David Lim

A16398479

04/18/25

MAE 242

Homework 1 Programming

1. Depth first search (DFS) algorithm

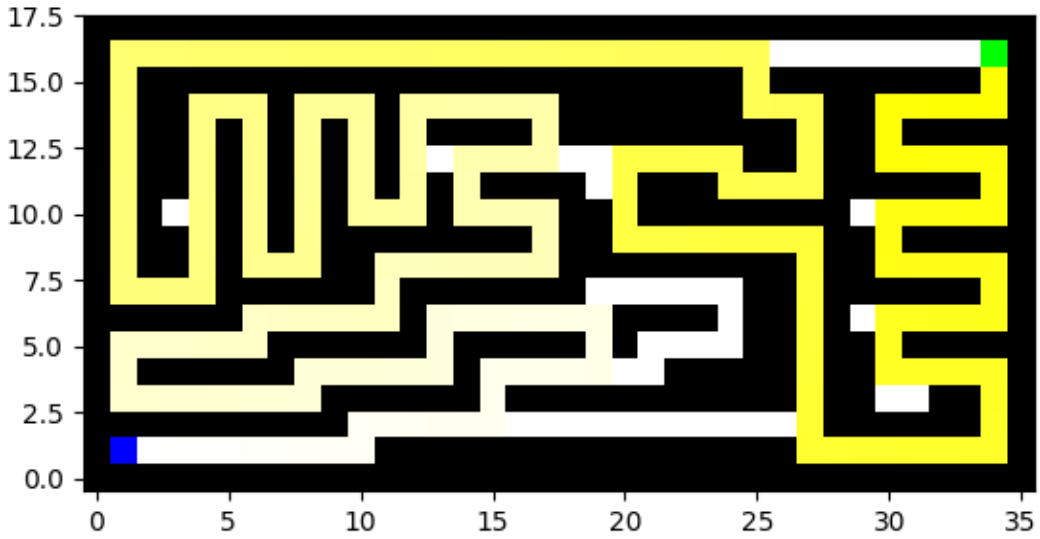

Figure 1. DFS algorithm applied to the medium maze.

Figure 1 shows a plot of the path returned by a depth first search (DFS) algorithm. Table 1 shows the performance of the algorithm. The algorithm was implemented with the preferred order of directions being north, south, east, and west. Given this preference, this path was expected for DFS. For example, at nodes (15, 2) and (17, 12), the algorithm decided to expand a path to the north instead of to the east, which could have led to a shorter path to the goal.

Table 1. DFS algorithm performance

Basic cost:	232 nodes
Explored (expanded):	247 nodes
Remaining in queue:	11 nodes

This was not the least-cost solution. Rather, it was nearly the greatest-cost solution (without self-intersection) since the preferred directions happened to be “unlucky” for this maze. The DFS algorithm prefers exploring in arbitrarily defined directions and ignores other directions which could be more optimal.

2. Breadth first search (BFS) algorithm

Figure 2. BFS algorithm applied to the medium maze.

Figure 2 shows a plot of the path returned by a breadth first search (BFS) algorithm. Table 2 shows the performance of the algorithm. The path obtained was the least-cost path of 68 nodes as expected. However, the BFS algorithm explored 225 nodes before finding the goal.

Table 2. BFS algorithm performance

Basic cost:	68 nodes
Explored (expanded):	225 nodes
Remaining in queue:	3 nodes

3. Dijkstra search algorithm

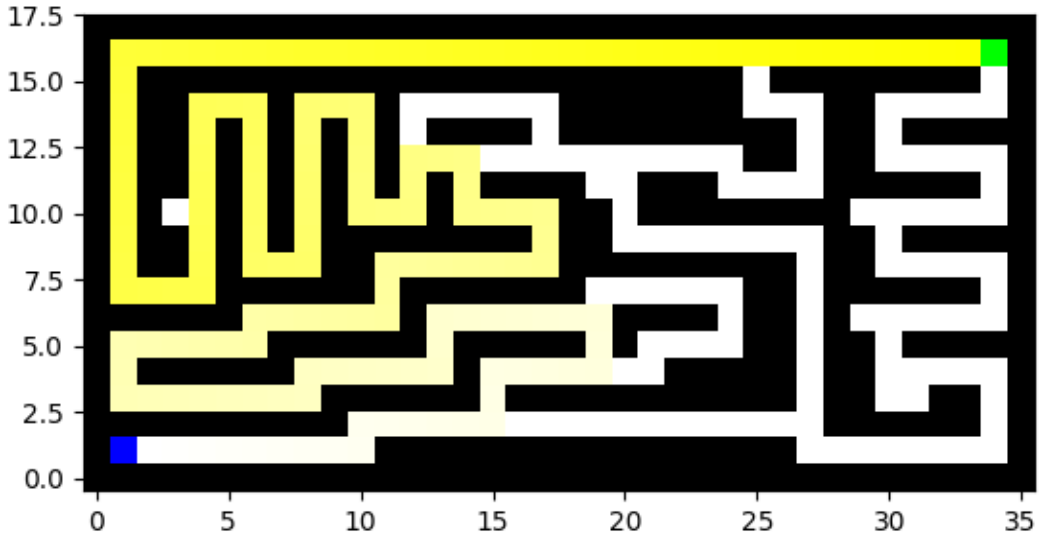

Figure 3. Dijkstra search algorithm applied to the medium maze with a cost function favoring the west.

Figure 3 shows a plot of the path retuned by the Dijkstra search algorithm with the “stay west” cost function. Table 3 shows the performance of the algorithm. The cost function forces the algorithm to find a path that “dwells” in the west, which was the most optimal in the “stay west” cost sense rather than the basic cost sense.

Table 3. Dijkstra search algorithm performance with “stay west” cost

Basic cost:	152 nodes
Stay west cost:	25875
Explored (expanded):	246 nodes
Remaining in queue:	1 node

Figure 4. Dijkstra search algorithm applied to the medium maze with a cost function favoring the east.

Figure 4 shows a plot of the path retuned by the Dijkstra search algorithm with the “stay east” cost function. Table 4 shows the performance of the algorithm. The cost function forces the algorithm to find a path that “dwells” in the east, which was the most optimal in the “stay east” cost sense rather than the basic cost sense.

Table 4. Dijkstra search algorithm performance with “stay east” cost

Basic cost:	74 nodes
Stay east cost:	13629
Explored (expanded):	101 nodes
Remaining in queue:	4 nodes

4. A* search algorithm

Figure 5. A* search algorithm applied to the medium maze with the Manhattan or Euclidean heuristic.

Figure 5 shows a plot of the path retuned by the A* search algorithm using either the Manhattan or Euclidean heuristic. Table 5 shows the performance of the algorithm using the Manhattan heuristic, and Table 6 shows the performance using the Euclidean heuristic. The path obtained using either heuristic was the optimal path with a basic cost of 68 as expected, since either heuristic was admissible and guaranteed optimality. However, the algorithm was slightly more efficient with the Manhattan heuristic by exploring 146 nodes instead of 152 nodes with the Euclidean heuristic.

Table 5. A* search algorithm performance with a Manhattan heuristic

Basic cost:	68 nodes
Explored (expanded):	146 nodes
Remaining in queue:	5 nodes

Table 6. A* search algorithm performance with a Euclidean heuristic

Basic cost:	68 nodes
Explored (expanded):	152 nodes
Remaining in queue:	5 nodes

The Manhattan heuristic led to a more efficient result for the maze because it was a better estimate of the cost to the goal. The Euclidean heuristic underestimates the cost to the goal by measuring the straight-line distance to the goal rather than the rectilinear distance that the agent would travel in the discretized space.