The purpose of this lab is to understand and utilize different search algorithms, including Depth First Search (DFS), Breadth First Search (BFS), Dijkstra's algorithm, and A* search, in the context of a grid-based "box world" environment. The lab focuses on analyzing the behavior and performance of these algorithms in finding paths between start and target positions.
Procedure:
1. Set up the lab environment by running the provided code files, including main.py, game.py, box_world.py, graph.py, and graphics.py.

2.Load and run the box world environment using the sample map files (map1.txt and map2.txt).

3.Observe the behavior of different search algorithms by cycling through them using the 'M' and 'N' keys.

4Modify the box world by setting box types (clear, mud, water, wall) and start/target positions using the provided controls (number keys and mouse clicks).

5. Analyze the search results, including the path, open and closed lists, and route.

6. Adjust the min_edge_cost variable in box_world.py to the correct value (1.0) based on the lowest possible edge cost in the defined box types.

7.Demonstrate cases where the old value (10.0) and the new value (1.0) of min_edge_cost lead to different optimal/non-optimal results in the A* search algorithm.

8.Uncomment the code in the reset_navgraph method of the BoxWorld class to add diagonal edges to the navigation graph and observe the impact on search results.

9. Optionally, create custom maps, add additional box types, costs, and colors, and implement an agent that follows the path.

Observations and Results:

The DFS algorithm explores the graph by traversing as far as possible along each branch before backtracking, resulting in a depth-first exploration of the search space.

The BFS algorithm explores the graph by visiting all the neighbors of a node before moving to the next level, resulting in a breadth-first exploration of the search space.
Dijkstra's algorithm finds the shortest path by maintaining a priority queue of nodes based on their tentative distances from the start node, ensuring the optimal path is found.
The A* search algorithm improves upon Dijkstra's algorithm by incorporating a heuristic function to estimate the cost from each node to the target, guiding the search towards the goal more efficiently.
Adjusting the min_edge_cost variable to 1.0 ensures that the heuristic cost estimate used by the A* algorithm is admissible, guaranteeing optimal paths. With the old value of 10.0, the A* algorithm may overestimate the cost and find suboptimal paths in certain cases.
Adding diagonal edges to the navigation graph allows for more direct paths between nodes, potentially reducing the overall path cost and improving search efficiency.

Conclusion:

This lab provided hands-on experience in understanding and implementing different graph search algorithms in a box world environment. By observing the behavior of DFS, BFS, Dijkstra's algorithm, and A* search, we gained insights into their exploration strategies and path-finding capabilities. The importance of setting appropriate edge costs and heuristic functions for optimal performance, particularly in the case of A* search, was highlighted. The lab also demonstrated the impact of graph structure, such as adding diagonal edges, on search results. Overall, this lab reinforced the concepts of graph search algorithms and their application in path planning problems.