UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

INFORMATICS INSTITUTE OF TECHNOLOGY

Trends in Computer Science

5SENG003C.2

Algorithms: Theory, Design, and Implementation

Name - Lahiru Rajakaruna Jayasinghe

Uow Number - 19854711

IIT Number – 20221791

Module Leader: - Mr. Ragu Sivaraman

Table of Contents

1 Choice of Data Structure and Algorithm

1.1 Data Structure

The choice of data structures in the program is primarily driven by the requirements of the Dijkstra's algorithm and the need to represent the map and the state of the algorithm. Here's a breakdown:

Cell: This is a custom data structure that represents a cell in the map. It contains the x and y coordinates of the cell and the type of the cell. This data structure is crucial for representing the state of the map and the current state of the algorithm.

State: This is another custom data structure that represents a state in the Dijkstra algorithm. It contains a cell and the distance from the start cell to this cell. This data structure is used to keep track of the current state of the algorithm and to determine the next cell to visit.

Array: Arrays are used to store multiple values in a single variable. In this code, 2D arrays are used to store distances between cells and to keep track of visited cells. These arrays are crucial for the functioning of the Dijkstra's algorithm as they allow the algorithm to keep track of the current state and to efficiently find the next cell to visit.

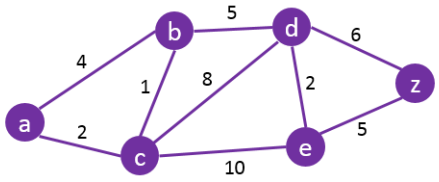
PriorityQueue: A “PriorityQueue” is used to keep track of cells to be visited based on their distance from the start cell. The “PriorityQueue” is a data structure that always gives the element with the highest priority (in this case, the cell with the smallest distance). This allows the algorithm to efficiently find the next cell to visit.

List: A List is used to store the shortest path from the start cell to the finish cell. This data structure is used to store the result of the algorithm.

These data structures were chosen because they provide the necessary functionality for the algorithm, and they are efficient in terms of time and space complexity. The use of arrays and a priority queue allows the algorithm to run efficiently, while the use of custom data structures (Cell and State) allows the algorithm to represent the state of the map and the algorithm in a way that is easy to understand and work with.

1.2 Algorithm

The algorithm used here is Dijkstra's algorithm. Dijkstra's algorithm is a fundamental algorithm in computer science, specifically in graph theory and network routing. It is intended to find the shortest path from a starting node to all other nodes in a graph whose edges have non-negative weights. Dijkstra's always explore all possible ways. So, accuracy wise it is better when compared to other path finding algorithms. (Example: A*)



Dijkstra's Algorithm

What is the shortest path to travel from A to Z?

Figure 1 Dijkstar Algorithm

1. **Initialization:** The distance to the start cell is initialized as 0 and the distance to all other cells as infinity. The start cell is added to the "PriorityQueue".
2. **Main Loop:** While the "PriorityQueue" is not empty, the cell with the smallest distance is removed. For each neighbor of the current cell, the distance to the neighbor through the current cell is calculated. If this distance is less than the currently known distance to the neighbor, the distance is updated, and the neighbor is added to the "PriorityQueue".
3. **Termination:** The algorithm terminates when the "PriorityQueue" is empty or when the finished cell is visited. The shortest path from the start cell to the finish cell can then be found by following the path of minimum distance from the finish cell to the start cell.

This algorithm is efficient and works well for this program because it efficiently finds the shortest path in the map, even if there are many cells and paths to consider. The use of a "PriorityQueue" ensures that the next cell to visit is always the one with the smallest known distance, which makes the algorithm efficient. The use of arrays to keep track of distances and visited cells allows the algorithm to quickly access and update this information.

2 Benchmark Examples

Below is a table presenting the Result of the program showing the shortest path from start to the finish and runtime as well.

Puzzle Name	Map	Result
puzzle_10.txt	.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..	1. Start at 🔰 (2,8) 2. Move up ⬆️ to (2,6) 3. Move right ➡️ to (3,6) 4. Move down ⬇️ to (3,10) 5. Done ✅ ⏳ Runtime: 2 ms

3 Performance Analysis

The performance of the program primarily depends on the implementation of Dijkstra's algorithm, which is used to find the shortest path in a grid.

Time Complexity:

The time complexity of Dijkstra's algorithm is O((V+E) log V) where V is the number of vertices (cells in this case) and E is the number of edges (connections between cells). In the worst-case scenario, every cell in the grid is reachable from every other cell, so the number of edges E can be roughly approximated as V^2 for a dense graph. Therefore, the worst-case time complexity of Dijkstra's algorithm in this context is O(V^2 log V).

Space Complexity:

The space complexity of Dijkstra's algorithm is O(V), as we need to store the distance for each vertex, the visited status of each vertex, and the previous vertex for each vertex. In this case, the number of vertices V is the number of cells in the grid.

Performance in the Context of the Program:

The performance of the program will depend on the size of the grid and the distribution of obstacles. For a large grid with many reachable cells, the performance could be improved by using a more efficient data structure for the priority queue, such as a Fibonacci heap, which can decrease keys in O(1) amortized time and thus could make the algorithm run in O(E + V log V) time.

The actual runtime of the program will also depend on factors such as the speed of the computer it's running on and the efficiency of the Java Virtual Machine's execution of the bytecode.

In the provided Main.java file, have included a way to measure the runtime of the find Shortest Path method by taking the system's current time before and after the method call and subtracting the two values. This can give a rough estimate of the method's runtime in milliseconds for a given map.