GTU Department of Computer Engineering

CSE 222/505 – Spring 2020

Homework 8

Buğra Eren Yılmaz

1801042669

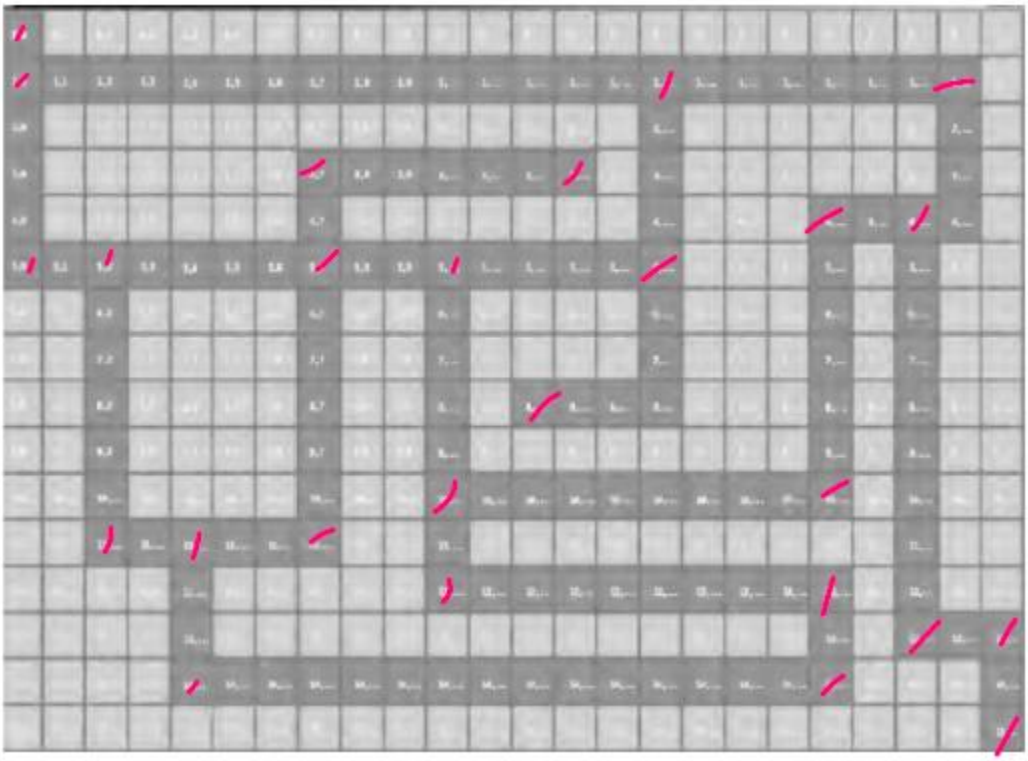
Q3

1. Class Diagrams

2. Problem Solution Approach

The problem at the hand was solving a 2 dimensional rectangular maze by creating a graph of junction points.

First of I needed a way to construct the graph representation of the maze. To achieve that, first I started to find out if a maze cell is junction or not. I considered the junction points as the following.



To construct a graph, such that edges of the graph is only between these junction point, I used the book ADT for graph. There was this method of loadEdgesFromFile

and it was not implemented in the book, rather it was left for the book reader to implement as a programming project. So I implemented that method inside AbstractGraph first. That way I had a way to construct graphs from file.

After that I created my MazeSolver interface which specified the need for a maze solver. Then I created the AbstractMazeSolver base class which specifies the fundamental construction of MazeSolver and basic helper methods to analyze maze cells to find out if it is a valid maze cell, a junction or even find out neighbors of a maze cell.

This AbstractMazeSolver also solves the problem of loading a maze file inside memory. It loads given maze text file into memory and stores the maze file content inside 2d list structure.

For these ADTs I created the MazeVertex class and MazeEdge class which represents these maze properties. MazeVertex holds the coordinate inside of maze too. MazeEdge holds MazeVertex classes.

Then I started to implement the concrete class implementation which is DijkstrasMazeSolver class which extends AbstractMazeSolver.

DijkstrasMazeSolver had 2 jobs actually, 1 is to analyze the loaded in-memory 2d list structure of maze and construct the correct string input for AbstractGraph classes loadEdgesFromFile method.

Since that methods specifies a special kind of input for graph creation like below.

1 0 1

1 2 4

3 4 1

Which means 1 and 0 has edge that has weight of 1 and it goes…

So it does not contain any information other than this id of vertex.

But I needed a way to know which vertex is at which coordinate since I am interested in finding the shortest path. For achieving this I hold the vertex list inside DijkstrasMazeSolver class and fill that up with MazeVertex objects. Then used this lists index values for vertex ids inside Graph. That way if a vertex inside graph is 3, then the corresponding MazeVertex is inside this list at index 3.

For constructing these MazeVertex list and MazeEdge list of. I developed an algorithm that is like DFS but different. It goes like this.

Start from vertex V which is top of the stack

Go right from V until, not valid cell or junction cell met

If junction cell was met, create edge between that cell and V

Push this new cell to stack, to be traversed on next loop

Go left from V until….

If junction cell was met…

Push this new cell…

Go down…

…

Go up…

…

This way I construct the MazeEdges and have the correct edges for graphs. Then I construct the correct input string for AbstractGraph class from this edges.

After the creation of correct graph with correct edges, I apply the Dijkstras algorithm which is provided inside the book, and find out the shortest path and return it as a string.

The program prints the shortest path of any given valid maze file. It also prints the constructed edges of the graph for showing that the graph only contains edges from junction to junction.

3. Results

All edges:

(0,0) -> (1,0){1.0}

(1,0) -> (0,0){1.0}

(1,0) -> (1,15){15.0}

(1,0) -> (5,0){4.0}

(1,15) -> (1,0){15.0}

(1,15) -> (1,22){7.0}

(1,15) -> (5,15){4.0}

(5,0) -> (1,0){4.0}

(5,0) -> (5,2){2.0}

(5,2) -> (5,0){2.0}

(5,2) -> (5,7){5.0}

(5,2) -> (11,2){6.0}

(5,7) -> (5,2){5.0}

(5,7) -> (5,10){3.0}

(5,7) -> (3,7){2.0}

(5,7) -> (11,7){6.0}

(11,2) -> (5,2){6.0}

(11,2) -> (11,4){2.0}

(11,4) -> (11,2){2.0}

(11,4) -> (11,7){3.0}

(11,4) -> (14,4){3.0}

(11,7) -> (11,4){3.0}

(11,7) -> (5,7){6.0}

(14,4) -> (11,4){3.0}

(14,4) -> (14,19){15.0}

(14,19) -> (14,4){15.0}

(14,19) -> (12,19){2.0}

(12,19) -> (14,19){2.0}

(12,19) -> (12,10){9.0}

(12,10) -> (12,19){9.0}

(12,10) -> (10,10){2.0}

(10,10) -> (12,10){2.0}

(10,10) -> (5,10){5.0}

(10,10) -> (10,18){8.0}

(5,10) -> (10,10){5.0}

(5,10) -> (5,15){5.0}

(5,10) -> (5,7){3.0}

(10,18) -> (10,10){8.0}

(10,18) -> (4,18){6.0}

(4,18) -> (10,18){6.0}

(4,18) -> (4,21){3.0}

(4,21) -> (4,18){3.0}

(4,21) -> (4,22){1.0}

(4,21) -> (13,21){9.0}

(4,21) -> (3,21){1.0}

(4,22) -> (4,21){1.0}

(4,22) -> (1,22){3.0}

(13,21) -> (4,21){9.0}

(13,21) -> (13,23){2.0}

(13,23) -> (13,21){2.0}

(13,23) -> (15,23){2.0}

(1,22) -> (4,22){3.0}

(1,22) -> (1,15){7.0}

(5,15) -> (1,15){4.0}

(5,15) -> (5,10){5.0}

(5,15) -> (8,15){3.0}

(8,15) -> (5,15){3.0}

(3,7) -> (5,7){2.0}

(3,7) -> (3,21){14.0}

(3,21) -> (3,7){14.0}

(3,21) -> (4,21){1.0}

(15,23) -> (13,23){2.0}

Shortest path: (15,23) -> (13,23) -> (13,21) -> (4,21) -> (4,22) -> (1,22) -> (1,15) -> (1,0) -> (0,0)