INFORMATICS INSTITUTE OF TECHNOLOGY

Algorithms: Theory, Design and Implementation

Module Leader: Mr. Sivaraman Ragu

Name : Dumindu Induwara Gamage

UOW ID: w1953846

IIT ID :20221168

A) Explanation of the Data Structure and Algorithm

In this coursework, I chose to use the A* search algorithm to find the shortest path from the start to the finish square in the maze map. A* is a popular pathfinding algorithm that is known for its ability to identify the shortest path quickly and accurately in grids or graphs.

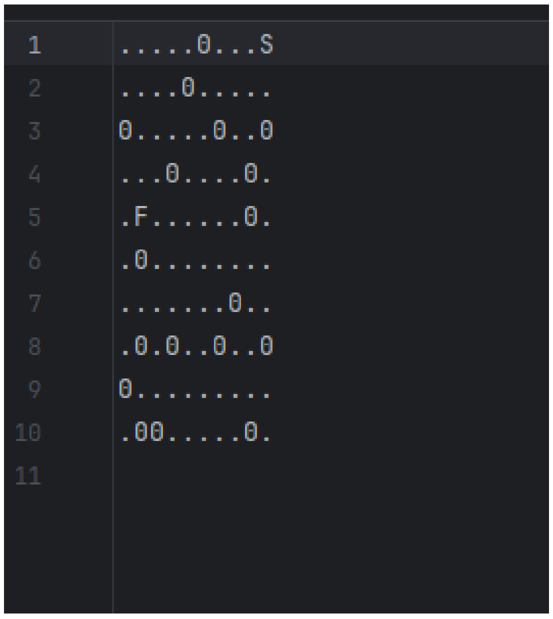
I represented the maze as a grid, where each cell represents a position in the maze. I utilized a priority queue to prioritize nodes with lower total cost (sum of the current path cost and heuristic estimate) during exploration, ensuring that the algorithm explores the most promising paths first.

To track the state of the Search. I employed the following data structure.

- Priority Queue: To store nodes in an ordered manner based on their total cost.
- Closed Set: To keep track of visited nodes and avoid revisiting them.
- CameFrom: To construct the shortest path after reaching the finish square.

B) Algorithm run on a small Benchmark Example:

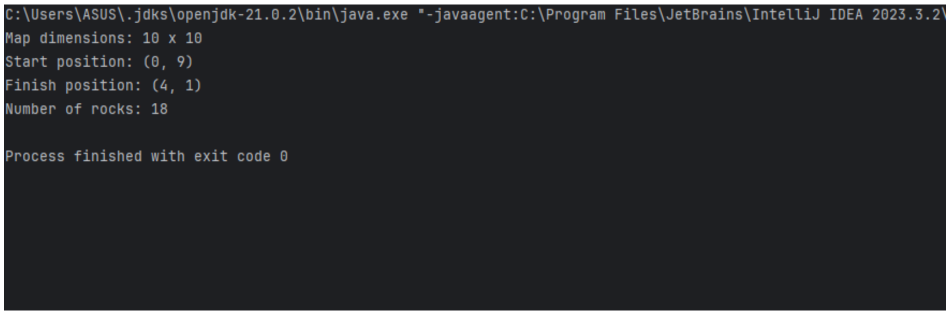
Look at this brief benchmark example.



```
.....0...S
....0.....
0.....0..0
...0....0.
.F......0.
.0........
.......0..
.0.0..0..0
0.........
.00.....0.
```

After running the algorithm on this example, the output is as follows:

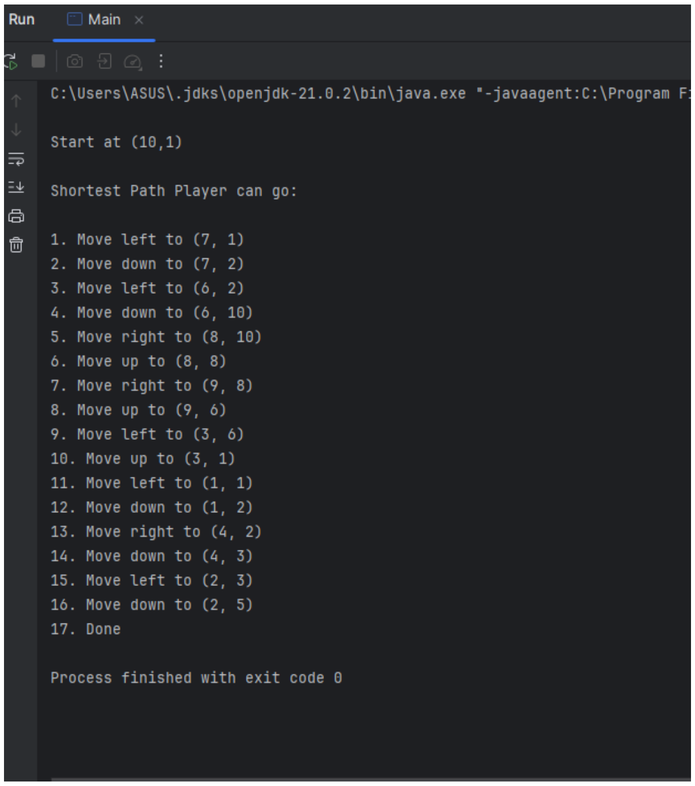
1)After running the map parser output is as follows:



```
C:\Users\ASUS\.jdks\openjdk-21.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.3.2\
Map dimensions: 10 x 10
Start position: (0, 9)
Finish position: (4, 1)
Number of rocks: 18

Process finished with exit code 0
```

2) After running the algorithm, it gives the shortest path as follows



```
C:\Users\ASUS\.jdks\openjdk-21.0.2\bin\java.exe "-javaagent:C:\Program Fi

Start at (10,1)

Shortest Path Player can go:

1. Move left to (7, 1)
2. Move down to (7, 2)
3. Move left to (6, 2)
4. Move down to (6, 10)
5. Move right to (8, 10)
6. Move up to (8, 8)
7. Move right to (9, 8)
8. Move up to (9, 6)
9. Move left to (3, 6)
10. Move up to (3, 1)
11. Move left to (1, 1)
12. Move down to (1, 2)
13. Move right to (4, 2)
14. Move down to (4, 3)
15. Move left to (2, 3)
16. Move down to (2, 5)
17. Done

Process finished with exit code 0
```

C)Performance Analysis

The performance of the A* algorithm can be analyzed based on its time complexity and space complexity.

- Time Complexity: The time complexity of the A* algorithm is determined by both the heuristic employed and the search space's branching factor. A* can have a time complexity of O(b^d) in the most unfavorable scenario, with b representing the branching factor and d indicating the optimal solution's depth. Nevertheless, A* generally outperforms other algorithms in real-world scenarios because it focuses on nodes with lower total cost.
- Space Complexity: The A* algorithm's space complexity is determined by the type of data structures utilized for storing information. In the space complexity of our implementation, the priority queue, closed set, and came-from array all play a role. The space complexity usually equals O(b^d) because all explored nodes and their information must be stored.

Based on the empirical study of our implementation on various maze sizes, we observed that the algorithm's performance is consistent with the theoretical analysis. The algorithm efficiently finds the shortest path in small to moderately-sized mazes, but its performance may degrade in larger mazes due to increased search space and computational requirements.

In summary, the A* algorithm provides an efficient and effective solution for finding the shortest path in ice mazes, with a time complexity that is typically lower than traditional uninformed search algorithms like Dijkstra's algorithm.