# Data

Results for destination point: (36.359595,-82.398868)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Run Time (ms) | Route Distance | Steps Taken | Route Image |
| BFS | 997.998 | 7448.816000000002 | 39 |  |
| DFS | 11.999 | 25413.508 | 148 |  |
| Random-Depth | 56.998 | 41156.432 | 183 |  |

Results for destination point: (36.342513,-82.373483)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Run Time (ms) | Route Distance | Steps Taken | Route Image |
| BFS | 1359.998 | 5652.981999999999 | 39 |  |
| DFS | 184.996 | 18784.899999999994 | 120 |  |
| Random-Depth | 811.996 | 70023.504 | 315 |  |

Results for destination point: (36.320831,-82.277667)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Run Time (ms) | Route Distance | Steps Taken | Route Image |
| BFS | 590.984 | 4653.238000000001 | 26 |  |
| DFS | 360.003 | 46090.71599999999 | 330 |  |
| Random-Depth | 50.999 | 33408.25099999999 | 482 |  |

Results for destination point: (36.316574,-82.352577)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Run Time (ms) | Route Distance | Steps Taken | Route Image |
| BFS | 57.998 | 2235.292 | 15 |  |
| DFS | 437.995 | 34256.70000000002 | 240 |  |
| Random-Depth | 404.999 | 77301.24800000004 | 333 |  |

Results for destination point: (36.301605,-82.337822)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Run Time (ms) | Route Distance | Steps Taken | Route Image |
| BFS | 33.995 | 3265.7579999999994 | 14 |  |
| DFS | 667.996 | 66881.21399999999 | 460 |  |
| Random-Depth | 116.008 | 56108.86899999994 | 303 |  |

Results for destination point: (36.347904,-82.400772)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Run Time (ms) | Route Distance | Steps Taken | Route Image |
| BFS | 1343.000 | 7448.816000000002 | 39 |  |
| DFS | 32.000 | 25413.508 | 148 |  |
| Random-Depth | 826.000 | 63748.13599999998 | 209 |  |

# Conclusion

### Search Space

The search space itself is a bidirectional weighted graph consisting of nodes which correspond to intersections from map coordinate data, and edges correspond to the roads between those intersections. By building the nodes in this way, we essentially have a simplified graph of Johnson City that can be traversed using nodes and edges. Nodes can be identified using gps coordinates, and the map overlay provided by the library osmnx which contains the graph design data. The underlying graph itself, made by the networkx library, can then be navigated by finding neighbors and traversing edges.

### Best Algorithm

In this set of performance tests, there is a run-away winner: Bread First Search. This makes sense, as BFS essentially surveys the route options around them, continually expanding out until the goal is found. That means in our map example. We find the destination quickly and with a near optimal route because expanding outwards is a broader search method than pursuing one route to the end before attempting another.

Depth first search performs quite poorly, in that it takes much longer and travels much further. This makes sense as it essentially returns the first in order path that takes you to goal and is therefore unlikely to ever be optimal in a real-world example such as this.

Finally, the last algorithm (described as Random-Depth) is essentially a modification of DFS in which we search the depth of the graph but instead of going with the first edge out of the node, we take a random path. As a result, this occasionally outperforms DFS and occasionally underperforms against DFS. This is because while DFS will return you the first path to goal, this essentially returns a random path to goal. Compared to BFS, this is consistently way worse.