

*Alexandria University Faculty of Engineering Computer and Systems Engineering Dept. CS122: Data Structures*

Maze Report

**Name: Mohamed Ayman Farrag Ahmed**

**Id: 64**

1. **Data Structures used:**
   1. **Stacks:**

Custom built Stacks which were made in a previous assignment were used in the implementation of the DFS (Depth first Search) as a container for the nodes.

The stack was based on a linked list.

**Stacks supported:**

* + 1. Pushing new objects to the stack.
    2. Popping Objects from the stack.
    3. Peeking at the object on the top of the stack.
  1. **Queues:**

Custom built Queues which were made in a previous assignment were used in the implementation of the BFS (Breadth first search) as a container for the nodes.

The queue was based on a linked list.

**Queues supported:**

* + 1. Queuing new objects to the Queue.
    2. Dequeuing the object in the front of the queue.
    3. Peeking at the object in the front of the queue.
  1. **Hash Map:**

Java Built-in Hash map were used in both the implementations of the DFS and the BFS to record the Parent-Child relation between the nodes to be used in the end to trace back the path followed from the Entrance Point to the Exit Point.

**Hash Map Supported:**

1. Hashing a node using its Parent as a value and the child as a key.
2. Getting the value using the child as a key.

-Some other Data structures were used but not as much as the previously mentioned data structures.

**i.e.:**

1. **Point:**

To store the coordination of the nodes as an x-y Pair integer value.

1. **Arrays:**

An Array of String was used to store the Maze.

A Boolean array was used to Avoid Cycles in the traversing of the graph.

1. **Algorithms Used:**
   1. **Depth first Search:** is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.

In this case the root was the entrance of the Maze ‘S’ and it explored until it found the end ‘E’.

The Algorithm depended mainly on the Stack data structure where Recursion wasn’t used.

Algorithm Complexity Worst Case: O (|E|)

Where E: is the number of Edges.

* 1. **Breadth first Search:** is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key) and explores the neighbor nodes first, before moving to the next level neighbors.

The Algorithm depended mainly on the Queue data structure.

Algorithm Complexity Worst Case: O (|E|)

Where E: is the number of Edges.

1. **Main Methods:** 
   1. private void validateSize(int row, int col)

Method used to validate the size Entered by the user.

* 1. Private String[] getMap(File maze)

Method used to read the map from a File based input and return a 1D array of Strings describing the Maze.

* 1. Private Point findStart(String[] mazeMap)

Method used to search the Maze for the Entrance and validates the existence of one Entrance and one or more Exits.

* 1. Private Boolean validPoint(int row, int col)

Method used to verify whether a Point coordinate on the Maze is a valid point to be visited or not.

* 1. Private int[][] reformArray(int[][] mazePath)

Method used to reform the Array validating no null reference existing.

* 1. Private int[][] traceMaze(Map mazeTrace)

Method which uses the Map and Point related by Parent-child relation to find the Path from the Entrance to the Exit and returns a 2D Array Describing it point by point.

1. **Sample Runs:**
   1. **Maze:**

5 5

##..S

..#..

.##..

E....

..###

DFS:

0,4

1,4

2,4

3,4

3,3

3,2

3,1

4,1

4,0

3,0

BFS:

0,4

1,4

2,4

3,4

3,3

3,2

3,1

3,0

* 1. Maze:

10 5

....S

.####

.....

####.

.....

.####

.....

####.

.....

E.###

DFS:

0,4

0,3

0,2

0,1

0,0

1,0

2,0

2,1

2,2

2,3

2,4

3,4

4,4

4,3

4,2

4,1

4,0

5,0

6,0

6,1

6,2

6,3

6,4

7,4

8,4

8,3

8,2

8,1

9,1

9,0

BFS:

0,4

0,3

0,2

0,1

0,0

1,0

2,0

2,1

2,2

2,3

2,4

3,4

4,4

4,3

4,2

4,1

4,0

5,0

6,0

6,1

6,2

6,3

6,4

7,4

8,4

8,3

8,2

8,1

9,1

9,0

* 1. Maze:

5 5

##..E

##.##

E.S..

.###.

.###.

DFS:

2,2

1,2

0,2

0,3

0,4

BFS:

2,2

2,1

2,0