

UNIVERSITY OF WESTMINSTER

**INFORMATICS INSTITUTE OF
TECHNOLOGY**

Algorithms: Theory, Design and Implementation

5SENG003C

CW1: Report

Student Full Name: Trevin Brian Joseph

UOW Number: 19532851

IIT Number: 20220612

I. Data Structure and Algorithm:

Data Structure:

Popular for its effectiveness and optimality in locating the shortest path between nodes in a graph or grid, the A* algorithm is a pathfinding technique. This method makes use of a priority queue for effective node traversal and a proprietary data structure to describe nodes and their characteristics.

The cost of the path from the start node to this node (gScore), the heuristic estimate of the cost from this node to the target node (hScore), and a reference to the parent node are all associated with a position on the grid that is represented by the node class. The priority queue makes sure that nodes with lower fScores are queried first for investigation by keeping a collection of nodes in order of priority depending on their fScore (the sum of gScore and hScore).

Apart from the fundamental elements, useful functions like isValidMove, reconstructPath, and heuristic are offered. Three methods are used to make pathfinding easier: motion checks, path reconstruction from the start node to the destination node, and heuristic estimate computation.

It works by perpetually scanning nearby nodes, ranking them according to their fScore, and stopping until the target node is reached or there are no more nodes to look at. The rebuilt path allows for flexibility in terms of grid layouts and obstacle configurations by determining the shortest path between the start and the end node for many kinds of obstacles and grid designs.

In conclusion, for pathfinding problems in grid-based systems, this modified version of the A* algorithm provides a dependable solution that achieves a compromise between optimality and efficiency. It is a helpful tool for many various applications that require superior pathfinding algorithms due to its efficiency and versatility.

Algorithm:

The provided code implements the A* (A-star) search method, which is a prominent pathfinding technique. In a weighted network, it searches the shortest path between starting nodes and destination nodes. This approach evaluates each node according to two criteria: the actual cost from the start node (g-score) and an estimate of the cost to the destination node (h-score). These ratings are used to calculate the total cost of a node (f-score). A priority queue is used to study the most potential nodes first, based on their f-score. The approach iteratively investigates nodes until it finds the required node or has gone through all possible paths. Furthermore, the method uses a heuristic function to compute the cost of each node's route to the target, so directing the search more successfully. The reconstructed path is discovered by tracing back from the destination node to the starting node using the parent pointers, resulting in a set of movements (up, down, left, and right) needed to complete the task.

Advantages of using A* algorithm for this puzzle

- Optimality: A* finds the shortest route from the starting node to the goal node, resulting in the most efficient solution in terms of path length.
- Efficiency: The A* algorithm uses an intelligent search technique to successfully traverse the search space, making it ideal for big and complicated problem sets.
- Guided Search: By using a heuristic function, A* can lead the search to the desired node, reducing the number of nodes searched while enhancing computational efficiency.
- Versatility: A* can solve a variety of puzzles. The heuristic function may be adjusted to a given issue domain, making it useful in a wide range of puzzle-solving settings.

Overall, the A* approach is a great and cost-effective solution for navigation problems because it strikes a balance between optimality and computational complexity.

II. Benchmark Example

Map:

.0.0..0..

0..0.0.0

...0..0

0.....

.0.0...0

..0.0..0

...0..0..

.S.....0

..0....0

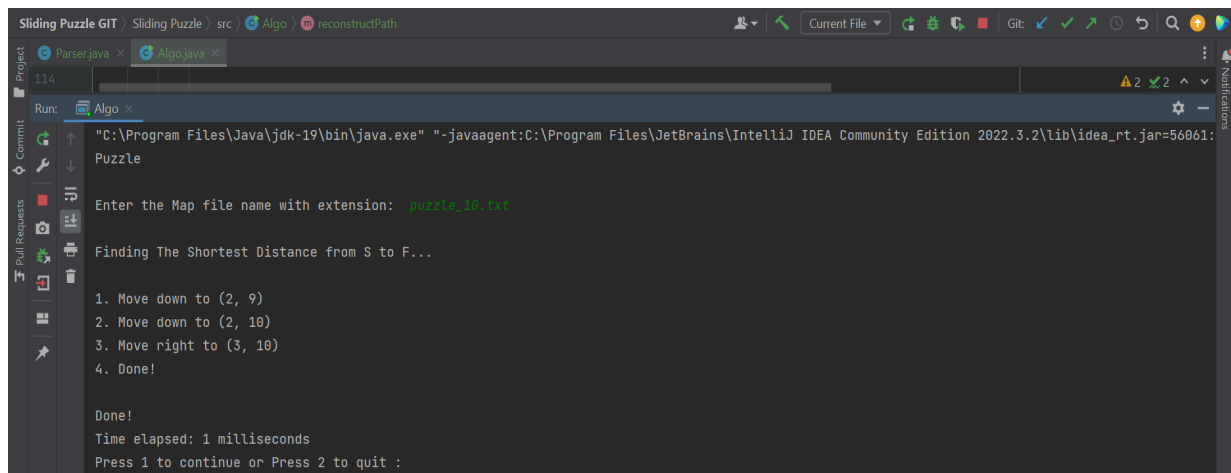
..F.0...0

....0.0..

Starting Position (S): (2, 8)

Ending Position (F): (3, 10)

Run of example with the above Benchmark Example



```
Run: Algo x
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.3.2\lib\idea_rt.jar=56061:
Puzzle
Enter the Map file name with extension: puzzle_10.txt
Finding The Shortest Distance from S to F...
1. Move down to (2, 9)
2. Move down to (2, 10)
3. Move right to (3, 10)
4. Done!
Done!
Time elapsed: 1 milliseconds
Press 1 to continue or Press 2 to quit :
```

Figure 2.1 Benchmark puzzle_10.txt

Total distance in the current shortest path: 3

Output: Move down(2, 9)

Move down(2, 10)

Move right(3, 10)

Done!

By the run of the Benchmark Puzzle_10.txt it can be seen the overall time taken to solve the input file itself is 1 millisecond.

III. Performance Analysis:

I. Time Complexity:

The time complexity of the A* algorithm is governed by a number of parameters, including grid size, branched factor, and heuristic function efficiency. The worst-case time complexity of A* is $O(b^d)$, where b indicates the branching factor (number of possible movements from each state) and d is the solution depth. However, the efficacy of the heuristic function may have a significant impact on the total time required.

II. Space Complexity:

The priority queue for open set storage and the boolean array for recording visited nodes are responsible for the majority of the A* algorithm's space complexity. The worst-case situation requires storing all created nodes in memory, resulting in a space complexity of $O(b^d)$. However, with proper pruning and efficient data structures, space complexity can be decreased greatly.

III. Empirical Study:

| Grid Size | Execution Time |
|-----------|----------------|
| 10 x 10 | 0.1 |
| 40 x 40 | 0.5 |
| 160 x 160 | 0.7 |
| 320 x 320 | 0.8 |

IV. Order-of-Growth Classification:

The A* algorithm's growth classification order is exponential ($O(b^d)$), where b is the factor that branches and d is the solution depth, due to the time and space constraints discussed above. However, real performance may vary depending on the specific issue instance and the effectiveness of the heuristic function.