TOADS AND FROGS JUMPING PUZZLE: A PROBLEM THAT EMPLOYS GRAPH THEORY

THE INTIAL STATE

THE GOAL STATE (*which has to be achieved by moving these toads and frogs following certain rules of the puzzle***)**

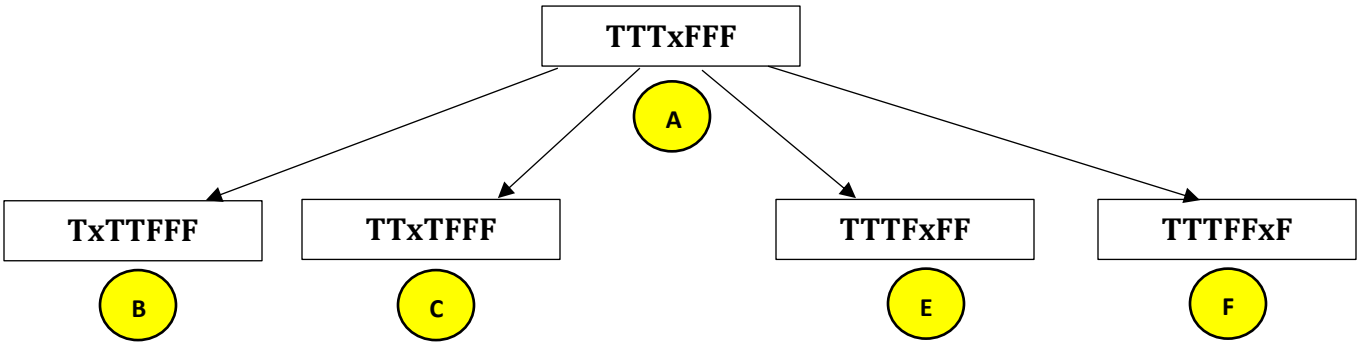
The rules are:

1. The toads and frogs can only move in FORWARD direction, means that a toad can only move towards the right and a frog can only move towards the left.
2. Each toad or frog can either move one step forward, or leap over another toad or frog to occupy the empty space present.
3. The puzzle cannot be solved further if goal state is not achieved and no permitted move is available.

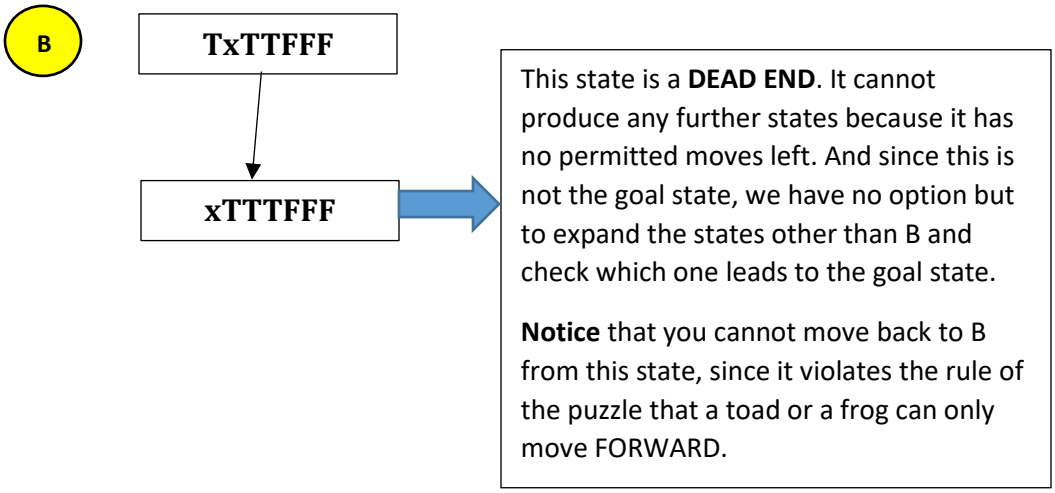
THE SOLUTION

We will represent toads with the letter '**T**'; frogs with the letter '**F**' and empty space with the letter '**x**'. Therefore, our initial state is "**TTTxFFF**" and our goal state is "**FFFxTTT**".

How can this puzzle be solved using graph theory? Let's take a look at in how many ways we can modify the initial state.



The initial state A has option of 4 permitted moves and therefore can generate FOUR new states, each of which may, or may not lead to the goal state. Let's try expanding state B.



By now we have realized that each state generates new states depending on the number of permitted moves it has, and once a new state is generated, you cannot move back to the previous state because it violates one rule of the puzzle. These are very important points for making a decision about what type of graph should be used to solve this puzzle.

Each state would act as a node in our graph. Since you can move from previous state to next state, but vice versa is not possible, this implies one-directional relationship between the nodes of graph.

What would be the **algorithm** to generate this graph? It is very simple:

1. Generate new states from each state already in the adjacency list. (Initially this would only be the initial state)
2. If any of the new states is already present in the adjacency list, create an edge between its parent state and itself, otherwise, introduce it in the adjacency list and then create the edge.
3. Stop the procedure if all states in the adjacency list have produced their all possible new states, and all DEAD ENDS have been discovered (one of these would be the goal state).

One important point to keep in mind is that each state in our graph may have more than one parent state, means that there is a possibility that there exist multiple routes between any two states inside the graph.

Once we have generated our graph, we have to find a path from the initial state to the goal state. This requires understanding of some algorithms of data structures which I won't cover in detail here. This algorithm would be **THE BREADTH FIRST SEARCH**, which traverses a graph breadth wise, while storing predecessor of each node in an array and keeping record of predecessor of each node. On reaching the goal state, you can **trace back** your path to the initial state. This path would be the shortest of all possible paths and therefore will be called **OPTIMAL SOLUTION** to the puzzle problem.

SO WHAT IS THE OPTIMAL SOLUTION TO THIS PUZZLE PROBLEM?

TTTxFFF---> TTxTFFF---> TTFTxFF---> TTFTFxF---> TTFxFTF ---> TxFTFTF---> xTFTFTF---> FTxTFTF ---> FTFTxTF ---> FTFTFTx ---> FTFTFxT ---> FTFxFTT---> FxFTFTT ---> FFxTFTT---> FFFTxTT ---> FFFxTTT

This is the optimal solution to this puzzle. It took 15 moves to reach the goal state. I am attaching the code of solution with this report. The code is in Java programming language.