CSCE 221 Project

August 6, 2017

Due August 7th

Laura Austin UIN 524006473

User Name: laustin254 E-mail address: laustin254@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, read more: Aggie Honor System Office

Type of sources		
People	Callen McCauley Aaron Ingram	
Web pages (provide URL)	reading files creating text file	error checking depth first search
Printed material	"*Discrete Structures*"	
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 *Laura* *Austin* Date 7/11/2017

1. Description of the assignment problem:

The assignment for this project was to design an algorithm to find the path from entrance to the exit of any given maze that consists of rooms connected by doors. This maze will consist of doors labeled0 through$n^2 - 1$. The entry to the maze starts at room zero and ends at room $n^2 - 1$. This maze is an undirected graph, and rooms represent the vertices or nodes, the edges are doors, or pathways. The output required is a printed adjacency list, length of the entry-exit path, and the vertices/room numbers on the entry-path exit.

2. Description of data structures and algorithms used in this problem:

(a) The definition of the data structures used:

vector ADT: Referring to the lecture slides, this data type extends the notion of array by storing a sequence of objects, and my objects for this project are all type ints. In vector data type, an element can be accessed, inserted, removed by calling it's index. Elements can be added without having to allocate new memory, like in regular arrays. That was the biggest benefit of using vectors in this project, since there will be unknown test files by the user. It is much easier to let the data type increase with the size that the user inputs rather than having to account for the possible sizes that a user can input.

I used both ordinary vectors, as well as a vector of vectors. A vector of vectors specifically used fo building the adjacency matrix.

(b) The vector ADT implementation in C++:

Abstract Data Structure's implementation in C++ is recognized as a class, according to slide 6 out of 27 on lecture notes "Data Structures & Algorithms." Furthermore, the implementations are containers, which include the sequences, under which vector is classified.

(c) The algorithms used to solve the maze:

The main algorithm I implemented was Depth-First Search (DFS). I used DFS since it was the simplest for me to understand while meeting the criteria of the assignment. Using DFS, there will always be a path printed out to reach the end of the maze. The green comments on the right side are the psuedo code which describe the C++ implementation of my DFS on the left.

```
// ....................DFS.................................
vector<int> findpath(int nodes, vector<vector<int>>& vec){
    cout <<  endl;
    cout << "The path from room 0 to room 15 is: " << endl <<endl;
    vector<int> vectori;                        // check if visited
    vector<int> path;                           // put path
    int j = 0;
    int i = 0;
    path.push_back(0);                          // initialize the first node
    vectori.push_back(0);                       // debugging vector (ignore)
    while ((i != nodes-1)){                     // This while loop implements DFS
        if (vec[i][j] == 1){                    // if there is an edge
            if (finde(path, j) == true){        // if edge is already been visited
                j++;                            // go to next number in adj list
            }
            else if (finde(path, j) == false){  // if edge has not been visited
                vectori.push_back(i);           //
                path.push_back(j);              // add edge to path
                i = j;                          // set edge equal to i
                j = 0;                          // reset edge back to zero for more
            }                                   // checking
        }
        else if ((j == nodes)){                 // if edge = 15 rooms
            j = 1 + path[path.size()-1];        // set edge to the second to last index in path
            path.pop_back();                    // remove the last index (edge) in path
            i = path[path.size()-1];            // set i to the last index in path
            vectori.pop_back();                 // debugging vector (ignore)
        }
        else j++;                               // if vec[i][j] is 0, go to next edge value

    }
```

(d) Since I use a vector of vectors, and each vector must be traversed for every room n, and each vector contains n containers, then the worst case or Big-O of my algorithm is $O(n^2)$. The best case, which is taught in class, $O(V + E)$, where V is vertices (or nodes) and E is edges (or doorways). Ultimately, in my implementation, the matrix must be built, which takes $O(n^2)$ time. So, for each run of my .cpp file, the time complexity is $O(n^2) + O(V + E)$.

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

(a) Description of classes or interfaces used:

I used no classes, and only used 3 functions and an int main( ) to implement them. There is a boolean function, finde(vector<int> vec, int e), which returns true if the element being searched IS in the path vector, or returns false if the element being searched is NOT in the path vector. This is implemented in all the other two functions because it is necessary for keeping track of visited nodes and conditions in which the algorithm should back-track.

Any discussion of exceptions are done in part 6 of this report.

There is a function which returns a vector of vectors, printpath(int nodes, vector<vector<int>> mazey, const vector<int> &path), which returns a maze. Here's an example of the output:

```
The path is:

o o x x
x o x x
o o x x
o o o o
```

Where o's represent rooms, and x's represent anything otherwise (as specified on the assignment sheet).

The other function which was mentioned in problem 2 part c, was the DFS algorithm, findpath(int nodes, vector<vector<int>>& vec). It returns a vector, which is very important because in order to print to a text file, I needed to obtain that vector a few times within the int main( ). Because I have it returning the vector that contains the pathway, I can use the STL vector function size() to get the length of the pathway, which was a criteria on the assignment.

(b) Class declarations (if any)

There are no class declarations, as I do not implement a class. Everything that must be included from STL is included at the top of the maze.cpp file:

#include<vector>
#include<iostream>
#include<fstream>
#include<sstream>
#include<string>
#include<cmath>
#include<algorithm>

(c) Features of C++ programming paradigms:

I am not using templates, as this is not a generic program, it only implements type ints in the vectors.

In terms of object oriented programming, I do use a function that is implemented in other functions, and hence characteristics used, which is a part of the definition of object oriented programming.

4. User guide description how to navigate the program:

(a) Compiling the program:

The folder that the maze.cpp will be contained in is Austin_Laura_Proj. It will contain maze.cpp, test1.txt, test2.txt, and this pdf file. In order for the program to properly take the users text file, let's call it "userfile.txt", userfile.txt must be in Austin_Laura_Proj folder. Otherwise, the program will not produce the desired output.

Once the userfile.txt is in Austin_Laura_Proj, the user can get to the folder by typing in putty,

cd Austin_Laura_Proj

then to compile the user can type

g++ -std=c++11 maze.cpp

and press enter. The program should compile if all the files are within the same folder.

(b) Running the program:

In the putty command line, the user must type

./a.out main.cpp

For test case 1, the user will be prompted to type in a filename. Simply type:

test1.txt

and for test case 2, similarly type

test2.txt

Then the screen will indicate the file was successfully open, and the MENU should pop up.

The users options are to type either **1**, **2**, **3**, or **4**.

If the user chooses **1**, **2**, **3**, then the program will do the specified function, and then re-print the MENU requesting for another input. This will happen until the user types **4**, which will end the program.

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

(a) The type of files used are an input text file, only one .cpp file, and an outputted text file called "mazedata.txt".

The test1.txt contains:

16
0 1 0 0
0 1 1 1
0 1 1 1
0 0 0 1
0 1 0 0
1 0 1 1
1 0 1 0
0 0 1 0
0 1 1 0
1 0 0 1
1 1 0 0
1 0 0 1
1 1 0 0
0 1 0 1
0 1 0 1
0 0 0 1

and test2.txt contains:
16
0 0 1 0
0 1 1 0
0 1 1 1
0 0 0 1
1 1 0 0
1 0 0 1
1 1 1 0
0 0 0 1
0 1 0 0
0 1 1 1
1 1 0 1
0 0 0 1
0 1 0 0
1 1 0 1
0 1 0 1
0 0 0 1

(b) The input file format will be in the form "filename.txt", for example, in the putty cmd prompt:

Enter the file name (e.g.: filename.txt): **test1.txt**

and then the user would press "enter" on their keyboard, and the menu would pop up thereafter, assuming their input file is found and that the user's input number of nodes is a perfect square.

(c) The possible cases in which my program will end:

If the menu pops up, and the user types in a type char instead of a type int, then the program will end. I tried to do a comparison like this:

```
//_____________________________________________________________
if ((typeid(state) == typeid(char)) || state>4 || state <0){
    cout << "Woops! Type an integer in 1-4\n";
}
```

But the comparison still doesn't work, so the program will crash if the user types an "s" for example, instead of a "1".

6. Types of exceptions and their purpose:

Some exceptions I included was

```
catch (exception& e)
{
    cerr << "error: " << e.what() << '\n';
    return 1; // 1 indicates failure
}
catch(...)
{
    cerr<< "Oops: unknown exception!\n";
    return 2; // 2 indicates failure
}
```

Otherwise, most of my possible errors I took care of within the functions or the main. Here is the list of them:

- if the inflile fails, if (infile.fail()) will catch it and require the user to re-input a file. This prevents a nonexistent file from being called, causing a crash.
- if the infile does not fail, but the node number is not a perfect square, then the program will require the user to once more re-input a file, until the file meets the conditions at which maze can be computed.
- There is an exit function in the menu, the user has to type 4 for it.
- If the user types a state that is less than 1 or greater than 4, the program prints "Woops! Type an integer in 1-4", and prints the menu again.
  - however, I go into detail in number 5 part C about how I tried to prevent an error if a user inputted a char, or basically any typeid that is not int. I kept the code within my .cpp, but unfortunately the program will end if the user types a char in the menu.

7. Examples of output in Putty and the text file output:
checking for errors like a non-perfect number of rooms:

```
[laustin254]@linux2 ~/maze> (17:49:03 08/06/17)
:: ./a.out
Enter the file name (e.g.: filename.txt): te
input error. Enter the file name (e.g.: filename.txt):
test1.txt
Room number is not a perfect square.
Enter the file name with a perfect square room count (e.g.: filename.txt):
test2.txt
File successfully open
............MENU.............
 1 - Print The Adjacency List/Matrix for graph
 2 - Find the Length of the Entry-Exit path
 3 - Print Out Room Numbers on Entry-Exit path
 4 - Exit
1
```

checking for errors such as a mistaken file name:

```
[laustin254]@linux2 ~/maze> (21:49:55 08/06/17)
:: ./a.out main.cpp
Enter the file name (e.g.: filename.txt): WRONGFILENAME
input error. Enter the file name (e.g.: filename.txt):
test.blah
input error. Enter the file name (e.g.: filename.txt):
test.tsx
input error. Enter the file name (e.g.: filename.txt):
test1.txt
File successfully open
............MENU.............
 1 - Print The Adjacency List/Matrix for graph
 2 - Find the Length of the Entry-Exit path
 3 - Print Out Room Numbers on Entry-Exit path
 4 - Exit
```

Option 1 on test file test2.txt:

```
test2.txt
File successfully open
............MENU.............
 1 - Print The Adjacency List/Matrix for graph
 2 - Find the Length of the Entry-Exit path
 3 - Print Out Room Numbers on Entry-Exit path
 4 - Exit
1

The adjacency matrix of 16 nodes is:

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 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
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 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 0 0 0 1 0 1 0 0 1 0 0
0 0 0 0 0 0 1 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 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 1 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
```

Options 2 and 3 on test2.txt:

```
............MENU.............
 1 - Print The Adjacency List/Matrix for graph
 2 - Find the Length of the Entry-Exit path
 3 - Print Out Room Numbers on Entry-Exit path
 4 - Exit
2

The path from room 0 to room 15 is:

0 4 5 1 2 6 10 9 13 14 15

Length of Entry-Exit path is 11.

............MENU.............
 1 - Print The Adjacency List/Matrix for graph
 2 - Find the Length of the Entry-Exit path
 3 - Print Out Room Numbers on Entry-Exit path
 4 - Exit
3

The path from room 0 to room 15 is:

0 4 5 1 2 6 10 9 13 14 15

Length of Entry-Exit path is 11.

The path is:

o o o x
o o o x
x o o x
x o o o
............MENU.............
 1 - Print The Adjacency List/Matrix for graph
 2 - Find the Length of the Entry-Exit path
 3 - Print Out Room Numbers on Entry-Exit path
 4 - Exit
4
```

And a screenshot of what mazedata.txt will look like for test1.txt :

```
The entered text file:
0100
0111
0111
0001
0100
1011
1010
0010
0110
1001
1100
1001
1100
0101
0101
0001

The adjacency matrix of 16 nodes is:

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

Path is:
0 1 5 9 8 12 13 14 15
Length of path is: 9
The path is:

o o x x
x o x x
o o x x
o o o o
```

And a screenshot for what mazedata.txt will look like for test2.txt:

maze.cpp | test1.txt | test2.txt | mazedata.txt

```
The entered text file:
0010
0110
0111
0001
1100
1001
1110
0001
0100
0111
1101
0001
0100
1101
0101
0001

The adjacency matrix of 16 nodes is:

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

Path is:
0 4 5 1 2 6 10 9 13 14 15
Length of path is: 11
The path is:

o o o x
o o o x
x o o x
x o o o
```