Contents

INTRODUCTION

This document contains a detailed description about the search algorithms used in these particular question and reports the average time, length, and hopes(the average number of nodes passed to reach the end node) taken for each algorithms to complete there tasks. There are Four algorithms that have been used to complete searching tasks for this assignment.

1. Breadth First Search

Breadth-first search (**BFS**) is an algorithm for searching a tree and weightless graph data structures for nodes satisfying a given property. It starts at the a given starting node and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. A queue is needed to keep track of the child nodes that were encountered but not yet explored.

2. Depth First Search

Depth-first search (**DFS**) is an algorithm for traversing or searching tree or weightless graph data structures. The algorithm starts at ta given starting node and explores as far as the given target/end node along each branch before backtracking.

3. Dijkstra’s Algorithms

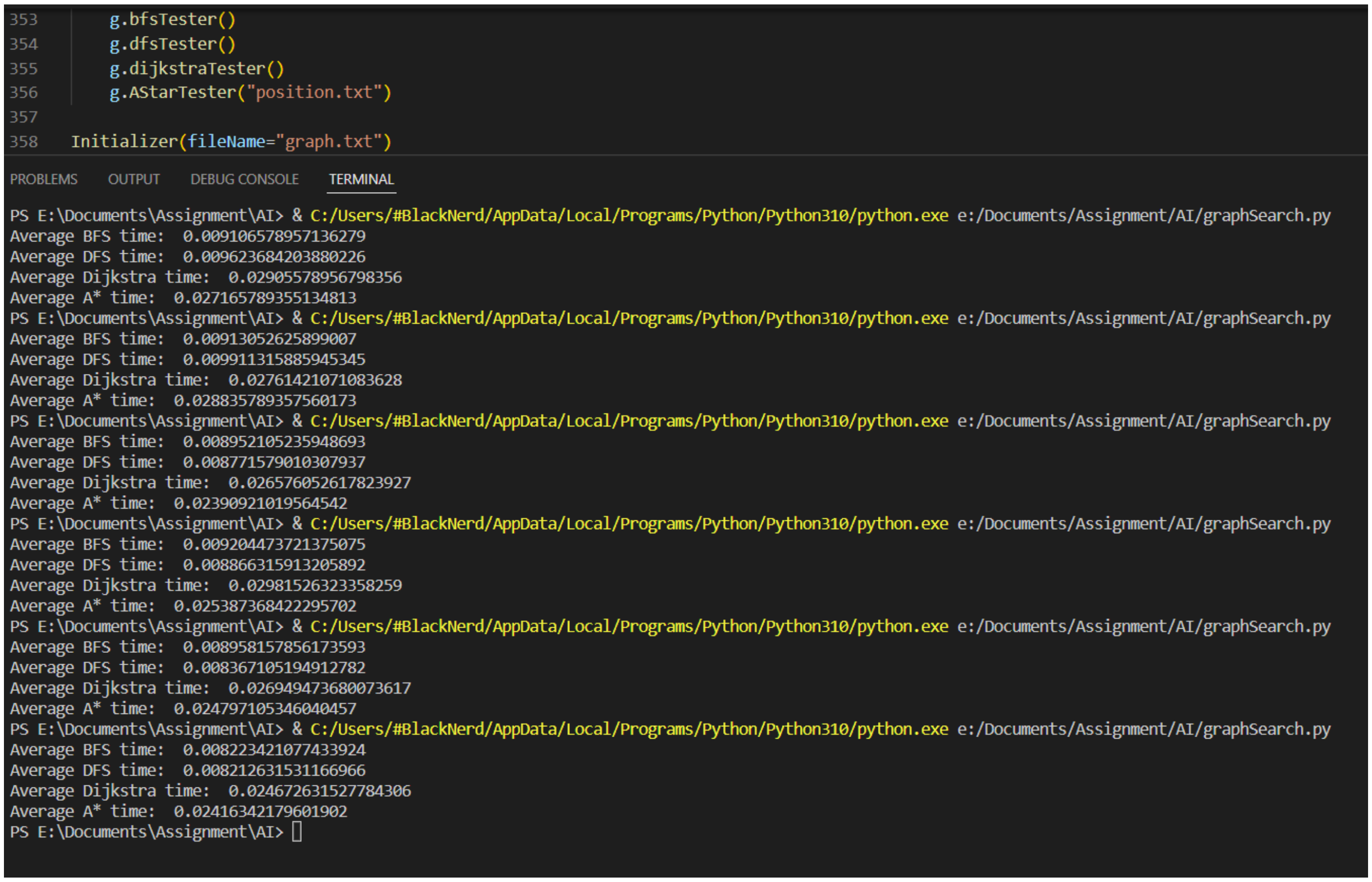
Dijkstra's algorithm is an algorithm we can use to find shortest distances or minimum costs depending on what is represented in a graph. You're basically working backwards from the end to the beginning, finding the shortest leg each time. We maintain two sets, one set contains vertices included in the shortest-path tree, other set includes vertices not yet included in the shortest-path tree. At every step of the algorithm, we find a vertex that is in the other set (set of not yet included) and has a minimum distance from the source.

4. A* Algorithm

It is a searching algorithm that is used to find the shortest path between an initial and a final point. It is a handy algorithm that is often used for map traversal to find the shortest path to be taken. A* was initially designed as a graph traversal problem, to help build a robot that can find its own course. It still remains a widely popular algorithm for graph traversal.

All this Algorithms were implemented and then tested and the benchmark results where were recorded. The results were found to be as stated below.

AVERAGE TIME



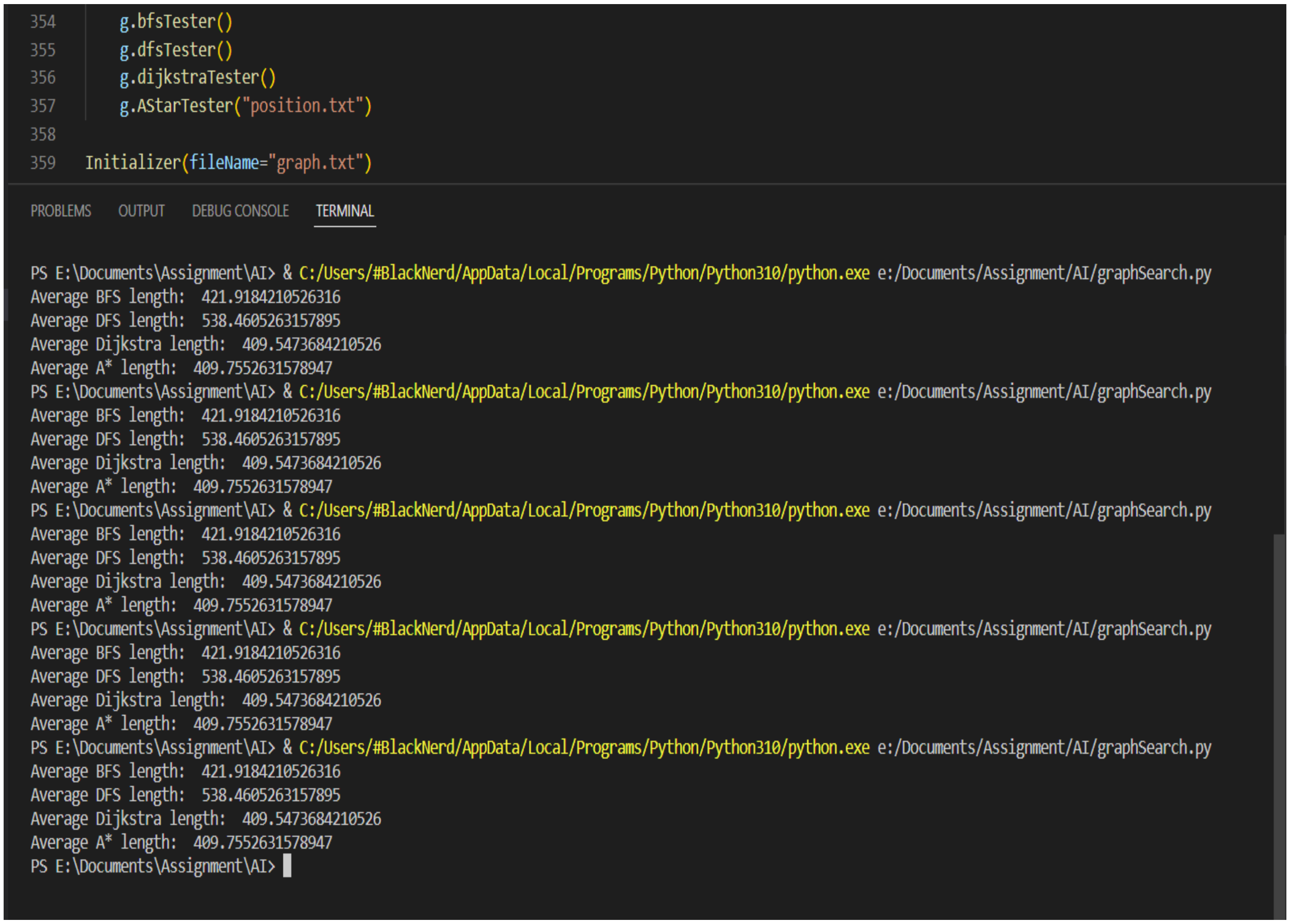
```
    g.bfsTester()
    g.dfsTester()
    g.dijkstraTester()
    g.AStarTester("position.txt")

Initializer(fileName="graph.txt")
```

```
PS E:\Documents\Assignment\AI> & C:/Users/#BlackNerd/AppData/Local/Programs/Python/Python310/python.exe e:/Documents/Assignment/AI/graphSearch.py
Average BFS time:  0.009106578957136279
Average DFS time:  0.009623684203880226
Average Dijkstra time:  0.02905578956798356
Average A* time:  0.027165789355134813
PS E:\Documents\Assignment\AI> & C:/Users/#BlackNerd/AppData/Local/Programs/Python/Python310/python.exe e:/Documents/Assignment/AI/graphSearch.py
Average BFS time:  0.00913052625899007
Average DFS time:  0.009911315885945345
Average Dijkstra time:  0.02761421071083628
Average A* time:  0.028835789357560173
PS E:\Documents\Assignment\AI> & C:/Users/#BlackNerd/AppData/Local/Programs/Python/Python310/python.exe e:/Documents/Assignment/AI/graphSearch.py
Average BFS time:  0.008952105235948693
Average DFS time:  0.008771579010307937
Average Dijkstra time:  0.026576052617823927
Average A* time:  0.02390921019564542
PS E:\Documents\Assignment\AI> & C:/Users/#BlackNerd/AppData/Local/Programs/Python/Python310/python.exe e:/Documents/Assignment/AI/graphSearch.py
Average BFS time:  0.009204473721375075
Average DFS time:  0.008866315913205892
Average Dijkstra time:  0.02981526323358259
Average A* time:  0.025387368422295702
PS E:\Documents\Assignment\AI> & C:/Users/#BlackNerd/AppData/Local/Programs/Python/Python310/python.exe e:/Documents/Assignment/AI/graphSearch.py
Average BFS time:  0.008958157856173593
Average DFS time:  0.008367105194912782
Average Dijkstra time:  0.026949473680073617
Average A* time:  0.024797105346040457
PS E:\Documents\Assignment\AI> & C:/Users/#BlackNerd/AppData/Local/Programs/Python/Python310/python.exe e:/Documents/Assignment/AI/graphSearch.py
Average BFS time:  0.008223421077433924
Average DFS time:  0.008212631531166966
Average Dijkstra time:  0.024672631527784306
Average A* time:  0.02416342179601902
PS E:\Documents\Assignment\AI> 
```

The average time taken to implement each algorithms was calculated using the tester methods namely, bfsTester(), dfsTester(), dijkstraTester(), AstarTester(). This tester methods iterate through each nodes twice to find the time it takes to find a path between the nodes and append them to a temporary time list. The average of this list is then calculated to find out the average time it took to find path. The average times taken by each algorithms to find path were found to be as presented in the screenshot. The test was run multiple times to see if there will be any difference. From the results, it can be clearly inferred that the average time taken for each algorithms were found to be slightly different in each run. This is due to the CPU performance and overload of RAM. But the information provided was enough to evaluate the effectiveness of the algorithms. The BFS and DFS traversal and searching algorithms were actually found to take similar execution time. There were times where the BFS algorithm had a faster execution time than the DFS and vice versa but clear consistent difference were not registered when tried repeatedly. The similarity in the average times indicate that the number of nodes and edges are very close to eachother and hence both algorithms' traversing method, whether it is layer by layer or crossing between layers did not cause a significant difference. But out of the six tries, the DFS algorithm was found to be faster than the BFS algorithm four times indicating that the graph may have slightly more layers than branches.

The Dijkstra and A* search algorithms, however, where found to be different. From the above screenshot, it can be clearly seen that most of the time the A* algorithm has a smaller average time than the Dijkstra algorithm. This actually makes a lot of sense if we look at the way this algorithms find the shortest path between given nodes. Dijkstra's algorithm find shortest path by registering the closest nodes to the starting nodes step by step. This means, the algorithm is not trying to find the shortest path to the target node but the nodes that are the closest to it step by step according to there closeness. And this may take a lot of time specially if the target node is far from the starting node. The A* algorithm ,however, resolves this issues by calculating a guessed distance across each path from start node to target node hence making it focus more on finding shortest distances between the nodes rather than finding and registering shortest distance from the starting node. But ,as you can see from the screenshot, The A* algorithm as not always faster than the Dijkstra algorithm. This anomalies are due to implementation methods, CPU/RAM congestion or they just might be anomalous results. All in all, it is safe to say that A* algorithm is faster than Dijkstra's algorithm in most cases and the graph was found to in accordance with this theory.

AVERAGE LENGTH



```
    g.bfsTester()
    g.dfsTester()
    g.dijkstraTester()
    g.AStarTester("position.txt")

Initializer(fileName="graph.txt")
```

```
PS E:\Documents\Assignment\AI> & C:/Users/#BlackNerd/AppData/Local/Programs/Python/Python310/python.exe e:/Documents/Assignment/AI/graphSearch.py
Average BFS length:  421.9184210526316
Average DFS length:  538.4605263157895
Average Dijkstra length:  409.5473684210526
Average A* length:  409.7552631578947
PS E:\Documents\Assignment\AI> & C:/Users/#BlackNerd/AppData/Local/Programs/Python/Python310/python.exe e:/Documents/Assignment/AI/graphSearch.py
Average BFS length:  421.9184210526316
Average DFS length:  538.4605263157895
Average Dijkstra length:  409.5473684210526
Average A* length:  409.7552631578947
PS E:\Documents\Assignment\AI> & C:/Users/#BlackNerd/AppData/Local/Programs/Python/Python310/python.exe e:/Documents/Assignment/AI/graphSearch.py
Average BFS length:  421.9184210526316
Average DFS length:  538.4605263157895
Average Dijkstra length:  409.5473684210526
Average A* length:  409.7552631578947
PS E:\Documents\Assignment\AI> & C:/Users/#BlackNerd/AppData/Local/Programs/Python/Python310/python.exe e:/Documents/Assignment/AI/graphSearch.py
Average BFS length:  421.9184210526316
Average DFS length:  538.4605263157895
Average Dijkstra length:  409.5473684210526
Average A* length:  409.7552631578947
PS E:\Documents\Assignment\AI> & C:/Users/#BlackNerd/AppData/Local/Programs/Python/Python310/python.exe e:/Documents/Assignment/AI/graphSearch.py
Average BFS length:  421.9184210526316
Average DFS length:  538.4605263157895
Average Dijkstra length:  409.5473684210526
Average A* length:  409.7552631578947
PS E:\Documents\Assignment\AI> 
```

The tester methods where re-used to calculate the average distances taken by each search Algorithms. However, the implementation of calculating the lengths for each combination of nodes where different for each algorithms. The BFS and DFS algorithms search for the minimum path by assuming that the weight of each node is 1. Because of this, a new loop was added to calculate the length of each path taken by these algorithms in each of the algorithms and their functions where made to return both the path and the length. This lengths where then added to the length list and then the average value of that list was taken to find the average distance taken by each algorithms.

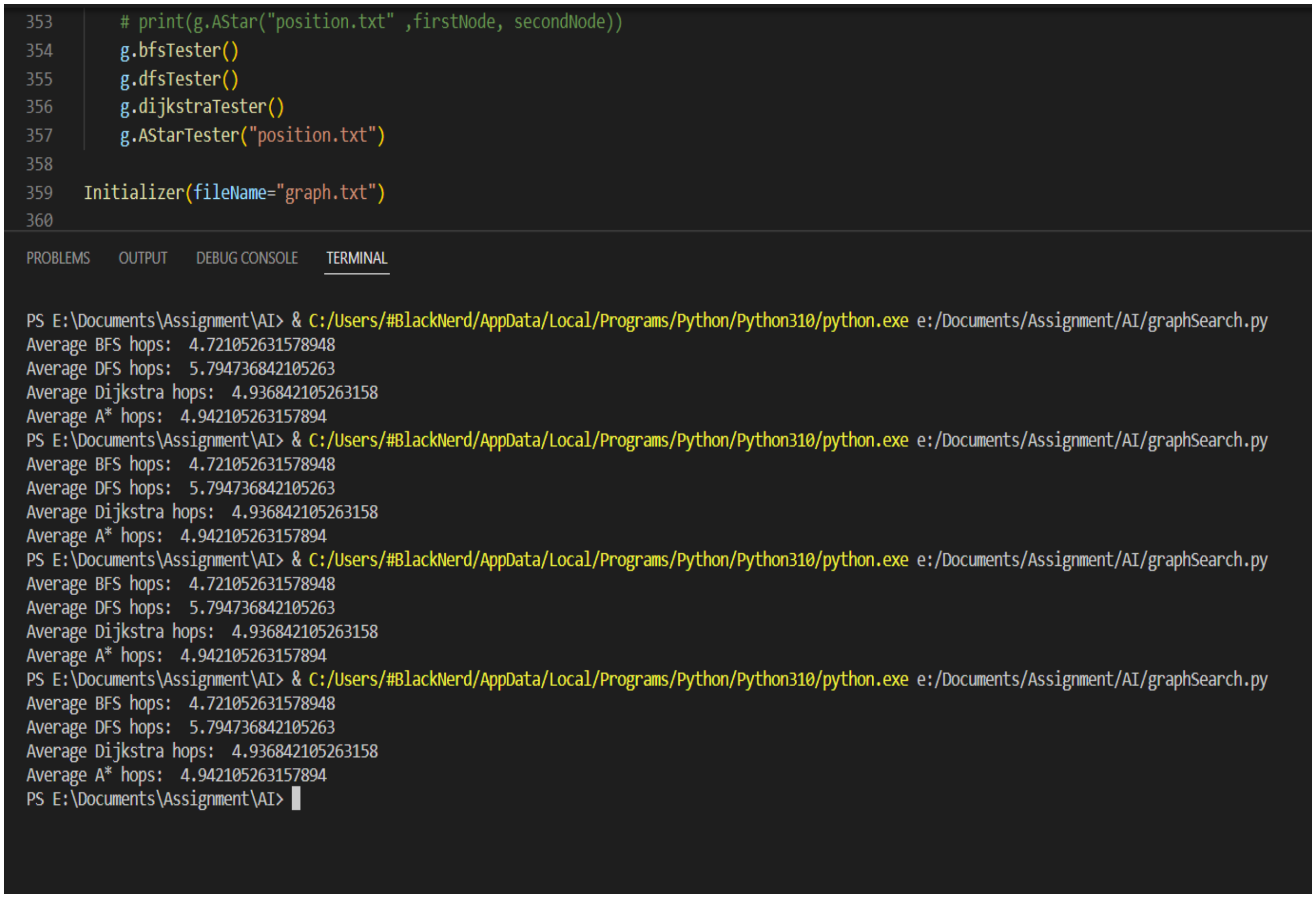
The Dijkstra’s and A* algorithms ,however, did not need to calculate the length by creating a different loop because they consider the weight, which means the find the best path by finding out which path has the least weight. So for this algorithms, the length was already calculated and registered and so were added to the return of their functions.

This calculated Average lengths of each algorithms were then outputted to the Terminal (as it can be seen from the screenshots). The test was ran 5 times to see if there will be difference in output and it was found that the distance remained the same. This was the expected output as we were calculating the average distance for the same graph. The results of each algorithms' test where then compared and the following conclusions where made.

The lengths of the BFS and DFS algorithms where found to be significantly different. The BFS algorithm's average length(421.91 km) was found to be much smaller than the DFS algorithm's average distance(538.46 km). this also was found to comply with the expected outcome. For a weightless graph, the BFS algorithm finds the shortest distance but the DFS just traverses across different layers until it finds the target. Even though, our graph was weighted, the BFS will always search the path with the least amount of nodes where as the DFS will just traverse until it finds. So the Expected result was to have a smaller average length using the BFS algorithm and the graph was found to comply with this logic. It is important to note that the BFS was expected to have the shorter distance compared to DFS not the other algorithms.

The lengths of The Dijkstra's and A* algorithms, however, did not completely comply with the expected result. It was expected that this two will have the shortest average result possible. The Dijkstra's algorithm, in fact, gave out the smallest possible average length which was found to be approximately 409.55 km. the A* algorithm, however, gave out an average distance that is a little higher than the Dijkstra's(409.75 km). This marginal difference(0.2 km difference in length) between this two algorithms came because of the quality of the heuristic value. The heuristic value is basically a guess made to decide which one of the alternative paths is predicted to be smaller than the rest. This is crucial in finding the best path and any deviated guess(bad heuristic value) may result in a wrong path. So that 0.2km difference is believed to come from longer paths taken by the A* algorithm due to the heuristic value.

AVERAGE NUMBER OF HOPS



```
    # print(g.AStar("position.txt" ,firstNode, secondNode))
    g.bfsTester()
    g.dfsTester()
    g.dijkstraTester()
    g.AStarTester("position.txt")

Initializer(fileName="graph.txt")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\Documents\Assignment\AI> & C:/Users/#BlackNerd/AppData/Local/Programs/Python/Python310/python.exe e:/Documents/Assignment/AI/graphSearch.py
Average BFS hops:  4.721052631578948
Average DFS hops:  5.794736842105263
Average Dijkstra hops:  4.936842105263158
Average A* hops:  4.942105263157894
PS E:\Documents\Assignment\AI> & C:/Users/#BlackNerd/AppData/Local/Programs/Python/Python310/python.exe e:/Documents/Assignment/AI/graphSearch.py
Average BFS hops:  4.721052631578948
Average DFS hops:  5.794736842105263
Average Dijkstra hops:  4.936842105263158
Average A* hops:  4.942105263157894
PS E:\Documents\Assignment\AI> & C:/Users/#BlackNerd/AppData/Local/Programs/Python/Python310/python.exe e:/Documents/Assignment/AI/graphSearch.py
Average BFS hops:  4.721052631578948
Average DFS hops:  5.794736842105263
Average Dijkstra hops:  4.936842105263158
Average A* hops:  4.942105263157894
PS E:\Documents\Assignment\AI> & C:/Users/#BlackNerd/AppData/Local/Programs/Python/Python310/python.exe e:/Documents/Assignment/AI/graphSearch.py
Average BFS hops:  4.721052631578948
Average DFS hops:  5.794736842105263
Average Dijkstra hops:  4.936842105263158
Average A* hops:  4.942105263157894
PS E:\Documents\Assignment\AI> 
```

The testing function were, yet again, used to measure the total number of hops made (number of nodes passed to find the path). This did not require any kind of updating to our existing functions of the Algorithms and was calculated by identifying the length of the path. These number of hops where then registered to a list, which was then used to calculate the average number of hops. The results found where as follow.

The BFS was expected to have the least number of hopes as it doesn't consider weight and tries to find path with the least number of node. The implemented algorithm confirmed this theory and found the average number of hops by the BFS algorithm to be 4.72. The DFS algorithm was expected to have the highest number of hops as it just tries to find target node by searching the neighbor of the last added without considering anything else. This was also confirmed by our implementation of the algorithm and the average number of hops made by the DFS algorithm in our case where found to be 5.79.

The Dijkstra and A* algorithms were expected to have the same average number of hops which is higher than BFS and lower than DFS. But the outcome was not entirely as expected. The results where found to be very close but still had a very marginal difference(4.936 and 4.942 respectively). This deviation was again cause by the quality of the heuristic value. Some of the paths found by these algorithms are a little different due to the quality of the heuristic value causing difference in the number of hops as well.

CONCLUSION

Most of the benchmark results where actually similar to the theoretically and calculated (expected) outcomes. Some anomalies were found due to PC specifications, heuristic values and the like. But based from our benchmark we can generalize the following points:

- It is better to use BFS algorithm to find a node that is in a close branch/layer to the starting node(if the node is shallow with respect to starting node) and the graph has no weight (weight =1).

- It is better to use DFS for nodes that are far deeper compared to the starting node (if the nodes are in a very far layer) and the graph has no weight (weight =1).

- It is better to use dijkstra's algorithm to find the shortest path from starting node to all the other nodes in a weighted Graph.

- It is better to use A* algorithm to find the shortest distance between two nodes in a weighted graph (as long as you are using a high quality heuristic values).