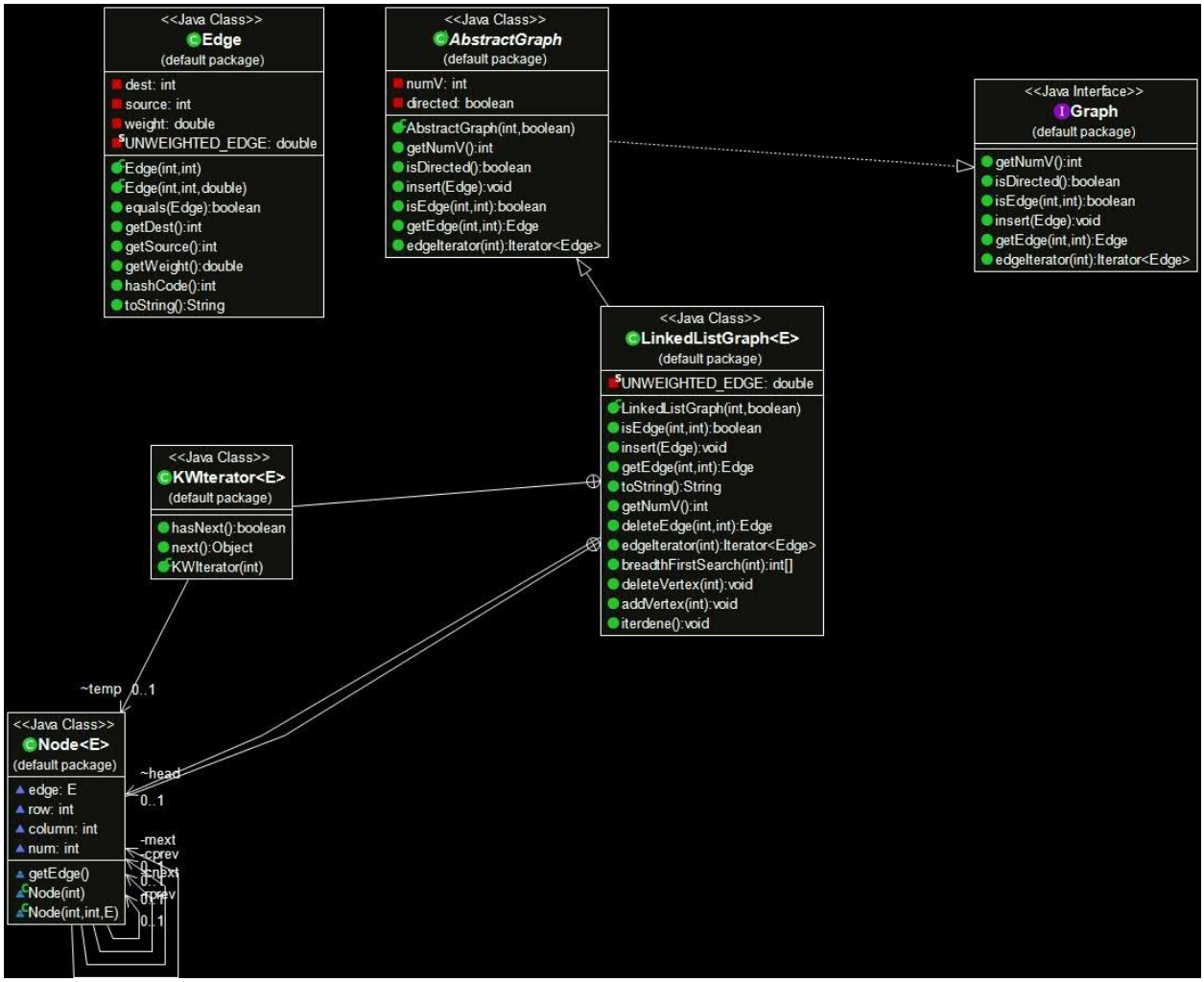
GIT Department of Computer Engineering
CSE 222/505 - Spring 2020
Homework 8 Report

161044063

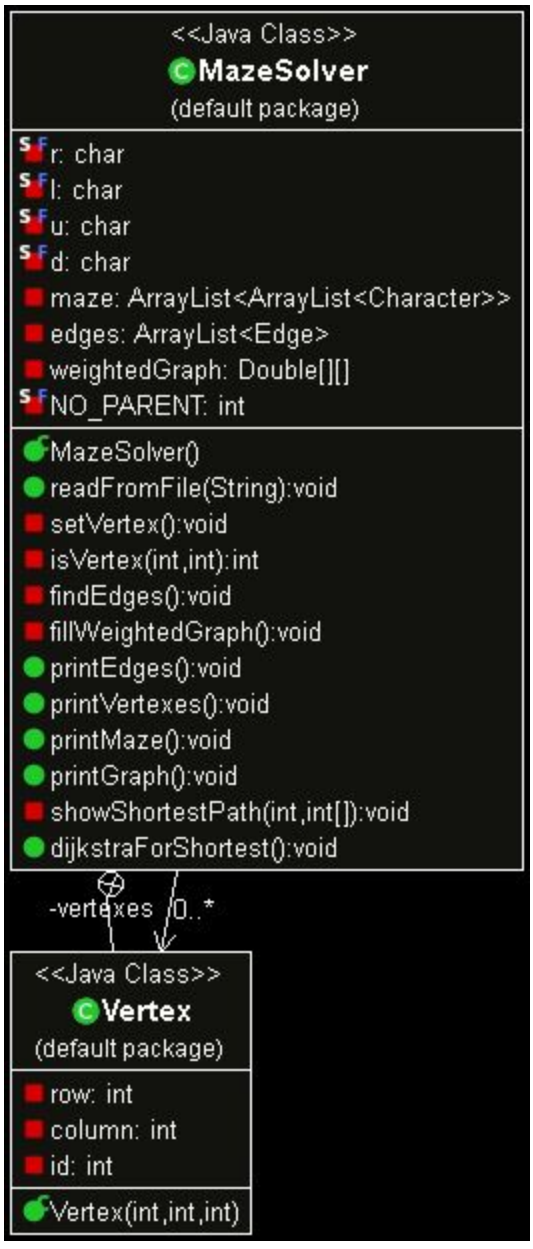
ZAFER ALTAY

1-)CLASS DIAGRAM

FOR PART2:



FOR PART3:


<<Java Class>>
MazeSolver
(default package)
r: char
l: char
u: char
d: char
maze: ArrayList<ArrayList<Character>>
edges: ArrayList<Edge>
weightedGraph: Double[][]
NO_PARENT: int
MazeSolver()
readFromFile(String):void
setVertex():void
isVertex(int,int):int
findEdges():void
fillWeightedGraph():void
printEdges():void
printVertexes():void
printMaze():void
printGraph():void
showShortestPath(int,int[]):void
dijkstraForShortest():void
-vertexes
0..*
<<Java Class>>
Vertex
(default package)
row: int
column: int
id: int
Vertex(int,int,int)

2-)TEST CASES

For LinkedListGraph:

TEST ID	T.SCENERIO	TEST DATA	EXPECTED R.	PASS/FAIL
T0	**create list, add a few elements and print them**	**add (0,1) (0,3) (0,2) (1,3) (1,2) (2,4) (3,1) (3,2)**	**Edge 0--> Edge 1--> edge: Source: 0, Destination 1, Weight: 1.0 edge: Source: 3, Destination 1, Weight: 1.0 Edge 2--> edge: Source: 0, Destination 2, Weight: 1.0 edge: Source: 1, Destination 2, Weight: 1.0 edge: Source: 3, Destination 2, Weight: 1.0 Edge 3--> edge: Source: 0, Destination 3, Weight: 1.0 edge: Source: 1, Destination 3, Weight: 1.0 Edge 4--> edge: Source: 2, Destination 4, Weight: 1.0**	**PASS**
T1	**test bfs and print the shortest path**	**start index=0**	**The shortest path is: 2 4**	**PASS**
T2	**test the iterator to show**	**for source 1**	**Source Dest Weight ---> 1 2 1.0 Source Dest Weight ---> 1 3 1.0**	**PASS**

T04	**delete vertex and print others**	**delete 1**	**Edge 0--> Edge 1--> edge: Source: 0, Destination 1, Weight: 1.0 edge: Source: 3, Destination 1, Weight: 1.0 Edge 2--> edge: Source: 0, Destination 2, Weight: 1.0 edge: Source: 3, Destination 2, Weight: 1.0 Edge 3--> edge: Source: 0, Destination 3, Weight: 1.0 Edge 4--> edge: Source: 2, Destination 4, Weight: 1.0**	**PASS**

For MazeSolver:

TEST ID	T.SCENERIO	TEST DATA	EXPECTED R.	PASS/FAIL
T0	**Reading data from file**	**abc.txt**	**1s and 0s in the file we read**	**PASS**
T1	**show all vertexes**		**Vertex id: 0 Row: 0 Column: 0 Vertex id: 1 Row: 1 Column: 0 Vertex id: 2 Row: 1 Column: 15 Vertex id: 3 Row: 3 Column: 13 Vertex id: 4 Row: 4**	**PASS**

			Column: 21 Vertex id: 5 Row: 5 Column: 2 Vertex id: 6 Row: 5 Column: 7 Vertex id: 7 Row: 5 Column: 10 Vertex id: 8 Row: 5 Column: 15 Vertex id: 9 Row: 8 Column: 12 Vertex id: 10 Row: 10 Column: 10 Vertex id: 11 Row: 11 Column: 4 Vertex id: 12 Row: 15 Column: 23	
T2	**show all edges**		**0.Edge sources: 0 Destination: 1 Weigth: 1.0 1.Edge sources: 1 Destination: 2 Weigth: 15.0 2.Edge sources: 1 Destination: 5 Weigth: 6.0 3.Edge sources: 2 Destination: 4 Weigth: 11.0 4.Edge sources: 2 Destination: 8 Weigth: 4.0 5.Edge sources: 3 Destination: 6 Weigth: 8.0 6.Edge sources: 4 Destination: 10 Weigth: 17.0 7.Edge sources: 4 Destination: 12 Weigth: 13.0 8.Edge sources: 5 Destination: 6 Weigth: 5.0 9.Edge sources: 5 Destination: 11 Weigth: 8.0 10.Edge sources: 6 Destination: 7 Weigth: 3.0 11.Edge sources: 6 Destination: 11 Weigth: 9.0 12.Edge sources: 7 Destination: 8 Weigth: 5.0 13.Edge sources: 7 Destination: 10 Weigth: 5.0 14.Edge sources: 8 Destination: 9 Weigth:**	**PASS**

			6.0 **15.Edge sources: 10** **Destination: 11 Weigth: 31.0**	
T3	**Show shortest path**		**Vertex** **0 -> 12** **Distance** **40.0** **Path** **0 1 2 4 12**	**PASS**

3-) Running command and results

For LinkedListGraph:

```
First we create an undirected list
We insert edge according to the sample in pdf   T0
Edge 0-->
Edge 1-->
        edge: Source: 0, Destination 1, Weight: 1.0
        edge: Source: 3, Destination 1, Weight: 1.0
Edge 2-->
        edge: Source: 0, Destination 2, Weight: 1.0
        edge: Source: 1, Destination 2, Weight: 1.0
        edge: Source: 3, Destination 2, Weight: 1.0
Edge 3-->
        edge: Source: 0, Destination 3, Weight: 1.0
        edge: Source: 1, Destination 3, Weight: 1.0
Edge 4-->
        edge: Source: 2, Destination 4, Weight: 1.0

We test bfs and print the shortest path to the screen   T1
The shortest path is:
2
4
We write an extra function and test the iterator to show that it works  T2
Source Dest Weight ---> 1 2 1.0
Source Dest Weight ---> 1 3 1.0
Now we test delete vertex and print other  T3
Edge 0-->
Edge 1-->
        edge: Source: 0, Destination 1, Weight: 1.0
        edge: Source: 3, Destination 1, Weight: 1.0
Edge 2-->
        edge: Source: 0, Destination 2, Weight: 1.0
        edge: Source: 3, Destination 2, Weight: 1.0
Edge 3-->
        edge: Source: 0, Destination 3, Weight: 1.0
Edge 4-->
        edge: Source: 2, Destination 4, Weight: 1.0
```

For MazeSolver:

```
First of all, we create an object.
Now we are reading from the file for maze. (We need not add txt while writing the file name)
We print the maze on the screen to test reading  T0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1
0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 1
0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1
1 1 0 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 0 1 1 0 1 1
1 1 0 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 0 1 1 0 1 1
1 1 0 1 1 1 1 0 1 1 0 1 0 0 0 0 1 1 0 1 1 0 1 1
1 1 0 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1 1 0 1 1
1 1 0 1 1 1 1 0 1 1 0 0 0 0 0 0 0 0 1 1 0 1 1
1 1 0 0 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1
1 1 1 1 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 1 1
1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0
1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
```

```
Now we will show all vertexes.   T1
Vertex id: 0 Row: 0 Column: 0
Vertex id: 1 Row: 1 Column: 0
Vertex id: 2 Row: 1 Column: 15
Vertex id: 3 Row: 3 Column: 13
Vertex id: 4 Row: 4 Column: 21
Vertex id: 5 Row: 5 Column: 2
Vertex id: 6 Row: 5 Column: 7
Vertex id: 7 Row: 5 Column: 10
Vertex id: 8 Row: 5 Column: 15
Vertex id: 9 Row: 8 Column: 12
Vertex id: 10 Row: 10 Column: 10
Vertex id: 11 Row: 11 Column: 4
Vertex id: 12 Row: 15 Column: 23
```

```
Now we will display all the edges   T2
0.Edge sources: 0 Destination: 1 Weigth: 1.0
1.Edge sources: 1 Destination: 2 Weigth: 15.0
2.Edge sources: 1 Destination: 5 Weigth: 6.0
3.Edge sources: 2 Destination: 4 Weigth: 11.0
4.Edge sources: 2 Destination: 8 Weigth: 4.0
5.Edge sources: 3 Destination: 6 Weigth: 8.0
6.Edge sources: 4 Destination: 10 Weigth: 17.0
7.Edge sources: 4 Destination: 12 Weigth: 13.0
8.Edge sources: 5 Destination: 6 Weigth: 5.0
9.Edge sources: 5 Destination: 11 Weigth: 8.0
10.Edge sources: 6 Destination: 7 Weigth: 3.0
11.Edge sources: 6 Destination: 11 Weigth: 9.0
12.Edge sources: 7 Destination: 8 Weigth: 5.0
13.Edge sources: 7 Destination: 10 Weigth: 5.0
14.Edge sources: 8 Destination: 9 Weigth: 6.0
15.Edge sources: 10 Destination: 11 Weigth: 31.0


Now, by applying the dijkstra algorithm, we print the shortest path on the screen.   T3


Vertex                          Distance        Path
0 -> 12                         40.0            0 1 2 4 12
```

4-)Problem Solution Approach

For MazeSolver at q3:

First, I defined the necessary elements for the class.These were the corners, edges, the weighted graphic, and the elements I would keep from the file.In addition, I created an inner class for vertexes.I kept the coordinates and id of the Vertex in the data I read from this class file.After that I went on reading from the file.

readFromFile : In this method, I read each line from the file and added it to our character-character maze variable.Then I called the setVertex method and found the vertexs, then called the findEdges method and found the edges.

setVertex : In this method,I checked the circumference for all zeros, if there is one times or more than 3 zeros around this is a crossroads and it would also be vertex. I added Vertex with its features on the vertexes list.I did the control by checking the each lines of the maze in turn.

isVertex : In this method,I checked the vertexes list with the data I received in the parameters of the function. If there is, I returned true.If there is not,I returned null.

findEdges : In this method, if there is a value of 0 in the right upper left lower neighbor squares of a vertex starting from the upper left corner in the maze, I followed the 0's for all. But I put a prohibited movement in order not to go in the direction I came from, I determined the opposite direction from the direction I came to be prohibited, so I did not count the same edge twice.For each step, I checked if the edge I was in was vertex. The end of every 0 of course goes to a vertex. If I came over the vertex, this is an edge.I counted for every 0, I increased by 1 for each step, and this was weighted.Thus, we have determined all features such as weight, source, and destination for an edge and added it to the list of edges.After determining all the edges, I called the fillWeightedGraph method for fill the weighted graph.

fillWeightedGraph : In this method, I brought an edge from the list of edges. And I placed the weight on the weighted graph, according to its source,destination and weighted properties.I placed the same weight in the symmetrical region of the same location, and I did this for all edges.

***I used the dijkstra's algorithm to find the shortest path for this chart because each edge has weights.**

For LinkedListGraph:

I first created an inner node class for my linked list.In this class, I kept links, one edge object, row and column.In the constructor method, I added node to the rnext and cnext of the head as much as the number of corners I received.These nodes represent source and destination vertexes.

insert : I created a node according to the source and destination of the edge we took as a parameter, and proceeded to the relevant numbers in the nodes representing the vertexes and connected them.If directed, I connected rprev and cprev to null, if undirected, prevs links to the previous one.

getEdge : I reached the location specified by parameters using rnext and cnext and returned the edge of that noded, if any.

deleteEdge : Using rnext and cnext, I reached the location specified by the parameters and removed the node if any.While removing the node, I also made the necessary linked operations.

deleteVertex : According to the sequence number of vertex given, I removed the correct vertex for both source and distance.

addVertex : I added a new vertex to the end of the lists (ie both for source and destination).

Deep first search and file reading sections are missing from my list.