# Part 1

Question 1:

1. The breadth-first search algorithm uses a queue structure which means that the first node in is the first one out (FIFO). The depth-first search algorithm uses a stack structure so that the last one in is the first one out (LIFO).

Advantages of BFS are:

* It is complete, it is guaranteed that if a solution exists, it will find it.
* If multiple solutions exist, the most optimal one is found.

Disadvantages of BSF are:

* BFS is more time consuming compared to DFS
* If the solution is at the end of the map, it will search all the cells before reaching it
* Consumes more memory since it stores all the cells and its neighbours that need to be visited.

Advantages of DFS are:

* Memory requirement is dependant on the number of nodes and is less compared to BFS.

Disadvantages of DFS are:

* It isn’t complete, it may not find a solution even if one exists.
* May get stuck on an infinite loop.
* Not always optimal, it may not always find the shortest path.



|  |  |  |
| --- | --- | --- |
| Chart, histogram  Description automatically generated | Chart  Description automatically generated | A picture containing text, clock, scoreboard  Description automatically generated |
| Figure 1: slightly inefficient path using BFS | Figure 2: inefficient path using DFS | Figure 3: inefficient search using BFS |

|  |  |  |
| --- | --- | --- |
| Planner | Sum of all path costs | Sum of all cells visited |
| Breadth-first search | 580.47 | 13050 |
| Depth-first search | 7538.8 | 32296 |

Table 1

As we can tell from the results found in Table 1, breadth-first search is the better performing algorithm, since the path costs and the number of cells visited are significantly lower compared to the depth-first search algorithm. Even though breadth-first seems to have significantly better results than depth-first search, it’s still in some cases not sowing the most optimal path (Figure 1). In Figure 1, the path is not optimal and instead it’s going too far into the right. The reason for that is since duplicates are not resolved the planner uses the first path found. Additionally, the breadth-first planner doesn't consider path cost, which means the path found may not be the shortest. You can tell by looking at Figures 2 and 3 that depth-first search is highly dependent on the start and goal locations. If the goal happens to be in a different direction than the planner, it may result in a large search.



|  |  |  |
| --- | --- | --- |
| Planner | Sum of all path costs | Sum of all cells visited |
| Breadth-first search | 580.47 | 13050 |
| Depth-first search | 7538.8 | 32296 |
| Dijkstra’s | 1374.1 | 19062 |

Question 2:

A Star

* 1. [Insert Image of the pseudocode]

The A star algorithm would be implemented similar to the Dijkstra algorithm. Q would be implemented as a Priority Queue, where Q.GetFirst() retrieves the state with highest priority.

In this case, the priority is determined by an estimate of the minimal total path length from current state to the goal states. The estimate is calculated by adding the actual Cost to Come from the start state to that state with a estimated Cost to Go to the goal state calculated using a heuristic, and is inserted along with the state using Q.Insert().

Resolving duplicate is done in the same fashion as in Dijkstra, where the path with the lower cost to get to that state is chosen, the parent of the cell is updated to the parent which gives that lower path cost, and the corresponding cell within the Priority Queue is updated with that lower path cost.

* 1. When a heuristic is admissible, it means that it never gives an over-estimate to the actual cost-to-go value from current state to the goal and is non-negative.

An example of an admissible heuristic is the Euclidean distance because firstly it is never negative due to the addition of the inner squares, and also it never over estimates the cost to go from current state to the goal state as the path between two points can only be as little as the straight line distance between them.

An example of inadmissible heuristic is a heuristic which yields the Euclidean distance, but perceives one or more of the states on the only actual optimal path as having a very high cost (e.g. in extreme cases higher than the maximal total path length of the given problem) because although it is always positive, this heuristic yields an extensive over-estimate to the Cost to go of the current state to the goal.

This reason why admissible heuristics are important is because taking the two function described above, when using the inadmissible heuristic the A star algorithm could reach the goal state before fully exploring the actual optimal path due to the heuristic’s perceived high cost on the optimal states hence would have the lowest priority within the priority queue. This means that the algorithm would not provide an optimal path with minimal cost to the goal, which is undesired. On the other hand, for admissible heuristics such as Euclidean distance, would allow the A star algorithm to find an optimal path because …

An example of an inadmissible heuristic which still guarantees to provide an optimal solution is an admissible function with an additive constant, such as the Euclidean distance f(x) with a constant c = maximal path length of the given problem. Hence the total estimated path cost becomes cost\_to\_come + f(x) + c. The constant wouldn’t affect the way in which the algorithm explores as it wouldn’t change the priority of the states, however it makes the heuristic inadmissible as the heuristic now yields an overestimate to the path.

* 1. The optimal heuristic that A\* can use is … Could it be used in practice? Yes/No because ….
  2. See code
  3. Evaluate performance of implementation …