

1. Search strategy used, implementation details, data structures, time/space complexity.

We implemented two search strategies during our experiments, Breadth-First Search (BFS) and A* search. We ended up using the BFS strategy.

First, we came up with an uninformed strategy, the BFS. We came up with this idea as all edge costs are equal if we treat each action (moving adjacent and jump sequence) as 1 cost. Given the circumstances, BFS should yield a lowest-cost path.

To implement this, we hash the current board to mark visited grids and initiated a FIFO queue to maintain the current position of the red frog, current board status and all valid moves from this node using helper functions. Note that we also adopt depth-first search (DFS) to store all possible moves for one position in one of the helper functions. Then, we loop through each move in all the possible moves. If the new board has not occurred before after the move is applied, we add it to our queue and keep searching using the same method. We keep looping through the queue and applying the possible moves until the first time we reach the bottom row. If a solution is found, we return the moves, and none otherwise.

The time and space complexities are both $O(b^d)$, as followed from the standard BFS. Here, the branching factor b is the number of actions for each state. The adjacent action for moving is 5. The jump sequence accounts for the rest of the possible actions. For our single frog problem, this is fairly small. The number of frogs influences the possible jump sequence, and possible sequences are fairly limited since the direction and blue frogs' position are fixed (our red frog cannot go back and has no need to jump over the same frog). Thus, even though we find jump sequences using a DFS helper function, the effect on the performance is negligible. However, it is still required to check for all 5 directions if a jump is not valid. Thus, the branching factor is ≥ 10 (and should be very close to 10 on average). In worse cases, depth $d = 8 * 8 = 64$ as the length of the board is 8. This is the total number of reachable states. However, 64 is a very high estimate because the average case of depth is likely to be around 8, as the frog keeps going downwards instead of visiting every node. The depth estimation is even less if jump sequences exist.

Then, we experimented with the A* search. We first define an appropriate Manhattan Distance function as the heuristic function and a heap as the priority queue, ensuring that we are always exploring the path with the lowest f-score (total cost). We use the queue to store the current f-score, g-score, frog position, board status and valid moves. Then we loop through the heap, get all possible moves using a DFS helper function and apply each move to find the path with the lowest f-score. We end the loop until we reach the bottom row.

The space and time complexities are both $O(b^d)$, and the priority queue is of $O(\log N)$. However, in practice, A* search usually performs better than BFS due to the heuristic function and priority queue, such that it will try to search for the lowest cost path every move and thus exploring much fewer nodes.

We ended up adopting the BFS approach, as it always gives us a correct and optimal solution. Although A* search is more efficient, it was hard to define a heuristic function that is admissible for jump-over sequences while being close to the actual cost before even seeing the board. This makes A* search suboptimal in edge cases. When we try to define a heuristic function that underestimates the cost all the time, the performance gets worse very quickly and has no advantage over BFS, while not promising

100% optimality either.

2. If you did not use a heuristic-based approach, justify this choice.

We ended up using BFS which is not a heuristic-based approach, as we encounter difficulties defining a heuristic function that is always admissible when considering jump-over sequences.

In our A* search attempt, we test with a modified Manhattan distance as our heuristic function. We first compute the vertical distance from the current frog's position to the bottom row. Then, we measure the current horizontal distance by calculating how far the frog is from the two middle columns (column 4 and 5). Finally, we take the minimum of these two values as our heuristic.

However, this heuristic function tends to overestimate in cases where there is a long jump-over sequence. Consider cases where the red frog starts on positions (r, c) where $r < (BOARD_N)/2$ (4), $c < 3$ or $c > 5$, but can jump consecutively over blue frogs that end up being at least 2 distances vertically downward. The whole action counts as 1 cost, but the heuristic could easily overestimate the jump sequence at a much greater cost. For A* to be optimal, the heuristic must be admissible but still being as close as possible to the actual cost, otherwise the it cannot guarantee a better performance than BFS. Therefore, we decided to go with BFS instead.

3. Imagine that all six Red frogs had to be moved. Discuss how this impacts the nature of the search problem, and how solution would need to be modified in order to accommodate it.

For a naive suboptimal solution, we can do BFS for each frog once. The issue is instead of a single-agent problem, having multiple frogs could cause scenarios where the orders of frogs is important. The red frogs can block each other when some frog move before other frogs that are supposed to move first, blocking their possible paths to the bottom row. In one word, a solution might exist but not to be found by this approach, as we have not handled the order of movement from different frogs. However, this approach has relatively small impact on performance, since we just need to run BFS once for each frog.

If we aim for an optimal solution, we can track state of all frogs and their possible moves at the same time. It will be very expensive in time/space complexities and branching factor and depth will both multiply by 6 in worst cases, given that we have $O(b^d)$ for both. This could lead to exponential growth for both time and space complexities, making it very impractical in larger boards with more frogs. The state complexity of the frogs will also rise from $O((BOARD_N)^2)$ to $O(6(BOARD_N)^2)$ to track the positions of all 6 frogs and modify the board, as well as other frogs' jump sequences. On the bright side, it may fix many issues we listed in the previous approach.

Additionally, we also need to change the goal state check to check if all 6 frogs are at the bottom row in response to the new rule. We might also change the hash function for the board and the DFS valid-move storing helper function to accommodate the frogs. For our search logic, we now need to handle jump-overs between the red frogs instead of just blue frogs and track their progress simultaneously.

Reference:

LLMs have been used to improve readability and formatting in the codes. We also used LLMs to generate the skeleton of the A* search algorithm for experimentation purposes. However, A* search was not accepted as the final solution, and neither is part of the formal submission to the problem. It is used to demonstrate our thought process and complement the report.