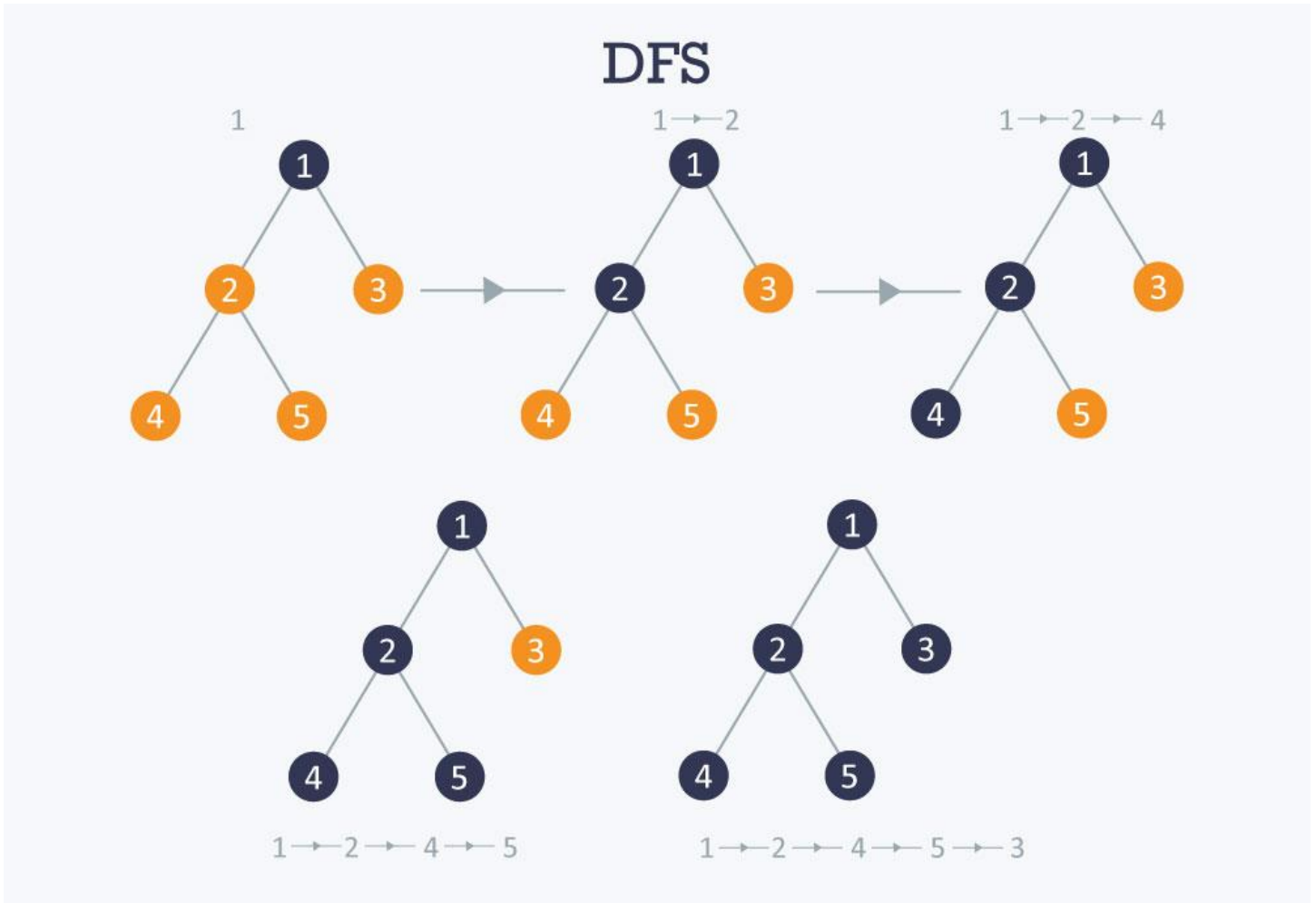
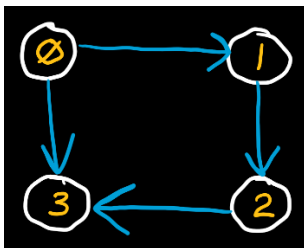


# 1310 PROGRAM 4 – Depth First Search of a Graph



## 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 print the adjacency matrix. Then create an adjacency list and print the adjacency list. Then, print out the vertices using the depth-first search (DFS) algorithm.



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
```

### Adjacency List...

```
0--->1--->3--->NULL
1--->2--->NULL
2--->3--->NULL
3--->NULL
```

Now traversing & printing graph vertices with DFS.

```
0      1      2      3
```

## FILES GIVEN & ASSUMPTIONS YOU CAN MAKE

- You are provided with the Stack class.
- 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 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

## 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)
- 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
- 

## 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 graph information (example: graph\_1.txt).
- Read in the number of vertices from the file and store in a variable. You will use this variable to create your matrix object and then your list object.
- Read each pair of vertices from the file that makes each edge and then add that edge to the matrix. Then add that edge to the adjacency list as well.

- Print the adjacency matrix.
- Print the adjacency list.
- Create a stack object to use in the DFS algorithm.
- You may also want to dynamically create a Boolean array (I called mine visited) which is the size of the number of vertices to keep track of which vertices have been visited.
- Use the stack to implement the Depth-First-Search algorithm. Your solution may either be recursive, or not.
- Print out the vertex when you push it to the stack (meaning it has been visited).

## 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
- **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

### PRIVATE ATTRIBUTES:

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

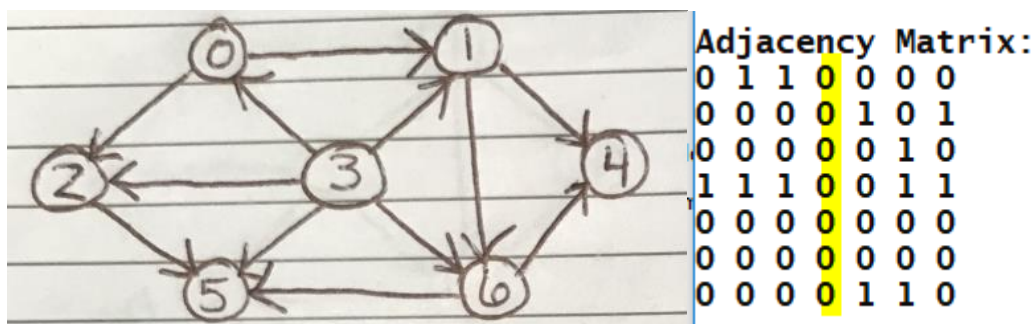
### PUBLIC MEMBER FUNCTIONS:

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

## EXTRA CREDIT OPPORTUNITY (10 POINTS)

### (5 POINTS) STARTING VERTEX

Instead of always starting with zero vertex, determine which vertex to start with so that all nodes are included. In order to get full credit, you must create a function in the Adjacency Matrix class called `getFirstVertex()`, which will check to see if there are any columns in the matrix that are all zeros (meaning that no other vertices have an edge that has a directional arrow pointing to this particular vertex in this column). This function should return the integer vertex that should be the starting point for your DFS algorithm. If there are no columns with all zeros, then the function should return zero. Here is an example of a graph & adjacency matrix for which the first vertex should be 3 instead of zero:



Now traversing & printing graph vertices with DFS.

3            0            1            4            6            5            2

### (5 POINTS) ACYCLIC GRAPH

Determine & indicate if a graph is acyclic (contains a cycle). **You may only implement the acyclic graph feature if you have already implemented the starting vertex feature.** Below is an example of a graph that is acyclic and how your output should look if you implement this extra credit feature. To test, use the given `graph_3.txt` or `graph_4.txt` because they both contain a cycle. The other example graphs do not.

0	1	1	0	0	0	0
0	0	0	0	1	0	1
0	0	0	0	0	0	1
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	1	0	0

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
```

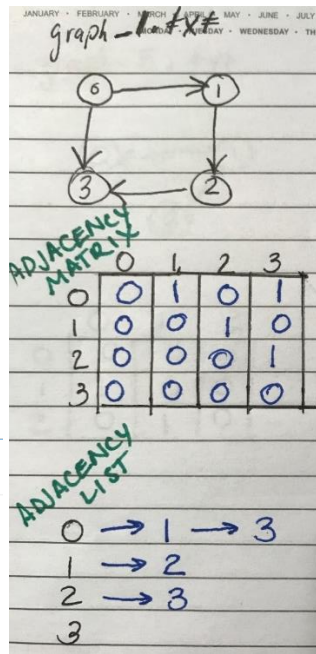
Now traversing & printing graph vertices with DFS.

3            0            1            4            6            5            2

Ah shooty pooty. This graph has a cycle.

### GRAPH TEXT FILES

GRAPH\_1.TXT



graph\_1 - Notepad

File Edit Format View Help

```

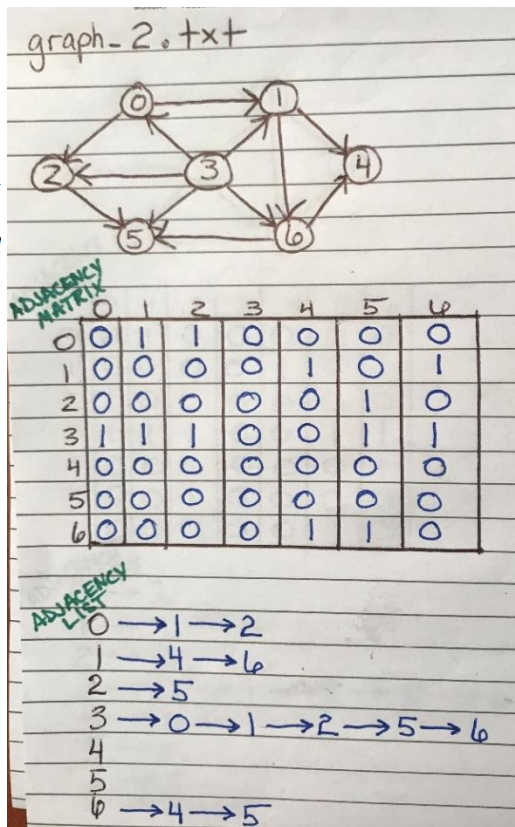
4
0 1
1 2
2 3
0 3

```

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

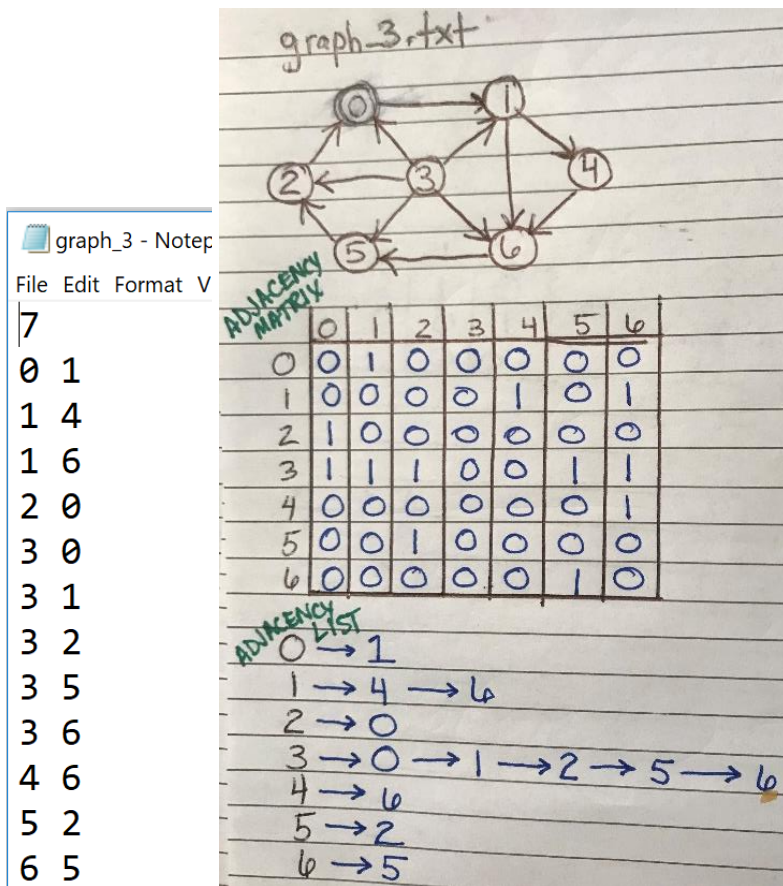
File Edit Format View

```

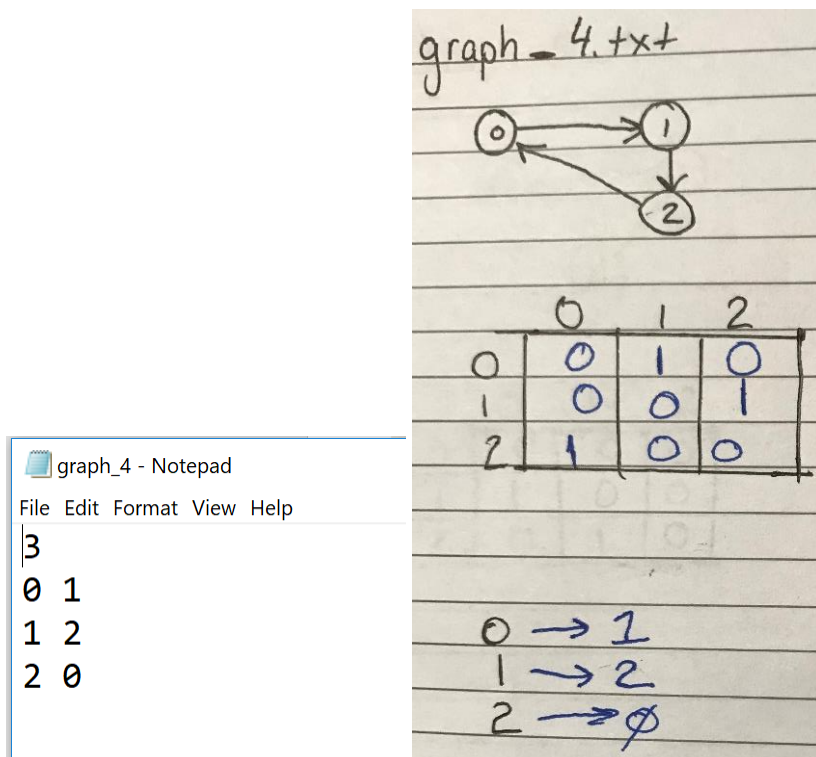
7
0 1
0 2
1 4
1 6
2 5
3 0
3 1
3 2
3 5
3 6
6 4
6 5

```

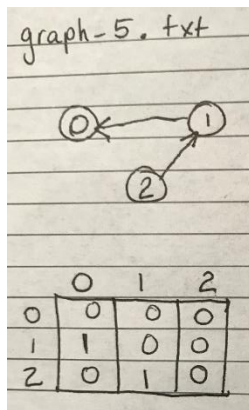
## GRAPH3.TXT (CONTAINS A CYCLE)



#### GRAPH4.TXT (CONTAINS A CYCLE)



#### GRAPH5.TXT



graph\_5 - Notepad

File Edit Format View Help

```
3
2 1
1 0
```

## SAMPLE OUTPUT WITHOUT EXTRA CREDIT OPTIONS

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 traversing & printing graph vertices with DFS.

```
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 traversing & printing graph vertices with DFS.

```
0      1      4      6      5      2
```



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 traversing & printing graph vertices with DFS.

```
0      1      4      6      5      2
```

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 traversing & printing graph vertices with DFS.

```
0      1      2
```

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 traversing & printing graph vertices with DFS.

```
0
```



## SAMPLE OUTPUT (WITH EXTRA CREDIT APPLIED)

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 traversing & printing graph vertices with DFS.

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

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 traversing & printing graph vertices with DFS.

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

Ah shooty pooty. This graph has a cycle.

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 traversing & printing graph vertices with DFS.

```
0      1      2
```

Ah shooty pooty. This graph has a cycle.

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 traversing & printing graph vertices with DFS.

```
2      1      0
```