CMSC398L Presentation

Gavin Hung 119207666

Problem

CSES Problem Set: Building Roads
https://cses.fi/problemset/task/1666

There are n cities with m roads between cities. There is a new goal of allowing every city to be accessible to every other city, meaning that there is a route between any two cities. What is the minimum number of roads required to accomplish this. Also print out the valid roads.

Input

The first line will have two integers n and m: the number of cities and the number of roads. The cities will be numbered from 1 to n.

Then, the following m lines will have two integers, which represent the two cities that are connected by the road.

Input Constraints

$$1 \leq n \leq 10^5$$

$$1 \leq m \leq 2 * 10^5$$

$$1 \leq a, b \leq n$$

Output

Print out the minimum number of required roads.
Then, print valid roads between two cities. Any valid solution will work.

Example

Input

In this example, there are 4 cities with 2 roads. There is a road connecting cities 1 and 2, and another road connecting cities 3 and 4.

```
4 2
1 2
```

```
3 4
```

Output

The output of this example is to add 1 new road between cities 2 and 3, allowing routes between any two cities.

```
1
2 3
```

Solution/Technique

The goal of the problem is for the graph to be one connected component.

When I first read this problem, I thought about implementing union find to separate the graph into different components. Then, I would connect the parent of one connected components to the parents of the other connected components.

However, I realized that performing a depth first search will essentially do the same thing in regards to finding the different connected components. The solution I went with has a visited array to keep track of visited nodes and performs a depth first search on all of the unvisited nodes. In the code, this translates to starting a depth first search if the node hasn't been visited yet by trying to start a depth first search on all of the n nodes. When a depth first search is started at a node, I save that node.

Once the depth first searches have been done, I print out the number of connected components - 1 to signify the number of roads that have to be added. The roads will be connecting the first node with the source of all of the other connected components.

Complexity

O(N) Space to keep track of visited nodes. There is also an array to keep track of the source of each depth first search.

O(N) Time, since we have a visited array to prevent running the depth first search more than once. Thus, every node will only be visited once.

C++ Code

```cpp
#include <climits>
#include <iostream>
#include <vector>
```

```cpp
using namespace std;

void dfs(int src, vector<vector<int>> &adj, vector<bool> &visited) {
  if (visited[src]) {
    return;
  }
  visited[src] = true;

  int neighbors = adj[src].size();

  for (int i = 0; i < neighbors; i++) {
    if (!visited[adj[src][i]]) {
      dfs(adj[src][i], adj, visited);
    }
  }
}

int main(int argc, char *argv[]) {
  int n, m;
  cin >> n >> m;

  vector<vector<int>> adj(n + 1);

  int a, b;
  for (int i = 0; i < m; i++) {
    cin >> a >> b;
    adj[a].push_back(b);
    adj[b].push_back(a);
  }

  int components = 0;
  vector<bool> visited(n + 1, false);
  vector<int> componentNodes;
  for (int i = 1; i <= n; i++) {
    if (!visited[i]) {
      dfs(i, adj, visited);
      components++;
      componentNodes.push_back(i);
    }
  }

  components--;
  cout << components << endl;
  for (int i = 1; i <= components; i++) {
    cout << componentNodes[0] << " " << componentNodes[i] << endl;
    }
  }
```