Algorithms: Theory, Design and Implementation

5SENG003C

Coursework : Report

Full Name : Dissanayaka Mudiyanselage Dhananjika Niwarthani

A short explanation of the choice of data structure and algorithm.

I have provided a Java solution for the provided problem. In that solution, I have used the Breadth-First Search (BFS) Algorithm as the algorithm of this solution. As data structures, I have used Queue, 2D Array, Array Lists, Linked List.

The Algorithm used in this solution is,

Breadth-First Search (BFS) Algorithm:

BFS is chosen for maze-solving due to its efficiency in finding the shortest path in unweighted graphs or mazes. The algorithm explores positions level by level, ensuring that the shortest path is found before longer paths. BFS utilizes a queue data structure to manage the exploration process, maintaining a systematic approach to searching for the shortest path. Its effectiveness and efficiency make it a suitable choice for navigating through the maze and finding the shortest path from the start to the end point.

The data structures used in this solution are,

Queue (Linked List):

A queue data structure is used to manage the positions to be explored during the BFS (Breadth-First Search) algorithm. Specifically, a Linked List is utilized as the underlying data structure for the queue, as it provides efficient insertion and removal of elements from both ends of the queue. Linked List's **addLast()** method is used to enqueue elements (add positions to the end of the queue), and **pollFirst()** method is used to dequeue elements (remove and return the first position in the queue).

2D Array:

A two-dimensional array (int[][]) is used to represent the puzzle. Each cell of the array corresponds to a position in the maze, allowing for easy access and modification of maze elements. The 2D array is used to store the maze layout, including walls, open paths, start and end points, and visited positions.

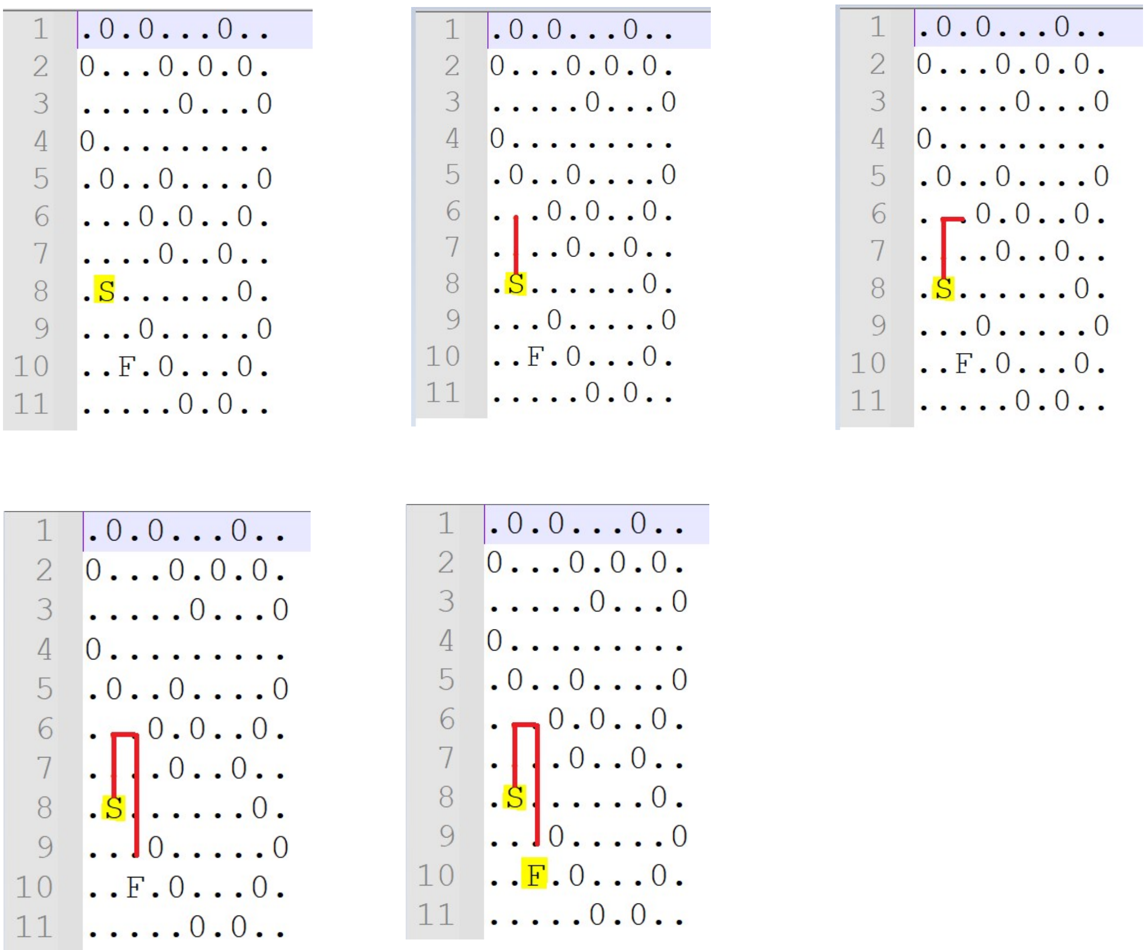
Array List:

Array Lists are utilized to store the path and directions found during the maze-solving process. Two Array Lists (path and directions) are used to store the coordinates of positions along the shortest path and the corresponding directions to navigate through the maze. Array Lists provide dynamic resizing, allowing for flexible storage of path information as the algorithm progresses.

A run of the algorithm on a small benchmark example

As a small example, I used first text file of benchmark_series.It's height is 11 and width is 10.

As instructions,



Here the starting position is "S". Hence the position (2,8)

Then move to up until hit the rock. Hence the position (2,6)

Then move to right until hit the rock. Hence the position (3,6)

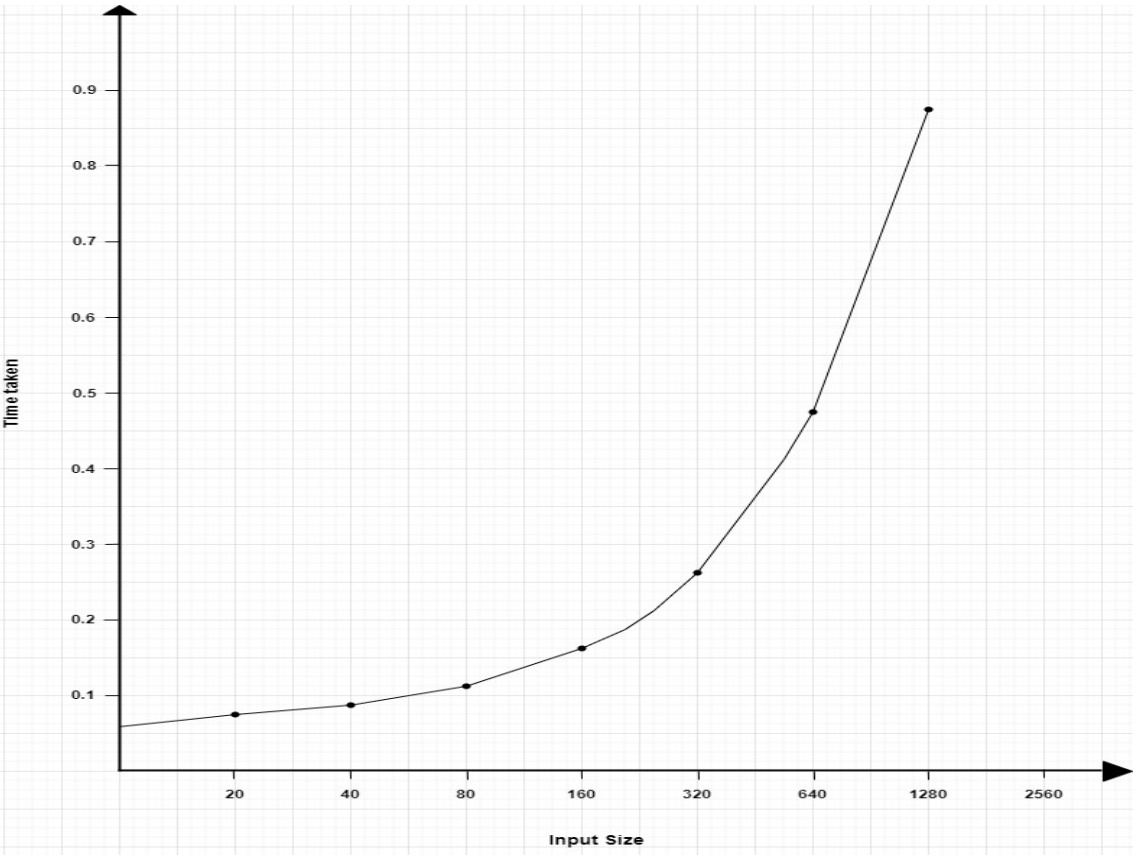
Then move to down until hit the rock. Hence the position (3,9)

Here the finishing position is "F". Hence the position (3,10)

```
File Name: puzzle_10.txt

Start Point: (2,8) | End Point: (3,10)

3 moves

Start at (2,8)
Move UP to (2,6)
Move RIGHT to (3,6)
Move DOWN to (3,10)
Done!
```

A performance analysis of the algorithmic design and implementation

Input Size	Time taken (seconds)	Run Time Ratio
20	0.08	
40	0.09	1.09
80	0.11	1.16
160	0.16	1.45
320	0.26	1.59
640	0.47	1.82
1280	0.87	1.85
2560	1.85	2.11



According to the empirical approach and the graph, if the input quantity is "n" then the increment of time taken is **"n^2"**. According to this, this is a **quadratic** complexity class. Here the Big-O notation is **($O(n^2)$).**