**Methodology**：

In this programming assignment, I am using Python to perform the searches. I defined a Node class which contains 7 member variables: leftB, leftW, leftC, rightB, rightW, rightC, and depth which are indicating a state of the problem: the boat on the left bank, the wolves on the left bank, and the chicken on the left bank and so on. Depth is indicating which level the Node is on the tree which will be used in IDDFS to control the loop. Also, in this Node class, a initializer to initialize all 7 member variables, assignment operator overloading, ‘>’ operator overload and ‘<’ operator overload. I did these because I found these comparison operators are useful. PrintNode() method will going to print the content of the Node to the screen. Get\_string() method will format the data stored in member variables and output a beautiful string that can be output to a file later. Heuristic\_eval() method will calculate the heuristic value of the node, and return it to astar search. I will take about it more later.

Before to perform any search, a command line checking is needed to make sure user has provided enough command line arguments. Then check if start and goal state file are existed to avoid error in the runtime. Identify which mode user typed in, and direct to corresponding search method.

In BFS, create two arrays: one is storing visited nodes, and one is storing the nodes need to expand, and put initial state into both array. This step will be repeated in all searches. If the queue is not empty then we pop out the first node in the queue to see if that is the goal node. If it is not a goal node, we expand that node (expand in BFS means put new Nodes in the end of the array) and keep going to next loop till we find goal.

In DFS, create two arrays, same operations as BFS, but while expanding a Node, put new Nodes in the front.

In IDDFS, create two same arrays, this time I used nested loop. Outer loop is the loop that controls the max depth of the search tree, and I set the upper limit to be very large. And inner loop does the checking and expanding things. Things are different here. To check if the Node reaches the max depth limitation, I need a way to know the depth of a Node. Then member variable depth can be used here. I treat the depth of the initial node to be 1, and new Nodes will copy the depth from their parent and + 1. A new condition is here: if the node that we are exploring reaches our max depth limitation, we don’t add its children to the queue, otherwise its children will break the rule. Since IDDFS is doing DFS in limited version, we put new Nodes in the front of the queue, too.

In astar search, the important thing is defining an admissible heuristic function. The heuristic I did here is adding the number of all items on the left bank and subtracts the number of all items on the right bank. Since I assume goal state will have empty right bank. Priority queue is also used to replace the queue in other searches, because heuristic value determines which Node worth explore. The overall flow of the algorithm is basically the same as BFS and DFS.

**Results:**

|  |  |  |  |
| --- | --- | --- | --- |
| BFS | Test#1 | Test#2 | Test#3 |
| # of nodes expanded | 14 | 110 | 760 |
| # of nodes on the solution path | 14 | 110 | 760 |

|  |  |  |  |
| --- | --- | --- | --- |
| DFS | Test#1 | Test#2 | Test#3 |
| # of nodes expanded | 11 | 82 | 562 |
| # of nodes on the solution path | 11 | 82 | 562 |

|  |  |  |  |
| --- | --- | --- | --- |
| IDDFS | Test#1 | Test#2 | Test#3 |
| # of nodes expanded | 11 | 72 | 562 |
| # of nodes on the solution path | 98 | 2639 | 160751 |

|  |  |  |  |
| --- | --- | --- | --- |
| ASTAR | Test#1 | Test#2 | Test#3 |
| # of nodes expanded | 14 | 121 | 939 |
| # of nodes on the solution path | 14 | 121 | 939 |

**Discussion and Conclusion:**

I found that IDDFS always has best solution in all searches algorithms but it takes longer, especially in test#3, it takes much much longer than others. Astar search always has the worst results in all algorithms, it is not what I expected. The heuristic evaluation must be bad due to this.

I think iddfs can perform better if we put larger starting max depth limitation initially for large test, this could saves a lot of time. If data is expected to take short path, we could use smaller starting place.