Mathematic for Computer Science: Hanoi's Tower Problem

Vu-Lam DANG

Oct 26, 2018

Hanoi Tower is a classical mathematics/computer science problem. Given a set of n disks and k pegs, the goal of the puzzle is to move a stack of disks from the initial (or Departure) peg to the target (or Arrival) peg. Furthermore, the disks must be stack in an increasing order by weight.

1 Solution for $n = 5$ and $k = 3$

```
Initial state:

    1 -> [1 -> 2 -> 3 -> 4 -> 5]
    2 -> []
    3 -> []

Move list:

    (1,3) (1,2) (3,2) (1,3) (2,1)
    (2,3) (1,3) (1,2) (3,2) (3,1)
    (2,1) (3,2) (1,3) (1,2) (3,2)
    (1,3) (2,1) (2,3) (1,3) (2,1)
    (3,2) (3,1) (2,1) (2,3) (1,3)
    (1,2) (3,2) (1,3) (2,1) (2,3)
    (1,3)

Total move: 31
```

2 Classical Problem

In order to solve the problem for $[5, 3]$, we divide the problem into subproblems. First, we move 4 disks to the intermidiate peg. The 5^{th} disk (the largest disk) can now be move to the destination peg. The 4 disks stack is then moved onto the destination peg.

This algorithm give us a recursive solution to Hanoi Tower problem. Each iteration give us 2 subproblems (one for move $n-1$ disks to intermediate peg, and one more to move said stack to the destination). Thus, the complexity for this solution is $\Theta(2^n)$.

In fact, because each iteration the algorithm call exactly 2 subproblems, the number of move required to move disks from one peg to another is strickly $2^n - 1$ (minus one because the original state is called once).

3 Coding of the position

In order to encode a state of the game we can ultilize 3 linked list to decode the state of the pegs. The head of the list refer to the disk on top of said stack (FILO), thus one can avoid puting a larger disk on top of the smaller disk. In this case, memory space complexity is $\Theta(n)$

In order to encode a move we simply need the original and destination pegs, or 2 variables. The disk at the head of the original peg is detached and transfer to the destination peg.

```
Example:
    State:
        1 -> [1 -> 2 -> 3]
        2 -> []
        3 -> [4 -> 5]
    Move(1,3)
        1 -> [2 -> 3]
        2 -> []
        3 -> [1 -> 4 -> 5]
    Move(1,2)
        1 -> [3]
        2 -> [2]
        3 -> [1 -> 4 -> 5]
    Move(3,2)
        1 -> [3]
        2 -> [1 -> 2]
        3 -> [4 -> 5]
```

4 Variation: Unbounded number of pegs

For a variation where $k > n$ the puzzle become trivial. A simple solution where all $n - 1$ disks are distributed throughout the intermediate pegs, leave room for the n^{th} disk to arrive at the destination peg before recollecting the stack at the destination peg is completed in $2n - 1$ move, will have time complexity of $\Theta(n)$.

```
Example: n=5, k=6
    Initial state:
        1 -> [1 -> 2 -> 3 -> 4 -> 5] //Departure
        2 -> []
        3 -> []
        4 -> []
        5 -> []
        6 -> [] //Arrival

    Move list:
        (1,2) (1,3) (1,4) (1,5) (1,6) //Dispertion

    Intermediate state:
        1 -> [] //Departure
        2 -> [1]
        3 -> [2]
        4 -> [3]
        5 -> [4]
        6 -> [5] //Arrival
```

```
Move list:
    (2,6) (3,6) (4,6) (5,6) //Recollection

Final state
    1 -> [] //Departure
    2 -> []
    3 -> []
    4 -> []
    5 -> []
    6 -> [1 -> 2 -> 3 -> 4 -> 5] //Arrival
```

5 Improved solutions

The two cases $H(n, \Theta(n)) = \Theta(n)$ and $H(n, \Theta(2^n)) = \Theta(2^n)$ represent 2 extreme of the input space for Hanoi's Tower problem, the best and worst case respectively. These 2 cases orcur when $k > n$ or $k = 3$ respectively, as shown in previous sections.

Consider a game where $n = \Delta(k-1)$ where $\Delta(k)$ is the triangular number of k. An algorithm to solve $H(\Delta(k-1), k)$ is stated as following: For each immidate peg j between $k-2$ and 1, build a j-high column of disks until only one disk left (the largest disk) in the Departure stack - this disk is now moved to the arrival peg. Finally, each immidate peg j between 1 and $k-2$ is deconstructed and stacked up at the final destination.

A graphical demonstration:

```
Original state
    1:=====.....======= (n disks) // Departure
    2:                            // Arrival
    3:
    .
    .
    k:

Immidiate state:
    1:=                           // Departure
    2:                            // Arrival
    3:=====.....===== // j=k-2 disks
    4:====.....===== // j=k-3 disks
    5:===.....===== // j=k-4 disks
    .
    .
    .
    k:= // j=1 disks

Final state:
    1:                            // Departure
    2:=====.....======= (n disks) // Arrival
    3:
    .
    .
    k:
```

Remark: For j from k-2 to 1, the member disks of the next column is larger than the previous column. For example j = k-2 will have k-2 disks from 1 to k-2; j = k-3 will have k-3 disks from k-2 to 2 k -5

From section 4, we show that the construction of a n disks high stack with $k > n$ stack have complexity of $\Theta(n)$. Therefore, for each immidiate stack in the previous algorithm have its complexity of $\Theta(j)$ where j is the height of the stack (and also the position of the peg). Thus, the total complexity of this case is $\sum_{j=1}^{k-2} \Theta(j) = \Theta(n)$.

In conclusion, for $n \leq \Delta(k-1)$ with k is a constant will have a linear cost solution (complexity of $\Theta(n)$). Furthermore, if $n > \Delta(k-1)$ the solution will no longer be linear, as it will require stack higher than j disks, which in turn require more costly subproblem similar to the classical problem.