CSCE 221 Cover Page
Final Project

First Name Jonathan Last Name Westerfield UIN 224005649

User Name jgwesterfield E-mail address jgwesterfield@gmail.com

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: Aggie Honor System Office

Type of sources	Online	Lecture Slides
People		
Web pages (provide URL)	The Crazy Programmer http://www.thecrazyprogrammer.com	
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 Jonathan Westerfield Date 7/9/2017

Part I

How to Compile and Run

1. To compile, use <g++ -std=c++11 *.cpp -o main>.
2. To run, use <./main>.
3. The program will ask for a particular input file to run. There are 2 input files:
   - The provided input file: "inputMaze.txt".
   - The input file I created: "inputMaze2.txt".
   - WARNING: YOU MAY HAVE TO USE THE ABSOLUTE FILE PATH OF THE FILES TO MAKE IT FUNCTION PROPERLY!!!
4. From there, the menu will pop up. To choose items from the menu, input 1, 2, 3, or 4.
   - Choice 1 will output the adjacency matrix.
   - Choice 2 will print out the length of the path.
   - Choice 3 will print out the rooms traveled along the path to exit the maze.
   - Choice 4 will terminate the program.
5. The output file will automatically be generated upon typing in the input file name.

Part II

Data Strucutres Used to Represent the Graph and Graph Operations

First the input of the graph was read in a format of 4 digits. Each digit in the input represented a door for North, East, South, West respectively. Ex. -1001. The example indicates that there was an opening for this node to the North and West. After the input was read in from an input file, the information was put into an adjacency matrix. This was because an adjacency matrix was easier to implement. The adjacency matrix is made up of a 2 dimensional vector due to how flexible a vector was over using an array. For the graph operations (traversing and finding the exit), a depth first search based on a stack was used, however, the stack was implemented using a vector instead of the stack from the standard library. To print out the path, the contents of the stack (vector) was printed out. To print out the maze after it had been solved, a 2 dimensional

vector of chars was compared to the vector of the path. If an element from the path stack matched the position in the 2 dimensional char vector, the location in that 2d vector was outputted as an 'O', else it was outputted as an 'X'.

Part III

My Algorithm and its Time Complexity

1. My Code:

```
bool myGraph::depthFirstSearch(int i) // the starting node
{

   // gives the length of the path that it finds
   ++pathLength;
   // counts the number of times this runs
   int j;
   pathRoomNumbers.push_back(i);
   visited[i] = 1;
   printf("\n%d",i);
   if(i == n - 1)
   {

      return true; // means the end of the maze was found
   }
   bool found = false;
   for(j = 0; j < n; j++)
   {

      if(!visited[j] && adjMatrix[i][j] == 1)
      {

        if(depthFirstSearch(j))
        {
          found = true;
          return found;
        }
      }
```

```
    }
    if(!found)
    {
        pathRoomNumbers.pop_back();
    }
    return found;
}
```

2. My implementation is a recursive implementation of the Depth First Search algorithm. The algorithm works by traversing the adjacency matrix. The vector "pathRoomNumbers" is a vector that behaves like a stack. The room that we are currently visiting is pushed onto the stack to keep track of where we are. We then use the "visited" vector in order to check that we have visited this particular vertex. If we have reached the end of the maze $(n-1)$, then we exit the function and return, which propogates back up the recursion. We then use a for loop to traverse the adjacency matrix. We make sure that the vertex we are in hasn't been visited and go to it. The depthFirstSearch function is then called recursively until the end of the maze is found. The next part allows us to account for dead ends in the maze. If there is a dead end, the pathRoomNumbers vector (implemented like a stack) pops off the last element. Since this is performed recursively, elements will continue to be popped off the vector until a new potential path the end is found. When the function is completely finished, the pathRoomNumbers will only have the rooms that make up a successful path. The length of the path can be found by simply finding the length of the pathRoomNumbers vector.

3. I initially used a depth first search alogorithm in order to find a path out of the maze. This was the simplest to implement and the most direct since its job is to find AN exit path. Its only shortcoming is that it doesn't find all of the exit paths and store which path was the shortest. However, to make up for this, the algorithm is simpler than a breadth first search implementation. The algorithm works by using the adjacency matrix and visting every single node in the maze. It traverses the maze until it finds the exit of the maze. Because of this implementation, the run time of the algorithm is $O(n^2)$. While the maze may be of size 16, a 4x4 maze, the adjacency list is of size n^2. This is the reason that the runtime is $O(n^2)$ because we traverse then entire matrix to find a path.

Part IV
Test Cases for Verification

1. For testing, I used the maze that was already given to us and a maze that I constructed on my own. Unlike the maze provided to us, the maze I

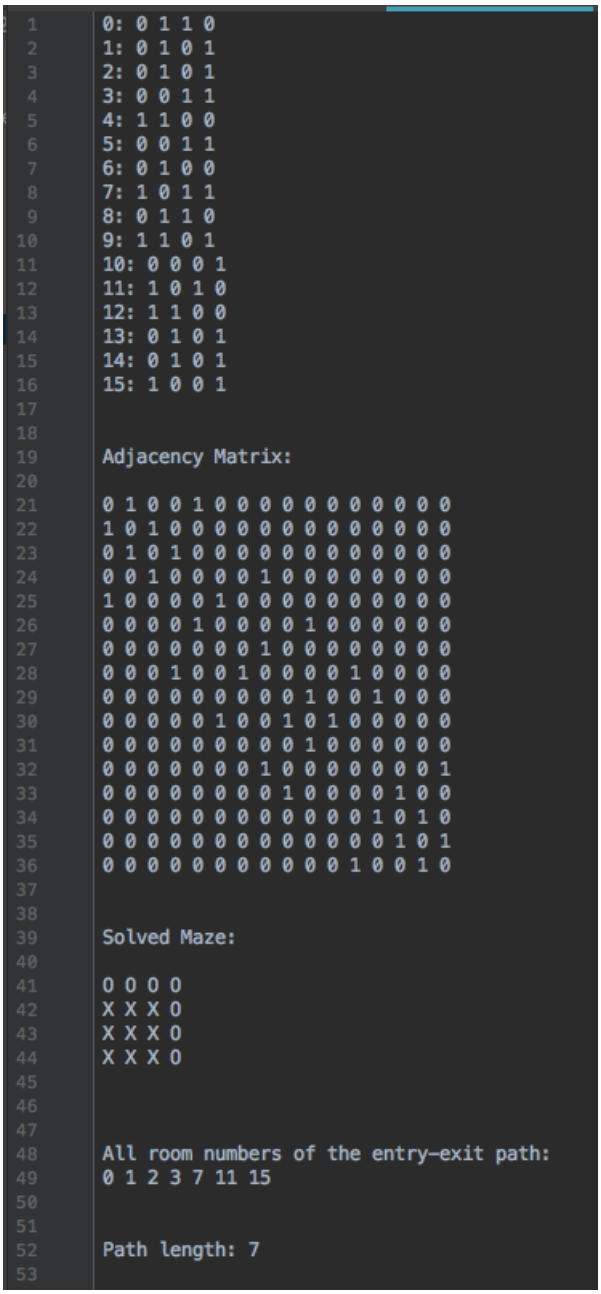
created has 2 exit paths of different lengths. My algorithm was able to find a path through the maze. The output file for the given maze is here:

(a)

```
0: 1 1 0 0
1: 0 1 1 1
2: 0 1 1 1
3: 0 0 0 1
4: 0 1 0 0
5: 1 0 1 1
6: 1 0 1 0
7: 0 0 1 0
8: 0 1 1 0
9: 1 0 0 1
10: 1 1 0 0
11: 1 0 0 1
12: 1 1 0 0
13: 0 1 0 1
14: 0 1 0 1
15: 0 0 0 1


Adjacency Matrix:

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


Solved Maze:

O O X X
X O X X
O O X X
O O O O



All room numbers of the entry-exit path:
0 1 5 9 8 12 13 14 15


Path length: 9

```

2. The output file for the second maze (my maze) is here:

(a)



3. The output for the program using the provided maze (with the menu) is here:

(a)

```
Make your choice

(1) Print out Adjacency Matrix

(2) Print length of the entry-exit path

(3) Print out all room numbers on the entry exit path

(4) EXIT PROGRAM

Please choose what you want (1, 2, 3): 1

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


Make your choice

(1) Print out Adjacency Matrix

(2) Print length of the entry-exit path

(3) Print out all room numbers on the entry exit path

(4) EXIT PROGRAM

Please choose what you want (1, 2, 3): 2

Entry-exit path length: 9


Make your choice

(1) Print out Adjacency Matrix

(2) Print length of the entry-exit path

(3) Print out all room numbers on the entry exit path

(4) EXIT PROGRAM

Please choose what you want (1, 2, 3): 3

All room numbers of the entry-exit path:
0 1 5 9 8 12 13 14 15


Make your choice

(1) Print out Adjacency Matrix

(2) Print length of the entry-exit path

(3) Print out all room numbers on the entry exit path

(4) EXIT PROGRAM

Please choose what you want (1, 2, 3): 4

EXITING PROGRAM
```

4. The output for the program using the maze I created (with the menu) is here:

(a)

```
(1) Print out Adjacency Matrix

(2) Print length of the entry-exit path

(3) Print out all room numbers on the entry exit path

(4) EXIT PROGRAM

Please choose what you want (1, 2, 3): 1

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


Make your choice

(1) Print out Adjacency Matrix

(2) Print length of the entry-exit path

(3) Print out all room numbers on the entry exit path

(4) EXIT PROGRAM

Please choose what you want (1, 2, 3): 2

Entry-exit path length: 7


Make your choice

(1) Print out Adjacency Matrix

(2) Print length of the entry-exit path

(3) Print out all room numbers on the entry exit path

(4) EXIT PROGRAM

Please choose what you want (1, 2, 3): 3

All room numbers of the entry-exit path:
0 1 2 3 7 11 15


Make your choice

(1) Print out Adjacency Matrix

(2) Print length of the entry-exit path

(3) Print out all room numbers on the entry exit path

(4) EXIT PROGRAM

Please choose what you want (1, 2, 3): 4

EXITING PROGRAM

Process finished with exit code 0
```

5. Output Verification:

(a) As you can see from the output file for maze 1 (the provided maze), the output is identical to the specification listed in the instructions. Also since there was only a single path to the exit, the fact that the algorithm made it to the end without going outside of the maze.

(b) For the maze I provided, the algorithm made it to the end of that as well. While the algorithm was able to find the shortest path out of the maze I created, it still cannot find the shortest path. The reason it found it in this maze was due to the way the algorithm interacted with the adjacency list. The shortest path happened to be the first choice when choosing a path and since it made it to the end, there was no reason to go down the other path in the maze.

(c) The path lengths are also correct for both mazes due to the way it is calculated. The rooms that are along the path (including the starting room) are added to a stack (using a vector). To find the length of the path, all thats needed is to count the number of elements on the stack.

(d) Another way to verify if the path that is found is correct, it is also possible to actaully draw out the maze to see what paths were actaully possible. I did this to create the maze I used to test the algorithms correctness.