(RRR) - Restoring Rainbow Roads

A lot of what's interesting about programming is the ability to solve problems quicker using more efficient algorithms. In this assignment you will attempt to implement multiple solutions of a simple question. Questions to be solved as a part of the assignment are marked in red. Ensure to add readable comments and a one page report, including how long it takes your program to run on a small or large testcase for each implementation.

Post the events of Thor (2011), the bifrost is shattered into pieces and the realms are disconnected with no way to get to each other. The norse gods decide to reconnect the worlds using smaller rainbow roads between each pair of realms. They hire Svadilfari, the swift and intelligent horse to build these roads. It's a very time consuming task to build roads between a pair of realms, so Svadilfari would like to minimise the number of roads to be built.

A pair of realms are connected, if there exists a sequence of roads you can take to get from one planet to another.

The gods decide to send a set of roads they'd like to be built. Svadilfari will only build roads when necessary. **Essentially, if a pair of planets are already connected, we need not actually build the road there.**

<u>Input</u> -> Run the program, your input will be of the following format, to be read from the file roads.txt

The first line will denote the number of planets and the number of **Requests** from the norse gods, to be denoted by N and M respectively. The planets follow a 0 based number system, meaning they are numbered by 0 to N-1.

The next M lines contains a pair of numbers, denoted by x and y. This denotes the pair of planets that a road must be built between.

<u>Output</u> -> For each of the M lines, in a single concatenated line, print a 1 if a road will be built between those 2 realms and a 0 if a road will not be built between them. It should appear as a sequence of 1's and 0's in the first line of the file. The second line needs to contain the amount of time it took your program to run in milliseconds, rounded up to the nearest integer. The output is to be written to the file rollnumber_quiz2_q1_output.txt. (A sample of the input and output have been supplied below.)

What could you try to do? The underlying problem is to be understood as putting items into collections, also known as sets. Initially, each item, which are realms, are present in its own unique set. As you connect sets, or build rainbow roads, you combine sets to form larger sets. If 2 elements are already a part of the same set, there is no need to combine any set.

Attempt 1:

As the realms are numbered from 0 to n-1, perhaps you could initialise an array which denotes which set each index is supposed to be a part of.

Example input file: (contents of roads.txt)

```
10 5
1 2
1 3
6 7
1 7
2 6
```

Array with 10 elements.

Array is initialised with element = index:
A = [0,1,2,3,4,5,6,7,8,9]

1) Connect 1 and 2
   A = [0,1,1,3,4,5,6,7,8,9] - prints 1
2) Connect 1 and 3
   A = [0,1,1,1,4,5,6,7,8,9] - prints 1
3) Connect 6 and 7
   A = [0,1,1,1,4,5,6,6,8,9] - prints 1
4) Connect 1 and 7
   A = [0,1,1,1,4,5,1,1,8,9] - prints 1
5) Connect 2 and 6
   A = [0,1,1,1,4,5,1,1,8,9] - prints 0

The Output in this case should be: (contents of ee19b147_quiz2_q?_output.txt)

```
11110
40 ms
```

In place of the question mark, substitute 1,3 or 4 according to the question number.

1) Implement this algorithm successfully. The code should be stored in rollnumber_quiz2_q1.c

2) Can you generate a worst case sample for this algorithm? Assume n is a power of 2. Also, store an example of a worst case input in a file with 16 cities and 15 roads, in a file named roll_number_bad_roads.txt.

Attempt 2:

Look up the quick weighted union find method.

3) Define a struct for each city, and implement the above algorithm using structs and pointers. The code should be stored in rollnumber_quiz2_q3.c

4) Implement the above algorithm in the popular fashion, i.e with only a root and weight array. The code should be stored in rollnumber_quiz2_q4.c