CSCE 221 Cover Page
Programming Assignment #5

First Name **Shikhar** Last Name **Baheti** UIN **627007339**

User Name **shikhar** E-mail address **shikhar@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. According to the University Regulations, Section 42, scholastic dishonesty are including: acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. Disciplinary actions range from grade penalties to expulsion read more: Aggie Honor System Office

Type of sources			
People			
Web pages (provide URL)	stackexchange.com	geeksforgeeks.org	
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."

Electronic signature Shikhar Baheti Date April, 27, 2020

Shikhar Baheti

CSCE 221 – 501

Directed Acyclic Graph (DAG)

Programming Assignment – 5

Description:

This assignment required us to find a topological ordering in a Directed Acyclic Graph (DAG). A graph is an abstract data type that is used to relate data. Graphs can be used to represent scenarios such as connecting airline flights, internet, connecting cities, etc. A directed graph with no cycles is called a Directed Acyclic Graph. A topological sort takes a directed acyclic graph and produces a linear ordering of all its vertices such that if the graph *G* contains vertices (u, v), *u* comes before *v* in the ordering. This assignment required us to create a DAG from an input file, topologically sort it, and output the results from the topological sort.

Implementation:

There were several functions that were implemented in this assignment:

- buildGraph():
    - I used C++ features such as strings, input/output stream from files, vectors, and lists to create this function. This function reads an input file and processes the data in that file to create a Directed Acyclic Graph. The first character denoted in a line is the vertex number and the following characters (until termination of line by '-1') are the vertexes that our vertex points to (adjacent to). The vertex itself is added to a vector called node_list whereas the adjacent vertices are stored in the adj_list vector for that vector.
- displayGraph():
    - This function iterates through all the vertices in the vector node_list and outputs the adjacent vertices of each vertex.
- compute_indegree():

    - This function iterates through all the vertices in the vector node_list and calculated and then increments the indegrees for all the vertices.
- topological_sort():
    - The function topological sort creates a queue and pushes the vertices with indegrees zero in them. The function then (while the queue is not empty) dequeues the first vertex and pushes the vertices adjacent to that initial vertex in the queue if their indegree, decreased by one, is equal to zero. The function then pushes each such vertex in a different vector top sort and increments the counter. Lastly, the function checks for cycles in the graph and throws an exception if it detects a cycle.
    - Using a queue versus using a stack:
        - The function utilizes a queue instead of a stack for its first in first out operations. To topologically sort, initially, vertices with zero indegrees are enqueued onto the queue and then later dequeued to further process their indegrees.
        - We could have used a stack. We would push vertices on the bottom of a stack and process them as first in last out. In the end, we would have to empty the stack upside down to produce the desired topological sort.
- print_top_sort():
    - This function iterates through the vector top_sort and prints out each vertex in its topological ordering.

C++ Features:

Several C++ features were used to complete this assignment.

- Vectors: Vectors were used to store data such as node_list, adj_list, and top_sort.
- Lists: Lists were used to dynamically allocate memory to store the adjacency list for each of the vertices in node_list vector.
- Queue: A queue was used to enqueue and dequeue vertices to produce the topologically sorted output.
- Input/output stream and strings: These were used to process data from the files and correctly store them in vectors for further processing.

- Iostream: Iostream was utilized to display/print vertices with their adjacent lists and output the topological sorting.
- Exceptions: Exceptions were used and thrown when a graph would detect a cycle while topologically sorting itself.

Input and output data:

We assumed that all the data in each input file comprised only of integers that followed the same format of first character in a line being a vertex and until the line hits a "*-1*", the following numbers were the vertices that the initial vertex is adjacent to.

Cycles:

Since the function topological_sort() should only optimally run for the amount of times the vertices in the graph, if the counter is greater or less than the number of vertices in the graph, our function will detect a cycle in the graph because the function would endlessly loop between vertices going back and forth enqueuing and dequeuing.

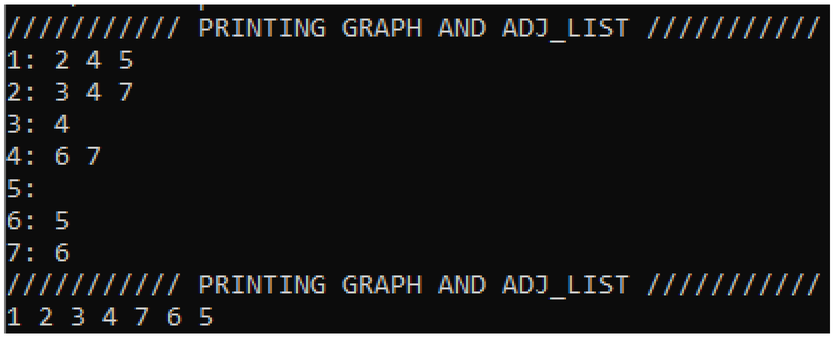
Runtime:

- buildGraph():
  - This function has a runtime of *O (V + E)* for V being the number of vertices in the graph and E being the number of edges in the graph.
  - This is so because the function iterates through all the vertices in each file and then for those vertices, iterates through the edges that connect each adjacent vertex.
- displayGraph():
  - This function has a runtime of *O (V + E)* for V being the number of vertices in the graph and E being the number of edges in the graph.
  - This is so because the function iterates through all the vertices in the vector node_list and then for those vertices, iterates through the edges that connect each adjacent vertex in the adj_list vector.
- compute_indegree():
  - This function has a runtime of *O (V + E)* for V being the number of vertices in the graph and E being the number of edges in the graph.

    - This is so because the function iterates through all the vertices in the vector node_list and then for those vertices, iterates through the edges that connect each adjacent vertex in the adj_list vector and increments their indegree.
- topological_sort():
    - This function has a runtime of *O (V + E)* for V being the number of vertices in the graph and E being the number of edges in the graph.
    - The function iterates through each vertex in the vector node_list and then pushes each vector with indegree of zero to the queue. Then, it iterates through all the adjacent vertices of that given vector and further processes their indegree to be pushed into the queue.
- print_top_sort():
    - This function has a runtime of *O (V)* for V being the number of vertices in the graph.
    - This is because it iterates through the vector top_sort which contains all the topologically sorted vertices.

Testing:

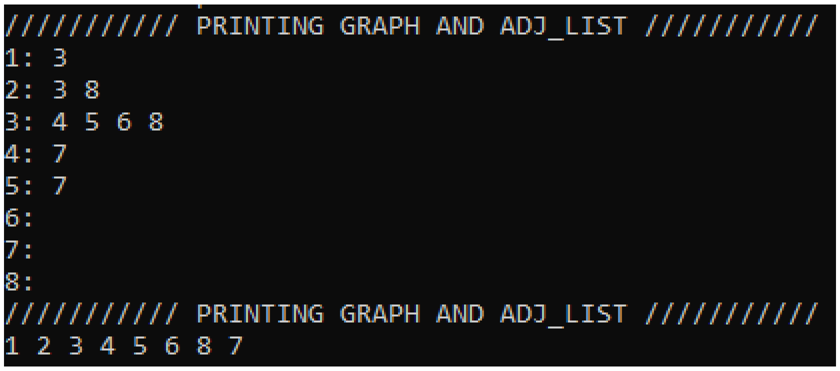
Case 1:

- Output:
    - 

```
/////////// PRINTING GRAPH AND ADJ_LIST ///////////
1: 2 4 5
2: 3 4 7
3: 4
4: 6 7
5:
6: 5
7: 6
/////////// PRINTING GRAPH AND ADJ_LIST ///////////
1 2 3 4 7 6 5
```

Case 2:

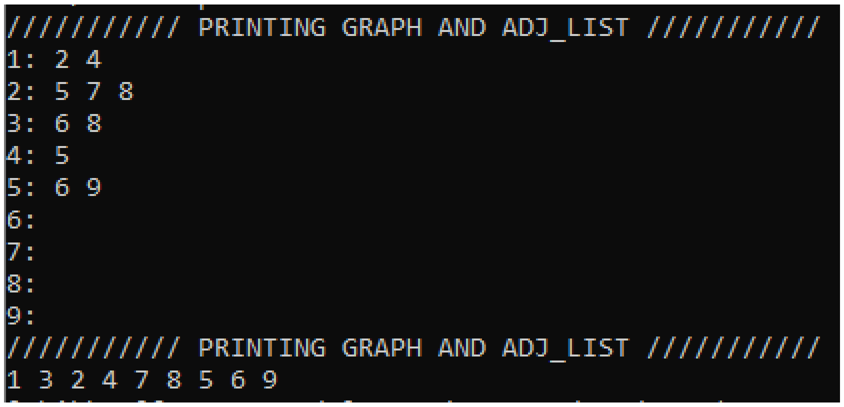
- Output:

- 

```
/////////// PRINTING GRAPH AND ADJ_LIST ///////////
1: 3
2: 3 8
3: 4 5 6 8
4: 7
5: 7
6:
7:
8:
/////////// PRINTING GRAPH AND ADJ_LIST ///////////
1 2 3 4 5 6 8 7
```

- Description: Samantha should take the courses in the following order:
  - CSCE121 ➔ CSCE222 ➔ CSCE221 ➔ CSCE312 ➔ CSCE314 ➔ CSCE313 ➔ CSCE411 ➔ CSCE315

<u>Case 3</u>:

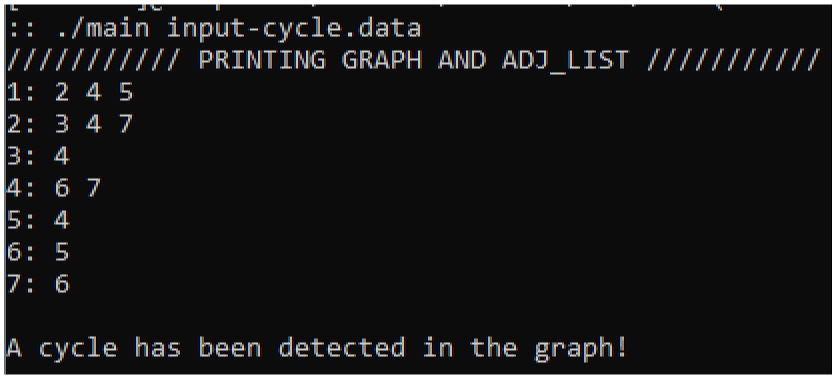
- Output:
  - 

```
/////////// PRINTING GRAPH AND ADJ_LIST ///////////
1: 2 4
2: 5 7 8
3: 6 8
4: 5
5: 6 9
6:
7:
8:
9:
/////////// PRINTING GRAPH AND ADJ_LIST ///////////
1 3 2 4 7 8 5 6 9
```

- Description: Samantha should also take the foreign languages courses in the following order:
  - LA15 ➔ LA22 ➔ LA16 ➔ LA31 ➔ LA127 ➔ LA141 ➔ LA32 ➔ LA126 ➔ LA169

<u>Case 4</u>:

- Output:

- 

```
:: ./main input-cycle.data
/////////// PRINTING GRAPH AND ADJ_LIST ///////////
1: 2 4 5
2: 3 4 7
3: 4
4: 6 7
5: 4
6: 5
7: 6

A cycle has been detected in the graph!
```

- Description: Since a cycle was detected in the graph, an exception was thrown.

Conclusion:

Through the means of this programming assignment, we learned how Directed Acyclic Graph (DAG) work. We also found the topological ordering of each of the input files and calculated that the worst-case run-time for each function is of O(V+E) time complexity and space complexity of also, O(V+E).