"Tower of Hanoi" implementation in C++

Milos Vukadinovic
American University in Bulgaria

COS460
Algorithms
Prof. Emil Kelevedjiev

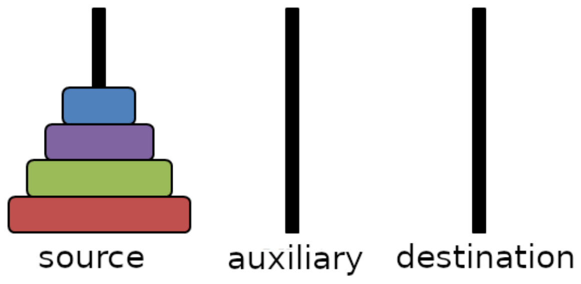
April 18, 2020

1 Assignment

The Tower of Hanoi is a mathematical game or puzzle. It consists of three rods and a number of disks of different sizes, which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on leftmost rod, the smallest at the top, thus making a conical shape. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

- Only one disk can be moved at a time. list
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
- No larger disk may be placed on top of a smaller disk.

My assignment is to implement solution to the "Tower of Hanoi" in C++ with simple graphic animation
I will implement various approaches to the problem and compare their efficinecy.



2 Recursive approach

We will name the rods from left to right: source rod, auxiliary rod, destination rod. Following is recursive algorithm:

```
recursive(n,source,target,spare)
        if(n==1) move from source_rod to target rod
        else{
                //recursively move all but the biggest
                recursive(n-1,source,spare,target)
                // move biggest dist to the target
                recursive(1,source,target,spare)
                // recursively spare to target
                recursive(n-1,spare,target,source)
        }
```

Now let's find out what is the minimum number of moves we need to perform in order to solve the puzzle. We have the following reccurence relation.
T(n)=T(n-1)+1+T(n-1)=2*T(n-1)+1
Using telescopic method we see the following:

T(n) = $2^0 + 2^1 + 2^2 + ... + 2^{n-1} \implies T(n) = \sum_{i=0}^{n-1} 2^i$

Consider M = $2^0 + 2^1 + 2^2 + ... + 2^n \implies M = \sum_{i=0}^{n} 2^i$

Then we know $M = T(n) + 2^n$ and $2 * T(n) = M - 1$

Exchanging M in second eq. $2 * T(n) = T(n) + 2^n - 1 \implies T(n) = 2^n - 1$
In conclusion, the minimum number of moves we need to solve the puzzle is $2^n - 1$ where n is the number of disks.

We can confirm the result using induction. Starting from the reccurence relation $T(n) = 2 * T(n-1) + 1$

- Basis step: $T(0) = 0, T(1) = 1$ It is trivial that to solve a puzzle with 0 disks we need 0 moves and for 1 disk we need 1 move.
- Induction Hypothesis: $T(n) = 2^n - 1$
- Induction step: $T(n+1) = 2*T(n)+1 \implies T(n+1) = 2^n - 2 + 1 \implies T(n+1) = 2^n - 1$ So we confirm the result we obtained earlier.

3 Iterative approach

The list of moves given by our recursive approach has many regularities. We can observe them and come up with an iterative approach to the problem. I will explain one of the simplest iterative approaches. We proved that the minimum number of moves we need to perform in order to solve the puzzle is $2^n - 1$. Our approach will be to alternate moves between smallest and second smallest accessible disks. In other words, we will first move smallest accessible disk to the first rod on the right that allows us to perform legal move, and then we will move second smallest accessible disk and move it to the first rod on the left that allows us to perform legal move. However, if starting number of elements is even, always move smallest to left and second smallest to the right. Also, consider our rods to be in the field of n=3, in other words rod on the left of 0 is 2, and rod on the right of 2 is 0.

By observing the results from our previous approach, we can see that if the number of a move is m (zero indexing)

- if m divided by three gives a remainder 0, perform a move between source and destination rod
- if remainder is 1 perform a move between source and auxiliary rod
- if the remainder is 2 perform a move is between auxiliary and destination rod.

Notice that I did not say perform a move from-to rod. A move between 2 rods is defined as either a move from 1st rod to the 2nd rod or from 2nd rod to the 1st rod, depending what rules of the game allow. This is a pseudo code for the algorithm.

```
for i=0;i < (1 << n)-1 ;i++
        // If the number of disks is even
        // switch aux and dest rod.
        if(n%2==0) swap(aux,dest)
        if(i%3==0) legal move between source and destination rod
        if(i%3==1) legal move between source auxiliary rod
        if(i%3==2) legal move between auxiliary destination rod
```

4 Binary Solution

Now that we now from the iterative approach that we can sort each move in three groups based on the number of a move, we will use bits to solve the problem.
This is the description of our binary representation of the problem:

- There is one bit for each disk
- The leftmost bit represents the largest disk and the rightmost smallest
- If a bit has value 0 zero it means that a disk is on leftmost rod, and if it has value 1 on the rightmost rod
- A bit with the same value as the previous one means that it is stacked on the top of the previous one on the same rod (i.e all zeros mean that all disks are on the leftmost rod and all ones rightmost rod)
- A bit with a different value to the previous one means that corresponding disk is one position to the left or right of the previous one. In order to determine we use this rule:
  Let's say that we want to determine a position of m-th disk (m-th bit) then let n be the number of bits in the interval (0,m) such that their value is the same as the value of preceding bit. Add 1 to n if the largest disk is on the leftmost rod(i.e. if leftmost bit is 0). If n is even, the disk is located one rod to the left, if n is odd, the disk located one rod to the right (in case of even number of disks and vice versa otherwise).

Example: Describe the situation of 25th move, when we have 5 disks.
25_{10}=11001_2

- Largest disk is on the dest peg
- Disk two is stacked on the largest disk on dest
- Disk three is 0 (n=1 $\Longrightarrow$ go right) it's on the src
- Disk four is one (n=1 $\Longrightarrow$ stay in place) it's on the src
- Disk five is one (n=2 $\Longrightarrow$ go left) it's on the dest

In order to find what movement did we perform in m-th step we use the following formula. We perform movment from
(m & m -1) % 3 to (m + (m & -m)) %3)

5 Results

In this section we describe the results.

6 Conclusions

We worked hard, and achieved very little.