McKenna Jones
Spike Madden

CS331 Assignment #1: Uninformed and Informed Search

Methodology

As the assignment description suggests, we began by looking at the GraphSearch psuedocode from the slides. This psuedocode ended up being the basis for each of our different algorithms. Each different implementation shares the idea of having a data structure to hold the current expanded nodes, called the fringe, and another data structure to hold all visited nodes. How we used these data structures in each algorithm differs.

We began by defining a *State* class. This class is used to represent the various nodes on the graph. It holds the number of missionaries and cannibals on each side of the bank, as well as the position of the boat (left or right). The next major piece of code is the *expand* function. This function takes a *State* and tries to expand the maximum number of 5 children states, making sure that each one is valid.

The implementation of depth first and breadth first are extremely similar. Both use a simple list as the fringe. In breadth first, expanded nodes are added to the end of the list. In each cycle, a node is retrieved from the *beginning* of the list. In our implementation of depth first, expanded nodes are also added to end of the list, but in reverse order. We then retrieve the last element from the list in each cycle. This simple difference results in slightly different performance and behavior.

Iterative deepening depth first search is essentially a mix of breadth first and depth first, so we were able to reuse most of our code from depth first. The difference is the addition of a limit. In each cycle the limit increases by one, and we only expand nodes that are at a level less than the current limit. The logic for expanding is the same and depth first, as the name of the algorithm suggests.

For A*, it was necessary to define a heuristic. We decided to keep ours as simple as possible. Our heuristic assigns a score to each node, based on how close it is to the goal state.

Here is the simple calculation:

```
score += goal.misLeft - cur.misLeft
score += goal.canLeft - cur.canLeft
```

It finds the difference between the number of missionaries on the left bank in the goal state, and the current state, and does the same for cannibals. These differences are then added

together. Then instead of using a simple list for the fringe, we use a priority queue. When retrieving items from the queue, we always retrieve the item that is closest to the goal, and then expand its nodes.

Results

Results for testing with *goal3.txt* and *initial3.txt*

Algorithm	Nodes Expanded	Time Spent (seconds)
Depth First	558	0.00857
Breadth First	2184	0.03404
Iterative Deepening	606367*	8.38669
A*	673	0.02146

* IDDFS expanded 2180 nodes at level 554

Discussion

All of our results were as expected. Depth first is extremely fast as long as there's no infinite path for the algorithm to get stuck on. Breadth first focuses more on completeness in exchange for efficiency which results in a slower time than depth first search. A* was fast with our well defined and admissible heuristic function. The only slightly surprising result was the iterative deepening depth first search. It's significantly slower than the other three algorithms and expands a much larger amount of nodes.

Conclusion

Depth first performed the best for the missionaries and cannibals problem that was presented to us. This was expected as long as we didn't run into any infinite loops, which we didn't. Depth first search expanded 558 nodes and took 0.00857 seconds to find the solution. We can also look at the Big O's for space and time complexity to see that this behavior was expected. Depth first search has a time complexity of $O(b^d)$ and a space complexity of O(bd), where b is the branching factor, d is the goal state depth. Compared to the other uninformed search algorithms' Big O's for time and space complexity, it's expected for depth first search to perform the best.