Informatic Institute of Technology In collaboration with the University of Westminster

Algorithms: Theory, Design, and Implementation (5SENG003C.2)

Module Leader: Sivaraman Ragu

Task 05: Brief Report

Student Name: Banula Perera

Student IIT Number: 20212085

Student UoW Number: W1871527

a) A short explanation of your choice of data structure and algorithm.

Data Structures Used

Three data structures were used for different functions in the coursework.

- 2D Array - When working with grids it can be advantageous to use a 2D array as a data structure. This structure is like a table and allows for organized and accessible data in rows and columns. Because of its efficient indexing, it is well-suited for scenarios where data is naturally organized in a two-dimensional grid or matrix. In the course work, a 2D array was used to store the values of the maze, along with the x, y coordinates, and distance of each node. This decision was made based on the requirements of the project.

- Priority Queue – A priority queue is the perfect tool for scenarios where elements must be processed based on their priority level. By processing high-priority items first, algorithms like Dijkstra can be easily implemented. The real advantage of a priority queue is how it efficiently manages and processes elements according to their priority, leading to significant performance improvements in various applications. With a priority queue, no need to manually sort elements every time they're added or processed. With the utilization of Dijkstra's algorithm and the implementation of a priority queue, the project has been able to meet its requirements.

- Array List – The Java ArrayList class is like an array with no size limit. It uses a resizable array for storing the elements, making it more flexible. An ArrayList named "path" is used to store the points x and y coordinates of the shortest path and display the output, which is the x and y coordinates of the shortest path along with the direction travelled after each point.

Algorithm Used

Dijkstra's algorithm is a widely used method for finding the shortest path from a starting node to all other nodes in a weighted graph. The algorithm explores neighboring nodes iteratively and updates the shortest path to each node until it reaches the destination node, or all reachable nodes have been visited. It is known for its ability to find the shortest path in non-negative weighted graphs and is therefore popular in applications where optimal paths are essential due to its efficiency. When it comes to implementing this coursework, Dijkstra's algorithm is undoubtedly the best choice. Its numerous advantages, when compared to other options, make it the clear winner.

Advantages of Dijkstra's algorithm

- The process of implementing it is simple and straightforward, with clear steps to follow that make it easy for anyone to get started and see results quickly.
- Dijkstra's algorithm can handle small graphs more efficiently.
- Dijkstra's algorithm finds the shortest path in a graph with non-negative edge weights.

b) A run of your algorithm on a small benchmark example. This should include the supporting information as described in Task 4.

```
Welcome to the Sliding Puzzle Game!
Please share the file path with me: /Users/banulaperera/Desktop/5SENG003W_CW_w1871527_Source_code/5SENG003W_CW_w1871527_Sorce_Code/maze/benchmark_series/puzzle_10.txt

********** Path from the start to the finish **********
1. Start at (2,8)
2. Move up to (2,6)
3. Move right to (3,6)
4. Move down to (3,10)
Done!
**********************************************************
Time taken to execute the algorithm: 5 milliseconds.
```

Figure 1. Run of the algorithm on a small benchmark example (puzzle_10.txt)

```
Do you want to see the visual traveled path of the maze(Y/N)?: y

********** Visual traveled path of the maze **********
.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..
**********************************************************

Thank you for playing the Sliding Puzzle Game!
```

Figure 2. Visual travelled path of the maze

Explanation of the code

The Java code utilizes the BufferReader class to read the user-provided text file and validate the maze format. Once validated, the data is stored in a 2D array and passed through Dijkstra's algorithm. The algorithm considers the x and y coordinates of each point in the maze and stores the resulting values in a separate 2D array. Initially, all distances are set to infinity. The algorithm then checks each neighbouring point to determine if it has been visited and adds it to a priority queue. The minimum distance value is selected from the queue, and the algorithm repeats the process, checking the neighbours of that point until it reaches the finish. The resulting values are stored in a list and the path is printed.

c) A performance analysis of your algorithmic design and implementation. This can be based either on an empirical study, e.g., doubling hypothesis, or purely theoretical considerations, as discussed in the lectures and tutorials. It should include a suggested order-of-growth classification (Big-O notation).

INPUT	TIME 1 (mili)	TIME 2 (mili)	TIME 3 (mili)	AVERAGE	RATION CHANGE	LOG 2 VALUE
10	4	4	4	4	-	-
20	5	5	4	4.666666667	1.166666667	0.222392421
40	5	5	5	5	1.071428571	0.099535674
80	7	5	7	6.333333333	1.266666667	0.341036918
160	6	9	9	8	1.263157895	0.337034987
320	13	13	13	13	1.625	0.700439718
640	27	26	28	27	2.076923077	1.054447784
1280	64	69	73	68.66666667	2.543209877	1.346650524
2560	236	288	261	261.6666667	3.810679612	1.930048317

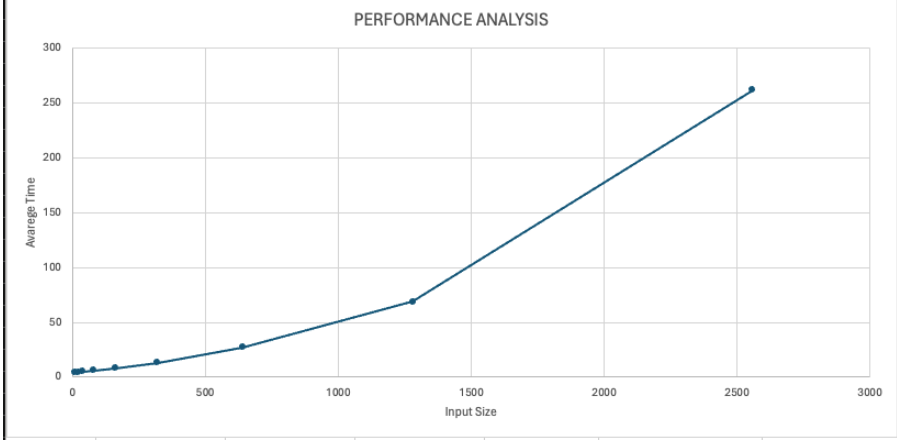

Figure 3 Performance Analysis of the Algorithm

Dijkstra's algorithm typically has a time complexity of O((V+E) log V), where V is the number of vertices and E is the number of edges. However, the Sliding Mechanism used in this implementation slides in all four directions until it hits a wall or rock in the getNeighbors method. In the worst-case scenario, it could traverse the entire height or width of the maze, resulting in a complexity of O(W+H), where W is the width of the maze and H is the height of the maze. The number of vertices (V) in this case is the product of the height and width of the maze (HW), and the number of edges (E) can be at most 4 times the number of vertices (4V) since each cell can have up to 4 neighbours. Thus, the time complexity of this implementation of Dijkstra's algorithm can be considered as O((V+E) log V) = O((HW + 4HW) log (HW)) = O(HW log(HW)), taking into account the specifics of the Sliding Mechanism.