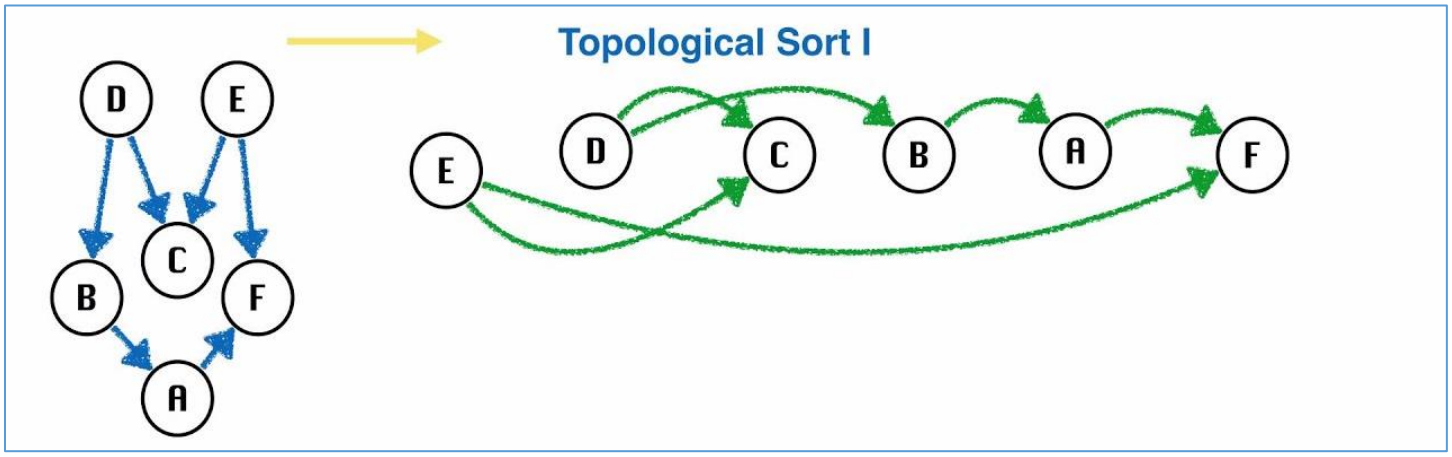
1310 PROGRAM 4 – Topological Sort



DESCRIPTION

Write a C++ program that can read a digraph's number of vertices and the edges from a text file (example below), create an adjacency matrix, and then use the depth-first search (DFS) method to topologically sort the vertices.

graph_1.txt - Notepad

File Edit Format View Help

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

```
Adjacency Matrix:
0       1       0       1
0       0       1       0
0       0       0       1
0       0       0       0
```

```
The graph vertices topologically sorted are: 0  1  2  3
```

Ten extra credit points will be awarded for also creating an adjacency list of the digraph.

```
Adjacency List...
0--->1--->3--->NULL
1--->2--->NULL
2--->3--->NULL
3--->NULL
```

FILES GIVEN & ASSUMPTIONS YOU CAN MAKE

- You may assume that all digraphs have only integer vertices.
- You may assume that all digraphs contain at least two vertices, zero and one.
- No integers are skipped. For example, a digraph with 7 vertices contains vertices 0, 1, 2, 3, 4, 5, & 6.
- You are given the Stack template class, which you will use during depth-first-search of the graph.
- You are given the text files which your program will be tested with: graph_1.txt, graph_2.txt, graph_3.txt, graph_4.txt, and graph_5.txt

GRAPH.CPP

This file contains the main function. It should include the other header files that you will use in your program.

Here is the order of how your main function should go:

- Ask the user for the filename of the file that contains the # of vertices in the graph and then the edges.
- Read in the number of vertices, which you should use to create your matrix (and list if you are doing the extra credit option)
- Read each pair of vertices that makes each edge and then add that edge to the matrix (and add that edge to the adjacency list if you are doing the extra credit option)
- Print the adjacency matrix (and print the adjacency list if you are doing the extra credit option)
- Sort the list topologically using a stack. Hint: you should dynamically allocate a visited array the size of the number of vertices and also an array the size of the number of vertices that holds the topologically sorted vertices.
    - Make sure to print if you find a cycle and tell the user if the graph can't be sorted if it is not a DAG

GRAPHMATRIX.H

PRIVATE ATTRIBUTES:

- int ** vertexMatrix – this is going to be a 2D array of integers (the matrix), which will be dynamically allocated in the constructor.
- int numVertices
- int numEdges

PUBLIC MEMBER FUNCTIONS:

- **constructor** – accepts an integer (the number of vertices in the graph), sets the private attribute numVertices, dynamically allocates a 2D array, sets all elements of the 2D array to zero
- **destructor** – deletes 2D array
- **addEdge** – accepts two vertices – sets the 2D array to 1 (instead of zero) at that element in the matrix
- **printGraph** – prints the matrix
- **getFirstVertex** –this will either get the vertex that doesn't have any edges pointing to it (only has edges pointing away from it or it will return vertex zero.
- **isThereAnEdge** – this will accept a row & column index and will return true if the matrix element is equal to 1 (there is an edge) or false otherwise.

GRAPHLIST.H (FOR EXTRA CREDIT ONLY – NOT MANDITORY)

PRIVATE ATTRIBUTES:

- ListNode structure (containing integer value & pointer to next ListNode
- ListNode ** headArray; (array of linked lists)
- int numVertices
- int numEdges

PUBLIC MEMBER FUNCTIONS:

- **constructor** – accepts an integer (the number of vertices in the graph), sets the private attribute numVertices, dynamically allocates an array of pointers to ListNodes
- **destructor** – deletes linked lists
- **addEdge** – accepts two vertices – create the node & add it to appropriate linked list
- **printGraph** – prints the matrix

GRAPH TEXT FILES

GRAPH_1.TXT

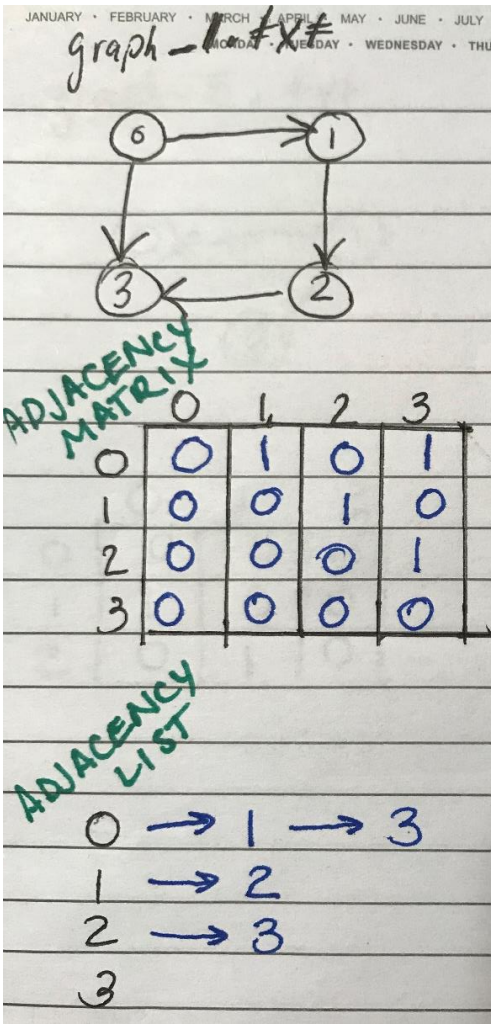
graph_1 - Notepad

File Edit Format View Help

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

The **4** is how many vertices there are.

The other numbers represent the **edges**. For example, there is an edge from 0 to 1.

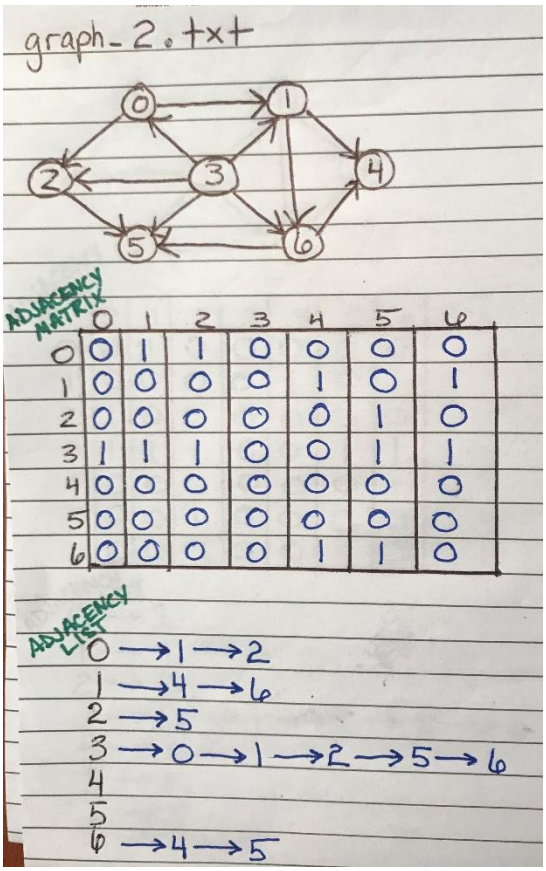


GRAPH_2.TXT

graph_2 - Notepad

File Edit Format View Help

```
7
0 1
0 2
1 4
1 6
2 5
3 0
3 1
3 2
3 5
3 6
6 4
6 5
```

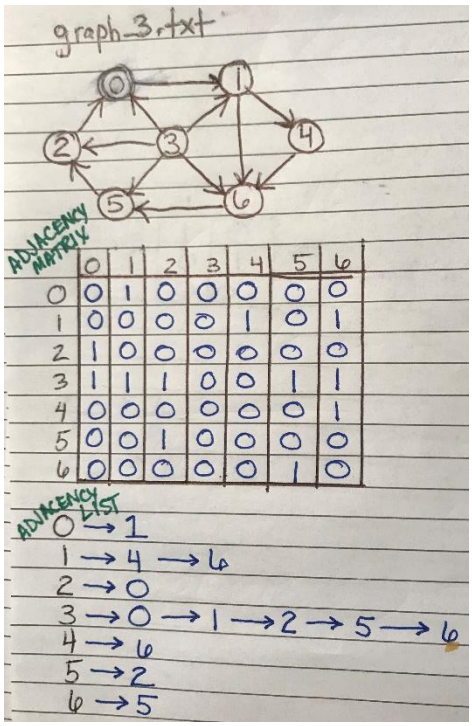


GRAPH3.TXT

graph_3 - Notepad

File Edit Format View Help

```
7
0 1
1 4
1 6
2 0
3 0
3 1
3 2
3 5
3 6
4 6
5 2
6 5
```

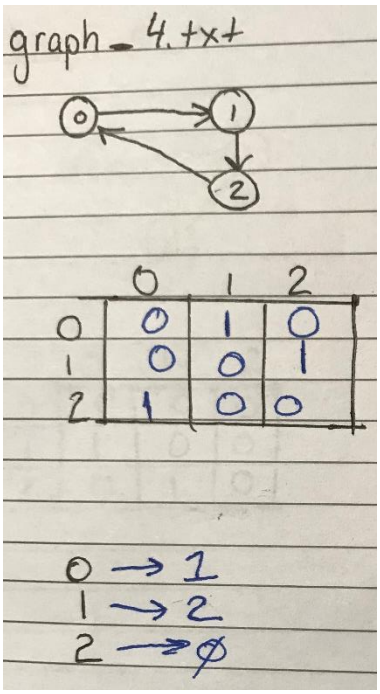


GRAPH4.TXT

graph_4 - Notepad

File Edit Format View Help

```
3
0 1
1 2
2 0
```

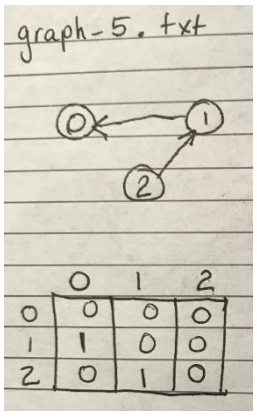


GRAPH5.TXT

graph_5 - Notepad

File Edit Format View Help

```
3
2 1
1 0
```



SAMPLE OUTPUT

```
C:\Users\acrockett\Desktop\CSC\CSC Fall 2018\CSC2400\PROGRAMS\PROGRAM THREE>a

Enter the name of your file that contains the graph vertices:  graph_1.txt

Adjacency Matrix:
0       1       0       1
0       0       1       0
0       0       0       1
0       0       0       0

Adjacency List...
0--->1--->3--->NULL
1--->2--->NULL
2--->3--->NULL
3--->NULL

Now topogocially sorting graph.

The graph vertices topologically sorted are: 0  1  2  3
```

```
Enter the name of your file that contains the graph vertices:  graph_2.txt

Adjacency Matrix:
0       1       1       0       0       0       0
0       0       0       0       1       0       1
0       0       0       0       0       1       0
1       1       1       0       0       1       1
0       0       0       0       0       0       0
0       0       0       0       0       0       0
0       0       0       0       1       1       0

Adjacency List...
0--->1--->2--->NULL
1--->4--->6--->NULL
2--->5--->NULL
3--->0--->1--->2--->5--->6--->NULL
4--->NULL
5--->NULL
6--->4--->5--->NULL

Now topogocially sorting graph.

The graph vertices topologically sorted are: 3  0  2  1  6  5  4
```

C:\Windows\System32\cmd.exe

```
Enter the name of your file that contains the graph vertices:  graph_3.txt


Adjacency Matrix:
0       1       0       0       0       0       0
0       0       0       0       1       0       1
1       0       0       0       0       0       0
1       1       1       0       0       1       1
0       0       0       0       0       0       1
0       0       1       0       0       0       0
0       0       0       0       0       1       0


Adjacency List...
0--->1--->NULL
1--->4--->6--->NULL
2--->0--->NULL
3--->0--->1--->2--->5--->6--->NULL
4--->6--->NULL
5--->2--->NULL
6--->5--->NULL


Now topogocially sorting graph.

Ah shooty pooty.  This graph has a cycle.

The graph can't be topologically sorted because it is not a DAG.
```

```
C:\Users\acrockett\Desktop\CSC\CSC Fall 2018\CSC2400\PROGRAMS\PROGRAM THREE>a

Enter the name of your file that contains the graph vertices:  graph_4.txt


Adjacency Matrix:
0       1       0
0       0       1
1       0       0


Adjacency List...
0--->1--->NULL
1--->2--->NULL
2--->0--->NULL


Now topogocially sorting graph.

Ah shooty pooty.  This graph has a cycle.

The graph can't be topologically sorted because it is not a DAG.
```

```
C:\Users\acrockett\Desktop\CSC\CSC Fall 2018\CSC2400\PROGRAMS\PROGRAM THREE>a

Enter the name of your file that contains the graph vertices:  graph_5.txt

Adjacency Matrix:
0       0       0
1       0       0
0       1       0

Adjacency List...
0--->NULL
1--->0--->NULL
2--->1--->NULL

Now topogocially sorting graph.

The graph vertices topologically sorted are: 2  1  0

C:\Users\acrockett\Desktop\CSC\CSC Fall 2018\CSC2400\PROGRAMS\PROGRAM THREE>
```

WHAT TO TURN IN

Zip ALL the files necessary to run your program **(listed below)** and then upload it to the program 4 assignment folder in ilearn by the due date.

- graph.cpp (source file containing main function)
- GraphMatrix.h (header file containing class that will create the adjacency matrix)
- GraphList.h (header file containing class that will create the adjacency list) – for extra credit only – you do not have to do this part.
- Stack.h (given)
- test_1.txt (given)
- graph_1.txt (given)
- test_2.txt (given)
- graph_2.txt (given)
- test_3.txt (given)
- graph_3.txt (given)
- test_4.txt (given)
- graph_4.txt (given)
- test_5.txt (given)
- graph_5.txt (given)
- Makefile (given) – makefile that I will use to test your program
- runProgram (given) – batch file that I will use to test your program