ECM2423 Coursework Exercise

Running My Code

To run the algorithms, simply execute the solve_mazes.py script. You will be prompted to provide some information to specify which maze and algorithm to run. Firstly, you will need to enter the name of the maze file, such as "maze-Easy.txt". Next, you will be asked to choose which algorithm to run. Lastly, you will have the option to output the maze solution to a file. It's worth noting that since I have already run both algorithms on all mazes, entering "no" may save you time, especially for larger mazes. Once the program finishes executing, it will display some insightful statistics regarding the search algorithm utilised on the given maze.

1.1

The maze solver problem can be framed as a search problem where the goal is to find the path from the start node to the goal node. The problem involves exploring the maze by traversing through the available paths and avoiding walls or blocked paths. To solve this problem, a search algorithm can be implemented to find the optimal path. The maze can be represented as a graph where each node represents a position in the maze, and the edges between nodes represent the possible movements from one position to another.

1.2

The depth-first algorithm is a traversal algorithm that visits all the nodes of a graph or a tree by exploring as far as possible along each branch, before backtracking to explore other branches. It can start at any node in the graph or tree, and the order of visited nodes is determined by the order in which unvisited neighbours are discovered.

Depth-first search statistics

Maze	maze-Easy	maze-Medium	maze-Large	maze-VLarge
Path	solutions/solution-dfs-maze-Easy	solutions/solution-dfs-maze-Medium	solutions/solution-dfs-maze-Large	solutions/solution-dfs-maze-VLarge
Nodes Explored	77	6525	71837	123290
Nodes in Path	27	641	1050	4085
Execution Time (secs)	0.000217	0.046875	0.4375	5.375

My depth-first search algorithm is implemented in the file dfs.py. I drew the following observations about the algorithm's performance based on the results:

- Nodes Explored: As expected, the number of nodes explored increases with the size of the maze, because the algorithm searches the entire maze, exploring more paths.
- Nodes in Path: Again, the number of nodes in the path from start to end increases with the size of the maze. This indicates that the path in larger mazes is longer and more complex than in smaller ones.
- Execution Time: As the size of the maze increases, the execution time of the algorithm increases dramatically. The results indicate that the algorithm's performance degrades significantly as the size of the maze increases, potentially making it unsuitable for larger mazes.

1.3

Another algorithm that can be used to solve mazes is an A* algorithm. It is a heuristic search algorithm that uses both the actual cost of a path from the start node to the current node, and an estimated cost from the current node to the goal node. This estimated cost is usually calculated using a heuristic function such as Euclidean distance or Manhattan distance. The A* algorithm selects the next node to explore based on the sum of the actual cost and estimated cost.

I expect the A* algorithm to perform better than depth-first search in terms of the number of nodes visited, nodes in the path, as well as a better execution time. Depth-first search may traverse long paths in the maze without finding the goal node, while A* uses the heuristic function to guide the search towards the goal node. The heuristic function can often prune large parts of the search space, resulting in a more efficient search. Additionally, A* guarantees finding the shortest path if the heuristic function is admissible (i.e. it never overestimates the cost to reach the goal node).

A* search statistics

Maze	maze-Easy	maze-Medium	maze-Large	maze-VLarge
Path	solutions/solution-astar-maze-Easy	solutions/solution-astar-maze-Medium	solutions/solution-astar-maze-Large	solutions/solution-astar-maze-VLarge
Nodes Explored	56	2059	42006	274136
Nodes in Path	27	321	974	3691
Execution Time (secs)	0.001346	0.015625	0.40625	9.83317

My A* search algorithm is implemented in the file astar.py. I drew the following observations about the algorithm's performance based on the results:

- Nodes Explored: Nodes in Path: As expected, the number of nodes explored increases with the size of the maze, because the algorithm has to explore a larger number of nodes in the search space to find the optimal path.
- Nodes in Path: Again, the number of nodes in the path from start to end increases with the size of the maze. This indicates that the optimal path in larger mazes is longer and more complex than in smaller ones.
- Execution Time: As the size of the maze increases, the execution time of the algorithm increases. The algorithm must evaluate the cost function at each node, which can be time consuming for large mazes.

Based on the results, the A* algorithm generally performs better than the depth-first search algorithm. The algorithm will always find the optimal path, so has fewer nodes in the resulting path than depth-first. A* also typically explores less nodes than the depth-first search, as depth-first may traverse long paths in the maze without finding the goal node, while A* uses the heuristic function to guide the search towards the goal node. The heuristic function can often prune large parts of the search space, resulting in a more efficient search.

The one result that surprised me was how A* has a significantly worse performance on maze-VLarge than depth-first. It takes nearly double the amount of time to execute as depth-first, and explores more than twice as many nodes to find the solution. This may be happening because my heuristic function, which calculates the manhattan distance, is not well suited to solving maze-VLarge, causing it to explore more nodes than necessary. It is also possible that the difference in execution time is due to factors such as my implementation, and could be improved with more efficient data structures.

Overall, the A* algorithm is generally a more efficient way of solving mazes compared to the depth-first algorithm, and also always finds the optimal path. However, the performance of the A* algorithm may not always be the fastest algorithm for solving very large or complex mazes.

Further Experimentation

To experiment further, I wrote a breadth-first algorithm to solve the mazes. The breadth-first algorithm is a traversal algorithm that visits all the nodes of a graph or a tree by exploring all the nodes at a given depth level before moving to the next level. It starts at a given node in the graph or tree and explores all the nodes at a distance of one edge away before moving on to explore nodes at a distance of two edges away, and so on. The order of visited nodes is determined by the order in which nodes are discovered at each level, and it guarantees to visit all nodes in the shortest possible path from the starting node.

I expect my breadth-first algorithm to perform worse than both depth-first and a* for nodes explored, and execution time. However if there aren't any cycles in the maze, breadth-first will always find the shortest path, unlike depth-first.

Breadth-first search statistics

Maze	maze-Easy	maze-Medium	maze-Large	maze-VLarge
Path	solutions/solution-bfs-maze-Easy	solutions/solution-astar-bfs-Medium	solutions/solution-bfs-maze-Large	solutions/solution-bfs-maze-VLarge
Nodes Explored	76	8447	81990	796521
Nodes in Path	27	321	974	3691
Execution Time (secs)	0.00007	0.03125	0.296875	32.65625

Looking at the results, the breadth-first algorithm did find the shortest path through every maze, while also having faster execution times for most of the mazes. On the other hand, for the very large maze the execution time was significantly higher than both depth-first and A*. This could have happened because it explores all nodes at a given depth level before moving to the next level. As a result, breadth-first can use a lot of memory to store the nodes at each level and can become very slow as the number of nodes as well as levels increases.

After analysing the outcomes of all the algorithms and mazes, unexpected results were observed. To ensure the precision of future algorithm testing, it would be valuable to execute the algorithms on a diverse range of mazes with varying structures and sizes. This approach would help to determine whether the previous results were consistent or if those specific algorithms were incompatible with the maze structure at hand.