GIT Department Of Computer Engineering
CSE 222/505 - Spring 2020
Homework 8 Report

Fatih Kaan Salgır
171044009

PART 3

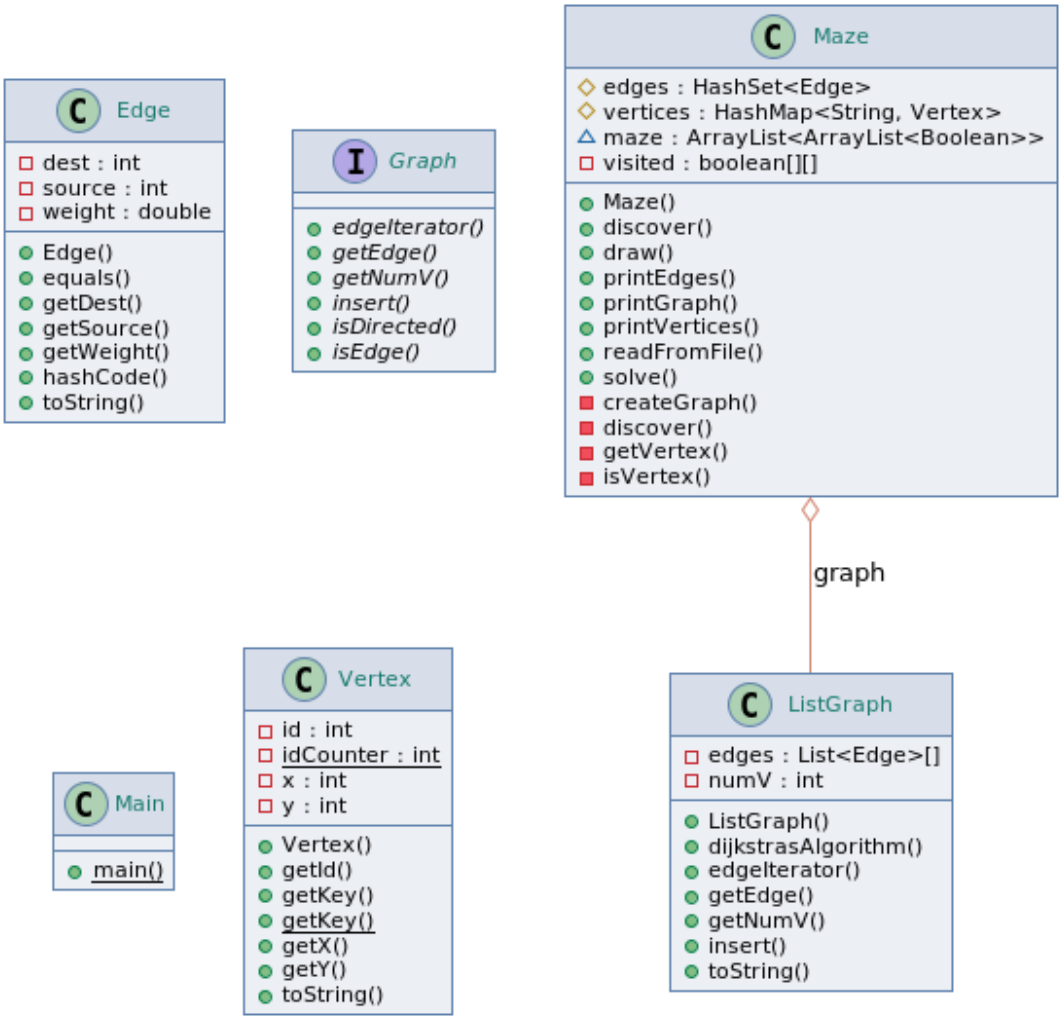
1 Problem Solution Approach

First I have read the file to a 2 dimensional boolean `ArrayList`. Then I have call the recursive `discover()` method which starts form beginning position and moves through 0's (false) in 2D `ArrayList`, identify vertexes and create edges. There are some cases to consider in this method. First we cannot simply check if node is visited before checking if it is a vertex. Because in this case some edges would be missing. When we find a vertex, before creating new vertex we must check if we added this vertex to vertices list. To get the vertex we need it's x and y coordinates. Therefore we can use a `Map` data structure, with keys are coordinates and values are vertices. Another case to consider when new vertex is found is, checking if last vertex is the vertex we found. In this case we will return because vertexes don't have edges to itself. Also I have used `Set` data structure to store edges to avoid duplicate.

Second step is creating the graph with the edges and the vertices we have found. I have used list graph representation, because by looking the map we can see that graph will be sparse, there are not many edges compare to number of vertices.

For finding the shortest path I have used the dijkstras algorithm. By using the output of dijkstras algorithm I have created a Map structure which consist of source - destination. Searching in this map and putting into a stack gives us reverse of the shortest path. Therefore to obtain shortest path, I have poped the items in the stack and print them out.

2 Class Diagrams



3 Test Cases

I have created some text cases for some adding and removing operations which is present on my main method as commented out.

Test Case ID	Test Scenario	Test Steps	Expected Results	Actual Results	Pass/Fail

T01	Maze with upper way blocked	`Maze m3 = new Maze("maze3.txt");`
		`m3.draw();`
		`m3.solve();`
			`0.....`
			`1.....14.11..8.....`
			`.....`
			`.....13.....`
			`.....9..`
			`..2....12..5....6.....`
			`.....`
			`.....`
			`.....7.....`
			`.....`
			`.....4.....`
			`....3.....`
			`.....`
			`.....`
			`.....`
			`.....10`
			`**SHORTEST PATH**`
			`0 1 2 12 5 6 8 9 10`
				As expected	Pass

T02	Maze with illegal character	`Maze m2 = new Maze("maze2.txt");`	`IllegalArgumentException: File must contain only 1's and 0's`	As expected	Pass

4 Running and Results

```
1  Maze maze = new Maze("maze.txt");
2  maze.draw();
3  maze.printVertices();
4  maze.printEdges();
5  maze.printGraph();
6  maze.solve();
7
8  System.out.println("T1");
9  Maze m3 = new Maze("maze3.txt");
10 m3.draw();
11 m3.solve();
12 System.out.println("T2");
13 Maze m2 = new Maze("maze2.txt");
14
```

Listing 1: Testing methods in main method

Output of the test cases;

```
0.....................
1.............8........
.....................
............12.........
.....................9..
..2....11..5....6........
.....................
.....................
............7...........
.....................
..........4.............
....3...................
.....................
.....................
.....................
.....................10
```

Listing 2: Visual representation of found vertices in maze

```
[Vertex{x=0, y=0, id=0}
, Vertex{x=1, y=0, id=1}
, Vertex{x=11, y=4, id=3}
, Vertex{x=1, y=15, id=8}
, Vertex{x=5, y=2, id=2}
, Vertex{x=10, y=10, id=4}
, Vertex{x=5, y=7, id=11}
, Vertex{x=8, y=12, id=7}
, Vertex{x=5, y=10, id=5}
, Vertex{x=4, y=21, id=9}
, Vertex{x=15, y=23, id=10}
, Vertex{x=5, y=15, id=6}
, Vertex{x=3, y=13, id=12}
]
```

Listing 3: Coordinates and id's of vertices

```
[Edge{dest=1, source=0, weight=1.0}
, Edge{dest=2, source=1, weight=6.0}
, Edge{dest=3, source=2, weight=8.0}
, Edge{dest=4, source=3, weight=31.0}
, Edge{dest=5, source=4, weight=5.0}
, Edge{dest=1, source=8, weight=15.0}
, Edge{dest=6, source=5, weight=5.0}
, Edge{dest=7, source=6, weight=6.0}
, Edge{dest=4, source=9, weight=17.0}
, Edge{dest=2, source=11, weight=5.0}
, Edge{dest=8, source=6, weight=4.0}
, Edge{dest=3, source=11, weight=9.0}
, Edge{dest=11, source=5, weight=3.0}
, Edge{dest=9, source=8, weight=11.0}
, Edge{dest=10, source=9, weight=13.0}
, Edge{dest=12, source=11, weight=8.0}
]
```

Listing 4: Visual representation of found vertices in maze

```
0: (1, 1.0)
1: (0, 1.0) (2, 6.0) (8, 15.0)
2: (1, 6.0) (3, 8.0) (11, 5.0)
3: (2, 8.0) (4, 31.0) (11, 9.0)
4: (3, 31.0) (5, 5.0) (9, 17.0)
5: (4, 5.0) (6, 5.0) (11, 3.0)
6: (5, 5.0) (7, 6.0) (8, 4.0)
7: (6, 6.0)
8: (1, 15.0) (6, 4.0) (9, 11.0)
9: (4, 17.0) (8, 11.0) (10, 13.0)
10: (9, 13.0)
11: (2, 5.0) (3, 9.0) (5, 3.0) (12, 8.0)
12: (11, 8.0)
```

Listing 5: Created graph shows source and destination vertices with weights

```
0 0 0.0
1 0 1.0
2 1 7.0
3 2 15.0
4 5 20.0
5 11 15.0
6 5 20.0
7 6 26.0
8 1 16.0
9 8 27.0
10 9 40.0
11 2 12.0
12 11 20.0
```

Listing 6: Output of Dijkstras algorithm

```
0 1 8 9 10
```

Listing 7: Shortest path