Informatics Institute of Technology

Department of Computing

Algorithm Coursework Report

Module	: Algorithms: Theory, Design and Implementation
Module Code	: 5SENG003C
Module Leader	: Ragu Sivaraman
Date of Submission	:04/24/2024
Student ID	: IIT No-20221574/ UoW No-W1999471
Student First Name	: Milni
Student Surname	: Yaddehi
Batch	: L5 CS
Tutorial Group	: Group H

Task 5

Part A

Data Structure

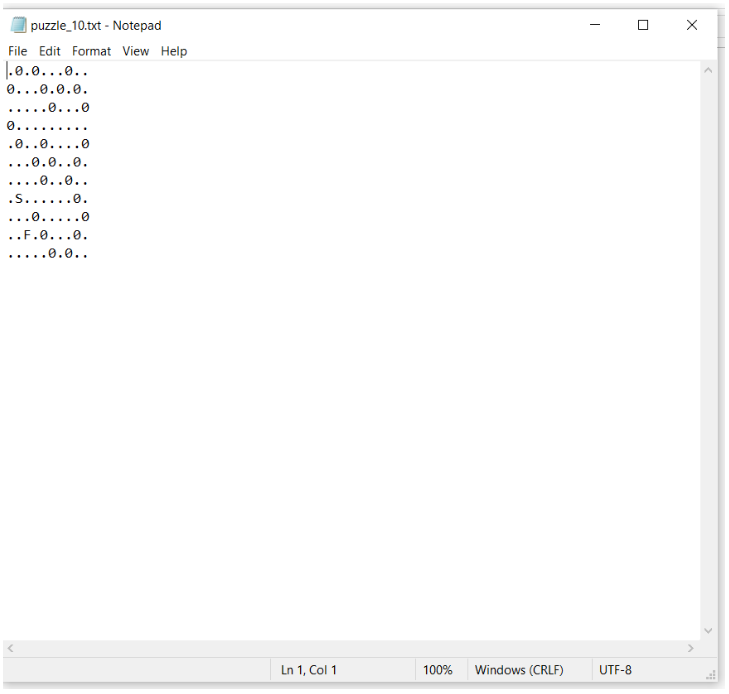
The selection of the data structure is based on how effectively the maze puzzle can be represented and altered. The main data structure used is a two-dimensional integer array (int[][] maze), which is used to describe the maze layout in which every cell represents one of the following states: wall, free path, start point, or end point. With constant-time access to individual cells provided by this array-based representation, navigating and modifying the maze during the solving process can be done with ease. To facilitate quick access to each line of the maze, an ArrayList (lines) is also used to store the lines read from the input file. This dynamic list format makes parsing operations simpler and allows for greater flexibility when managing inputs of varying lengths. In addition, the PathFinder class's implementation of a queue data structure makes it easier to apply the Breadth-First Search (BFS) pathfinding approach.

Algorithm

Regarding the choice of the algorithmic approach, the selection of Breadth-First Search (BFS) is notable as an appropriate option for discovering the shortest path inside the maze problem. BFS is a methodical traversal algorithm that proceeds from the starting point to the target destination, exploring nodes one level at a time. The shortest path is always followed in the first instance of reaching the end point, as ensured by BFS's breadth-first exploration prioritization of nearby cells. Because of this feature, BFS is especially well-suited for situations involving the solving of mazes in which the maze structure is represented as an unweighted graph free of cycles. Furthermore, BFS guarantees optimality in pathfinding without requiring heuristics, which makes it a dependable and effective solution for maze puzzles. The combination of the BFS algorithm with appropriate data structures such as arrays and queues, achieves an effective and scalable solution for solving maze puzzles.

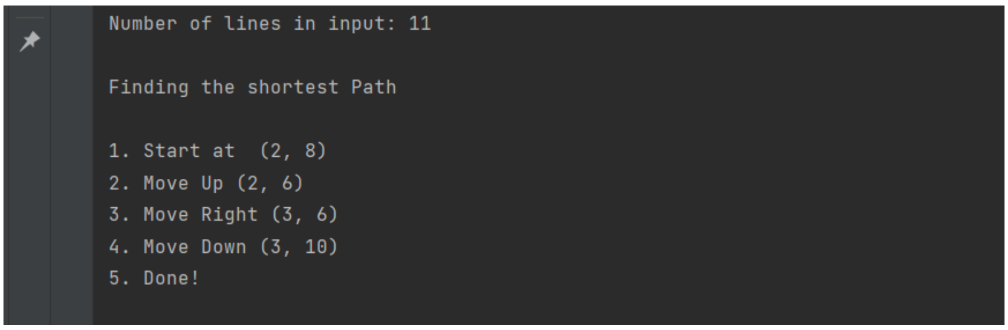
Part B

Example Puzzle



```
.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..
```

Output- Shortest path Found for the Puzzle



```
Number of lines in input: 11

Finding the shortest Path

1. Start at  (2, 8)
2. Move Up (2, 6)
3. Move Right (3, 6)
4. Move Down (3, 10)
5. Done!
```

In this benchmark example, we have a small maze represented as a 10x10 grid. The maze contains walls represented by '0', free paths denoted by '.', the start point indicated by 'S', and the goal point marked as 'F'. This maze presents a simple yet illustrative scenario for testing the efficiency of the pathfinding algorithm.

After importing the maze from the input file, the algorithm starts to run. Using a Breadth-First Search (BFS) based technique, it then moves on to exploring potential routes from the starting point. By avoiding walls and marking visited cells to prevent revisiting, this approach methodically explores nearby cells. Up until the algorithm hits the objective or runs out of possible routes through the maze, exploration will continue.

The program goes backwards to find the shortest route from the objective to the beginning after it reaches the goal point. The path is represented as a string with every step from the beginning to the end specified. To guarantee that the algorithm moves from the beginning to the end of the maze as effectively as possible, this shortest path acts as the best path.

The result displays the shortest path's successful navigation through the maze from the starting point to the target point. It shows the series of actions performed, including movements and direction changes, that lead to the accomplishment of the objective. In addition, even in a small-scale situation, the recorded time elapsed shows how effective the algorithm is in solving the maze.

Part C

Empirically, the algorithm's performance can be assessed on mazes of various sizes. We examine how the algorithm's performance scales with input size by testing with increasing maze dimensions (10x10, 20x20, 30x30, etc.). Understanding real-world efficiency and scalability is gained from analyzing these measurements.

We take the algorithm's time complexity into theoretical considerations. In a maze graph, V is the number of vertices (cells) and E is the number of edges (connections). The time complexity of the Breadth-First Search (BFS) algorithm, which is used for maze traversal, is usually O(V + E). The temporal complexity of a labyrinth with dimensions of n x m is O(n x m) since V and E are both proportional to the maze's area. In the worst-case situation, where there are all connections within the maze, the algorithm visits each cell once, resulting in a time complexity that is linear 0(n) in relation to the size of the maze.

In conclusion, the algorithm performs as O(n x m) according to the proposed order-of-growth classifying (Big-O notation), which suggests a linear relationship between time complexity and maze size.