***Maze solver***

***Amr Mohamed Nasr 50***

**Used data structures:**

1. Stack (Based on single linked list)
2. Queue (Based on double linked list)
3. MazePoint(A data structure that contains 3 variables, x, y & parent which is from the type MazePoint)

**Algorithm used:**

1. Begin
2. S: empty Data Structure of cells
3. T = (the start tile ’S’)
4. S.add(T)
5. Mark T as visited
6. While S is not empty
7. T = S.remove()
8. Mark T as visited
9. if T is target cell
10. Mark that goal is reached
11. Break
12. for each valid unvisited neighbour N of T
13. S.add(N)
14. End While

The above implementation is valid for both DFS and BFS. The dierence is based on the order of visiting cells as the remove() function will change when you're using a stack or a queue. To retrieve the path you used to reach the target cell, you shall add more variable to each cell, which means who is the parent cell who made that one reachable. When you reach the target cell, you can nd backward who is it's parent and similarly untill reaching the starting cell.

**Assumption:**

Directions start from north, east, south then west.

**Functions:**

1. Char [] [] getMaze(File file):

Reads the maze available in a certain file and checks if it is a valid maze and returns the maze map as a 2 dimensional char array and defines the x and y of the start place.

1. boolean dFSsolver(final int yc,

final int xc, final char[][] map,final int n, final int m):

Recursive function that follows the DFS algorithm.

**Depth First Search**

A DFS of a maze involves following some path through the maze as far as we can go, until a dead end or previously visited location is met. When this occurs, the search backtracks to its most recent choice and tries a different path instead.

**Breadth First Search**

BFS proceeds differently: it visits the locations in order of their distance from the starting point of the search. First it visits all locations one step away, then it visits all locations that are two steps away, and so on, until the exit is found. Because of this, BFS has the nice property that it will naturally discover the shortest route through a maze.

**Tests**

**Test 1:**

5 5

S#..E

..#..

.##..

.....

..###

***DFS:***

{{0,0},{1,0},{2,0},{3,0},{3,1},

{3,2},{3,3},{2,3},{1,3},{0,3},{0,4}

}

***BFS:***

{{0,0},{1,0},{2,0},{3,0},{3,1},

{3,2},{3,3},{2,3},{1,3},{0,3},{0,4}

}

**Test 2:**

5 5

S#..E

..###

.##..

.....

..###

***DFS:***

Null

***BFS:***

Null

**Test 3:**

3 3

S..

...

E..

***DFS:***

{{0,0},{0,1},{0,2},{1,2},{2,2},

{2,1},{1,1},{1,0},{2,0}}

***BFS:***

{{0,0},{1,0},{2,0}}

**Test 4:**

5 5

#...S

..##.

.##..

E.#.#

....#

***DFS:***

{{0,4},{1,4},{2,4},{2,3},{3,3},

{4,3},{4,2},{4,1},{3,1},{3,0}}

**Test 5:**

5 5

#...S

..#..

.##..

E.#.#

....#

***DFS:***

{{0,4},{1,4},{2,4},{2,3},{1,3},

{0,3},{0,2},{0,1},{1,1},{1,0},{2,0},{3,0}}