The Programming Assignment Report Instructions
CSCE 221

1. The description of an assignment problem.
   The purpose of this assignment was to implement and create a DAG (directed acyclic graph) using a edge, vertex and graph class and a file of inputs. The program is to output the graph data structure and the correct order of numbers when performing a depth first search on the input file. The DAG was to be implemented using a combination of vectors, linked lists and multiple classes.

2. The description of data structures and algorithms used to solve the problem.

   (a) Provide definitions of data structures by using Abstract Data Types (ADTs)
   Edge Class

   Edge()//Holds values of a start, end, and weight for each edge.

   Vertex Class

   Vertex()//Holds a value for the label of the vertex.
   void connectTo()//connects a vertex to another vertex in the graph

   Graph Class

   Graph()//Holds a value of the file that the graph is being constructed from
   void buildGraph()//Builds a DAG constructed from the file held in the Graph class constructor
   void displayGraph()//Outputs the DAG constructed from the buildGraph() function to the console.

   Other Functions

   void DFS()//Takes in a graph and uses topological ordering to order the vertices in the DAG using the depth first search method.

(b) Write about the ADTs implementation in C++.
Edge Class

Edge()//Sets the constructor for the edge class to accept three integers for the start, end and weight of each edge.

Vertex Class

Vertex()//Sets the constructor for the vertex class to accept one integer for the label of the vertex.
void connectTo()//Creates a new edge with the vertex calling the function as the beginning of the edge and the integer value set in the function definition as the vertex for the end of the edge. The edges are unweighted so a value of 0 is given for the weights.

Graph Class

Graph()//Sets the constructor for the graph class to hold the ifstream for the file that the DAG will be based off of.
void buildGraph()//The function begins by uisng getline to count the number of lines in the file, which contains how many vertices will be in the DAG, the function then contstructs a vector of vertices containing the given labels by the linecounter. The program then uses a combination of nested while loops, istringstream and getline to parse the file and use the connectTo() function to connect the corresponding vertex to all the vertices that are listed in the same row until the row is terminated by a -1.
void displayGraph()//Uses a for loop to parse through the edgeList of each vertex, popping off the first element of each edgeList and separating the values with an arrow. When an edgeList is empty, the next vertex is output to the console until all vertex's edgeLists are empty.

Other Functions

void DFS()//The function begins by setting all of the timestamps of each vertex to -1 using a for loop. The first vertex is then accessed using a while loop and the closest vertex it is set with an edge to is then set as the new vertex. With each change of vertex the beginning time stamp is set for the vertex. When the first while loop comes into contact with a vertex that contains to edges to point to or a vertex who's sole edge's time stamp has already been marked, the first while loop is broken and the second while loop begins. In this loop, the vertices are treversed backward in order to set the second timestamp for each vertex. When a finish time stamp is set, the data structure is sorted through using nested for loops to find the highest vertex that contains an instance where the previous vertex was set as the end of an edge. If the begin time stamp has yet to be set for a specific vertex, that timestamp is set before the finish timestamp and the vertex must find another vertex that does not have a begin timestamp or else the finish timestamp will be the begin time plus one. When both begin and finish timestamps are not -1, the vertex is pushed into the front of a linked list containing the integer values for all the vertices. This process is repeated in a while loop until the inital vertex is reached. Finally, the linked list is displayed on the console if the number of digits in the list are equivalent to the number of vertices, otherwise a message is displayed to the user that no topological order was found.

(c) Describe algorithms used to solve the problem.
The algorithms in this assignment were implemented using for and while loops. For more information read part 2b on the corresponding function.

(d) Analyze the algorithms according to assignment requirements.
void connectTo()//$O(1)$-no loops in the function
void buildGraph()//$O(e+v)$-where e is an edge and v is a vertex, counts each edge of each vertex.
void displayGraph()//$O(e+v)$-where e is an edge and v is a vertex, counts each edge of each vertex.
void DFS()//$O(e+v)$-where e is an edge and v is a vertex, counts each edge of each vertex.

3. A C++ organization and implementation of the problem solution

   (a) Provide a list and description of classes or interfaces used by a program such as classes used to implement the data structures or exceptions.
   The program used a series of vectors that contained integers and linked lists.

   Vectors-The vector of vertices was used to hold the correpsonding vertex's information as well as all the edges that the vertex pointed to contained within a linked list.
   Linked List-an STD linked list was used to hold the Edges of each vertex. The list had the standard operations of a linked list and contained the start and end of each edge in the corresponding vertex.

   (b) Include in the report the class declarations from a header file (.h) and their implementation from a source file (.cpp).
   On top of including the STD vector class and the Graph, Vertex and Edge .cpp files, sstream and fstream were included as well. sstream was used to parse each row of the input file whereas fstream allowed for ifstream to be used and getline to parse through each row of the file.

   (c) Provide features of the C++ programming paradigms like Inheritance or Polymorphism in case of object oriented programming, or Templates in the case of generic programming used in your implementation.
   This assignment focused on the paradigms of polymorphism and inheritance.

   Polymorphism-The assignment would have not been possible if it were not for the capabilities of the vector and list class. The STD vector and list held values for of the vertex class and edge class type. The ability that the vector class can adapt to the variable type put into it allows for polymorphism to be seen in the assignment.
   Inheritance-The assignment used heavily from the inheritance paradigm. The program was structured to where the edge class was contained in the vertex class and the vertex class was contained in the graph class. The heirarchy of derived classes is the principle that inheritance is based off of. Thus, this assignment would not have been made possible without the paradigm of inheritance.

4. A user guide description how to navigate your program with the instructions how to:

   (a) compile the program: specify the directory and file names, etc.
   Compile the program in the Kaiser-Alexander-A6 using the command make. Make sure the input.data file and the graph, vertex, edge and main header and .cpp files are in the same directory as well.

   (b) run the program: specify the name of an executable file.
   Run the program after compiling the file using the method above using the command ./main input.data.

5. Specifications and description of input and output formats and files

   (a) The type of files: keyboard, text files, etc (if applicable).
   The program inputs a file in given in the same directory as the program that contains muliple numbers per row ending with a -1 to terminate the line.

   (b) A file input format: when a program requires a sequence of input items, specify the number of items per line or a line termination. Provide a sample of a required input format.
   The program uses getline and istringstream in order to parse throughout the files. There are mulitple items per line and getline terminates when it runs out of rows to parse through and istringstream terminates when it runs into a -1. The input format is as follows.

   ./main input.data

   Type the name of the file in the console after ./main to use the file for the DAG

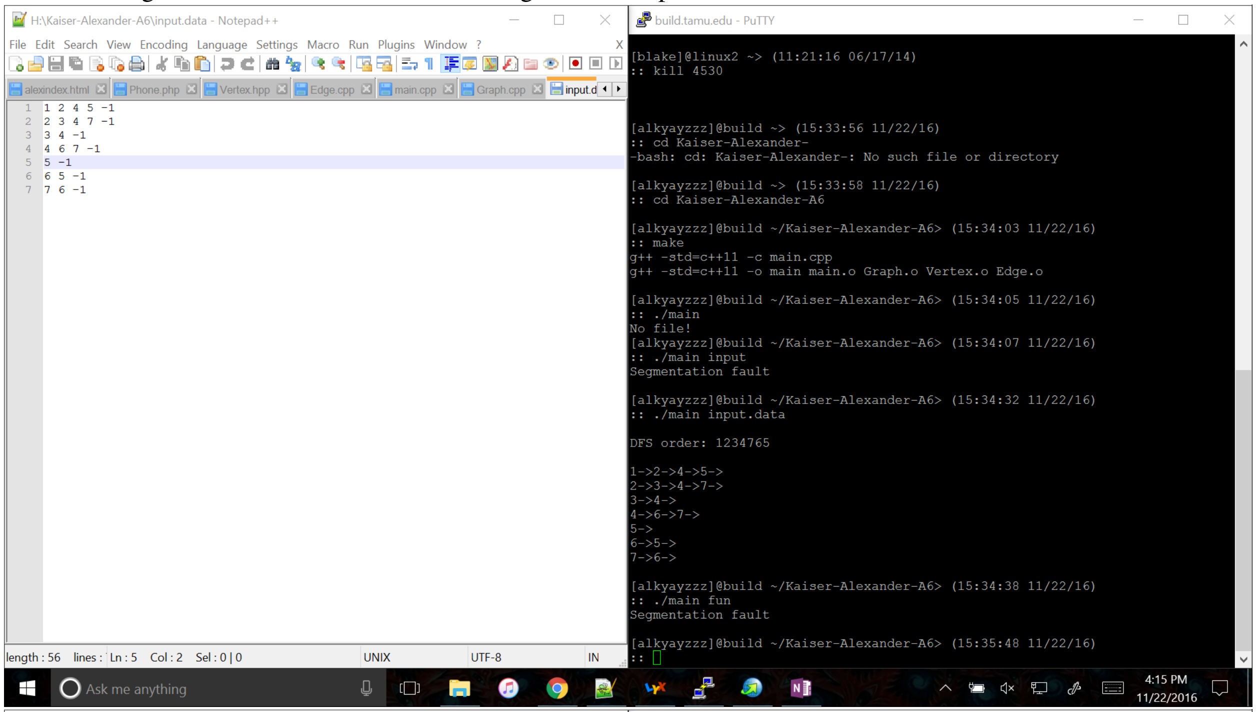
   (c) Discuss possible cases when your program could crash because of incorrect input (a wrong file name, strings instead of a number, or such cases when the program expects 10 items to read and it finds only 9.)
   The only time that the file could crash would involve a misplacement of files in the directory. If the file is not in the same directory as the program, the program will not run and an error will be thrown. Also, if the file given does not correctly implement a DAG, then the program will not function correctly. All other cases of crashing are accounted for with exceptions.

6. Provide types of exceptions and their purpose in your program.

   (a) logical exceptions (such as deletion of an item from an empty container, etc.).
   In order to eliminate logical exceptions in the program, if statements are used to ensure that the data is being handled responsibly. For example, if the user forgot to add a file name to the end of ./main, "No File!" will be output and the program will exit.

   (b) runtime exception (such as division by 0, etc.)
   To prevent runtime exception, a try-catch loop was implemented in the main function. When a runtime exception is encountered (such as a segmentation fault) the program is terminated completely and a reason is given as to why the termination occurred.

7. Test your program for correctness using valid, invalid, and random inputs (e.g., insertion of an item at the beginning, at the end, or at a random place into a sorted vector). Include evidence of your testing, such as an output file or screen shots with an input and the corresponding output.
   The following shows test trials for the DAG using different input files.



H:\Kaiser-Alexander-A6\input.data - Notepad++

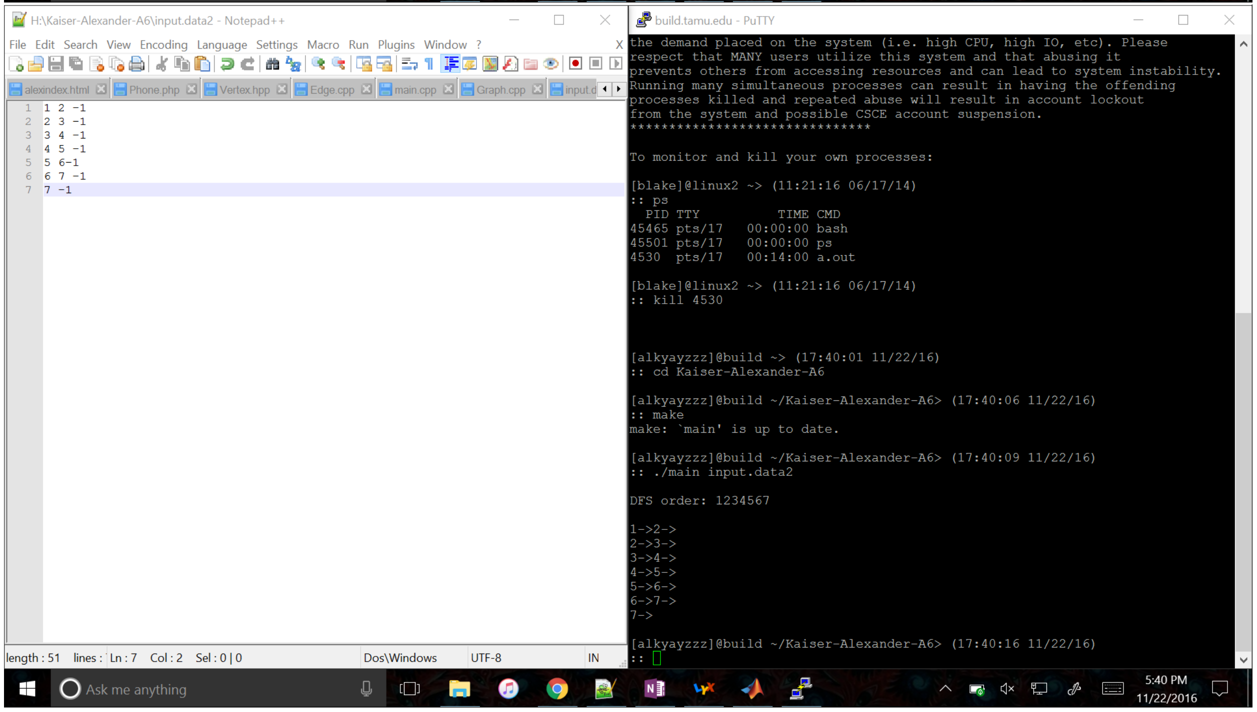
```
1 2 4 5 -1
2 3 4 7 -1
3 4 -1
4 6 7 -1
5 -1
6 5 -1
7 6 -1
```

length : 56   lines : Ln : 5   Col : 2   Sel : 0 | 0   UNIX   UTF-8   IN

build.tamu.edu - PuTTY

```
[blake]@linux2 ~> (11:21:16 06/17/14)
:: kill 4530


[alkyayzzz]@build ~> (15:33:56 11/22/16)
:: cd Kaiser-Alexander-
-bash: cd: Kaiser-Alexander-: No such file or directory

[alkyayzzz]@build ~> (15:33:58 11/22/16)
:: cd Kaiser-Alexander-A6

[alkyayzzz]@build ~/Kaiser-Alexander-A6> (15:34:03 11/22/16)
:: make
g++ -std=c++11 -c main.cpp
g++ -std=c++11 -o main main.o Graph.o Vertex.o Edge.o

[alkyayzzz]@build ~/Kaiser-Alexander-A6> (15:34:05 11/22/16)
:: ./main
No file!
[alkyayzzz]@build ~/Kaiser-Alexander-A6> (15:34:07 11/22/16)
:: ./main input
Segmentation fault

[alkyayzzz]@build ~/Kaiser-Alexander-A6> (15:34:32 11/22/16)
:: ./main input.data

DFS order: 1234765

1->2->4->5->
2->3->4->7->
3->4->
4->6->7->
5->
6->5->
7->6->

[alkyayzzz]@build ~/Kaiser-Alexander-A6> (15:34:38 11/22/16)
:: ./main fun
Segmentation fault

[alkyayzzz]@build ~/Kaiser-Alexander-A6> (15:35:48 11/22/16)
::
```

4:15 PM
11/22/2016



H:\Kaiser-Alexander-A6\input.data2 - Notepad++

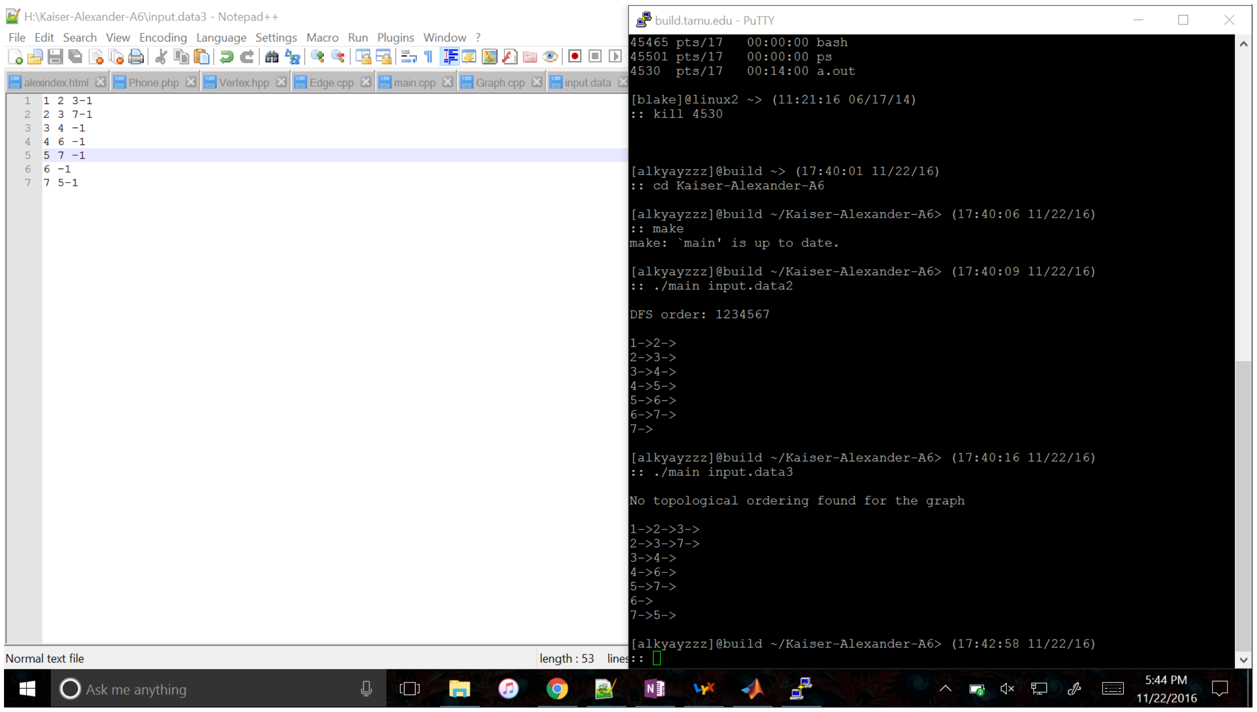
```
1 2 -1
2 3 -1
3 4 -1
4 5 -1
5 6-1
6 7 -1
7 -1
```

length : 51   lines : Ln : 7   Col : 2   Sel : 0 | 0   Dos\Windows   UTF-8   IN

build.tamu.edu - PuTTY

```
the demand placed on the system (i.e. high CPU, high IO, etc). Please
respect that MANY users utilize this system and that abusing it
prevents others from accessing resources and can lead to system instability.
Running many simultaneous processes can result in having the offending
processes killed and repeated abuse will result in account lockout
from the system and possible CSCE account suspension.
********************************

To monitor and kill your own processes:

[blake]@linux2 ~> (11:21:16 06/17/14)
:: ps
  PID TTY          TIME CMD
45465 pts/17   00:00:00 bash
45501 pts/17   00:00:00 ps
4530  pts/17   00:14:00 a.out

[blake]@linux2 ~> (11:21:16 06/17/14)
:: kill 4530


[alkyayzzz]@build ~> (17:40:01 11/22/16)
:: cd Kaiser-Alexander-A6

[alkyayzzz]@build ~/Kaiser-Alexander-A6> (17:40:06 11/22/16)
:: make
make: `main' is up to date.

[alkyayzzz]@build ~/Kaiser-Alexander-A6> (17:40:09 11/22/16)
:: ./main input.data2

DFS order: 1234567

1->2->
2->3->
3->4->
4->5->
5->6->
6->7->
7->

[alkyayzzz]@build ~/Kaiser-Alexander-A6> (17:40:16 11/22/16)
::
```

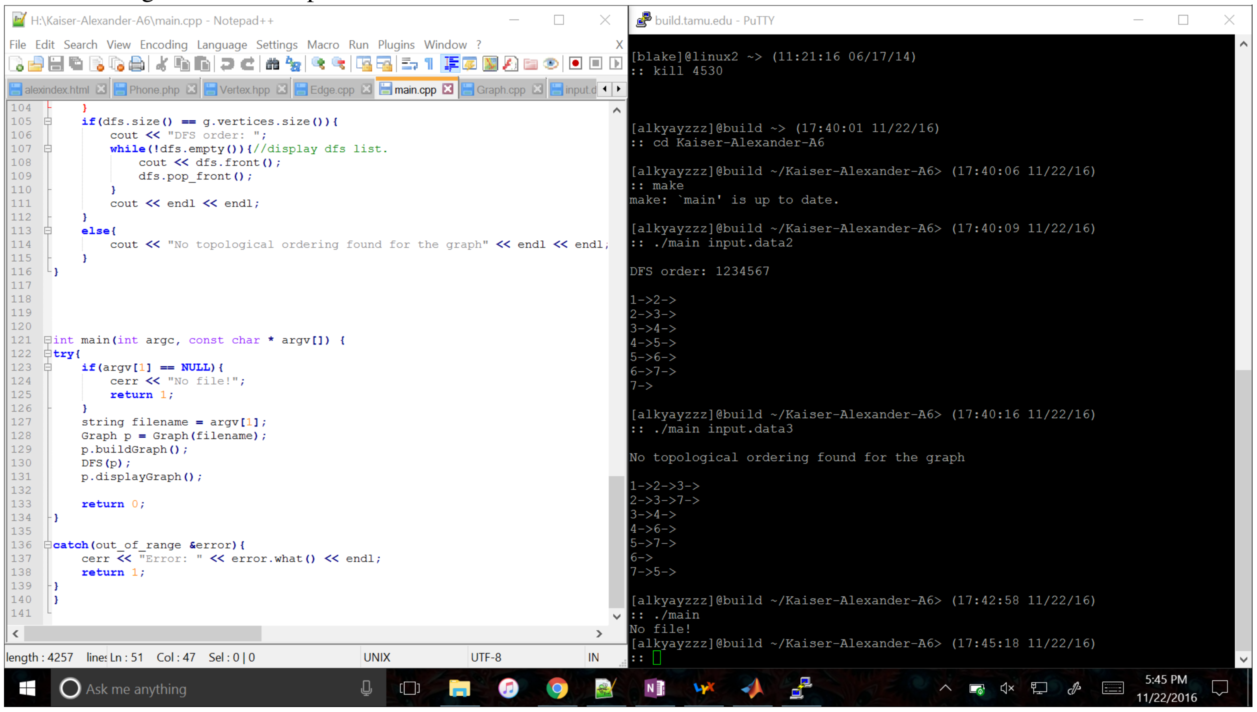
5:40 PM
11/22/2016



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

```
45465 pts/17   00:00:00 bash
45501 pts/17   00:00:00 ps
4530  pts/17   00:14:00 a.out

[blake]@linux2 ~> (11:21:16 06/17/14)
:: kill 4530


[alkyayzzz]@build ~> (17:40:01 11/22/16)
:: cd Kaiser-Alexander-A6

[alkyayzzz]@build ~/Kaiser-Alexander-A6> (17:40:06 11/22/16)
:: make
make: `main' is up to date.

[alkyayzzz]@build ~/Kaiser-Alexander-A6> (17:40:09 11/22/16)
:: ./main input.data2

DFS order: 1234567

1->2->
2->3->
3->4->
4->5->
5->6->
6->7->
7->

[alkyayzzz]@build ~/Kaiser-Alexander-A6> (17:40:16 11/22/16)
:: ./main input.data3

No topological ordering found for the graph

1->2->3->
2->3->7->
3->4->
4->6->
5->7->
6->
7->5->

[alkyayzzz]@build ~/Kaiser-Alexander-A6> (17:42:58 11/22/16)
::
```

The following shows the output for a file that does not exist.



```cpp
    }
    if(dfs.size() == g.vertices.size()){
        cout << "DFS order: ";
        while(!dfs.empty()){//display dfs list.
            cout << dfs.front();
            dfs.pop_front();
        }
        cout << endl << endl;
    }
    else{
        cout << "No topological ordering found for the graph" << endl << endl;
    }
}



int main(int argc, const char * argv[]) {
try{
    if(argv[1] == NULL){
        cerr << "No file!";
        return 1;
    }
    string filename = argv[1];
    Graph p = Graph(filename);
    p.buildGraph();
    DFS(p);
    p.displayGraph();

    return 0;
}

catch(out_of_range &error){
    cerr << "Error: " << error.what() << endl;
    return 1;
}
}
```

```
[blake]@linux2 ~> (11:21:16 06/17/14)
:: kill 4530


[alkyayzzz]@build ~> (17:40:01 11/22/16)
:: cd Kaiser-Alexander-A6

[alkyayzzz]@build ~/Kaiser-Alexander-A6> (17:40:06 11/22/16)
:: make
make: `main' is up to date.

[alkyayzzz]@build ~/Kaiser-Alexander-A6> (17:40:09 11/22/16)
:: ./main input.data2

DFS order: 1234567

1->2->
2->3->
3->4->
4->5->
5->6->
6->7->
7->

[alkyayzzz]@build ~/Kaiser-Alexander-A6> (17:40:16 11/22/16)
:: ./main input.data3

No topological ordering found for the graph

1->2->3->
2->3->7->
3->4->
4->6->
5->7->
6->
7->5->

[alkyayzzz]@build ~/Kaiser-Alexander-A6> (17:42:58 11/22/16)
:: ./main
No file!
[alkyayzzz]@build ~/Kaiser-Alexander-A6> (17:45:18 11/22/16)
::
```

8. Additional Questions.

   1.Describe a real life application where your program can be used.

   One real world application where my program could be used is a map route program like "Google Maps." The program would take into account all the roads that are connected across the US and calculate which path would take the user through the destinations that must be visited up to the final destination and back. If weights were added to the assignment, the shortest path could be calculated. This means that a shortest path could be calculated on the mapping software that would take the shortest distance to get to and from the destination the user wants to reach.