Project 1: Graph Search **Clinton Fernandes**

ME EN 6225 Motion Planning u1016390

1.1 : Run DFS on map0.txt

- Graph display : Proj1-1-1.png
- Path and set of states visited : Proj1-1-1.txt
- The path found is not the shortest path. This algorithm starts the search at the initial state/node and explores as far as possible along each branch before backtracking. Hence, there is a possibility that it may go down one specific path (e.g., left-most path) forever. For a large tree, a depth-first search may take an excessively long time to find even a very nearby goal node.

1.2 : Run DFS on map1.txt

- Graph display : Proj1-1-2.png
- Path and set of states visited : Proj1-1-2.txt
- The path found is not the shortest path as the this was a map with a large grid (or tree), the algorithm took an excessively long time and path even though the goal node was nearby. The idea of DFS is to make a path as long as possible, and then go back (backtrack) to add branches also as long as possible.

1.3 : Run DFS on map2.txt

- Graph display : No path exists for map2.txt
- Path and set of states visited : No path exists for map2.txt

1.4 : DFS algorithm explained in the pseudo code above adds search nodes for all actions

- As the array of actions has a sequence of u, d, l, r, hence while the code is running, the node of the last action (right) is given preference as it is on the top of the stack, hence the next node chosen is the right node, for most of the time. And because of this, even though there may be another node related to the up, down, left actions, which may give a node closer to goal, they are not chosen.
- To avoid this, the selection can be based on weights of path (or cost or distance) instead of a sequence of actions.

1.5 : Reverse the order of the actions explored by DFS and run it again on map1.txt

- Graph display : Proj1-1-5.png
- Path and set of states visited : Proj1-1-5.txt
- The visited set and the path, both change in the result.
- This is because, the search node saved last was of the up action, compared to the right action of the previous case. As the tendency to reach the goal was more, considering the up action first, hence the path and also the visited sets change.

1.6 : Extend DFS to iterative deepening DFS

2.1 : Run BFS on map0.txt

- Graph display : Proj1-2-1.png
- Path and set of states visited : Proj1-2-1.txt
- Yes, for the given graph of map0, this is the shortest path.
- The function of this algorithm is to find the shortest path. BFS starts traversing from the initial node and explores the search in the level by level manner, from the leftmost node, moving towards the right i.e. as close as possible from the root node.

2.2 : Run BFS on map1.txt

- Graph display : Proj1-2-2.png
- Path and set of states visited : Proj1-2-2.txt
- Yes, the algorithm has found out the shortest path. Same reason as above.

2.3 : Run BFS on map2.txt

- Graph display : No path exists for map2.txt
- Path and set of states visited : No path exists for map2.txt

2.4 : Compare Algorithms

- BFS vs DFS : The path found by the BFS algorithm was shorter compared to the one found by DFS. For the map1.txt file, in BFS, about two-thirds of the overall empty states were visited. But in DFS, almost all of the states were visited.
- We also know that BFS uses more memory as it uses queue data structure instead of the stack data structure used by dfs. DFS was easier to implement as it used stacks.
- BFS vs iterative deepening DFS : As opposed to DFS, the depth limit of search in iterative deepening can be controlled. This way, we can combine the space efficiency of depth-first search with the optimality of breadth-first methods.

3.1 : Run Uniform cost search on map0.txt

- Graph display : Proj1-3-1.png
- Path and set of states visited : Proj1-3-1.txt
- For the map0, the path found is the shortest. The cost of traversing from each node to the successive node was taken as 1. As the path is the shortest, so is it the lowest cost path. It is the main feature of the algorithm to determine a path to the goal state that has the lowest weight.

3.2 : Run Uniform cost search on map1.txt

- Graph display : Proj1-3-2.png
- Path and set of states visited : Proj1-3-2.txt
- Yes, the algorithm found out the shortest path for map1, and also the lowest cost path. Uniform cost search always expands the node with the lowest total path cost from the initial node.

3.3 : Run Uniform cost search on map2.txt

- Graph display : No path exists for map2.txt
- Path and set of states visited : No path exists for map2.txt
- No path is possible from initial to goal state for map2

4.1 : Implement a Euclidean distance heuristic for evaluating A*

- for map0.txt : Proj1-4-1-1.png , Proj1-4-1-1.png.txt
- for map1.txt : Proj1-4-1-2.png, Proj1-4-1-2.png.txt
- for map2.txt : no path exists
- The paths were similar in map0 but in map1 they were slightly different. But, the paths found out in both the methods were the shortest. A star explored less number of states than uniform cost search method.

4.2 : Implement a Manhattan distance heuristic for evaluating A*

- for map0.txt : Proj1-4-2-1.png, Proj1-4-2-1.png.txt
- for map1.txt : Proj1-4-2-2.png, Proj1-4-2-2.txt
- for map2.txt : no path exists
- The paths were similar in map0 but in map1 they were slightly different. But, the paths found out in both the methods were the shortest. A star explored less number of states than uniform cost search method.

4.3-1: Using Euclidean distance heuristic

- for map0.txt : Proj1-4-3-1-1.png, Proj1-4-3-1-1.png.txt
- for map1.txt : Proj1-4-3-1-2.png, Proj1-4-3-1-2.png.txt
- for map2.txt : no path exists

Using Manhattan distance heuristic

- for map0.txt : Proj1-4-3-2-1.png, Proj1-4-3-2-1.png.txt
- for map1.txt : Proj1-4-3-2-2.png, Proj1-4-3-2-2.png.txt
- for map2.txt : no path exists
- Yes, the heuristics are very much admissible for the new problem. Because we have seen that the visited node set is reduced compared to other methods.
- Also, it can be seen that the path found out is the shortest path. Without the heuristics, more number of nodes will be visited hence visited set increases. Also, the path found out may not be the shortest pat. The heuristics are important when traversing diagonally across nodes.

5.1 : The Hardest part of the assignment was learning the Python programming language and implementing it within few weeks

5.2 : I did not find anything to be easy

5.3 : The graphs showing the visited states and the path helped me a lot in understanding the methods.

5.4 : Implementing the Uniform Search Cost and A star search code was tedious, however I did not find anything not helpful to my understanding

5.5 : I had a really very hard time in learning and implementing codes in the python language.