UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

University of Westminster

Algorithms: Theory, Design and Implementation

5SENG003C.2

Coursework Report

Student Name : H.A.T.S.Samarakoon

Student ID Number : 20221322

UoW Number : w1956102

Tutorial Group : N

Table of Contents

1. Short explanation of Data Structure and Algorithm

Data Structure

- 2D Character Array (‘char[][]’):

  Represents the map, where each character indicates a cell type: ‘S’ (start), ‘F’ (finish), ‘0’ (wall), or walkable path.

- Priority Queue (‘PriorityQueue<int[]>’):

  Manages points on the map during exploration, ordered by priority (cost to explore).

- Boolean Array (‘boolean[][]’):

  Tracks whether a point on the map has been visited.

- Integer Array (‘int[][]’):

  Tracks the number of steps taken to reach each point on the map.

- HashMap (‘Map<String, String>’):

  Tracks the path taken during exploration, mapping each point to its previous point in the path.

These data structures enable the code to efficiently explore the map and find the shortest path from the start to the finish point.

Algorithm

The code uses the A* (A-star) pathfinding algorithm to find the shortest path from a start point (S) to a finish point (F) on a 2D grid map. The algorithm uses a priority queue to prioritize points based on the estimated total cost, the sum of the current cost to reach the point, and a heuristic estimate of the cost from the point to the finish point. The heuristic function used is the Manhattan distance. The algorithm explores valid neighboring points, updating costs and backtracking paths to determine the shortest route to the finish point.

2. Run of small Benchmark Example

File name – “puzzle_10.txt”

```
"C:\Program Files\Java\jdk-19\bin\java.exe"
Shortest path:
1. Move down to (10,2)
2. Done!

Process finished with exit code 0
```

3. Performance Analysis

The program implements a pathfinding algorithm that uses the A* (A-star) algorithm to find the shortest path from a start point (‘S’) to a finish point (‘F’) on a 2D grid map. The program uses a priority queue to prioritize cells based on a heuristic function and the cost from the start point.

The algorithm uses a priority queue to process nodes based on priority (cost + heuristic), with operations in O(log N) time. It traverses a 2D grid and explores each cell’s neighbors, totaling rows * cols cells. The heuristic function is the fast Manhattan distance calculation (O(1)). The overall time complexity is approximately O(N log N), where N is the total number of cells in the 2D map (rows * cols). This complexity stems from priority queue operations for each cell.

The algorithm uses a 2D array to represent the grid, a visited array, a steps array, and a path map for backtracking. The overall space complexity is approximately O(N), where N is the total number of cells in the map (rows * cols), due to the arrays and map used for tracking each cell's state.

The A* algorithm efficiently finds the shortest path in a graph or grid. The heuristic function guides the search, prioritizing nodes closer to the finish point. It is robust, handling different map configurations and sizes while finding the shortest path if one exists. The algorithm's performance depends on the heuristic function and the grid's quality. Empirically, it performs well across various map sizes and configurations, but performance can decline with very large grids or many obstacles.