# 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

Steps in the Depth-First algorithm:

1. First, select the first direction you encounter.
2. Search 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, return the path up to this node and terminate.
4. Otherwise, once you reach the end of this direction, backtrack until you find a new direction to explore. The search fails if no other directions are found, and it can be concluded that the goal is not solvable.
5. When a new direction is discovered, repeat steps 2-5.

### Part 2

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

The solution to the maze is printed to the file PathOutput.txt, as printing the larger mazes to the console would be impractical.

Identical pre-made PathOutput.txt solutions are also present under the paths folder for every maze type, for both A* and depth-first search.

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

### Part 3

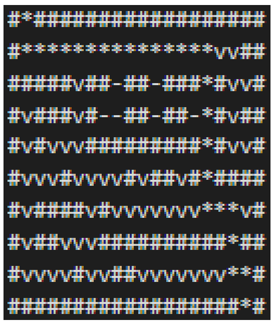
When testing the performance of my algorithm, I used the commands provided in the README.txt file. I also enabled maze output (mazeOutput = true in 'Constants.h').

#### *Statistics*

Easy Maze

| | |
|---|---|
| Number of nodes visited | 77 |
| Number of steps in final path | 27 |
| Execution time | 0.004s (mazeOutput = false)<br>0.006s (mazeOutput = true) |

Path travelled for the easy maze (see README.txt for legend):



```
#*##################
#***************vv##
#####v##-##-###*#vv#
#v###v#--##-##-*#v##
#v#vvv#########*#vv#
#vvv#vvvv#v##v#*####
#v####v#vvvvvvv***v#
#v##vvv##########*##
#vvvv#vv##vvvvvvv**#
##################*#
```

### *Analysis*

This result shows that on a small number of nodes, depth-first search (DFS) is very quick. Additionally, a large portion of the time is taken by outputting to the MazeOutput.txt file, which can be circumvented by setting mazeOutput = false in “Constants.h”.

Furthermore, the maze output shows that DFS explores nearly every node in the maze, leading to a huge number of total nodes explored, when only a small portion of them is included in the final path.

This suggests that an algorithm which uses a heuristic function may be able to produce more optimal results.

## Part 4

### *Statistics*

<u>Medium Maze</u>

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

<u>Large Maze</u>

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

<u>Very Large Maze</u>

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

### *Analysis*

The above statistics suggest that the depth-first search implementation is of polynomial time complexity.

The ratio between the **Medium** and **Large** mazes for execution time is 0.231/0.827 = ~0.279, and for size is 20000/180000 = ~0.111.

The ratio between the **Medium** and **Very Large** mazes for execution time is 0.231/51.577 = ~0.004, and for size is 20000/2000000 = 0.01.

Therefore, the ratio of size difference to execution time difference (ratio of ratios) for Medium and Large is 0.279/0.111 = ~2.514, and the ratio of size difference to execution time difference for Medium and Very Large is 0.004/0.01 = 0.4.

As such, the execution time difference increases faster than the number of nodes, hence a time complexity higher than O(n).

## Question 1.3

### Part 1

I will be using the A* search algorithm to solve the maze. The A* algorithm produces a more optimal solution than Depth-First search, since it evaluates the cost of travelling down each path when finding the solution. Rather than travelling down the nearest path until a wall has been reached, the A* algorithm uses a heuristic algorithm to determine which path to travel down next, based on the distance from the goal and the total length of the path so far.

The A* algorithm works as follows in simple terms:

1) Pick a start node.
2) Add all possible adjacent nodes from this node to the list of potential paths.
3) Select the adjacent node with the lowest cost using a heuristic algorithm, keeping track of the nodes that came before it with a linked list.
4) If the adjacent node is the goal, output the linked list keeping track of all the nodes before it in order.
5) Otherwise, go back to step 2.

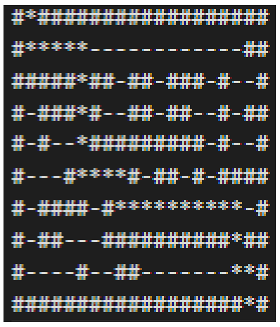
### Part 2

#### *Statistics*

<u>Easy Maze</u>

| | **A*** | **DFS** |
|---|---|---|
| Number of nodes visited | 36 | 77 |
| Number of steps in final path | 27 | 27 |
| Execution time | 0.005s | 0.006s |

Path travelled for the easy maze (see README.txt for legend):



```
#*##################
#*****------------##
#####*##-##-###-#--#
#-###*#--##-##--#-##
#-#--*#########-#--#
#---#****#-##-#-####
#-####-#**********-#
#-##---##########*##
#----#--##-------**#
##################*#
```

#### *Analysis*

For the **Easy** maze, A* performs slightly better than DFS, by 0.008s. Both algorithms output a path with the same number of steps in.

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

The A* algorithm explores far fewer nodes than my DFS algorithm, meaning its heuristic function has been successful.

## Part 3

### *Statistics*

Medium Maze

| | **A*** | **DFS** |
|---|---|---|
| Number of nodes visited | 2046 | 4782 |
| Number of steps in final path | 321 | 449 |
| Execution time | 0.067s | 0.231s |

Large Maze

| | **A*** | **DFS** |
|---|---|---|
| Number of nodes visited | 41900 | 22372 |
| Number of steps in final path | 974 | 1120 |
| Execution time | 9.031s | 0.827s |

Very Large Maze

| | **A*** | **DFS** |
|---|---|---|
| Number of nodes visited | 273919 | 468366 |
| Number of steps in final path | 3691 | 3745 |
| Execution time | 697.413s (mazeOutput = true)<br>656.968s (mazeOutput = false) | 9.023s (mazeOutput = true)<br>4.749s (mazeOutput = false) |

### *Analysis*

Up to the **Medium** maze, the A* algorithm performs relatively well. However, especially between the **Medium** and **Large** mazes, the difference in runtime speed is considerable, with a 0.004 ratio between time taken by Medium and time taken by Large.

The ratio of time taken between Medium and Large is ~0.004, while the ratio of their sizes is a much larger 0.1.

The implication of this is that, when increasing the size of a maze, the difference in time taken to solve the maze increases much faster than the difference in its maze size. Hence the A* algorithm has a much higher time complexity than O(n).

As such, the A* algorithm is considerably less suited to solving large mazes (in speed terms) than DFS.

However, the above results also show that the A* algorithm always produces a path that is equal to or better than the one produced by DFS.

## Part 4

### *Conclusion*

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. Therefore, for most cases, my Depth-First search algorithm is the better choice for solving mazes.

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.

## Further Experimentation, Analysis and Discussion

<u>Experimentation/Discussion</u>

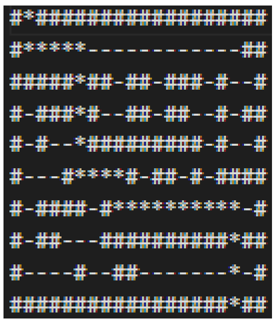
- To begin with, my A* algorithm was incredibly inefficient, and the **Very Large** maze was practically unsolvable.
- After experimenting with performance analysis tools, I was able to improve the performance by nearly 100 times with the following optimisations:
    - Using a single insertion sort pass and pushing from the front, rather than comparing every node's cost in the very large list of potential nodes.
    - Changed the loop checking from "has any node explored" to "has this node explored", as this had a high performance overhead which added up.
    - It was also unnecessary, as loops can only occur within a node.
    - Changed the potential_list type from <vector> to <deque> to improve efficiency of front and back operations.
- Additionally, I originally used node path size + Euclidian distance to the goal as a heuristic, which weighted the node path size higher than the distance to the goal.
    - This is because the Euclidian distance is a hypotenuse distance to the goal, whereas the node path size is not.
    - I replaced this heuristic with node path size + Manhattan distance, as Manhattan distance has better synergy with the node path size.

<u>Analysis</u>

- As my A* algorithm explores far fewer nodes than the DFS algorithm, I believe the heuristic function was sufficient.
- However, if I were to continue this project, I would reconsider many of the design choices I made, such as the decision to check every frame if a node has explored a given position.

<u>Testing</u>

- In order to test the algorithms, I changed the position of the goal node for each of the mazes, and the output was still successful.
- For example, A* on easy:



```
#*##################
#*****------------##
#####*##-##-###-#--#
#-###*#--##-##--#-##
#-#--*#########-#--#
#---#****#-##-#-####
#-####-#**********-#
#-##---##########*##
#----#--##-------*-#
#################*##
```

<u>Output</u>

All maze paths output to the file PathOutput.txt, and these results can be seen in the 'paths' folder.

All visualisations of the solved maze are outputted to MazeOutput.txt.

Snippets of the outputs for easy, normal and very large are below (for the A* algorithm):

*as_easy.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)
```

*as_normal.txt*

```
--- 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)
```

*as_vlarge.txt*

```
--- 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)
```