**1. With reference to the lectures, which search strategy did you use? Discuss implementation details, including choice of relevant data structures, and analyse the time/space complexity of your solution.**

We used the BFS (Breadth-First Search) algorithm for determining the path with the least moves, storing the coordinates of the red frog as the nodes. We use the position of the frog as our nodes instead of the state of the board. This is because in the problem with only 1 frog, and a fixed set of lily pads and non-moving blue frogs, the only changes made to the state of the board is when the previous lilypads disappear. However, the shortest path of the frog would not revisit a previous position/lilypad, so the disappearing lilypad is irrelevant to the calculation of the final path.

We chose this search strategy over several others due to its correctness, as it will always be able to identify the existence of a solution, and guarantee it is the optimal solution.
We used a queue data structure to store nodes in the correct order of visiting, and we used a python set to store all the nodes we have visited already.

For the generation of moves, generating single step moves was simple by iterating over all possible directions. For moves involving jumping, we used a recursive function which generates all possible nodes the red frog could arrive at through a single sequence of jumps.

The search space has 64 vertices, and at every node, the maximum number of moves is 64, for every possible next position it could visit. Let N = 64 be the size of the board. The time complexity of BFS is O(V + E), where V = N, and E <= N * N, since each node has at most N edges. So, the time complexity should be O( N + N * N ) = O(N^2). The space complexity of BFS is O(V), so the space complexity should be O(N).

**2. If you used a heuristic as part of your approach, clearly state how it is computed and show that it speeds up the search (in general) without compromising optimality. If you did not use a heuristic based approach, justify this choice.**

As part of our approach we did not use a heuristic, for several key reasons. The first being that our initial approach of BFS was implemented and reached all our test cases along with the given tests in a sufficient amount of time. Thus, we view the potential benefits of having a heuristic to be unnecessary.

Secondly, we thought that for this specific problem, the use of a heuristic search wouldn't be useful. We couldn't use the greedy algorithm, since it was not optimal.
We didn't use the A* algorithm, because condition on the heuristic function h(n) <= h*(n) would have made the heuristic function much less effective. Specifically, h*(n) could be low even when the red frog is far away from the destination due to jump moves. This makes it too difficult to quickly calculate an effective heuristic that properly estimates the actual distance without overshooting. On the other hand, if we tried, for example, to make h(n) <= 1 for all n, the

heuristic could be much too conservative, and the search would end up behaving similarly to a BFS. We believe that the additional cost of using a priority queue for the heuristic search outweighs the benefits.

**3. Imagine that all six Red frogs had to be moved to the other side of the board to secure a win (not just one). Discuss how this impacts the nature of the search problem, and how your team's solution would need to be modified in order to accommodate it.**

If we were to code a program where we have to find the optimal solution for 6 different frogs this would be unfeasible for our program and likely we would not be able to give a solution for it. This is because at each step we would have to iterate over each of the frogs each at a unique position. In our program we have at most 5 possible moves(if we do not count the jump moves) to check in the scenario. This is because given we would have 5 possible directions for each frog which would mean a branching factor of 5* 6 for each frog. This is 30 possible moves at least each turn. Just to demonstrate the difference in scale Ignoring the possibilities for jumps, a simple calculation for 1 frog has 5^7(78125) possible searches but 6 different frogs would mean 30^(7*6) (1.0941899e+62).

In order to accommodate this change numerous optimisations would need to be made such as using symmetry to get rid of checking the same board states. Moreover a heuristic like how close the frog is to the end could be used. Maybe even an iterative depth search to keep things on track and manageable, and possibly some pruning of states where after a certain milestone of moves the states without a sum of length of all the frogs to the end less than a certain value is removed. However these optimisations definitely mean that the solution if found due to the use of heuristics and pruning means that the solution may not be optimal.