否被访问过，对于一个未访问过的节点，我们将结果自增1，因为这肯定是一个新的连通区域，然后我们通过邻接链表来遍历与其相邻的节点，并将他们都标记成已访问过，遍历完所有的连通节点后我们继续寻找下一个未访问过的节点，以此类推直至所有的节点都被访问过了，那么此时我们也就求出来了连通区域的个数。

class Solution {

public:

int countComponents(int n, vector<pair<int, int> >& edges) {

int res = 0;

vector<vector<int> > g(n);

vector<bool> v(n, false);

for (auto a : edges) {

g[a.first].push\_back(a.second);

g[a.second].push\_back(a.first);

}

for (int i = 0; i < n; ++i) {

if (!v[i]) {

++res;

dfs(g, v, i);

}

}

return res;

}

void dfs(vector<vector<int> > &g, vector<bool> &v, int i) {

if (v[i]) return;

v[i] = true;

for (int j = 0; j < g[i].size(); ++j) {

dfs(g, v, g[i][j]);

}

}

};

**Solution 2:  Union Find (10 min)**

**Style 1: Simple Union Find**

class Solution {

public int countComponents(int n, int[][] edges) {

int[] parent = new int[n];

for (int i = 0; i < n; i++) {

parent[i] = i;

}

for (int[] edge : edges) {

int rootA = find(edge[0], parent);

int rootB = find(edge[1], parent);

if (rootA != rootB) {

parent[rootA] = rootB;

n--;

}

}

return n;

}

private int find(int x, int[] parent) {

if (parent[x] == x) {

return x;

}

// Note: Don't write as "return parent[x] = find(x, parent);"

return parent[x] = find(parent[x], parent);

}

// Alternative find style

private int find2(int x, int[] parent) {

while(parent[x] != x) {

parent[x] = parent[parent[x]];

x = parent[x];

}

return x;

}

}

**Style 2: Union Find with weighted union and path compression**

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

public class Solution {

public int countComponents(int n, int[][] edges) {

int[] parent = new int[n];

int[] rank = new int[n];

for (int i = 0; i < n; i++) {

parent[i] = i;

rank[i] = 1;

}

for (int[] edge : edges) {

int rootA = find(edge[0], parent);

int rootB = find(edge[1], parent);

// if (rootA != rootB) {

// parent[rootA] = rootB;

// n--;

// }

// Weighted union

if(rank[rootA] > rank[rootB]) {

parent[rootB] = rootA;

rank[rootA] += rank[rootB];

} else {

parent[rootA] = rootB;

rank[rootB] += rank[rootA];

}

n--;

}

return n;

}

private int find(int x, int[] parent) {

if (parent[x] == x) {

return x;

}

// Note: Don't write as "return parent[x] = find(x, parent);"

return parent[x] = find(parent[x], parent);

}

// Alternative find style

private int find2(int x, int[] parent) {

while(parent[x] != x) {

parent[x] = parent[parent[x]];

x = parent[x];

}

return x;

}

public static void main(String[] args) {

Solution s = new Solution();

int n = 5;

int[][] edges = {{0, 1}, {1, 2}, {3, 4}};

// int n = 5;

// int[][] edges = {{0,1},{1,2},{2,3},{3,4}};

int result = s.countComponents(n, edges);

System.out.println(result);

}

}

**Refer to**

<https://massivealgorithms.blogspot.com/2015/12/lintcode-431-find-connected-component.html>

<https://www.cnblogs.com/grandyang/p/5166356.html>

这道题还有一种比较巧妙的方法，不用建立邻接链表，也不用DFS，思路是建立一个root数组，下标和节点值相同，此时root[i]表示节点i属于group i，我们初始化了n个部分 (res = n)，假设开始的时候每个节点都属于一个单独的区间，然后我们开始遍历所有的edge，对于一条边的两个点，他们起始时在root中的值不相同，这时候我们我们将结果减1，表示少了一个区间，然后更新其中一个节点的root值，使两个节点的root值相同，那么这样我们就能把连通区间的所有节点的root值都标记成相同的值，不同连通区间的root值不相同，这样也能找出连通区间的个数。

class Solution {

public:

int countComponents(int n, vector<pair<int, int> >& edges) {

int res = n;

vector<int> root(n);

for (int i = 0; i < n; ++i) root[i] = i;

for (auto a : edges) {

int x = find(root, a.first), y = find(root, a.second);

if (x != y) {

--res;

root[y] = x;

}

}

return res;

}

int find(vector<int> &root, int i) {

while (root[i] != i) i = root[i];

return i;

}

};