­­Reid Russell

CSE 3353 Algorithms

Lab 2 Report

Looking at BFS and DFS, BFS will consistently return paths that are shorter than DFS, but the number of nodes visited has the potential to be much greater in large datasets. Datasets that are large and also have a higher amount of connections will increase this proportional to the amount of connections per node. DFS performs in a similar manner when dealing with large datasets, because it has a tendency to travel all the way down a dataset vertically before checking the nodes in the immediate vicinity. For this reason it is often a good idea to implement iterative deepening as to avoid DFS searching too deep within a graph.

Dijkstra and A\* are much better at taking a good guess as to which connection to follow given a source node. A\* is essentially Dijkstra except for the heuristic which is factored into the cost of traversing a connection- usually some type of distance. Heuristics can theoretically be anything but obviously there would not be a point in using some random metric.

When looking at how the adjacency list compares to the adjacency matrix, its quite clear that in larger datasets, the adjacency matrix performs much better in larger datasets because its lookup time is much faster than an adjacency list.

Below is the data I was able to collect over 100 randomized src and destination nodes. As you can see its not much. I believe it had something to do with my random number generator, and with the matrices- the reading in of the weights. When running the larger graph I was not able to get any data out, as it appeared to go into an infinite look in my BFS, but when run with random manually input src and dest nodes, it seemed to perform fine even on the larger datasets.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Search Algorithm 16 Nodes | | |
| Matrix | DFS Iterative | DFS Recursive | BFS Iterative | BFS Recursive |
| Nodes in Path | 2.21 | n/a | n/a | n/a |
| Nodes Explored | n/a | n/a | n/a | n/a |
| Execution Time | -7.93E-06 | n/a | n/a | n/a |
| Distance | n/a | n/a | n/a | n/a |
| Cost | n/a | n/a | n/a | n/a |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Search Algorithm 16 Nodes | | |  |  |
| Adjacency List | DFS Iterative | DFS Recursive | BFS Iterative | BFS Recursive | Dijkstra | A\* |
| Nodes in Path | 2.165 | n/a | n/a | n/a | 2.88667 | n/a |
| Nodes Explored | n/a | n/a | n/a | n/a | n/a | n/a |
| Execution Time | -7.93E-06 | n/a | n/a | n/a | n/a | n/a |
| Distance | n/a | n/a | n/a | n/a | n/a | n/a |
| Cost | n/a | n/a | n/a | n/a |  | n/a |

Obviously there is not much that can be discerned from my data, but while working through BFS, DFS, and Daijkstra’s there was a clear distinction as to the paths returned. In BFS after building a vector of paths travelled, when the destination was found you then had to iterate through that vector of paths and piece together the actual shortest path. The nice thing about DFS was that this was not necessary. Dijkstra doesn’t necessarily return the shortest path, but the mechanics of it are much more simple, making it much quicker.