UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

Informatics Institute of Technology

Department of Computing

Algorithms Theory Design and Implementation

5SENG003C.2

Module	: 5SENG003C.2 Algorithms Theory Design and Implementation
Module Leader	: Mr. Ragu Sivaraman
Date of submission	: 2024/04/24
Student ID	: 20221846 /w1985618
Student First Name	: Pawan
Student Surname	: De Silva
Tutorial Group	: SE G - D

Task 5

a) The breadth-first Search (BFS) algorithm solves sliding puzzles.

Overview of Algorithm and Data Structure

The sliding-puzzle program employs a sliding node approach and queue-based traversal technique to explore neighbouring nodes. It identifies the start and finish locations of the puzzle grid and then uses BFS principles to find the shortest path between them. Once at the finish point, the algorithm modifies the grid to display the shortest route, continuously updating the grid and path history.

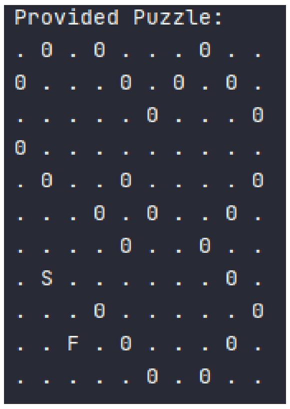
For optimal performance, the algorithm relies on several crucial data structures. A 2D Array stores puzzle elements and represents the grid layout. To avoid revisiting the same nodes, a Boolean Array tracks which locations have already been visited. Additionally, an ArrayList facilitates a more straightforward reconstruction of the traversal path, ensuring a smoother and more efficient pathfinding process.

Reason to Choose This Algorithm

The breadth-first search (BFS) technique finds the shortest path from a start location to the finish position in a puzzle. BFS works through the challenge level by level, visiting every neighbour at each distance level before moving on to the next. This systematic investigation ensures the shortest route is used to reach the final point first, using less memory than other graph search algorithms like Depth-First Search (DFS).

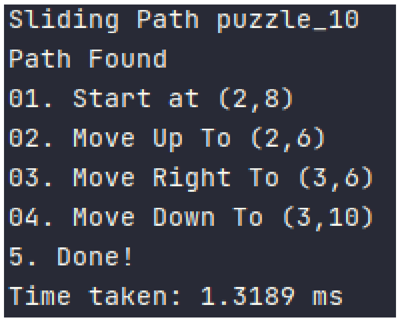
b) Benchmark Example

The shortest path for the provided benchmark file (puzzle_10.txt)



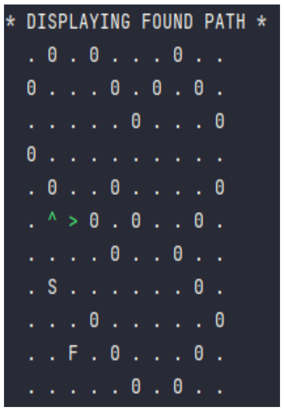
```
Provided Puzzle:
. 0 . 0 . . . 0 . .
0 . . . 0 . 0 . 0 .
. . . . . 0 . . . 0
0 . . . . . . . . .
. 0 . . 0 . . . . 0
. . . 0 . 0 . . 0 .
. . . . 0 . . 0 . .
. S . . . . . . 0 .
. . . 0 . . . . . 0
. . F . 0 . . . 0 .
. . . . . 0 . 0 . .
```

Figure 1 : Puzzle_20.txt



```
Sliding Path puzzle_10
Path Found
01. Start at (2,8)
02. Move Up To (2,6)
03. Move Right To (3,6)
04. Move Down To (3,10)
5. Done!
Time taken: 1.3189 ms
```

Figure 2: Shortest Path Steps



```
* DISPLAYING FOUND PATH *
 . 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 3: Display the Path

This example displays a map with 11 rows and 11 columns, as shown in Figure 1. Figure 2 illustrates the shortest path for the given puzzle, while the path with arrows is displayed on the opposite side.

The game begins at row number 8, column number 2 (S), and the player moves around until they reach the F point. However, to solve this puzzle the shortest way possible, the player should start at point number (2, 8) and move upwards until they reach the rock at (2, 5). The player should then stop at (2, 6). Next, they should turn right and head to position (3, 6) because the rock blocks their way at (4, 6). From there, they should move downwards until they reach the point where the "F" is located, which is at (3, 10). These are the shortest steps to solve the puzzle_10.txt file.

c) Algorithmic Design and Implementation Performance Analysis

Theoretical Considerations

- Algorithm Complexity: The number of cells (vertices) and connections between cells, denoted by V and E, respectively, determine the time complexity of the BFS algorithm.
- Big O Notation for classifying order of growth: Since V is the total number of game board cells, the spatial complexity is written as O (V). This is because every cell must have its unique characteristics noted, including the kind of terrain and traversal condition.
- Time Complexity: The formula to calculate the number of edges in a graph is O(V + E), where V is proportional to the size of the board, and E depends on the arrangement of obstacles. In the worst-case scenario where every cell is connected to every other cell, the value of E can be as large as V squared.

Empirical Study

- Variation in File Size: The program's performance was assessed using input files of different sizes. The application was executed for each file size to measure processing time and resource utilisation. After analysing the data, patterns and trends in performance changes were identified as the size of input files increased.
- Variation in Grid Dimensions: To evaluate the impact of grid size on program performance, we created input files with varying grid dimensions (varying numbers of rows and columns).
- Variation in Complexity: Input files with various obstacle configurations and locations were created to assess program performance at different levels of complexity.

Conclusion

The Sliding Puzzles Game program reads input files with a time complexity of O(n^2) (Big-O notation = Quadratic (O(n^2))) and then uses BFS, which has a time complexity of O(V + E), to find the shortest path. This approach works effectively, and empirical testing with various file and grid sizes confirms its scalability. The software offers a practical solution for sliding problems by balancing computing efficiency and usability. However, running on more significant and complex grids takes more time.

Input Size	1st Time (s)	2nd Time (s)	3rd Time (s)	Average (s)
11 x 11	0.0012	0.0012	0.00152	0.0013
21 x 21	0.0017	0.0013	0.0017	0.0015
42 x 42	0.0024	0.0022	0.0027	0.0024
84 x 84	0.00364	0.00287	0.00348	0.0033
168 x 168	0.00778	0.01094	0.00709	0.0086
336 x 336	0.0181	0.0234	0.0173	0.025
672 x 672	0.0494	0.0418	0.0448	0.045
1344 x 1344	0.0716	0.0625	0.0659	0.066

Table 1 : Different times with Average