1. Methodology

I. Breadth-First-Search

Breadth-First-Search algorithm uses FIFO data structure list of frontier in order to travel the graph breadth by breadth. To implement this algorithm, I used C++ <deque> library, and then used .push_back(), .front() and .pop_front() function to operate this library as queue. Furthermore, to track the solution path, I used <vector> library with structure of node in order to implement hash table of node.

II. Depth-First-Search

Depth-First-Search algorithm uses LIFO data structure list of frontier in order to travel the graph depth by depth. To implement this algorithm, I used C++ <deque> library, and then used push_back(), .back() and .pop_back() function to operate this library as stack. Furthermore, in order to track the solution path, I used <vector> library with structure of node to make hash table.

III. Iterative-Deepening-Depth-First-Search

Iterative-Deepening-Depth-First-Search algorithm is operating DFS algorithm for limited amount. I set the counter of DFS operation and the limitation which ends the DFS. Moreover, in order to prevent the infinite loop of DFS, I set the maximum depth of DFS as 1000, which is a great depth that can approximately verify there is no solution state in the graph. This algorithm iterates DFS until the counter reaches the limitation. Then limitation is incremented, and the list of frontier and visited nodes are cleared. After the reset, this algorithm does the same process with incremented limitation until the limitation reaches maximum depth.

IV. A* Search

A. Implementation

In the aspect of implementation, A* Search algorithm is similar to the BFS algorithm. A* Search algorithm uses priority queue to find the solution path. So, I used C++ <queue> library, and then set the priority element of node to be the value of evaluation function $f(n)$. Furthermore, I set the node which has less value of $f(n)$ to have higher priority in the queue.

B. Heuristic Function

$$f(n) = h(n) + g(n) = N(wolf\ in\ right\ bank)$$

$$h(n) = N(wolf\ in\ right\ bank), g(n) = 0$$

Since the graph is unweighted, I had to evaluate the cost of each traversal in the graph. The cost for each movement was uncertain so that I chose the direction of $f(n)$ to reasonably underestimate the cost, but not extremely underestimate the cost such as $f(n) = 0\, for\ every\ n$. Furthermore, I had to create the $h(n), and\ h(goal\ node) = 0$.

Then I realized that this ‘wolves and chickens puzzle’ has same structure with the 8-puzzle game in the lecture slide. Since the heuristic function which counts the misplaced tiles in the 8-puzzle game is admissible, I tried to set the amount of a variable as heuristic function of ‘wolves and chickens puzzle.’ In case of the 8-puzzle game, the number of misplaced tiles was the significant constraint to complete the puzzle. Thus, the significant constraint is the heuristic function of ‘wolves and chickens puzzle’, and the significant constraint for completing the puzzle was the number of wolves in the bank. Since all of the animals of this puzzle is conserved, I chose the number of wolves in the right bank which significantly affects the completion of the puzzle, or winning or losing of this game.

Thus,

$$f(n) = h(n), h(n) = N(wolves\ in\ the\ right\ bank,$$

$$f(goal\ node) = h(goal\ node) = 0.$$

2. Results

I. Node Expanded

	BFS	DFS	IDDFS	A* Search
Test 1	28	25	154	28
Test 2	315	191	5705	160
Test 3	3123	2507	1086910	2128

II. Depth(Number of Nodes in solution path)

	BFS	DFS	IDDFS	A* Search
Test 1	12	12	12	12
Test 2	34	38	38	34
Test 3	392	396	396	392

3. Discussion

In test case 1, DFS was the most efficient and effective search algorithm because DFS had the least node expansion and the least depth of solution path among the search algorithms. However, in the rest of the test cases, A* Search algorithm was the most efficient and the effective search algorithm because A* search algorithm had the least node expansion, and the least depth of solution path among the search algorithms. I think the heuristic function of A* Search algorithm is not sufficiently admissible in finding solution path for test case 1. However, the heuristic function of A* search was admissible for the rest of the test cases.

Before implementing the search algorithms, I expected A* Search algorithm will outperform rest of the search algorithms for all of the test cases, and IDDFS algorithm will be the worst search algorithm for all of the test cases because of iterating DFS for each depth of the graph. Furthermore, BFS will have less depth of solution path than DFS, and more node expansion than DFS. As a result, the expectation of algorithms exactly matched to the experiment results except the test case 1.

The interesting behavior happened in A* Search algorithm implementation. When I set $g(n)$ in extremely large positive number such as 1000, the result was same when $g(n) = 0$. I think this happened because $g(n)$ was unified to a single constant so that it did not affect the result of the algorithm.

4. Conclusion

A. What can you conclude from these results?

From the results, in uninformed search, DFS is the most effective search algorithm in the aspect of node expansion(memory usage), and BFS is the most effective search algorithm in the aspect of the depth of the solution path(time consumption for finding solution). However, IDDFS is the worst search algorithm in the aspect of both node expansion and depth of the solution path.

In informed search, A* Search algorithm outperformed the rest of the search algorithms when the heuristic function never overestimates the cost to reach goal. However, if the heuristic fails not to overestimate the cost to reach goal, then uninformed search outperforms A* Search. Thus, the performance of A* Search highly depends on the heuristic function.

B. Which search algorithm performs the best?

If we measure the performance of the algorithm in the aspect of node expansion, DFS performed best in test case 1, and A* Search performed best in the rest of the test cases.

If we measure the performance of the algorithm in the aspect of depth of solution path, BFS and A* Search performed best in the entire test cases.

Thus, in general, A* Search algorithm performed best because A* Search algorithm performed best in both aspects, except test case 1.

C. Was this expected?

I expected A* search algorithm will outperform rest of the search algorithms in every test cases. However, I did not expect the results of test case 1. Therefore, except test case 1, these were expected.