Solving Towers of Hanoi with A\* Algorithm

CS471

Kent Huynh

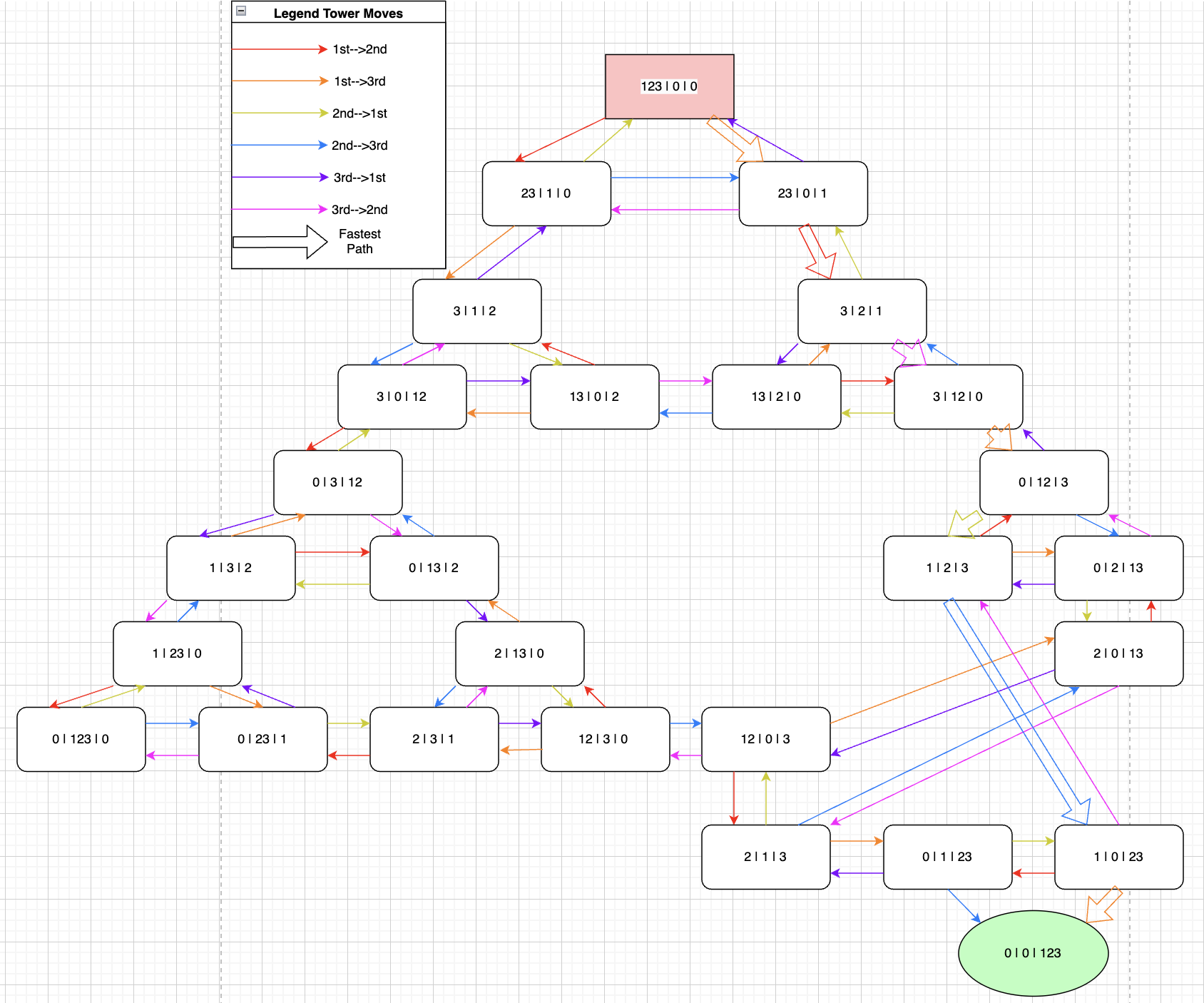
Computer Science (CS)

Check List

1. \_X\_ Did you follow all the requirements in implementing the system?
2. \_X\_ Did you create one Word file of your report with the cover page and section headers as specified?
3. \_X\_ Did you answer all questions per section?

**Section 1: Analysis of the Problem Space (6.1)**

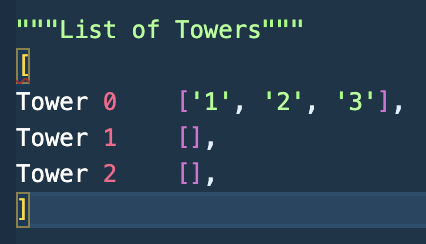
1. List all possible disk move actions/operators (what goes where) – note that only the disks at the top can move to another peg:
   1. 123 | 0 | 0
   2. 23 | 1 | 0
   3. 23 | 0 | 1
   4. 3 | 1 | 2
   5. 3 | 2 | 1
   6. 3 | 0 | 12
   7. 3 | 12 | 0
   8. 13 | 0 | 2
   9. 13 | 2 | 0
   10. 0 | 2 | 13
   11. 0 | 3 | 12
   12. 0 | 12 | 3
   13. 0 | 13 | 2
   14. 1 | 3 | 2
   15. 1 | 2 | 3
   16. 1 | 23 | 0
   17. 1 | 0 | 23
   18. 2 | 13 | 0
   19. 2 | 3 | 1
   20. 2 | 1 | 3
   21. 2 | 0 | 13
   22. 12 | 3 | 0
   23. 12 | 0 | 3
   24. 0 | 123 | 0
   25. 0 | 23 | 1
   26. 0 | 1 | 23
   27. 0 | 0 | 123
2. Therefore, the Branching factor: <= ??
   1. 2
3. Draw all states that are reachable/legal, and draw all possible arrows between them to create a graph. No node shall be shown twice (i.e. one node per state).
   1. Image below
4. Label the arrows with moves (you may use a “legend” to make it easy to label links).
   1. Color arrow shows what towers the move indicates
5. Mark the initial and goal states. Mark the shortest path from Initial to Goal.
   1. Initial state is red, Goal state is green, shortest path is marked with big arrow as shown in legend



**Section 2: Designing the Evaluation Function for A\* (6.2)**

1. Give the equation for the estimate function **h** (i.e. how do you guess the # of moves?)
   1. h = num of disks in first tower + (total num disks - num of disks in last tower)
2. Give an example **h** value and **g** value for **3** of the states in the above Problem Space.
   1. 23 | 0 | 1
      1. h = 2 + (3-1) = 4
      2. g = 1
   2. 0 | 12 | 3
      1. h = 0 + (3-1) = 2
      2. g = 4
   3. 1 | 3 | 2
      1. h = 1 + (3-1) = 3
      2. g = 5
3. Defend your decision for the **h** function:
   1. We are trying to relocate the disks from the first tower to the last tower
4. Is this a good estimate of how many more moves are required?
   1. It is not a good estimate of how many more moves are required, however it is a good value to estimate which move will get us to our goal
5. Is this h admissible? Explain why.
   1. h is admissible because we are trying to find the path that will get us to the goal in the least amount of moves and g in this situation is enough to do that

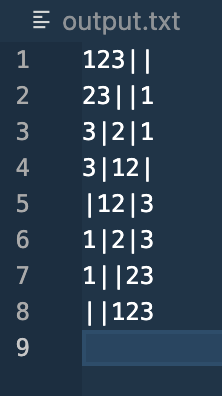
**Section 3: Implementation (Source Code is submitted separately) (6.3)**

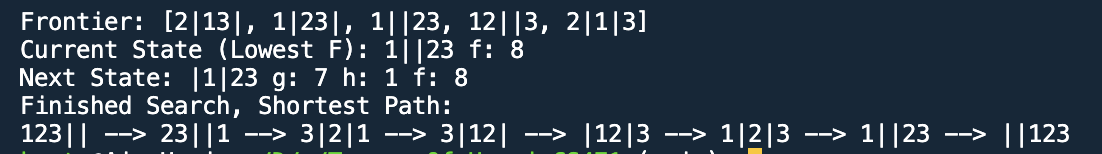
1. Must have functions dedicated to do the following. Write the name of the function next to each:
   1. Basic framework of puzzle Name: class Game:
   2. Generating all the new states Name: Game.get\_valid\_moves()
   3. Choosing the next state to expand Name: Game.move()
2. What data structure (type and name) did you use for each node/state?
   1. List - Game.towers
3. Give an abstract picture of it with examples values.
   1. 
4. What data structure (type and name) did you use to store all the Frontier nodes/states?
   1. List - open\_list

**Section 4: Testing and Results (6.4)**

1. Test your program **very** **thoroughly** and make sure the output matches your expectations.

Produce **a .txt file of test results** (recorded script compiled with g++).





1. Your analysis of the results:
   1. The program finds the solution with the shortest amount of moves and records the states of the path in order
2. Did it work as expected? If not, explain exactly where it failed. (This includes correctness of all the intermediate outputs)
   1. Yes
3. How many disk moves did it take to reach the goal?
   1. It took 7 moves to reach the goal
4. Is that optimal? Or a person can do with fewer moves? Check against your graph.
   1. That is the optimal amount of moves for a puzzle with 3 pegs and 3 disks, and also matches the shortest path in our graph

**Section 5: Ideas for Adding Machine Learning**

1. **Give one way to incorporate machine learning into your program to make it better?**

* What part of the program will the puzzle program update/adjust to improve itself?
* How and when would this learning happen in your program?

(Being advised by a human as it plays? Exactly at what point in the program?

Automatically by observing an outcome? Exactly at what point in the program?)

Be very specific **using an example.**

One way to incorporate machine learning into the program to make it better is to use a neural network to decide what moves should be made. By letting the neural network model train by completing the puzzle and rewarding it when it completes the puzzle in fewer moves, we can adjust its weights until it favors moves that lead to the optimal amount of moves to solve the puzzle.