Alexandria University
Faculty of Engineering
Computer and Systems Engineering Dept.
First Year

Name: Mahmoud Mohammed Hussein
ID : 73

Assignment #07

Maze Report.

Used Data Structure:

1. BFS
   To implement the breadth first search algorithm we only need to use a queue and an array
   The queue is to organize the order of visiting the neighbouring nodes in the maze.
   The arrays are used for getting the path which represent the order of the nodes that we have to visit in order to reach the target.
   For getting the path I used a 2D array of Points to save the parent of each node I visit A better way I have discovered later but I didn't use is to create a tree that stores only the nodes I've visited and their parents this data structure would extremely reduce the memory allocated compared to the Parent array.

2. DFS
   To implement the iterative depth first search algorithm we only need to use a stack and a parent array to store the path.
   The stack is to organize the order of visiting the neighboring nodes in the maze.

Used Algorithms:

1. BFS
   The breadth first search is an algorithm for traversing or searching tree or graph data structures.
   It depends on visiting the neighboring nodes level by level so when the target is found we get the shortest path from the source to the target.
2. DFS
   The depth first search is an algorithm for traversing or searching tree or graph data structures. It's a back tracking algorithm.
   It depends on deepening in the path till it is the wrong one then back to try another path and keeps trying till it founds the path or all the graph nodes are discovered.

Assumptions and necessary details:

- the implementation puts no constrains on the input size except the time complexity and the used memory.
- Using the iterative implementation of the DFS algorithm may be faster but using the recursion implementation is much easier to think.
- It's assumed that the maze may have more than one Ending point.
- It's assumed that the maze has only one start point.

A simple Comparison between the tow algorithms

Algorithm	DFS	BFS
Running Time	O (V+E)	O (V+E)
Pros	- Usually use less memory - Can find articulation points, bridges and strongly connected components	Can solve single source shortest path (unweighted graphs)
Cons	Cannot solve single source shortest path on unweighted graphs	Usually uses more memory so it's bad for large graphs
Code	Slightly easier to code	Slightly easier to code
Technique	Keeps deepening in one path till it's blocked then it tries another one	Visits the neighboring nodes level by level till it finds the Target so it obtains the shortest path

Sample runs:

Maze	DFS Solution	BFS Solution
5 5 .#..# .S#.# #.... ##.#. ###E.	(1,1) → (2,1) → (2,2) → (2,3) → (2,4) → (3,4) → (4,4) → (4,3)	(1,1) → (2,1) → (2,2) → (2,3) → (2,4) → (3,4) → (4,4) → (4,3)
4 6 #....S #.#.#. E#...# ...#E#	(0,5) → (0,4) → (0,3) → (1,3) → (2,3) → (2,4) → (3,4)	(0,5) → (0,4) → (0,3) → (1,3) → (2,3) → (2,4) → (3,4)
8 6 ####E# ...#.. .#..#. .S#... ###... .E#.#E .#..#.	(3,1) → (3,0) → (2,0) → (1,0) → (1,1) → (1,2) → (2,2) → (2,3) → (3,3) → (3,4) → (3,5) → (2,5) → (1,5) → (1,4) → (0,4)	(3,1) → (3,0) → (2,0) → (1,0) → (1,1) → (1,2) → (2,2) → (2,3) → (3,3) → (4,3) → (4,4) → (4,5) → (5,5)

```
19 20
###S....##.#..#..##.
####............##..
###....#..#.#....#.#
##....###......#...#
.####...#.....#.##E.
.###......#...#.#.#.
...##.#...#..#..#...
.....#.....#..#....#
.#.....##..#........
.##...E#......#....#
....#.......#......E
......##..#........#
......#.#.##....#...
..................#.
...##.##....#..###..
.##..#.........#...#
......#..#..###..#..
#......#...E#.......
.......###....#.#...
```

DFS Path

(0,3) → (0,4) → (0,5) → (0,6) → (0,7) →
(1,7) → (1,8) → (1,9) → (1,10) → (1,11) →
(1,12) → (0,12) → (0,13) → (1,13) → (1,14) →
(1,15) → (2,15) → (2,16) → (3,16) → (3,17) →
(3,18) → (4,18) →

BFS Path

(0,3) → (0,4) → (1,4) → (2,4) → (3,4) →
(3,5) → (4,5) → (5,5) → (5,6) → (5,7) →
(6,7) → (7,7) → (7,6) → (8,6) → (9,6) →