**PSG COLLEGE OF TECHNOLOGY**



Department of Information Technology

ADVANCED DATA STRUCTURES

**ASSIGNMENT PRESENTATION**

Topic : **Hackerrank Problem – Components in a Graph**

Group : 7

Submitted To:

Dr. Anitha Kumari K,

Department Of Information Technology.

PSG College Of Technology

Submitted By:

Sharath S V

17I340

3rd Year B.Tech IT-G2

PSG College Of Technology

**Problem Statement:**

**Components in a graph**

There are 2N values to represent nodes in a graph. They are divided into two sets G and B. Each set has exactly N values. Set G is represented by {G1, G2, ...., GN}. G can contain any value between 1 to N(inclusive). Set B is represented by {B1, B2, ...., BN}. B can contain any value between N+1 to 2N (inclusive). Same value can be chosen any number of times.

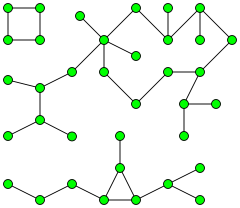
Here (G1, B1), (G2, B2), ..., (GN, BN) represents the edges of the graph. Your task is to print the number of vertices in the smallest and the largest connected components of the graph. Also considering the fact that single nodes should not be considered in the answer.

Link : <https://www.hackerrank.com/challenges/components-in-graph/problem>

**Solution-Description:**

**Pre-Requisite:**

Component in Graph :

In graph theory, a component, sometimes called a connected component, of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the super graph. For example, the graph shown in the illustration has three components. A vertex with no incident edges is itself a component. A graph that is itself connected has exactly one component, consisting of the whole graph

Disjoint-Sets:

A [disjoint-set data structure](http://en.wikipedia.org/wiki/Disjoint-set_data_structure) is a data structure that keeps track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets. A union-find algorithm is an algorithm that performs two useful operations on such a data structure:

**Find:** Determine which subset an element is in. This can be used for determining if two elements are in the same subset.

**Union:** Join two subsets into a single subset.

**Understanding The Problem:**

The graph has two sets of values each ranging from 1 to N, so there are 2N elements totally in the graph. So we will be having 2N elements as the capacity of our disjoint set data structure, where each node is represented by the array element’s index and its index is the node itself. While adding an edge we merge two nodes if they belong to separate groups and ignore the link if they belong to the same group, this is achieved by the union operation.To get the final size of the component we create another analogous array for maintaining size initialized with 1 and after merging one will have the sum of sizes and other will be zero denoting it belongs to another group.

**Code:**

import java.util.*\**;

class Solution{

public static *void* main(*String* []*args*)

    {

*Scanner* sf=**new** Scanner(System.in);

*int* a,b,x,y;

*int* n=sf.nextInt(); //input n links from user

*int* arr[]=**new** *int*[2\*n+1]; //total elements in disjoint set (first Array)

*int* brr[]=**new** *int*[2\*n+1]; //array to maintain size (second Array)

        for(*int* i=1;i<2\*n+1;i++)

        {

            arr[i]=i;

/\*initialize with itself (it doesnt belong to any other group,index is the element itself)\*/

            brr[i]=1;

//size of single component is 1

        }

        for(*int* i=0;i<n;i++)

        {

            a=sf.nextInt();

            b=sf.nextInt();

//a,b are link from node a to node b

            x=find(a,arr);

//find which group does the given element belong to

            y=find(b,arr);

//then group both of them to same group using union

            union(x,y,arr,brr);

        }

        System.out.print(findmax(brr,n)+" "+findmin(brr,n));

//findmax returns maximum sized component

//findmin returns minimum sized component

    }

/\* In the first array when the element matches the value at its index then the group to which it belongs is returned denoted by the array value.Otherwise the parent value of the node is given in the array’s value then we redirect to the parent’s element and check the matching with its index happens,then return the value \*/

    static *int* find(*int* *x*,*int* *aa*[])

    {

        while(x!=aa[x])

        {

            x=aa[x];

        }

        return x;

    }

/\* As discussed earlier union happens when both the elements belong to different group ,as find returns the group which the node belong to. So when group is different ,now on linking two nodes the second node is now grouped to first node using the first array (so all elements under second element also included by the redirecting parent edges). The count array i.e the second array we add the second element’s group size to first so that the size of the two groups are added into single group size. Now the second element does not have a group so we make it zero. Note that only the parents(in first array) have the size maintained in same index of second array. \*/

    static *void* union(*int* *x*,*int* *y*,*int* *aa*[],*int* *bb*[])

    {

        if(y!=x)

        {

            aa[y]=x;

            bb[x]=bb[x]+bb[y];

            bb[y]=0;

        }

    }

/\* The second array contains the sizes, so maximum size is one, when no elements are linked so we check that in the if condition as said in the question single elements are ignored so we ignore 1s, and also check the maximum size in the size array using naive method of iteration and checking maximum value at each point. And finally return the maximum \*/

    static *int* findmax(*int* *bb*[],*int* *n*)

    {

*int* max=0;

        for(*int* i=1;i<2\*n+1;i++)

        {

            if(bb[i]>1 && bb[i]>max)

            {

                max=bb[i];

            }

        }

        return max;

    }

/\*In the same way as maximum but as given in question single node should not be considered so we check the size array element greater than 1 and then check minimum and return it to main function \*/

    static *int* findmin(*int* *bb*[],*int*  *n*)

    {

*int* min=2\*n;

        for(*int* i=1;i<2\*n+1;i++)

        {

            if(bb[i]>1 && bb[i]<min)

            {

                min=bb[i];

            }

        }

        return min;

    }

}

**Overall Explanation of Code:**

Two arrays are used here one for nodes and their linking/edges where nodes are linked in a redirecting fashion, another array for size where all the redireted nodes finally end at a point and that point has the total size of that component/group.This method is similar to quick union method where we add the new element to a group in the form of a tree where the parent has the total size (size is maintained in the separate second array with same index positions), and same is followed here and finally the parents’ having maximum and minimum size is returned from the size array.

**Time complexity analysis:**

Find : O(N)

(Find has an average of log(N) complexity because its maintained like a tree and height of tree is its group finding path)

Union : O(1)

Find Maximum : O(N)

Find Minimum : O(N)

Find for N elements : O(NlogN)

Union for N elements : O(N)

Total Time Complexity : O(NlogN) + O(N) + O(N) + O(N)

: O(NlogN)

Time Complexity : O(NlogN)

**Number of test cases passed:**

All 38 testcases have been passed and score obtained is 50.

**Solution Link:**

Permalink : <https://www.hackerrank.com/challenges/components-in-graph/forum/comments/729166>

Github Link : <https://github.com/Suuareezz/ADS_presentation/tree/master>

Language Used : Java8

**Hacker rank ID:**

Suuarezz

**Result:**

Hence the problem ‘Component in a graph’ has been solved successfully using the concept of disjoint sets with all the time and space complexity constraints abided.