**Solving Tower of Hanoi with an   
A\* Algorithm**

**CS471 – Intro to Artificial Intelligence**

**Cody McKinney**

**Software Engineering Major**

Check List

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

Program in C++  
Report for  
Dr. Rika Yoshii  
12/16/2022

Contents

[**Puzzle System**](#_gjdgxs)3

[**Section 1: Analysis of the Problem Space (6.1)**](#_30j0zll)3

[**Section 2: Designing the Evaluation Function for A\* (6.2)**](#_1fob9te)5

[**Section 3: Implementation (6.3)**](#_3znysh7)7

[**Section 4: Testing and Results (6.4)**](#_2et92p0)10

[**Section 5: Ideas for Adding Machine Learning**](#_3dy6vkm)12

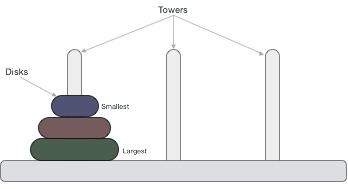
# **Puzzle System**

**Your task is to implement a puzzle agent using C++.**

**This time, you have to solve the 3 pegs 3 disks Towers of Hanoi using A\*.**

**The initial state is having all 3 disks on Peg 1.**

**The goal state is having all 3 disks on Peg 3.**



# **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:**

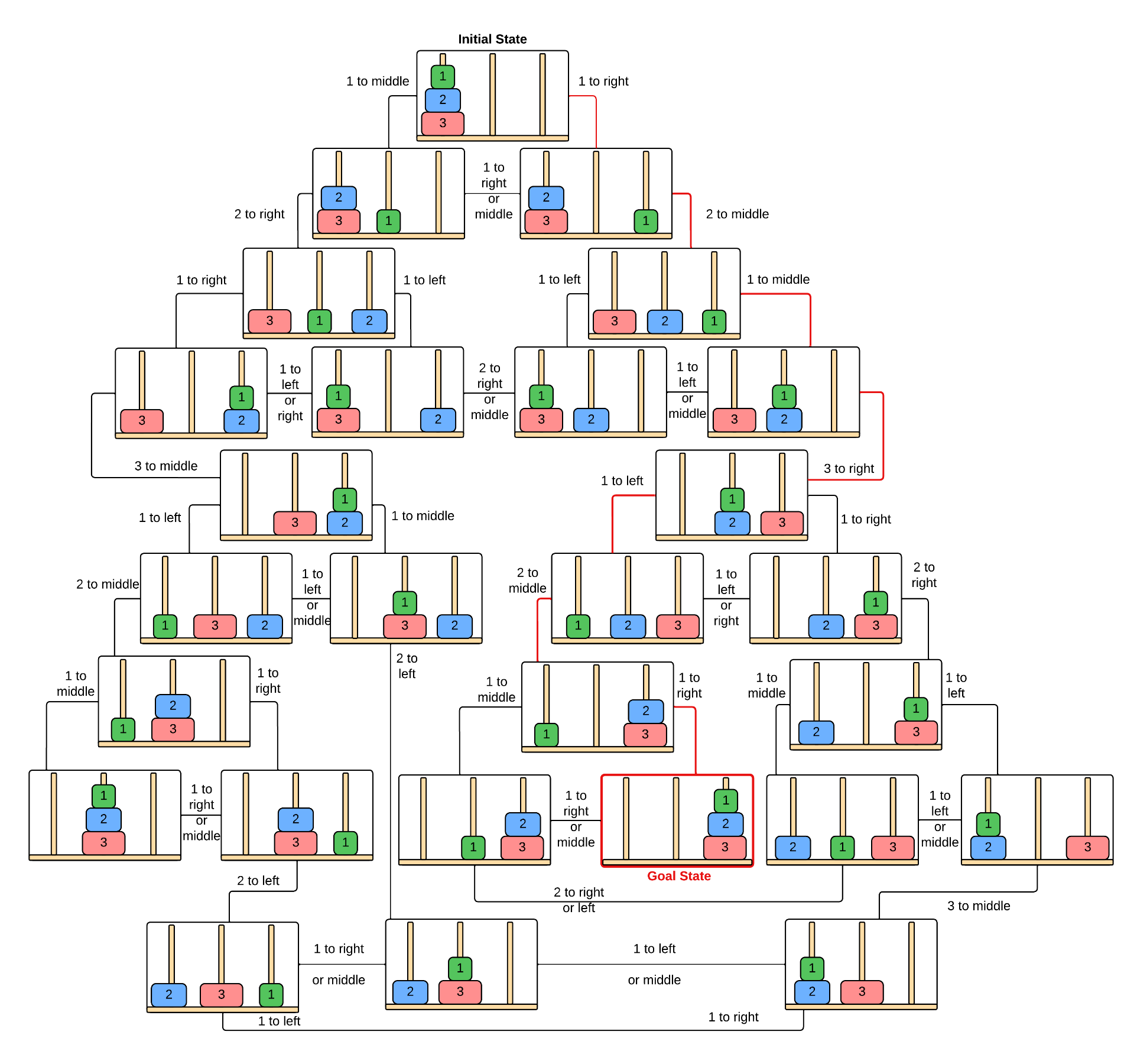
**a**: top disk on left peg to middle peg or right peg

**b**: top disk on middle peg to left peg or right peg

**c**: top disk on right peg to left peg or middle peg   
\*assuming that the move does not place a disk on top of a smaller disk

1. **Therefore, the Branching factor:** <= 3 (if you count the option to backtrack)
2. **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).**
3. **Label the arrows with moves (you may use a “legend” to make it easy to label links).**
4. **Mark the initial and goal states. Mark the shortest path from Initial to Goal.**

Diagram Next Page



\* Red path is shortest path to goal.

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

**Assume that g = number of disk moves so far**

**h = estimate of how many more moves from a given state**

**f = g+h is the goodness**

1. **Give the equation for the estimate function h (i.e., how do you guess the # of moves?)**

The estimate function h will be based on the number of disks on the first two pegs.

1. **Give an example h value and g value for 3 of the states in the above Problem Space.**Diagram

   Description automatically generated

(1.) There have been 3 moves to reach this state  
g = 3  
  
and 3 disks on the first two pegs h = 3

f = g + h = (3 + 3) f = 6  
  
This is a pretty close estimate because optimal  
solution is 7 turns.A picture containing diagram

Description automatically generated

(2.) It has taken 5 moves in order to reach this state.  
g = 5

and 2 disks on the first two pegs h = 2

f = g + h = (5 + 2) f = 7

7 is the optimal number of moves to solve so our evaluation function was not far off.

(3.) At this state we have done 4 moves.Diagram

Description automatically generated

g = 4

but there is only 1 disk on the first 2 pegs h = 1

f = g + h = (5 + 2) f = 7

This is not a great estimation from this position because  
there is still another 8 moves to be made from this state.   
  
However, from this state, we have already gone down a poorly chosen route, and will actually have to backtrack to get back to optimal route.

1. **Defend your decision for the h function:**
2. **Is this a good estimate of how many more moves are required?**

This is a good estimate in general because the number of disks plus the number of moves is very close to the total number of optimal moves for the solution. This is helpful when comparing states because we will always choose the route with the lowest f, and for most states the f value is pretty accurate to total moves.

1. **Is this h admissible? Explain why.**

This h function is admissible because our h function will never overestimate from any state. The number of disks on the first two pegs will always be less than the number of moves needed to solve.

# **Section 3: Implementation (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:** state **File:** hanoi.cpp

Stateholds the board it came from, and the current board, along with values for heuristic evaluation. Name: state File: hanoi.cpp

class state

{

public:

    board current;

    board camefrom;

    int g;

    int h;

    int f;

};

Boardholds the 3 stacks representing the pegs, implements the methods for moving pegs on the board. Name: board File: board.cpp

class board

{

private:

public:

    board();

    ~board();

    // pegs for tower of hanoi puzzle

    stack left;

    stack middle;

    stack right;

    // display board, with 3 pegs and disks

    void printBoard();

    // change position of a disk

    void changePeg(stack &curr, stack &target);

    // find a disk on the peg, el\_t is the element number of the disk

    string find(board &b, el\_t x);

    // determine if move is safe

    bool safemove(stack &curr, stack &target);

};

Stack used to represent each peg (Last-In-First-Out). Stack implementation is my   
stack lab submission from Dr. Yoshii’s 311 class with additions for the puzzle.   
 Name: stack File: stack.cpp

class stack

{

private:          // to be hidden from the client

    el\_t el[MAX]; // el is  an array of el\_t's

    int top;      // top is index to the top of stack

public: // available to the client

        // Add exception handling classes here

    class Overflow

    {

    };

    class Underflow

    {

    };

    // prototypes to be used by the client

    // Note that no parameter variables are given

    stack();  // constructor to create an object

    ~stack(); // destructor  to destroy an object

    void push(el\_t);

    void pop(el\_t &);

    void topElem(el\_t &);

    bool isEmpty();

    bool isFull();

    void displayAll();

    void clearIt();

    el\_t returnElement(int i);

};

* 1. **Generating all the new states Name:** generateAll(state old)  **File:** hanoi.cpp



* 1. **Choosing the next state to expand** **Name:** bestofFrontier() **File:** hanoi.cpp



1. **What data structure (type and name) did you use for each node/state?**
2. **Give an abstract picture of it with examples values.**Diagram

   Description automatically generated with low confidence

State object holds the board it came from, and the current board,   
along with values for heuristic evaluation.

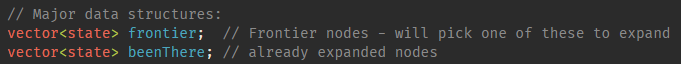
Board object holds the 3 stacks representing the pegs,   
implements the methods for moving disks on the pegs.

Stack used to represent the peg, used for Last-In-First-Out concept

In this image, this is the initial state, g = 0, h = 3, f = 3  
  
Below is class definition of the state and board objects.

1. **What data structure (type and name) did you use to store all the Frontier nodes/states?**

I used a vector of states called frontier, found in hanoi.cpp vector<state> frontier

****

**To compile:**  g++ stack.cpp board.cpp hanoi.cpp

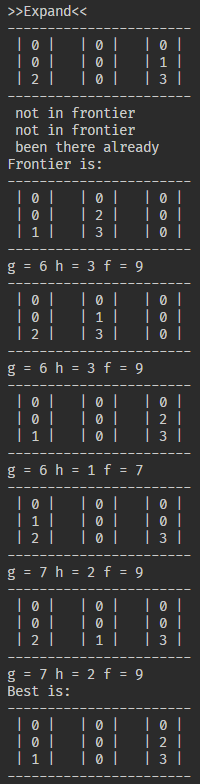
# **Section 4: Testing and Results (6.4)**

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++).**

test.txt included in submission.  
 output of traceback output of frontier after expanding

**A screenshot of a computer

Description automatically generated with medium confidence** ****

1. **Your analysis of the results:**
2. **Did it work as expected? If not, explain exactly where it failed. (This includes correctness of all the intermediate outputs)**

It works as expected by finding and displaying the optimal path correctly after expanding states.

1. **How many disk moves did it take to reach the goal?**

The optimal path that is found uses 7 disk moves and displays the moves and the states it came from in order to reach the goal state.

1. **Is that optimal? Or a person can do with fewer moves? Check against your graph.**

My program finds the optimal path and returns the 7 moves needed to reach goal state. A person cannot reach the goal state in less than 7 moves, so yes this is optimal.

# **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?**

To improve itself I would say the program would need to adjust the value of f by assigning a weight to states that it has previously tried expanding that led to a bad route. In the early stages of the program, a lot of the f values are close together or even equal. When the f value is equal, the algorithm picks at random a state to expand only to have to backtrack to this decision and make the other choice.

* **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.**

Ideally this should happen automatically by observing an outcome. When the program expands a state and finds the next 2 expansions have the same f value, if it chooses to expand one of those states that leads to a non-optimal route, the program should assign a weight value to that original state it selected when the f values matched. The first time it begins generating states, it may try expanding off a state that is a bad choice, if this is found, when it backtracks, the system should add a weight to the bad state. The next time it begins generating states, when it reaches the 2 choices that previously had the same f value, the state that leads to a bad path will have a weight assigned to it which will increase the f value, and the system will choose the optimal state.  
  
This method is almost as if by incorporating Machine Learning, the puzzle solver is creating its own heuristic value based on weights from previous attempts.