CS 331 Assignment #1

Methodology

Move and append method: To expand a node, I use this order to move (1-one chick, 2-one wolf, 3-two chicks, 4-two wolves, 5-one chick and one wolf). If the expanded node is in the visited array or it leads to lose, do not append it to the open (queue) array. Otherwise, append (this node, its parent) to closed (visited) array. For each appending, count+=1 for storing expanded nodes number.

Bfs:

With given start node and goal node. Initializing two arrays, queue and visited. Starting at the start node as the root and add to queue. Popping the 1st node in the queue and expanding it by 'move and append function'. Appending the node to visited. Each repeat will check if expanded nodes have a goal node. If yes, using closed array to find solution. Then repeating popping the 1st node from queue. If the while loop terminates, meaning there is no solution. Using the visited array to find solution.

Dfs:

With given start node and goal node. Initializing two arrays, stack and visited. Starting at the start node as the root and add to queue. Expanding it using move function which can return the allowed motion in next status and popping the last node in the stack. Appending the node to visited. Each repeat will check if expanded nodes have a goal node. If the node found goal,Backtracking to its parents until reach the root . The path would be the reverse order of moving wolves and chicken.

Iddfs:

With given start node and goal node. Initializing two arrays, stack and visited and set up a level limit . Starting at the start node as the root and add to

queue. Expanding it using move function which can return the allowed motion in next status and popping the last node in the stack. Appending the node to visited. Each repeat will check if either reach the goal or reach the limit. If it reach goal, then backtracking to find the path, if it reach the limit, then the code would travel all the node higher than that level. Then add the level until finding the goal or reach the bottom of the tree.

Astar:

With given start node and goal node. Initializing two arrays, open and closed. Open is for storing nodes that need to be considered (expanded). Closed is for storing nodes that need not to be considered (e.g. visited). Starting at the start node as the root and compute **Fscore** and add to open. Popping the node in the open array with lowest **Fscore** and expanding it by 'move and append function' (also compute **Fscore** for expanded nodes). Appending this node to closed. Each repeat will check if expanded nodes have a goal node. If yes, using closed array to find solution. Then repeating popping the lowest **Fscore** node from open. If the while loop terminates, meaning there is no solution. Using the closed array to find solution.

Compute **Fscore**: If moving from right bank, h(n) = (right chicks + right wolves - 2) * 2 + 1. If moving from the left bank, h(n) = (right chicks + right wolves) * 2. This estimates cost by ignoring the eating case. Next, g(n) is the current layer of the tree, since each expand only moves 1 step (1 layer).

Results

First # is solution nodes, second # is expanded nodes.

	Test1	**Test2**	**Test3**
bfs	**11, 14**	**39, 66**	**387, 1344**
dfs	**11, 15**	**39, 62**	**449, 1059**
iddfs	**11, 97**	**39, 1432**	**387, 326465**
astar	**11, 15**	**39, 49**	**387, 782**

Discussion

bfs: complete and optimal as expected.

dfs: complete and maybe optimal as expected.

iddfs: complete and optimal as expected.

astar: complete and optimal as expected.

Conclusion

What can you conclude from these results? Dfs may not find optimal solution. Bfs, iddfs, astar are complete and optimal. Also, dfs's running time is far larger than others. When test number gets bigger, aster will use least time to run.

Which search algorithm performs the best? Aster. Since aster almost has same running time with others in the low test number case. And its expanded nodes much less than others when number is bigger.

Was this expected? Yes, by comparing to algorithm books.