# Maze Solver

## Question 1.1

A search problem has a state space, nodes and arcs. Therefore, for the maze solver to be seen as a search problem, each of these items needs to be defined by the problem.

The state space of the maze solver would be a tree, with the branches being the arcs.

The arcs of the maze solver would be the adjacencies between the empty ‘-‘ characters.

The nodes in this representation would be the empty ‘-‘ characters.

## Question 1.2

### Part 1

1. The Depth-First algorithm operates by first selecting the first direction it encounters.
2. It then searches all the nodes along this direction, labelling each as “visited” once they have been passed for the first time.
3. If at any point the goal node is found along this direction, the path up to this node is returned and the algorithm terminates.
4. Otherwise, once it reaches the end of this direction, it backtracks until it finds a new direction to explore, and the search fails if no other directions are found.
5. When a new direction is discovered, the algorithm repeats steps 2-5.

### Part 3

In my implementation, I represented nodes by a 2D cartesian coordinate grid, with (0, 0) being the top left position in the maze.

More specific details are available in the README.txt file.

When testing the performance of my algorithm, I used the commands provided in the README.txt file.

I also enabled maze output (mazeOutput in ‘Constants.h’).

Easy Maze

|  |  |
| --- | --- |
| Number of nodes visited | 77 |
| Number of steps in final path | 27 |
| Execution time | 0.013s |

On a small number of nodes, the depth-first search is very quick, taking less than 0.01 seconds to run from compile time.

### Part 4

Medium Maze

|  |  |
| --- | --- |
| Number of nodes visited | 4782 |
| Number of steps in final path | 449 |
| Execution time | 0.231s |

Large Maze

|  |  |
| --- | --- |
| Number of nodes visited | 22372 |
| Number of steps in final path | 1120 |
| Execution time | 0.827s |

Very Large Maze

|  |  |
| --- | --- |
| Number of nodes visited | 468366 |
| Number of steps in final path | 3745 |
| Execution time | 4.749s (mazeOutput = false)  51.577s (mazeOutput = true) |

## Question 1.3

### Part 1

* The A\* algorithm would produce a more optimal solution than DFS, provided a good heuristic function is used.
* The A\* algorithm would be an improvement over depth-first search, especially on larger graphs.
* This is because there is no decision making in DFS; it just travels down the next direction until it hits a wall, and this is extremely inefficient in larger graphs, with the number of nodes needing to be explored being in the millions for the “maze-VLarge.txt”.
* The use of a heuristic algorithm (A\*) would drastically reduce this time for larger graphs.
* However, A\* may be slower on trivial graphs, due to the added time cost of a heuristic.

### Part 2

### Part 3

#### Statistics

Easy Maze

|  |  |  |
| --- | --- | --- |
|  | **A\*** | **DFS** |
| Number of nodes visited | 36 | 77 |
| Number of steps in final path | 27 | 27 |
| Execution time | 0.004s | 0.004s |

#### Analysis

For the easy maze, similar results were obtained for the depth-first and A\* algorithms – both outputted paths of length 27, and both had an execution time of 0.004 seconds.

These results suggest that for small mazes, my A\* and DFS algorithms have similar performance and optimality.

### Part 4

#### Statistics

Medium Maze

|  |  |  |
| --- | --- | --- |
|  | **A\*** | **DFS** |
| Number of nodes visited | 513 | 4782 |
| Number of steps in final path | 331 | 449 |
| Execution time | 0.014s | 0.055s |

Large Maze

|  |  |  |
| --- | --- | --- |
|  | **A\*** | **DFS** |
| Number of nodes visited | 40275 | 22372 |
| Number of steps in final path | 974 | 1120 |
| Execution time | 7.984s | 0.218s |

Very Large Maze

|  |  |  |
| --- | --- | --- |
|  | **A\*** | **DFS** |
| Number of nodes visited | 272249 | 468366 |
| Number of steps in final path | 3691 | 3745 |
| Execution time | 656.968s | 4.749s |

#### Analysis

Up to the **Medium** maze, A\* runs faster than DFS due to being more optimised. However, the A\* algorithm ultimately has a much higher time complexity than DFS, and so its runtime speed suffers drastically on the **Large** and **Very Large** mazes.

The A\* algorithm produces a more optimal path for every maze type, with the exception of the **Small** maze, where DFS finds the same route.

For larger mazes, DFS is much quicker and produces a minimally less optimal result, outputting a path of generally <10% more steps than that of the A\* search.

However, for situations in which the optimality of a maze solver is more important than its runtime speed, the A\* algorithm is a better choice.

#### Output

* All maze paths output to the file PathOutput.txt, and these results can be seen in the ‘paths’ folder.
* Snippets of the outputs for easy, normal and very large are below:

*as\_easy.txt as\_normal.txt as\_vlarge.txt*

### 

--- A\* SEARCH EASY [mazes/maze-Easy.txt] ---

(1, 0)

(1, 1)

(2, 1)

(3, 1)

(4, 1)

(5, 1)

(5, 2)

(5, 3)

(5, 4)

(5, 5)

(6, 5)

(7, 5)

(8, 5)

(8, 6)

(9, 6)

(10, 6)

(11, 6)

(12, 6)

(13, 6)

(14, 6)

(15, 6)

(16, 6)

(17, 6)

(17, 7)

(17, 8)

(18, 8)

(18, 9)

--- A\* SEARCH MEDIUM [mazes/maze-Medium.txt] ---

(1, 0)

(1, 1)

(1, 2)

(1, 3)

(2, 3)

(3, 3)

(3, 4)

(3, 5)

(4, 5)

(5, 5)

(5, 6)

(6, 6)

(7, 6)

(8, 6)

(8, 7)

(9, 7)

(9, 8)

(9, 9)

(9, 10)

(9, 11)

(9, 12)

(9, 13)

(9, 14)

(9, 15)

(9, 16)

…

(198, 99)

--- A\* SEARCH VLARGE [mazes/maze-VLarge.txt] ---

(1, 0)

(1, 1)

(1, 2)

(1, 3)

(1, 4)

(1, 5)

(1, 6)

(1, 7)

(2, 7)

(2, 8)

(2, 9)

(2, 10)

(2, 11)

(2, 12)

(3, 12)

(4, 12)

(5, 12)

(6, 12)

(6, 13)

(6, 14)

(6, 15)

(6, 16)

(5, 16)

(5, 17)

(4, 17)

(4, 18)

(4, 19)

…

(1880, 999)