Haewon Cho

Assignment Report

Methodology:

First, I created “node” class that has left side of chicken and wolf, right side of chicken and wolf, the boat location, depth of node, state, parent, action, key, and cost. State is for checking with other nodes’ state in explored list and frontier list. Parent and action are for tracking moves from the beginning. Key is for the hash table. And lastly, cost is f() for A-star search. I change the start files and goal files into nodes, and then call each search algorithm with the start node and the goal node.

BFS

First, I inserted the start node into frontier FIFO queue. I used dequeue, and when I take nodes, I used popleft(), so I can take out the first in node. I also inserted the start node’s state into frontier hash table using key. I set key to sum of right side of chicken and wolf and boat location. When boat is on right, it is 1. In this way, I could have from 0 to 7 keys to save each state. I also created explored dequeue and hash table just like frontier. When I popleft() a node from frontier queue, I also removed the node state from frontier hash table. When the node is not the goal, I inserted the node into explored dequeue and the node state into explored hash table. Then I called expand function for successors with current node. Then I check the successors, if they are in explored hash table or frontier hash table. I added successors that are not in both hash tables into frontier dequeue and frontier state hash table. When I found the goal node, then I called getresult() function with the node to track the actions from the beginning node to the result node using parent attribute of the node.

DFS

Everything is same with BFS, but the queue it is using is different. BFS used First-In First-Out queue, but DFS uses Last-In First-Out. So I used same dequeue, but when I take out nodes from frontier, I used pop(), so the last node in can be out first. In DFS, I did not use depth limit.

IDDFS

IDDFS is very similar with DFS that is using DFS algorithm with LIFO queue. I used “sys.maxsize” for the maximum depth limit. I called ndfs function with increasing n, start node, and goal node. In ndfs(), everything is same with DFS, but it checks if n is bigger than the currently popped node’s depth. If n is bigger, then the successors of the node can be expanded and added. However, if n is same or smaller than the node’s depth, then it starts from the beginning with increased n.

ASTAR

For ASTAR, I used priority queue, so it can choose the smallest cost. Everything is same with other search except for that it is using priority queue and it is not adding states into frontier hash table. It only checks explored hash table for the states because even if it is same states, there could be fewer cost node. I used h() = right side chicken + right side wolf – boat location (b = 1 when right, b = 0 when left). Therefore, the cost, f() = current node depth + h() because each next step costs 1. I chose the heuristic function because it is not overestimating the cost. Also, it is quite accurate at the end. When there are

one chicken and one wolf and boat on right, they can be moved by 1 move. 1 + 1 – 1 = 1. Also, for two chicken and boat on right, it can be 2 – 1 = 1. However, I don't think it is consistent because it has more expanded nodes than BFS and DFS. I was also thinking (right side chicken * right side wolf) as heuristic. However, I think it is overestimating the costs. Because when 99 chickens and 96 wolves, the cost is almost 10000. Maybe it also can work, but I wasn't sure, so I just chose safe one.

Results:

	Nodes for path test1	Expanded nodes test1	Nodes for path test2	Expanded nodes test2	Nodes for path test3	Expanded nodes test3
BFS	11	14	35	60	387	970
DFS	11	11	35	48	391	856
IDDFS	11	98	35	1186	389	206631
ASTAR	11	23	35	85	387	6294

Discussion:

I did not expect that IDDFS would be this slow for the test3. The time complexities are similar but as the number of nodes is getting bigger, IDDFS gets very slow. I also, thought ASTAR would have smaller number of expanded nodes, but it got way more than DFS and BFS. I am quite disappointed that I couldn't find the proper heuristic. I am wondering if multiply of the right side chicken and wolf would work. When one of them is 0 then we can add 1 for it to avoid 0. However, ASTAR found less nodes path to the goal than DFS, so I am wondering if it can be optimal with fewer nodes to path.

Conclusion:

I think ASTAR would be the best if the best heuristic can be found. With not the best heuristic function, my ASTAR found fewer nodes to path than DFS. Therefore, with a proper heuristic, ASTAR would be the best among the search algorithms. I expected that it would be the best, but I couldn't find the proper heuristic.