Demarco Guajardo
COMP 5600
HW1 - Search Report

I. Design Choices
   A. BFS
      1. I chose to use a double ended queue to implement the queue in BFS, which helped me append and pop from both ends of the queue.
      2. I chose to create a list called visited that told me when a certain node was visited so that I wouldn't revisit them.
      3. For each node, I made sure to explore the neighbors if they had not been visited.
   B. DFS
      1. I chose to use a list to implement the stack in DFS, which helped me append and pop from the end of the stack.
      2. I chose to create a list called visited that told me when a certain node was visited so that I wouldn't revisit them.
      3. For each node, I made sure to explore the neighbors if they had not been visited.
   C. A* Search
      1. I used a heap queue to implement the priority queue in A* Search, which helped me retrieve the node with the lowest cost.
      2. I chose to create a dictionary called the costSoFar dictionary that allowed me to keep track of the cost to reach a certain node.
      3. I chose to create a dictionary called the cameFrom dictionary that allowed me to reconstruct the path once the goal was reached.

II. Implementation Strategy
   A. All algorithms were implemented in Python. The graphs were represented using an adjacency list, and the node coordinates were used in regard to the heuristics I chose to use for A* Search. The performance of each algorithm was measured based on the time taken to find a solution. I used the "time" python tool to measure the execution time of each algorithm.

III. Performance Comparison
   A. TestCase_01
      1. BFS: 0.000406 seconds
      2. DFS: 0.000074 seconds
      3. A* w/ Euc: 0.002111 seconds
      4. A* w/ Man: 0.000074 seconds
      5. A* w/ Cheb: 0.000090 seconds
   B. TestCase_02
      1. BFS: 0.000694 seconds
      2. DFS: 0.000328 seconds
      3. A* w/ Euc: 0.001884 seconds
      4. A* w/ Man: 0.000172 seconds

5. A* w/ Cheb: 0.000199 seconds

C. TestCase_03
    1. BFS: 0.043751 seconds
    2. DFS: 0.016548 seconds
    3. A* w/ Euc: 0.001752 seconds
    4. A* w/ Man: 0.001341 seconds
    5. A* w/ Cheb: 0.001507 seconds

IV. Heuristics Chosen

A. Euclidean Distance
    1. Why I Choose It
        a) Euclidean measures the straight line distance between two points and is suitable when you can move diagonally, and a “robot”, in theory, can move across diagonal spaces, so I chose this heuristic.
    2. Admissible?
        a) Euclidean is admissible. It never overestimates the true cost.
    3. Did it help perform better than Uniformed Searches?
        a) It performed a little worse in Test Cases 01 and 02, but was better for Test Case 03 which gives me the impression that for more nodes, the more efficient it is.

B. Manhattan Distance
    1. Why I Choose It
        a) Manhattan measures distance between two points based on horizontal and vertical distances only. This is applicable to our robot scenario, so I went with this as well.
    2. Admissible?
        a) Manhattan is admissible. It never overestimates the true cost.
    3. Did it help perform better than Uniformed Searches?
        a) It performed the best in all three test cases.

C. Chebyshev Distance
    1. Why I Choose It
        a) Chebyshev measures the maximum distance of the absolute differences in the horizontal and vertical directions. It works best in environments focused primarily on left and right movements, and where diagonal movements are also allowed, so in a sense, works in scenarios similar to that of the other two heuristics.
    2. Admissible?
        a) Chebyshev is admissible. It never overestimates the true cost.
    3. Did it help perform better than Uniformed Searches?
        a) It performed better than Uniform in all three test cases, however for TestCase_01, specifically, it was worse than DFS.