CSCI 2270 – Data Structures

Instructor: Asa Ashraf

Assignment 8 - Graph

Due on Sunday, Nov 7 2021, 11:59PM

OBJECTIVES

1. **Applications of Depth First Traversal**

Overview

In this assignment, you will apply DFT for finding a path through a maze.

Graph Class

Your code should implement depth first traversal to search for a path through a maze starting at a source node (0, 0) and ending at a destination node (n-1, n-1). A header file that lays out this maze can be found in **Maze.hpp** on Canvas. *As usual, do not modify the header file. You may implement helper functions in your .cpp file if you want as long as you don't add those functions to the Maze class.*

Your maze will utilize the following struct:

```
struct vertex;

struct adjVertex{
    vertex *v;
};

struct vertex{
    int vertexNum;
    bool visited = false;
    vector<adjVertex> adj;

};
```

Consider a maze of 0s and 1s, where 0 indicates a possibility of a path, and 1 indicates a wall, and we are trying to search for a path starting from (x = 0, y=0) to (x = 4, y = 4).

x\y	0	1	2	3	4
0	0	1	1	0	1
1	1	0	0	0	1
2	1	0	1	0	0
3	0	1	0	1	1
4	1	0	1	0	0

This maze will be represented using a graph of 13 nodes (13 zeros or possibilities of path), where each node is numbered based on the position of the node in the maze (y + n*x) where n = number rows/columns in the maze. The following grid, numbers each node based on its position (For example, (2, 3) is 3+5*2 = 13). The walls are highlighted blue, and the open positions are highlighted red.

x\y	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14
3	15	16	17	18	19
4	20	21	22	23	24

This maze in its graph form is represented as following adjacency list:

0 → [6]
3 → [7, 8]
6 → [0, 7, 11]
7 → [3, 6, 8, 11, 13]
8 → [3, 7, 13, 14]
11 → [6, 7, 15, 17]
13 → [7, 8, 14, 17]
14 → [8, 13]
15 → [11, 21]
17 → [11, 13, 21, 23]
21 → [15, 17]
23 → [17, 24]
24 → [23]

The path highlighted in red is one potential path from 0 to 24:

0 → 6 → 7 → 8 → 13 → 17 → 23 → 24

Member Functions

int createDefaultMaze();**

- ➔ Using the private member **n** of the Maze class, creates a default maze of all 1s of size n x n, except for positions (0,0) and (n-1, n-1).
- ➔ Note: Think of int** as an array of arrays or an array of integer pointers (int*).
    - Allocate memory required by **n** `int*`, and
    - For each `int*` allocate memory required by **n** integers.
    - This results in a 2D array used to store 0s, and 1s for the maze.

void createPath(int i, int j);

- ➔ Create an open path at position (x = i, y = j) by inserting a 0 at that position in the maze.

void printMaze();

- ➔ Display the maze

Format for printing:

If we create a maze with the following structure

```
maze.createDefaultMaze(3);
maze.createPath(0, 0);
maze.createPath(1, 1);
maze.createPath(2, 2);
```

0	1	1
1	0	1
1	1	0

We print the maze in the following manner with spaces and pipes in between elements.

```
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
```

int findVertexNumFromPosition(int x, int y);

- ➔ Use the private member of class **n** to return the vertex number using the formula `y + num_rows_cols_in_maze * x`

void addVertex(int num);

- ➔ Add a new vertex with the given number to the graph.

void addEdge(int v1, int v2);

- ➔ Add an edge between from v1 to v2, and from v2 to v1 if the edge doesn't already exist.

void displayEdges();

➔ Display all the edges in the graph.

Format for printing:

Consider we create a graph with the following structure from the maze on the right:

```
graph.addVertex(0);
graph.addVertex(4);
graph.addVertex(8);

graph.addEdge(0, 4);
graph.addEdge(4, 8);
```

```
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
```

We print the edges in the following manner.

```
0 --> 4
4 --> 0 8
8 --> 4
```

vector<int> findOpenAdjacentPaths(int x, int y); `// provided to you`

➔ For a given position (x, y) in the maze, the function returns a vector of vertex numbers of all the open path positions in all 8 directions (North, South, East, West, North-West, North-East, South-West, and South-East).

void convertMazeToAdjacencyListGraph();

➔ For each position x,y in the maze, if the position is an open path (not a wall),

- find its vertex number in the graph
- find its adjacent vertices by checking for open paths in all 8 directions in the maze. Please use the provided helper function **findOpenAdjacentPaths.**
- Add the vertex, and its adjacent vertices to the graph.
- Add the edges between the vertex and its adjacent vertices to the graph.

bool checkIfValidPath();

➔ Check if the private member `vector<int> path` is a valid path:

  - First vertex must be 0
  - Last vertex must be (n^2 -1)
  - Every vertex must be an adjacent vertex to the previous vertex in the vector.
- ➔ If it is, return `true`, otherwise, return `false`.

bool findPathThroughMaze();

- ➔ This method returns True if it found a valid path through the maze, else it returns False.
- ➔ It also populates the private member **`path`** vector with vertex numbers from source to the destination.
- ➔ Implement Depth First Traversal on the graph using recursion.
- ➔ While performing depth first traversal on the graph, print information every time you reach a new vertex while exploring a path, and every backtracking step in the traversal using the following format on the next page.

```
// for the starting position (0, 0)
cout << "Starting at vertex: 0" << endl;

// when you reach a new position
cout << "Reached vertex: " << v->num << endl;

// when you backtrack to a previous position
cout << "Backtracked to vertex: " << v->num << endl;
```

Please note that once you are done with your assignment on code runner you need to click on 'finish attempt' and then 'submit all and finish'. If you don't do this, your submission will not be graded.