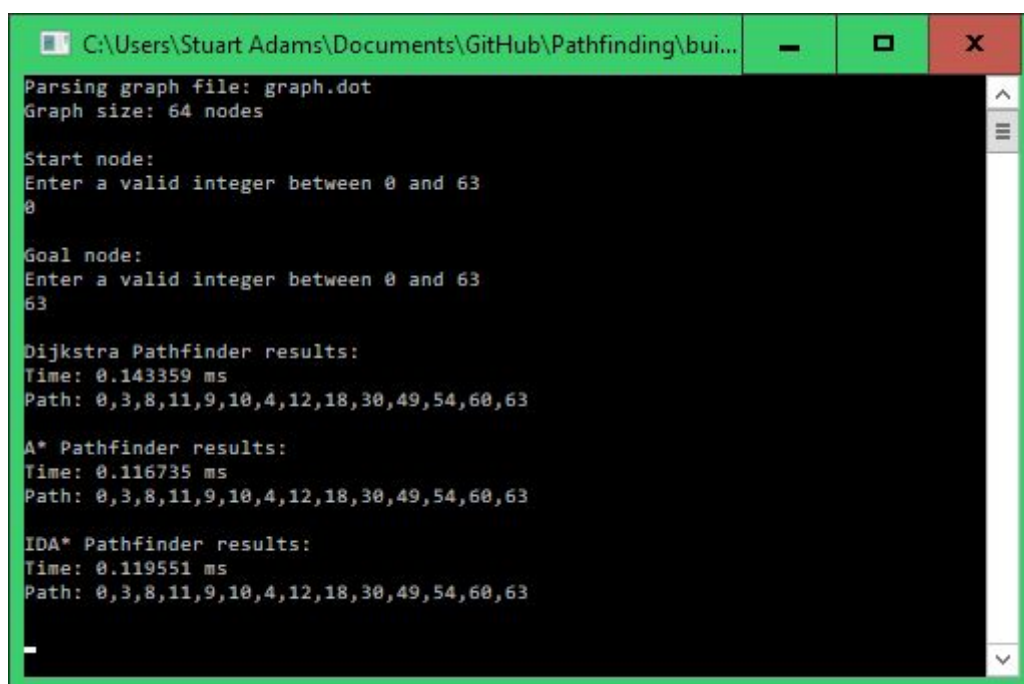


Our solution presents a simple pathfinding framework with a console application. The graph class can be constructed using the **DotParser** class, which loads the text of a specified .dot file into memory and extracts the node / edge data. We have created a pathfinder interface, **IPathfinder**. This is implemented by our pathfinders, **AStarPathfinder**, **DijkstraPathfinder** and **IDAStarPathfinder**. When "FindPath" is called, it takes the graph and start / goal indices as arguments. Once the shortest path is found, it returns the path as a `std::deque<int>`, with each element representing the index of each node in the path. This can then be used by the client code in whatever way it sees fit.

The program parses the example graph.dot file and then asks the user to enter the start and goal nodes. If the input is valid, it then runs the same problem using the different implementations of **IPathfinder**, and prints out the path found and the execution time in milliseconds.

A screenshot of a Windows console application window. The title bar shows the file path 'C:\Users\Stuart Adams\Documents\GitHub\Pathfinding\bui...'. The console output is as follows:

```
Parsing graph file: graph.dot
Graph size: 64 nodes

Start node:
Enter a valid integer between 0 and 63
0

Goal node:
Enter a valid integer between 0 and 63
63

Dijkstra Pathfinder results:
Time: 0.143359 ms
Path: 0,3,8,11,9,10,4,12,18,30,49,54,60,63

A* Pathfinder results:
Time: 0.116735 ms
Path: 0,3,8,11,9,10,4,12,18,30,49,54,60,63

IDA* Pathfinder results:
Time: 0.119551 ms
Path: 0,3,8,11,9,10,4,12,18,30,49,54,60,63
```

Example Output

We decided to implement the algorithms ourselves. Our approach is largely based on the examples provided by redblobgames. We have implementations for A*, Dijkstra and IDA*. We decided to try each because it would provide a better understanding of pathfinding algorithms as a whole, and their relative strengths and weaknesses.

We started with Dijkstra's algorithm, a common pathfinding solution that finds the shortest path from a given node to every other node in the graph. Our implementation is based around a priority queue, and returns once the path to a specific goal node is found. The algorithm begins with a start node and a list of its neighbors. At each step, the neighbour

with the lowest distance from the start is examined. This neighbour is marked "closed", and all of its neighbours are examined if they have not been already. We repeat this process until the destination is found. Because the lowest distance neighbours are considered first, the route taken when we find the destination for the first time is guaranteed to be the shortest path.

We then implemented A*. A* is a variant of Dijkstra's algorithm that achieves greater performance by using heuristics to guide its search. It assigns a weight to each node under consideration equal to the edge to that node plus the distance between the node and the goal. This allows A* to eliminate longer paths once the initial path is found, eliminating the need to examine many neighboring nodes. A* will always find the optimal solution so long as it exists and the heuristic supplied is admissible.

IDA* is an iterative deepening depth-first search that borrows A*'s use of heuristics to evaluate the remaining cost of traversal. Because it is depth-first, the memory usage is lower than A*. However, IDA* will explore the same node many times, making it typically slower than A*. It can be seen as a memory-efficient variant of A* - it does the same job as A*, but trades A*'s run-time efficiency for memory efficiency. This is generally visible in our program output. Like A*, IDA* will always find the optimal solution so long as it exists and the heuristic supplied is admissible.

Our project's only 3rd party dependency is glm, a header-only math library that we use for its vector type and distance function for use in heuristics functions.

Bibliography:

<https://www.raywenderlich.com/4946/introduction-to-a-pathfinding>

<https://www.redblobgames.com/pathfinding/>

<https://algorithmsinsight.wordpress.com/graph-theory-2/ida-star-algorithm-in-general/>