



**INFORMATICS  
INSTITUTE OF  
TECHNOLOGY**

**UNIVERSITY OF  
WESTMINSTER** 冊

## **Informatics Institute of Technology**

**5SENG003W – Algorithms: Theory, Design and  
Implementation**

**Module Leader: Mr. Ragu Sivaraman**

### **Coursework Report**

Name – Praveen de Silva.

UOW No – W1985643

IIT ID – 20221895

## Table of Contents

<b><u>1. DATA STRUCTURE AND AGORITHM .....</u></b>	<b><u>3</u></b>
<b><u>2. BENCHMARKS .....</u></b>	<b><u>4</u></b>
<b><u>3. PERFORMANCE ANALYSIS OF ALGORITHMIC DESIGN AND IMPLEMENTATION...</u></b>	<b><u>4</u></b>

## List of figures

Figure 1 - Constructor for BFS .....	3
Figure 2 - Maze structure .....	4
Figure 3 - output .....	4
Figure 4 - Performance Analysis Graph .....	5

# 1. Data Structure and Algorithm

## A. Algorithm

The chosen algorithm implements the Breadth-First Search (BFS) algorithm to efficiently find the shortest path in a maze. It is straightforward to implement and understand, ensuring the shortest path from the starting point to the goal. Compared to other algorithms like Dijkstra's algorithm and A\* algorithm, BFS tends to find paths with fewer steps, making it a suitable choice for finding the shortest path in a maze.

In addition, BFS can be readily parallelized to take advantage of several processors to speed up the search and is generally more memory economical. All of these benefits combined to make BFS a good option for determining the shortest path through a maze.

To use the BFS to discover the shortest path, one needs to initialize an instance of the class, provide the maze, and the starting and finishing positions, and then invoke the shortest distance method. As an example:

```
BFSAlgorithm shortPath = new BFSAlgorithm();
```

*Figure 1 - Constructor for BFS*

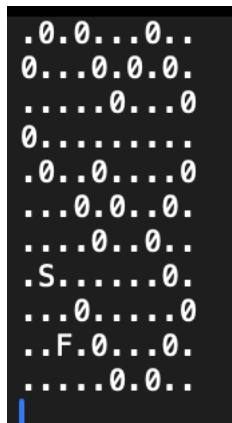
## B. Data Structures

The main data structures used in the code are a Queue and a 2D boolean array named visited. The Queue, implemented using the LinkedLists, stores the coordinates of the maze cells to be visited. It adheres to the First-In-First-Out (FIFO) sequence, which is necessary for traversing the Breadth-First Search (BFS). The visited array, a 2D boolean array, is used to record visited locations across the maze in order to prevent previously visited cells from being accidentally visited during traversal. This prevents redundant computations and ensures that each cell of the maze is explored exactly once. These data structures are crucial parts of the code's execution of the Breadth-First Search (BFS) method, which ensures accurate and speedy maze traversal in order to find the shortest path.

In the maze, coordinates are represented using the Coordinate class. It records details about a specific cell, such as its row and column, the number of steps needed to get there, the path followed, the maze's starting coordinates, and the path's step count. During the BFS traversal, this class is essential for building the path and storing data about each cell that is visited.

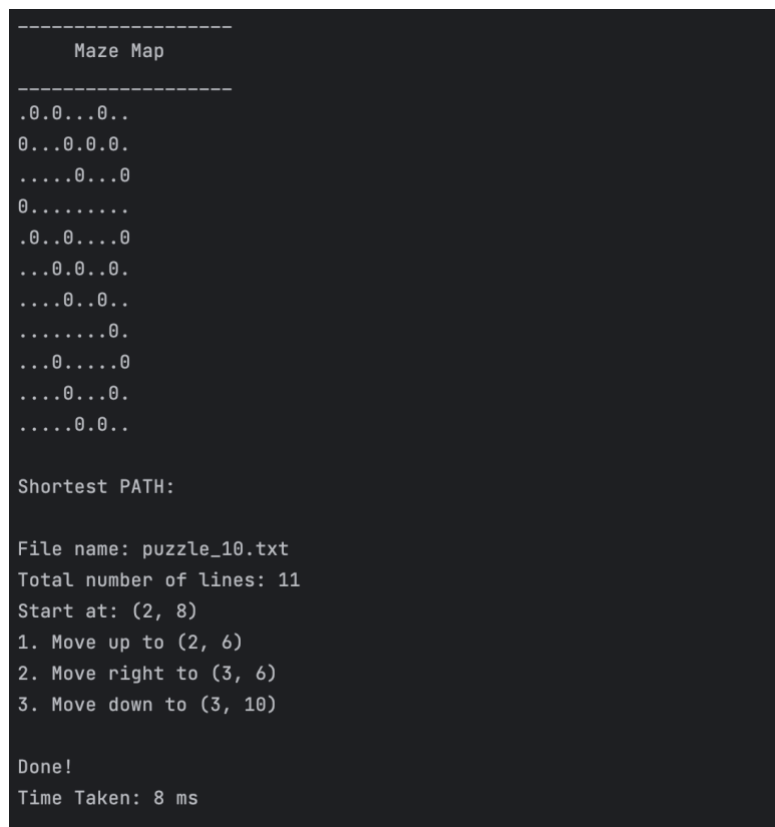
## 2. Benchmarks

A sample of the text file of the maze and its output of the bfs shortest bfs path is as below.



```
.0.0...0.
0...0.0.0.
....0...0
0.....
.0..0...0
...0.0..0
...0..0..
.S.....0
...0.....0
..F.0...0.
...0.0..
```

Figure 2 - Maze structure



```
-----
Maze Map
-----
.0.0...0..
0...0.0.0.
....0...0
0.....
.0..0...0
...0.0..0.
...0..0..
.....0.
...0.....0
...0...0.
....0.0..

Shortest PATH:

File name: puzzle_10.txt
Total number of lines: 11
Start at: (2, 8)
1. Move up to (2, 6)
2. Move right to (3, 6)
3. Move down to (3, 10)

Done!
Time Taken: 8 ms
```

Figure 3 - output

Figure 1 show the benchmark testing puzzle file which is puzzle\_10.txt file. And in the figure 2 it shows the output of the program. And it shows the bfs path including the maze, name of the file its number of lines and as well as the time it took . it also show the starting position.

There are three classes in this program which are the “main” method, And then “BFSAlgorithm” class and the “Puzzle” class. BFSAlgorithm class contains the algorithm and The main method displays a welcome message and menu options, allowing users to choose between playing Puzzle Mania, loading a puzzle, or exiting the program, while handling user input validation and executing the selected options accordingly, with an option to perform additional tasks after each operation. Puzzle class that allows users to load puzzles from text files, extract puzzle data such as starting and ending points, and represent the puzzle as a two-dimensional array. Multiple benchmark testing puzzle files are available, and after testing each file, the program displays the correct output with the correct bfs path and checks the time taken for each puzzle.

### 3. Performance analysis of algorithmic design and implementation

<i>Input puzzle</i>	<i>Time for the 1st run</i>	<i>Time for the 2nd run</i>	<i>Time for the 3rd run</i>	<i>Average Time</i>	<i>ratio</i>
10*10	5 ms	3 ms	0 ms	2.66 ms	
20*20	4 ms	5 ms	4 ms	4.33 ms	1.629
40*40	9ms	8ms	1ms	6 ms	1.385
80*80	9 ms	2 ms	1 ms	4 ms	0.667
160*160	13 ms	7 ms	6 ms	8.66 ms	2.165
320*320	37 ms	21 ms	14ms	24 ms	2.769
640*640	257 ms	219 ms	151 ms	209 ms	8.708
1280*1280	1101 ms	1063 ms	1035 ms	1066.33 ms	5.106
2560*2560	7687 ms	7512 ms	7498 ms	7565.66 ms	7.09

The ratios of consecutive items do not show a consistent growth based on the provided values and ratios. Rather, it appears that the ratios are rising. The ratios' growing tendency points to a quadratic relationship as opposed to a linear one. Thus, quadratic ( $O(n^2)$ ) is the best estimate for the chronological complexity.

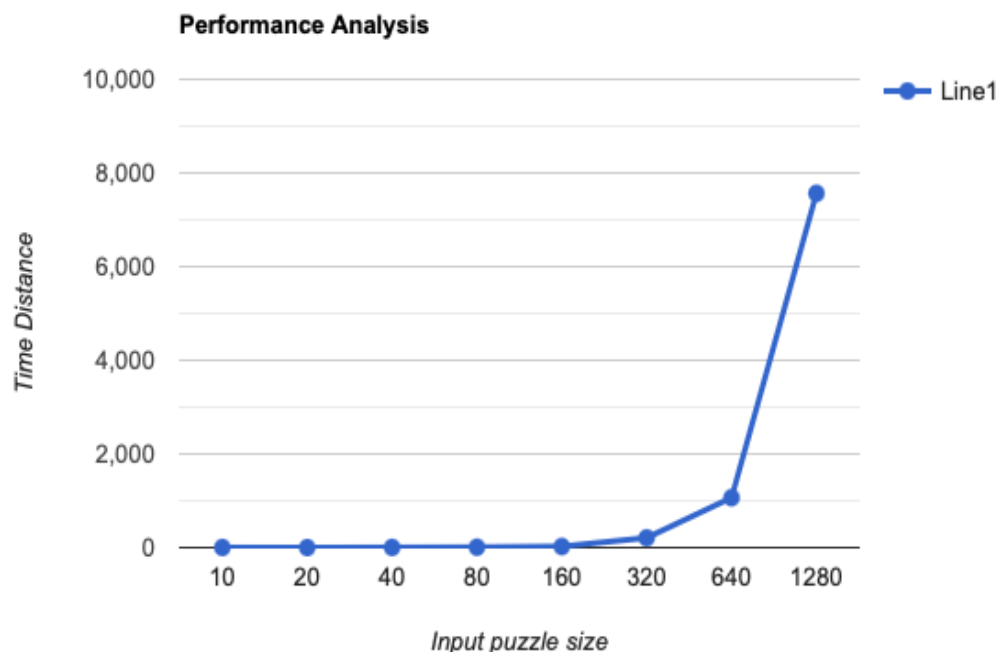


Figure 4 - Performance Analysis Graph