**BFS & DFS Maze Solver**

Depth First Search :

The main used data structure at BFS algorithum is Stack of the recursion function and a linked list.

* DFS takes a path until it found a barrier or the End if we are lucky.
* DFS in most cases wont found the shortest path and will take a very long.
* But DFS is more efficient than BFS at memory because it do not store a lot of pointers to the node which call or its parent.

Algorithum :

* Read(maze) // Reading file
* Change the data of this maze to a two dimensional array and find the entry point and number of exits to ensure that there will be an exit.
* Calling a recursive function giving it the x and y of the current point
  + If this point is outside the maze or visited before
    - Then return.
  + If this point is the end
    - then save it to the path then return.
  + Else
    - And it to the path and mark it as visited.
  + Call the function with the indexes of all neighbor nodes.
* End of the function.

“Simple Run” (DFS):

**The maze :**

**10 10**

**S......#..**

**.#..#...#.**

**.##...#...**

**#..#....#.**

**..###.....**

**#......#..**

**#.....#...**

**##.......#**

**E.......#.**

**##....#.#.**

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

The Breadth First Search :

main used data structure at BFS algorithum is Queue and a linkedlist

* BFS visits all child node of the current node.
* In this case it mostly gives the shortest path comparing with DFS.
* But it use a lot of memory.

Algorithum :

* Read(maze) // Reading file
* Change the data of this maze to a two dimensional array and find the entry point and number of exits to ensure that there will be an exit.
* Then create a new class called node to save the position of the current node and it’s parent whish is a pointer or reference to the same class.
* Call a recursive function by giving it the axis of the start node and a null parent pointer.
* In the function we set the node to visited
* Then visit all it’s child and enqueue it in the queue.
* Then dequeue a new node from the queue and call the function again giving it position of the new node and pointer to it as a new parent.
* In case of dequeue the End I will try to find all his parent and add it in the path.
* End of the function.

“Simple Run” (BFS):

1-

The maze :

**10 10**

**S......#..**

**.#..#...#.**

**.##...#...**

**#..#....#.**

**..###.....**

**#......#..**

**#.....#...**

**##.......#**

**E.......#.**

**##....#.#.**

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