CSCE 221 Cover Page
Homework Assignment #6

Oneal Abdulrahim 324007937

oneal.abdulrahim oneal.abdulrahim@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more in the Aggie Honor System Office http://aggiehonor.tamu.edu/

Type of sources	
People	Dr. Leyk Peer TA
Web pages (provide URL)	http://en.cppreference.com/w/ http://www.geeksforgeeks.org/c-plus-plus/
Printed material	Textbook
Other Sources	Dr. Leyk Slides

I certify that I have listed all the sources that I used to develop the solutions/code to the submitted work.

"*On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.*"

Your Name (signature) Oneal Abdulrahim Date 5 December 2016

Implementation, C++ Features, Assumptions on Input Data

In this programming assignment, we were tasked with implementing a Graph data structure and then providing a method for finding its topological order, assuming it was a directed, acyclic graph (DAG). In my implementation, I chose to follow the template code provided and create three classes: Edge, Vertex, and Graph. Edge objects had a property of their weight, as well as beginning and end vertex values. The Vertex objects had a vector of each connected edge, as well as a data label (integer). Finally, the Graph object had a vector of Vertex objects corresponding to every vertex in the graph.

The topological ordering algorithm that I chose to implement is Kahn's Algorithm. This is an iterative approach for graph traversal, as opposed to our implementation of depth-first-search. In contrast with DFS, Kahn's algorithm uses a Queue to store vertices whose in-degree values are zero, and checks its children. In-degree values correspond to the number of edges directed at the current vertex, so an in-degree of 0 means no edges directed at the current vertex. The algorithm first calculates every vertex's in-degree value, then checks its children vertices, and subtracts one from their in-degree value. If this is now 0, we enqueue this vertex, and so on. This way, when dequeued, the algorithm gives correct topological order. We must track cycles, too, to verify the integrity of DAG. If topological ordering fails, or there is a cycle, an error message is printed.

Some C++ features used include the usage of the STL, namely in vectors and queues. Included also is overloaded output operators << for all three classes. The assumptions made on input data is critical. When attempting to find a topological order for a graph, this program assumes the graph input:

- vertex data is consecutive, positive integers starting at 1
- data stream includes a series starting first with a vertex data label, then its edge endpoint values, then a -1, delimited by whitespace

Testing

Included below are some screenshots of three test cases. One is the example test case, the other two I created.

```
:: ./main test.data
VERTEX 1: 2 3
VERTEX 2: 4 5
VERTEX 3: 6 7
VERTEX 4: 8
VERTEX 5: 9 10
VERTEX 6: 10
VERTEX 7: 12
VERTEX 8: 11
VERTEX 9: 11
VERTEX 10: 12
VERTEX 11: 13
VERTEX 12: 13
Possible topological ordering:
1 2 3 4 5 6 7 8 9 10 11 12
```

This graph represents a minimum heap

```
:: ./main test.data
VERTEX 1: 2 3
VERTEX 2: 1 3
VERTEX 3: 1 2
Possible topological ordering:
Topological ordering failed!
```

This graph is clearly cyclical

```
:: ./main input.data
VERTEX 1: 2 4 5
VERTEX 2: 3 4 7
VERTEX 3: 4
VERTEX 4: 6 7
VERTEX 5:
VERTEX 6: 5
VERTEX 7: 6
Possible topological ordering:
1 2 3 4 7 6 5
```

This graph is our example data

Running Time & Algorithm Efficiency

When using topological ordering, the main concern is to not get lost in the recursion or checking of children vertices, that is, iterate over vertices unnecessarily. For Kahn's Algorithm, we rely on the number of vertices and edges. The definitive steps in the algorithm come from the calculation of the in-degree values. This step is time complexity $O(V + E)$ where V is the number of vertices and E is the number of edges. When calculating the in-degree values, we traverse every Vertex and check each Edge within it. Below is some pseudocode for the definitive step.

```
for every Vertex in graph:
          if there are Edges to check:
               for every Edge directed at current Vertex:
                    in_degree increment by 1
```

Other functions, like adding a vertex to the graph and adding an edge to a vertex, offer $O(1)$ running time due to the usage of vectors. Order is ignored.

Real World Applications

Topological ordering is important to shortest path algorithms, which can be used in exactly that way in real life; vertices can represent locations and edges can represent distances. C++ can use graphs in its linker to organize files and libraries within a program. Networking (computer, social) an easily be represented using graphs, and the topological ordering provides a hierarchic display, in a way. Topological ordering can also provide instruction ordering.