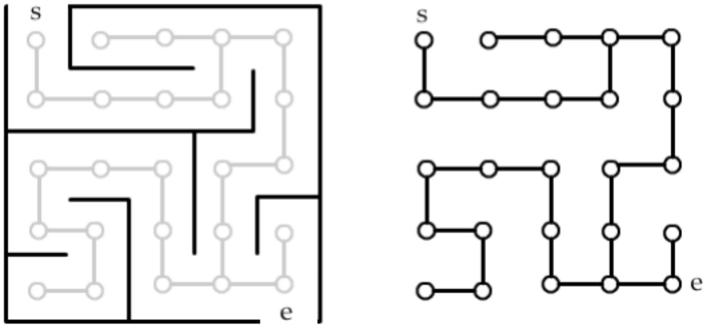
CME 2201 - Assignment 2

MAZE SOLVER

- A maze can be represented as a graph, if each juncture (intersection) in the maze is considered to be a vertex, and edges are added to the graph between adjacent junctures that are not blocked by a wall.



The mazes will always have a starting point at the upper left corner and an ending point at the lower right corner.

- In this project, you are expected to solve a given maze in four different ways, by first converting the maze into a graph, and then apply the following standard graph algorithms (please write them in *DirectedGraph*.java file that is under *GraphPackage* package):

  1. Depth-First-Search

/* **public QueueInterface<T> getDepthFirstSearch(T origin, T end)**

 * **return** depth first search traversal order between origin vertex and end vertex

 */

  2. Breadth-First-Search

/* **public QueueInterface<T> getBreadthFirstSearch(T origin, T end)**

 * **return** breadth first search traversal order between origin vertex and end vertex

 */

3. Shortest Path Algorithm

/* **public int getShortestPath(T begin, T end, StackInterface<T> path)**

***** **return** the shortest path between begin vertex and end vertex

*/

4. Cheapest Path Algorithm

In this part, the maze is reorganized in a way that there are some costs to visit a path during traversals. The gamer wants to the cheapest cost path at the end of traversal. The cost of each path is **randomly generated between 1 and 4**. The edge costs should be printed in the output of your program. According to these costs, the cheapest path and the least cost should be printed using your own written ***getCheapestPath*** method.

/** Precondition: path is an empty stack (NOT null) */

/* Use EntryPQ instead of Vertex in Priority Queue because multiple entries contain the same

* vertex but different costs - cost of path to vertex is EntryPQ's priority value

* **public double getCheapestPath(T begin, T end, StackInterface<T> path)**

* **return** the cost of the cheapest path

*/

Graph Implementation

Your implementation should work for every maze type (do not construct for a specific maze). You are given sample maze text files, you can use them to test your program.

GraphPackage and *ADTPackage* are given to you for your graph implementation. You **<u>must</u>** use their methods.

While constructing your graph, name each vertex according to its position (its row and column number) in the maze. For example, if there is a vertex on row one and column two, then the name of the vertex should be *1 – 2*.

Your search algorithms should use adjacency lists of vertices. Adjacency matrix of a graph should also be found. In this direction, write a function to create and return the adjacency matrix of the graph in *DirectedGraph*.java file that is under *GraphPackage* package.

A sample adjacency matrix:

	0-0	1-0	1-1	1-2	…
0-0		1			…
1-0			1	1	…
1-2					…
…	…	…	…	…	…

You can add your helper functions if you need in *DirectedGraph*.java and *Test*.java files. Write comments about their functionality. Do not change other classes.

Grading Policy

Job	**Percentage**
Maze to Graph	30%
Adjacency Matrix	10%
Depth-First Search Implementation	20%
Breadth-First Search Implementation	10%
Shortest Path Implementation	10%
Cheapest Path	20%

Due date

25.12.2021 Sunday 23:55. Late submissions are not allowed.

Requirements

• You need to implement base functions of a classical graph data structure and classes (do not extend an available Java Graph classes directly, just use the given *GraphPackage* and *ADTPackage*).

• Object Oriented Programming (OOP) principles must be applied.

• Exception handling can be used when it is needed.

<u>Your codes may be checked by automatic control. For this reason, the default print requirements are as follows when sending the code:</u>

- Adjacency Lists of Each Vertex of the Graph After Maze to Graph Operation
- Adjacency Matrix of the Graph After Maze to Graph Operation
- The number of edges found
- BFS output between the starting and the end points of the maze
- The number of visited vertices for BFS
- DFS output between the starting and the end points of the maze
- The number of visited vertices for DFS
- Shortest path between the starting and the end points of the maze
- The number of visited vertices for Shortest Path
- The cheapest path for the Weighted Graph
- The number of visited vertices for the Weighted Graph
- The cost of the cheapest path

- o Example output format for traversals (draw a line using “.” on the path)

```
#.###################
#.        # #     # #
#.##### # # # ##### #
#.# #   #       #   #
#.# # ### ### # ### #
#.........# # #     #
### #####.# ##### ###
#   # # #.  #   #   #
# ### # #.### # ## ##
#     #  .  # #     #
##### ###.####### # #
#        .    #   # #
# # #####.# ### #####
# # # # #.#   #   # #
# ### # #.### ### # #
#       #...#   #   #
# # ### # #.##### # #
# #   # # #...# # # #
### # # #####.# ### #
#   # #      ........
#####################
```

Submission

You must upload *DirectedGraph*.java and *Test*.java files as an archive file (.zip or .rar). Your archived file should be named as ‘studentnumber_name_surname.rar/zip’, e.g., 2007510011_Ali_Yılmaz.rar and it should be uploaded in the SAKAI portal. You can ask your questions from the “FORUM -> Homework 2 - Questions” topic.

Plagiarism Control

The submissions will be checked for code similarity. Copy assignments will be graded as zero, and they will be announced in the SAKAI portal.