**[mohamedrefat007](https://www.hackerrank.com/mohamedrefat007)**about 8 hours ago

Since the competition is over, and I've had some requests for the code, here it is, using Java 8:

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

public class Solution {

private static int[] id; // id[city] = id of connected component containing city

private static int[] size; // size[id] = number of cities in given component

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

int cities = in.nextInt();

int buses = in.nextInt();

//Write your code here

ArrayList<Integer>[] adjCities = (ArrayList<Integer>[]) new ArrayList[cities+1];

for (int c = 0; c <= cities; c++) {

adjCities[c] = new ArrayList<Integer>();

}

//initializing adjCities

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

int c1 = in.nextInt();

int c2 = in.nextInt();

adjCities[c1].add(c2);

adjCities[c2].add(c1);

}

//System.out.println(adjCities.length);

//Checking our adjCities

/\*for (int i = 0; i < adjCities.length; i++){

System.out.println(adjCities[i]);

}\*/

//System.out.println(adjCities[1].size());

//System.out.println(adjCities[1].get(1));

int connectedComponents = 0;

boolean[] visited = new boolean[cities+1];

id = new int[cities+1];

size = new int[cities+1];

for(int c = 1; c <= cities; c++) {

if(!visited[c]) {

dfs(c, adjCities, visited, connectedComponents);

connectedComponents++;

}

}

//System.out.println(connectedComponents);

//System.out.println(size.length);

/\*for (int i = 0; i < size.length; i++){

System.out.println(size[i]);

}\*/

int largestCC = Arrays.stream(size).max().getAsInt();

System.out.println(largestCC);

}

private static void dfs(int city, ArrayList<Integer>[] adjCities, boolean[] visited, int connectedComponents){

visited[city] = true;

id[city] = connectedComponents;

size[connectedComponents]++;

//System.out.println(city);

//System.out.println("visited[city] = true: "+city);

//System.out.println("adjCities[city].size(): "+adjCities[city].size());

//System.out.println(adjCities[2].get(0));

//System.out.println(adjCities[2].get(1));

for (int c = 0; c < adjCities[city].size(); c++){

if(!visited[adjCities[city].get(c)]){

//System.out.println("visiting adjCities[city.get(c)]: "+ adjCities[city].get(c));

dfs(adjCities[city].get(c), adjCities, visited, connectedComponents);

}

}

}

}

[**redligot2009**](https://www.hackerrank.com/redligot2009)about 7 hours ago

Nice problem! My solution involves going through all nodes and if the current node has not yet been added to a set, perform BFS while setting the parent of each node, and store the size of the set; meanwhile, if the current node has been added to a set, find the root node of the set that the current node belongs to, and then get the size of the root node set and check if it's greater than the maxSize thus far. [https://pastebin.com/AMBwr75F](https://www.hackerrank.com/external_redirect?to=https://pastebin.com/AMBwr75F)

1. #include <bits/stdc++.h>
2. #include <algorithm>
3. #define ii pair<int,int>
4. using namespace std;
6. int main() {
7. int n;
8. int m;
9. cin >> n >> m;
10. //sets of edges
11. set<int> edges[100001];
13. for (int i = 0; i < m; i++)
14. {
15. int a, b;
16. cin >> a >> b;
17. edges[a].insert(b);
18. edges[b].insert(a);
19. }
20. int maxSize = 0;
21. bool done[100001] = { false };
22. //the sets which are initially empty
23. set<int> s[10001];
24. //a vertex is marked as done when it has been added to a set in one of the sets in s
25. int parent[100001] = { 0 };
26. for (int i = 1; i <= n; i++)
27. {
28. //set parent of all vertices to itself initially
29. parent[i] = i;
30. }
31. for (int i = 1; i <= n; i++)
32. {
33. //if vertex i is not yet in any set
34. if (!done[i])
35. {
36. //mark vertex i as done
37. done[i] = true;
38. queue<int> q;
39. bool visited[100001] = { false };
40. visited[i] = true;
41. q.push(i);
42. //BFS
43. while (!q.empty())
44. {
45. int front = q.front();
46. q.pop();
47. s[i].insert(front);
48. //insert front into set s[i]
49. for (set<int>::iterator j = edges[front].begin(); j != edges[front].end(); j++)
50. {
51. int element = \*j;
52. if (!visited[element])
53. {
54. q.push(element);
55. visited[element] = true;
56. done[element] = true;
57. parent[element] = front;
58. //set parent of element to be front
59. }
60. }
61. }
62. int setsize = s[i].size();
63. //set maximum size of set
64. maxSize = max(maxSize, setsize);
65. }
66. else
67. {
68. //if vertex i is done
69. int curr = i;
70. s[curr].insert(i);
71. //insert vertex i in set s[i]
72. while (parent[curr] != curr)
73. {
74. //find the root
75. curr = parent[curr];
76. }
77. //get size of the root
78. int setsize = s[curr].size();
79. //set maximum size of set
80. maxSize = max(maxSize, setsize);
81. }
82. }
83. //print answer
84. cout << maxSize << endl;
85. return 0;
86. }

[**jalilalchy**](https://www.hackerrank.com/jalilalchy)about 8 hours ago

Solved in C++ Using union find/Disjoin Sets:

#include <bits/stdc++.h>

using namespace std;

class DisjointSet {

public:

class Node {

public:

long data;

Node \*parent;

int rank;

};

long findSet(long data) {

return findSet(nodeMap.find(data)->second)->data;

}

bool merge(long data1, long data2) {

Node \*node1 = nodeMap.find(data1)->second;

Node \*node2 = nodeMap.find(data2)->second;

Node \*parent1 = findSet(node1);

Node \*parent2 = findSet(node2);

if (parent1 == parent2) return false;

if (parent1->rank >= parent2->rank) {

parent1->rank = (parent1->rank == parent2->rank) ? parent1->rank + 1 : parent1->rank;

parent2->parent = parent1;

} else {

parent1->parent = parent2;

}

return true;

}

void makeSet(long data) {

Node \*node = new Node();

node->data = data;

node->parent = node;

node->rank = 0;

map<long, Node \*>::iterator it = nodeMap.begin();

nodeMap.insert(it, pair<long, Node \*>(data, node));

}

private:

map<long, Node \*> nodeMap;

Node \*findSet(Node \*node) {

Node \*parent = node->parent;

if (parent == node) return parent;

node->parent = findSet(node->parent);

return node->parent;

}

};

int main() {

int n;

int m;

DisjointSet d;

vector<long> answer;

cin >> n >> m;

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

d.makeSet(i);

answer.push\_back(0);

}

for (int route\_i = 0; route\_i < m; route\_i++) {

int u, v;

cin >> u >> v;

d.merge(u-1, v-1);

}

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

answer[d.findSet(i)]++;

}

sort(answer.begin(), answer.end());

cout << answer[answer.size() - 1];

return 0;

}