CSCE 221 Cover Page – Programming Assignment 5

First Name: Maria Last Name: Dmitrievskaia UIN: 927009911

User Name: E-mail address: dmimar382@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 on Aggie Honor System Office website: http://aggiehonor.tamu.edu/

Type of sources				
People				
Web pages (provide URL)				
Printed material				
Other Sources				

I certify that I have listed all the sources that I used to develop the solutions/codes 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: Maria Dmitrievskaia Date: 04/23/20

Description of implementation, C++ features used, assumptions on input data:

The purpose of this assignment was to implement a graph data structure and implement topological ordering on the graph. The program takes an input file and builds a corresponding graph. The graph is populated by reading from a text file with a fixed format. The results are two vectors, one of which stores each vertex in the graph, and the other of which stores the adjacent vertices. Once the graph is created, the user is able to see the graph in the following format with the help of the dispayGraph() function:

```
1:2 4 5
2:3 4 7
3:4
4:6 7
5:
6:5
7:6
```

The topological sort algorithm is based on the indegrees of each vertex. To compute the indegree, the program counts how many connections there are to each vertex. So, the vertices are sorted from the smallest indegree to the largest with the help of a queue. The queue assists in keeping track of which vertex should be next in the topological ordering. First, the topological sort function iterated through all the vertices to check if any vertex has an indegree of 0. If a vertex has an indegree of 0, it is added to the queue. The algorithm then continues, until the queue is empty and preforms the following operations.

1. Store the value for the front of the queue
2. Pop the queue
3. Go through the adjacency list for the stored value and find any values what have an indegree of 0 and push them to the queue.

This algorithm allows for topological ordering of values in a graph.

Some features used in this program involve vectors, strings, lists, queues, and input streams. An assumption made on the input data is that it will be in the correct format. Namely, each vertex is the first value of a new line; each line ends with a “-1” and a new line begins. All the values between the first value and the “-1” are adjacent to the vertex in that line.

Why does the algorithm use a queue? Can we use a stack instead?

The algorithm uses a queue in order to keep track of the next vertex in the topologically sorted list. Because of the First-In-First-Out structure of a queue, it makes it easy to use a queue to keep track of the next vertex in the sorting process. When the vertex has an indegree of 0, it gets pushed into the queue. Once the vertex is in the queue, in the next iteration of the while loop, the first value of the queue gets popped and its top_num gets updated. Thus, unless there is a cycle, the algorithm goes through all the vertices and computes their top_num. The algorithm can use a stack instead of a queue, as it can keep track of the next vertex as well. The stack has a First-In-Last-Out structure, so, the implementation would have to differ than that of a queue, however, the implementation would be valid. The implementation can use a bool value to keep track of the visited vertices. Then, until all the vertices are visited, the topological sort function will be recursively called; while exiting the from all the recursive calls, each vertex will be added to the stack, thus ensuring that the last vertex in the stack is the first item in the topologically sorted list.

Can you explain why and how the algorithm detects cycles?

The algorithm detects cycles in order to know if the graph can be topologically sorted. The algorithm works by visiting nodes with indegrees 0. If the algorithm ever arrives at a state where there are no nodes with indegrees 0, then a cycle has been detected and the graph can not be topologically sorted.

What is the running time for each function? Use the Big-O notation asymptotic notation and justify your answer.

The running time for the *compute_indegree()* function is O(|V| +|E|) because in order to compute the indegrees, the function must go through all of the vertices and their corresponding adjacency lists to calculate how many in degrees each vertex has.

The running time for the *topological_sort()* function is O(|V| +|E|) because in order to compute the top_num, the function must go through all of the vertices and their corresponding adjacency lists. This is done in order to keep track of the in degrees of each vertex and place the vertex within degree of zero into the queue. As stated before, the queue assists in keeping tack of which vertex is next in the topological sorting.

The running time for the *print_top_sort()* function is O(|V|) because in order to print the topological sort, the function must go through all of the vertices and display their top_nums.

The running time for the *buildGraph()* function is O(n), where n is the number of integers in the input file. This is because each valid integer need to be places into the graph, thus, the function must run through each number in the input file to determine whether to disregard the integer or where to place the integer in its corresponding place.

The running time for the *displayGraph()* function is O(|V| +|E|) because in order to display the graph, the function must go through all of the vertices and their corresponding adjacency lists to output them to the screen.

Testing and Screenshots of output

Input.data Output:

```
[dmimar382]@compute ~/CSCE221/Programming Assignments/PA5> (15:48:52 04/23/20)
:: make
rm -f main.o graph.o topological_sort.o  main
g++ -g -std=c++11 -c main.cpp
g++ -g -std=c++11 -c graph.cpp
g++ -g -std=c++11 -c topological_sort.cpp
g++ -g -std=c++11 -o main main.o graph.o topological_sort.o
./main input.data
1:2 4 5
2:3 4 7
3:4
4:6 7
5:
6:5
7:6
Topological sort: 1 2 3 4 7 6 5
```

Input2.data Output:

```
[dmimar382]@compute ~/CSCE221/Programming Assignments/PA5> (15:49:22 04/23/20)
:: make
rm -f main.o graph.o topological_sort.o  main
g++ -g -std=c++11 -c main.cpp
g++ -g -std=c++11 -c graph.cpp
g++ -g -std=c++11 -c topological_sort.cpp
g++ -g -std=c++11 -o main main.o graph.o topological_sort.o
./main input2.data
1:3
2:3 8
3:4 5 6 8
4:7
5:7
6:
7:
8:
Topological sort: 1 2 3 4 5 6 8 7
```

Input3.data Output:

```
g++ -g -std=c++11 -o main main.o graph.o topological_sort.o
./main input3.data
1:2 4
2:5 7 8
3:6 8
4:5
5:6 9
6:
7:
8:
9:
Topological sort: 1 3 2 4 7 8 5 6 9
```

Input-cycle.data Output:

```
[dmimar382]@compute ~/CSCE221/Programming Assignments/PA5> (13:26:49 04/26/20)
:: make
rm -f main.o graph.o topological_sort.o  main
g++ -g -std=c++11 -c main.cpp
g++ -g -std=c++11 -c graph.cpp
g++ -g -std=c++11 -c topological_sort.cpp
g++ -g -std=c++11 -o main main.o graph.o topological_sort.o
./main input-cycle.data
1:2 4 5
2:3 4 7
3:4
4:6 7
5:4
6:5
7:6
Cycle found: illegal input
Topological sort: 1 2 3 0 0 0 0
```

Conclusion:

Throughout this assignment, I have learned how to create a graph data structure and how to implement a topological sorting algorithm using the constructed graph. I am now familiar with this data structure and its advantages, as well as the topological sorting algorithm.