**FINDING PATH IN A MAZE BY DIFFERENT PATHFINDING ALGORITHMS(GRAPH THEORY)**

****

**Student Name: RAHUL SINGHAL**

**Student Enrolment No: 18103316**

**Subject Name:** Data Structure Lab

**Batch:** B10

**Semester: 3rd**

**Academic Session:** 2019 ODD

Maze Pathfinder

A C++ implementation of a Maze data structure using graphs with four pathfinding algorithms. The implementation of the data structure aims to be memory efficient. The code is designed to be easily adaptable to read and solve mazes of any format (for example from a file). Four pathfinding algorithms are implemented: Depth First Search, Breadth First Search, Dijkstra and A\*.

STRUCTURE OF CODE

How is the maze built?

The Maze class aims to minimize the number of Vertexes needed to store the maze. Vertexes are built only at cells where path decisions need to take place to reduces the number of iterations using any pathfinding algorithm. Between Vertexes exist Edges. Each Vertex has a maximum of four Edges. Edges store the Vertex they lead to and the distance to the Vertex. In the case where a Vertex has fewer than four Edges, Edges lead to a nullptr. Each Vertex will have at least one edge that points to another Vertex.

Building the Maze

You will need to build a class that inherits the IMazeReader class to read the maze. The IMazeReader class has two virtual functions as follows:

virtual void isVertex(int row, int column, bool & top, bool & down, bool & left, bool & right, bool & isGoal) = 0;

The function isVertex takes in a row and column positions of a cell in your maze as arguments along with four Booleans that represent the direction of movement that can occur at the cell and the Boolean isGoal representing the start or end points.

Using the maze above as example, at row zero column zero, (numbering starts from zero) you can move only downward and hence down would be set to true and other directions set to false. If the position at row zero column zero is your starting or ending point, isGoal additionally would be set to true as well. By default, all Boolean passed are set to false.

virtual void setSize(size\_t & row, size\_t & col) = 0;

The function should set row and column size of given your maze.

In the Project Files folder, you can find MazeStringReader Class that inherits the IMazeReader Class that reads a maze from a file as a string and implements the above functionality.

To build your maze, simply pass your IMazeReader as a parameter to Maze::setMaze as shown below. Taking MazeStringReader as an example

Maze maze;

MazeStringBuilder msb;

maze.setMaze(msb);

Start and end position

If the function IMazeReader::isVertex does not set start and end goals (by setting isGoal to true at a given position), they are set automatically based on the first and last Vertexes in stored in Maze. If the function IMazeReader::isVertex set isGoal to true for more that two position true, then start position is the first Vertex to be set as a goal and end position is the last Vertex to be set as goal. Vertexes are built at positions where isGoal is set to true, regardless if a different end position is later added and Vertex was otherwise redundant.

Path Finding

Four pathfinding algorithms are implemented. Depth First Search, Breadth First Search, Dijkstra and A\*. To find a path, run any of the pathfinding algorithms as follows.

1)Breadth First Search:It starts at the tree root , and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

2)DepthFirstSearch dfs;

dfs.findPath(maze.maze\_begin(), maze.maze\_end());

The calculated path is stored in a list and can be accessed by calling the following function.

list<Vertex\*> path = dfs.getPath();

3)Dijkstra :we generate a SPT (shortest path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source.

4)A\* is implemented using Manhattan distance. To add your own pathfinding algorithm, inherit the Path class. It gets the shortest path possible

SALIENT FEATURES:

* This project aims to explore different applications of graphs and incorporate it to study different pathfinding algorithms.
* It is basically a C++ implementation of a Maze data structure using graphs with four pathfinding algorithms.
* The implementation of the data structure aims to be memory efficient.
* The code is designed to be easily adaptable to read and solve mazes of any format.
* Different files will be made for building the maze, then finding path using different algorithms and finally printing it which will all be later imported into the driver program.
* Four pathfinding algorithms are implemented using this project:   
    
  Depth First Search  
  Breadth First Search  
  Dijkstra   
  A\*
* It will include the usage of data structures like graphs, lists, queues, and others if required.
* Templates will be used for performing functions on generic path rather than carrying it individually for each algorithm.
* Extensive use of effective data structures and multiple optimization routines would be done to provide a rich experience.