|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | dfs iterative | dfs recursive | bfs iterative | bfs recursive | dijkstra | a star |
| list(src->dest) | |  |  |  |  |  |
| nodes explored | 11.6 | 6.5 | 8.2 | 5.5 | 11.9 | 11.9 |
| execution time | 14.5 | 10.5 | 20.3 | 9.8 | 22.6 | 22.2 |
| total cost | 16.4 | 0 | 9.2 | 0 | 15.2 | 15 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | dfs iterative | dfs recursive | bfs iterative | bfs recursive | dijkstra | a star |
| matrix(src->dest) | |  |  |  |  |  |
| nodes explored | 9.2 | 18.8 | 6.1 | 12.6 | 8 | 8 |
| execution time | 15.5 | 34.3 | 23.8 | 17.7 | 29.2 | 28.5 |
| total cost | 0 | 0 | 0 | 0 | 0 | 0 |

Analysis: There were a lot of inconsistencies in terms of data produced by each respective algorithm. I was expecting the iterative implementations of both DFS and BFS to be much faster, but this was not the case. Upon examining the source code, I realized I actually optimized my recursive functions much better than the iterative ones. The optimization in the recursive functions were significant enough that they ended up performing better than iterative implementations of the same functions. The reason behind this oddity is simply that in the iterative implementations of DFS and BFS, nested loops kill a lot of time by going through each node or datapoint, whereas in the recursive implementations, the recursion only occurs when a node is selected to be of significance. However, this trend of recursive functions operating faster than iterative ones was not always the case. With adjacency matrix, the DFS recursive implementation was slower than its iterative counterpart. The reason for this is unclear other than the fact that structural as well as behavioral differences between the adjacency list and the adjacency matrix may have accounted for this inconsistency. Furthermore, when using the adjacency list, the recursive implementations of both DFS and BFS returned similar execution time while the recursive implementations of DFS and BFS that used adjacency matrix had a stark difference in their respective execution times (DFS took about twice as much time as BFS). The differences in execution times seem to be a result of different data points (src, dest) that were used to run the algorithms. However, even with the varying execution times, DFS explored more nodes on average than BFS. Dijkstra took the longest to execute with both adjacency list and adjacency matrix, which is logical because Dijkstra algorithm does not come up with the fasted solution but the shortest path, which requires node explorations, comparisons, and many other operations that other algorithms forego. The A star algorithm for both the list and the matrix executions performed better than Dijkstra in terms of execution time but got around the same number of nodes explored. As A star is a modified version of Dijkstra, and since its chief function is to run faster than Dijkstra even at the cost of not discovering the shortest path, the data collected accurately represents the pros and cons of using Dijkstra and A star.