Alan Neads
CS331

Uninformed and Informed Search

Methodology

I took each test case and modified the quantities of chickens and wolves to test various situations and to ensure the validity of the situation was correct. The validity was confirmed by watching the output solution path. The conditions tested for were invalid cases where a side of the river had more wolves than chickens, and the boat starting on an empty side of the river. Each of these cases were successfully detected and yields no solution. Implementing each search function was handled by Python3's deque from the collections library package, and a-star uses a custom made wrapper class for the required priority queue. The priority queue class made wraps around the functionality of Python3's heap queue, presenting the information in the required format to ensure the priority functionality based on the cost function. For retrieving the next node to do expansion upon was handled by pop and popleft functions respectively for each algorithm. Whether the algorithm used pop or popleft was determined by the method the algorithm searches the tree; in the case of depth first search it would explore the top of the stack first, whereas breadth first search explores the bottom of the stack first and adds more content for exploration onto the top. Given the nature of iterative deepening depth first search it's a mixture of this due to limiting the depth search. The depth limit of 2,000 was chosen based on the resulting solutions of BFS and DFS. With the completeness of IDDFS it should find the most optimal solution. Whereas with a-star the heuristic chosen was to take the cost of the parent and add the quantities of chickens and wolves remaining on the current side of the boat. This ensures that longer chains are less optimal and the actions that fill the boat are chosen first over single occupant actions.

Results

	BFS	**DFS**	**IDDFS**	**ASTAR**
Test 1	Expanded: 14 Solution: 11	Expanded: 11 Solution: 11	Expanded: 87 Solution: 11	Expanded: 14 Solution: 11
Test 2	Expanded: 72 Solution: 33	Expanded: 62 Solution: 47	Expanded: 1,262 Solution: 33	Expanded: 73 Solution: 33
Test 3	Expanded: 2,184 Solution: 377	Expanded: 2,062 Solution: 1,797	Expanded: 428,566 Solution: 377	Expanded: 2,185 Solution: 377

Discussion

From the beginning I expected a-star to perform the best, but I was shocked to find that in the second and third test cases it performed one expansion worse than breadth first search did. It would certainly be interesting to see each expansion that it differed by between each function to see where it evaluated a potential successor compared to the other. Whereas on the other end of the spectrum I was surprised just how quickly the expansions increased with a larger problem size for iterative deepening depth first search. The expansions initially were about eight times as much, but by the final test case it was two hundred times more expansions. The growth alone of expansions fits an exponential curve for growth and definitely places an upper limit on the maximum problem size attainable in a reasonable timeframe. Eventually you'll need to implement multithreading to make the algorithm operate quickly or the waiting will become unreasonable. As for behavior that I found strange was switching between popleft and pop for IDDFS, where each choice would produce dramatically different solutions. For the third test case with popleft it produces a solution of 377 nodes, akin to BFS, whereas with pop it produces a solution of 1,795 nodes. I verified it was clearing the queue and correctly flagging explored pathways, but it still ended up with drastically different solutions.

Conclusion

1. **What can you conclude from these results?**
   With the results I received with my implementation I can conclude that the choice of algorithm depends on your use case for the application. If the application requires the first solution found, ignoring response time, then depth first search would be great as in the test cases it found a solution with the least expansions. However, it's not the most optimal solutions. If completeness is required then breadth first search would produce the complete and optimal answer, but can consume large amounts of time to do so. Lastly if you need a combination of these traits you can either do intelligent searching with a-star, losing completeness, or go with IDDFS for completeness, but with more expansions.
2. **Which search algorithm performs the best?**
   With the least expansions and lowest node solution path it would be breadth first search. Followed closely with one extra expansion, astar.
3. **Was this expected?**
   I expected a-star to be the most optimal, but I think given the test cases the solution may have been quickly searchable with the BFS algorithm.