**Speed:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **BFS** | **DFS** | **Dijkstra (Manhattan)** | **Dijkstra (Weighted Diag.)** |
| **Map 1** | 0.143112 seconds | 0.110103 seconds | 0.156801 seconds | 0.219384 seconds |
| **Map 2** | 0.126486 seconds | 0.110184 seconds | 0.156681 seconds | 0.173456 seconds |
| **Map 3** | 0.134855 seconds | 0.109896 seconds | 0.156774 seconds | 0.188578 seconds |

The speed of DFS was fairly consistent and the fastest across all 3 forms of search algorithms. Following closely behind is BFS for the fact that the search patterns are very similar. In last would be Dijkstra, however this algorithm has weighted diagonals built-in where as BFS and DFS assume unweighted corners. Looking a more comparative limited version of the function shows that the speeds are only marginally slower than BFS and are almost identically consistent.

**Nodes visited:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **BFS** | **DFS** | **Dijkstra (Manhattan)** | **Dijkstra (Weighted Diag.)** |
| **Map 1** | 8250 Nodes | 4797 Nodes | 8278 Nodes | 8305 Nodes |
| **Map 2** | 7696 Nodes | 4089 Nodes | 7716 Nodes | 8023 Nodes |
| **Map 3** | 8107 Nodes | 4898 Nodes | 8167 Nodes | 7081 Nodes |

Most notably among these 3 algorithms is that DFS visits the lest number of nodes by almost ½. BFS and Dijkstra are close and separate in an undetermined manner (correlated to how diagonal the ideal straight line path is) when comparing BFS to the corner weighted Dijkstra.

**Computational Complexity:**

In terms of shear programming and computational complexity, BFS and DFS are almost identical when implemented in python. DFS can be argued to be a few clock cycles faster as it just has to pull the last entry from the stack rather than call the first entry and modify the queue length (although this can be optimized). Other than this the searches are nearly identical as the method is the same, but the following node is different. Dijkstra is the outlier in that it is significantly more computationally heavy as it has to keep track of a whole separate parameter (total path length) and also manage queue placement by path length. This leads to much more complexity in programming implementation and computation (although it is still just addition and callbacks).

Overall BFS is good for a quick and somewhat accurate search path where as DFS is the fastest at finding the end and gives an extremely poor path (basically unfollowable and useless for shortest path planning in a grid). Dijkstra is good overall and can easily implement weights and is only marginally slower than BFS with a near identical Node count but higher memory usage.

Referenced Material:

*Breadth first search*. Programiz. (n.d.). Retrieved February 13, 2023, from https://www.programiz.com/dsa/graph-bfs

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms*. Mit Press.

*Depth first search (DFS)*. Programiz. (n.d.). Retrieved February 13, 2023, from https://www.programiz.com/dsa/graph-dfs

Wikimedia Foundation. (2023, February 9). *Dijkstra's algorithm*. Wikipedia. Retrieved February 13, 2023, from https://en.wikipedia.org/wiki/Dijkstra%27s\_algorithm