Joshua Nelson

CSCE 146

Lab Report 11

Problem

Create an algorithm to find all cycles in a graph using depth first search.

Proposed Solution

Modify depth first search to be recursive (accept visited nodes as a parameter). Starting at an arbitrary origin node, perform depth-first search until encountering a node already visited. Create a circularly linked list extending from the visited node until the current. In order for the nodes to be in the correct order, create a copy of the visited nodes each time a different neighbor is visited.

Once the circularly linked lists representing the cycles have been created, traverse each starting from each point in the list; i.e. a list 1-2-3 would also be traversed 2-3-1 and 3-1-2.

Print the result.

Tests and Results

```
> java test/CycleTester
0
4
1
2
6
3
5

0 4 1 0
1 0 4 1
4 1 0 4
0 4 2 0
2 0 4 2
4 2 0 4
0 4 6 3 1 0
6 3 1 0 4 6
1 0 4 6 3 1
```

Joshua Nelson

CSCE 146

4 6 3 1 0 4
3 1 0 4 6 3
0 4 6 5 2 0
6 5 2 0 4 6
2 0 4 6 5 2
4 6 5 2 0 4
5 2 0 4 6 5

Problems Encountered

This lab was very challenging. My original solution performed a breadth first search starting from every node in the tree; for both efficiency and implementation reasons I decided to use circularly linked lists instead. Once I hit upon the idea of circularly linked lists, it was relatively easy to implement the algorithm. I encountered the problem that the visited nodes were not always in order, which is why the algorithm makes copies of the visited nodes.

Conclusions and Discussion

My implementation is very inefficient. As the problem did not require efficiency, this is acceptable, but I would like to improve it for the next homework. I could do so by using a double-linked list to store the parents of each node, instead of copying the list each time.

Additional Questions

1. Could BFS also be used to find cycles? Describe why or why not.

Yes, as long as all cycles are encountered at least once, any method of traversal would work.

2. How could your algorithm be modified to discover if there is a way back to the starting vertex?

Perform breadth first search; if node linked is start vertex, return circular list.