Crack-a-Hack

Maximal Tourism, Hackerrank

https://www.hackerrank.com/contests/rookierank-3/challenges/maximal-tourism

Course: Algorithmic Problem Solving
Course Code: 17ECSE309

Name: Akash Bengeri
USN : 01FE15BEC019

Solution 1

This solutions uses the concept of Disjoint sets by using parent node and child nodes.

```
#include <bits/stdc++.h>

using namespace std;

class DisjointSet {
public:
    class Node {
        public:
            long data;
            Node *parent;
            int rank;
    };

    long findSet(long data) {
        //Calls the private findset that uses nodes
        return findSet(nodeMap.find(data)->second)->data;
    }

    bool merge(long data1, long data2) {
        //Get the actual nodes from our long to Node map.
        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; //Shouldn't merge two nodes
in the same subset

        //Choose parent1 as the parent if it's rank is greater or equal to
parent 2's rank.
        if (parent1->rank >= parent2->rank) {
            //raise parent1's rank if there is a tie in ranks.
            parent1->rank = (parent1->rank == parent2->rank) ? parent1-
>rank + 1 : parent1->rank;
            parent2->parent = parent1;
        } else {
            //We are choosing parent2 as the parent here.
            parent1->parent = parent2;
        }
        return true;
    }

    //Add a new node to the space
    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; //This keeps track of all of our nodes, so
that we can easily iterate from 0 to n through hem
```

```
    Node *findSet(Node *node) {
        Node *parent = node->parent;
        if (parent == node) return parent;
        node->parent = findSet(node->parent); //Set the a nodes parent to
the result of the find.
        return node->parent;
    }
};


int main() {
    int n;
    int m;
    DisjointSet d;
    vector<long> answer;
    cin >> n >> m; //Get the number of nodes and vertices.

    //Create each node, and an answer associated with it's subset so far
not counting itself.
    for (int i = 0; i < n; i++) {
        d.makeSet(i);
        answer.push_back(0);
    }

    //Let's merge based on the vertices we are told exist.
    for (int route_i = 0; route_i < m; route_i++) {
        int u, v;
        cin >> u >> v;
        d.merge(u-1, v-1);
    }

    //Find the parent of each node, and increment the subset it belongs to
by 1.
    for(int i = 0; i < n; i++) {
        answer[d.findSet(i)]++;
    }

    //Sort and get the last element which is the max
    sort(answer.begin(), answer.end());
    cout << answer[answer.size() - 1];

    return 0;
}
```

References:

1) https://gist.githubusercontent.com/jaybyrrd/d6d5bce5c9864bb0fda50025da89182e/raw/fe8a85e64212078c9a9793b4fa246a58f8a4bd97/MaximalTourism.cpp
2) https://medium.com/@JayDByrd/disjoint-sets-and-maximal-tourism-medium-difficulty-problem-6e6155f73230
3) https://www.hackerrank.com/external_redirect?to=https://www.coursera.org/learn/algorithms-part1/lecture/RZW72/quick-union-improvements

Solution 2

One possible way to solve this problem is to iterate every location and let it be the start place, do a dfs and find out how many different locations it can reach. An crucial notice is that for the buses are all bilateral, if a location has been visited, we can just ignore it because the answer will not change.

```
#include <bits/stdc++.h>

using namespace std;

const int maxn = 100005;

vector<int> lk[maxn];
bool walk[maxn];
int n,m,ans,cnt;

void dfs(int now)
{
	if (walk[now]) return;
	++ cnt;
	walk[now] = 1;
	for(auto p : lk[now])
		dfs(p);
}

int main()
{
	scanf("%d%d", &n, &m);
	for(int i = 1,u,v;i <= m;i ++)
	{
		scanf("%d%d", &u, &v);
		lk[u].push_back(v),lk[v].push_back(u);
	}
	for(int i = 1;i <= n;i ++)
		if (!walk[i])
		{
			cnt = 0;
			dfs(i);
			ans = max(ans,cnt);
		}
	printf("%d\n", ans);
	return 0;
}
```

References :

1) https://www.hackerrank.com/contests/rookierank-3/challenges/maximal-tourism/editorial
2) https://github.com/prakashbh/day-today-codes/blob/master/32-dfs.c