CSE331 – Project 6: Unweighted Graphs

In this project you will complete an implementation of an efficient algorithm to find Euler Circuits in an unweighted graph. The project must be submitted to Handin no later than 11:59 pm Wednesday March 27, 2013 . Email the TA with any questions. Good Luck!

Project Description

You must implement an efficient algorithm to find an Euler Circuit in an unweighted undirected graph. An Euler Circuit is a path through a graph such that each edge is traversed only once and you end up at the same vertex you started from. A undirected graph, G=(V,E), contains an Euler circuit if and only if it is a connected graph and it has only vertexes with an even degree.

Your algorithm must run in $O(|V|+|E|)$ time, were **V** is the set of vertexes and **E** is the set of edges. It must accept a single command line argument giving the name of a text file with an adjacency list for a graph. Each line in the text file contains a number indicating which vertex it is, followed by numbers separated by spaces, indicating which vertexes this vertex has an edge to. Several example input files have been provided. Your executable must output the path it takes on a single line, listing the vertexes in the order they are visited in separated by a space. See Section 9.6.3 in the book on page 385 for more on Euler Circuits.

Programming Notes

For this project you have been given a great deal of latitude in the implementation, however the general restrictions in the project guidelines still apply. You must provide a Makefile (feel free to modify one from a previous project), and it must produce an executable called 'eulercircuit'.

The output format is one line of text, as described above. If the graph does not contain a valid Euler circuit than you must output the string '-1'. The executable file eulercircuit_linux_solution, in the project directory, demonstrates this behavior. For example:

```
<133 adriatic:~/Public/Project6 >eulercircuit_linux_solution figure9.70.txt
1 3 2 8 9 10 7 4 10 5 4 11 10 12 9 6 3 9 7 3 4 1
```

The executable lists the order in which each vertex is visited, and returns the starting vertex again at the end. You program must behave in the same way. *If your executable does not adhere to this output format it will not be graded.*

An additional executable has been provided for this project. The file eulerverifier_linux takes one of the properly formatted graph files from the command line, and a list of vertexes via standard input. For example, we can give it the same file as before, and enter the list generated by our solution into standard input:

```
<140 adriatic:~/Public/Project6 >eulerverifier_linux figure9.70.txt
1 3 2 8 9 10 7 4 10 5 4 11 10 12 9 6 3 9 7 3 4 1
1 3 2 8 9 10 7 4 10 5 4 11 10 12 9 6 3 9 7 3 4 1  is a valid Euler circuit.
```

Note that if you input the vertexes manually, you must put an extra space at the end. An easier way is to simply pipe the output from your executable to the verifier program, like so:

```
<141 adriatic:~/Public/Project6 >eulercircuit_linux_solution figure9.70.txt |
eulerverifier_linux figure9.70.txt
1 3 2 8 9 10 7 4 10 5 4 11 10 12 9 6 3 9 7 3 4 1  is a valid Euler circuit.
```

Also it will work if the graph does not contain a valid circuit:

```
<143 adriatic:~/Public/Project6 >eulercircuit_linux_solution invalid1.txt |
eulerverifier_linux invalid1.txt
Testing invalid graph.
Program returned -1.
```

Note that both commands on either side of the "|" are on the same line, the lines have been wrapped here.

To determine if a graph is valid, you must first make sure each vertex has an even degree, then you must perform a breadth first search to make sure all of the vertexes in the graph are connected. For this project you may use anything from the STL. I recommend using STL list to store your path, that way you can easily use the splice() function to merge paths together. STL list is also a doubly linked list, which means that you can erase items from it in constant time (using iterators), so it may also be ideal for storing adjacency lists.

We have provided you with a few graphs to test your projects with, but make sure to come up with your own test cases!

Project Deliverables

The following files must be submitted via Handin no later than 11:59 pm Wednesday March 27, 2013:

- *.cpp *.h – your source code files must use these extensions
- Makefile – produces the executable eulercircuit
- project6.pdf – files containing your answers to the written questions. (Must be a Portable Document Format file)

Written Questions

Submit the answers to these questions in a PDF file, called project5.pdf, along with your source code files. Any images or diagrams must be embedded in your PDF file, they will not be graded if they are separate files. For problems asking for a short paragraph, the paragraph should consist of 4-5 complete and concise sentences.

1. There are two types of professional wrestlers: "good guys" and "bad guys". Between any pair of professional wrestlers, there may or may not be a rivalry. Suppose we have n professional wrestlers and we have a list of r pairs of wrestlers for which there are rivalries. Give an $O(n+r)$-time pseudo-code algorithm that determines whether it is possible to designate some of the wrestlers as good guys and bad guys such that each rivalry is between a good guy and a bad guy. If it is possible to perform such a designation, your algorithm should produce it. Make sure you define your inputs and outputs and give a short paragraph describing your algorithm. (HINT: Your algorithm will need to incorporate a breadth first search.)
2. The following questions refer to the graph in Figure 9.82:
    a. Give the minimum spanning tree generated by Prim's algorithm.
    b. Give the minimum spanning tree generated by Kruskal's algorithm.
3. Give a pseudo-code algorithm to determine if an edge (u , v) in a depth-first search tree of a directed graph is a tree, back, cross or forward edge. Assume a modified version of the DFS algorithm on page 379, which marks the order visited, and the order fully explored. Assume that the base call to DFS has time=0.

```
void Graph::DFS(Vertex v, unsigned &time){
v.visited = true;
v.discovered = ++time;
 foreach Veretex w adjacent to v
 if(!w.visited)
      dfs(w, time);
 v.explored = ++time;
}
```

4. An Euler circuit of a connected *directed* graph G=(V,E) is a cycle that traverses each edge of G exactly once, although it may visit a vertex more than once. Note that this is different than an Euler circuit, which visits each edge of an *undirected* graph. Prove (by contradiction) that G contains an Euler path if and only if in-degree(v) = out-degree(v) for each vertex $v \in V$.
5. In breadth-first and depth-first search, an *undiscovered* vertex is marked *discovered* when it is first encountered, and marked *completely-explored* when it has been fully searched. At any given moment, various numbers of vertexes can be in any of these states. In all cases below, describe a graph on n vertexes with a particular starting vertex *v* that has the properties described.
    a. At some point during a breadth-first search, Θ(*n*) vertexes are simultaneously in the discovered state.
    b. At some point during a depth-first-search Θ(*n*) vertexes are simultaneously in the discovered state.
    c. At some point during a breadth-first-search Θ(*n*) vertexes are in the undiscovered state, Θ(*n*) vertexes are in the completely explored state, and only Θ(1) vertexes are discovered.