Homework 6, CSCE 240, Spring 2014

RULES

The rules are as in the previous assignments.

Assignment

Your assignment is to compute independent cycles in a graph using the union-find algorithm.

This will require (I think) you to use `map` and `vector` containers, iterators, and some overloaded operators.

ALL GRAPHS FOR THIS ASSIGNMENT ARE ASSUMED TO BE UNDIRECTED GRAPHS.

You must overload the output operator for the `Arc` class. If you do this Java-like, you can start by writing a `toString` function, and then your overloading just calls that function. That lets you write code to do the main functionality of the assignment first and deal with the overloading only when you are ready to do that.

Background

Finding cycles in graphs is a hugely important task, although you might not think of it in the abstract. Cycle finding would allow you to determine redundant paths in the USC computer network, for example, or in a larger network. Cycle finding is done in optimizing compilers to detect whether or not assignment of values to variables can be done in independent threads. Chip and board design often require one to verify that electrical loops are dealt with properly. And finding who is communicating with whom would probably be something that could be inferred from the documents Snowden released.

Union-Find

The union-find algorithm is one way to get an *independent* set of cycles, that is, a set of cycles such that all cycles in a graph can be expressed as a sum of cycles from the independent set.

The input to your program will be one number n that is the number of arcs in the graph. This will be followed by n lines of data, each of which is a pair of integers a, b, representing an arc (a, b) from node a to node b. REMEMBER THIS IS AN UNDIRECTED GRAPH, so a to b is the same as b to a. We will assume that the nodes in the graph are represented by integers so as to simplify things. The integers don't actually have to be consecutive, but they probably will be.

The algorithm proceeds by building trees that represent paths in the graph. Every node starts by being the root of its own tree. In my code, I represent this as an arc from the node to itself, but you could represent this as an arc from the node to a dummy value for a node. Either works just fine, and the only difference is the comparison you make in your function that returns whether or not a node is the root of its tree.

When you read in an arc (a, b), you call the `find` function to find the root of the tree in which a lives. Then you call the `find` function to find the root of the tree in which b lives.

If these two roots are different, then you do a "union" of the two trees so that the new tree now contains the arc. I recommend that you standardize your code as if you had a directed graph and that you write the code as if the arc went from the node with the larger value to the node with the smaller value. The graphs are undirected, and this convention does not affect the correctness of the algorithm, but it will make your coding simpler to have a convention. Otherwise, you would have to think harder about duplication of arcs and paths and you'd have to notice when an arc got turned around from the way it was written in the input file to the way it was represented in the computer. Don't make life more difficult than it already is.

Now, if you chase up to the root from a and b, and you find that both trees have the same node r as the root, that means that there is a path down from r to a and a path down from r to b, and thus that adding the arc from a to b would create a cycle.

Don't add the arc to the tree. Instead, output the (a, b) arc, and then the path from a to r and the path b to r. This is the list of arcs forming a cycle.

Examples

Let's say you have trees

$$(7,5)(5,3)(3,1)(1,1),$$

$$(4,2)(2,2),$$

$$(8,8)$$

and and you encounter an arc $(8,3)$.

Chasing the 3 to its root, we get 1. Chasing the 8 to its root, we get 8. These are different, so we do a union. We can think of this as if things were directed, with the 8 as the "from" node and the 3 as the "to" node. What we want to do is to union the $(8,3)$ arc so that the path to its root is

$$(8,3)(3,1)(1,1).$$

That's really just like setting up a linked list.

Now, if we had then encounter an arc $(7,8)$, we would chase the 7 and the 8 to the common root 1.

Instead of adding this arc to a tree, we would output

$$(8,7)(7,5)(5,3)(3,1),$$

and

$$(8,3)(3,1).$$

(In my code I put the cycle-making arc as the first arc of the first path.)

BONUS OPPORTUNITY: I will give you full marks if you were to output

$$(7,8),$$

$$(7,5)(5,3)(3,1)(1,1),$$

and

$$(8,3)(3,1)(1,1).$$

That is, I will not require you to output ONLY the exact cycle. If you want to quit when you output the existence of the cycle and the information needed to show the complete cycle, that's ok. I will provide 10 bonus points for trimming back the two paths to roots so as to write only the cycle.