AN IMPROVED ALGORITHM FOR FINDING OUT THE PEG STATUS AT ANY MOVEMENT OF M-TH DISK OF THE TOWER OF HANOI

Sardar Anisul Haque[1] **Md. Abdul Mottalib**[2]

Abstract

Tower of Hanoi is a familiar puzzle used to show the principle of mathematical induction, the power of recursive algorithms etc. Its general solution is based on recursion method that can be defined generally as the method of solving a problem by dividing it into two or more sub-problems, each of them are same like the original problem in nature, but smaller in size. This paper shows the status of all three pegs after each movement of any disk. It also computes the moves of a particular disk mathematically without solving all disks' movement. Based on this mathematical analysis a new algorithm is devised here.

I. Introduction

The Tower of Hanoi, a popular puzzle of the late nineteenth century consists of three pegs mounted on a board together with disks of different sizes, with the largest on the bottom [1,2,4,5,12]. No existing algorithm is able to describe the status of pegs at any movement of a particular disk without solving all disks movement. Its general solution is based on recursion method that can be defined generally as the method of solving a problem by dividing it into two or more sub-problems, each of them are same like the original problem in nature, but smaller in size [1,2,4,5,12].

Many researchers have tried to state good solutions of the tower of Hanoi problem in different solution [3,6-11]. In [6] a non-recursive solution of multi-peg tower of Hanoi has been given, where intermediate peg is more than one. If we apply this analysis in our single temporary peg tower of Hanoi problem it will compute all disks movement to tell the description of each peg after each movement. Also in [3] that computes all disks to solve a single disk's movement.

The rules of the tower of Hanoi can be summarized as follows: If n disks are arranged on the first peg largest at the bottom and each disk is sitting on a large disk then transferring of the n disks to the third peg will characterize the problem according to the following five conditions:

---

1. Student, Department of Computer Science and Information Technology, Islamic University of Technology, Board Bazar, Gazipur 1704, Bangladesh
2. Head, Department of Computer Science and Information Technology, Islamic University of Technology, Board Bazar, Gazipur 1704, Bangladesh

i) Only one disk at a time may be moved;
ii) Only a topmost disk may be moved;
iii) A disk can not be placed on a disk smaller than itself;
iv) A minimum number of disk moves are to be moved; and
v) Only one intermediate location may be used.

Now have a look to the problem. If total disk is 3 (i.e n=3) then to solve this problem, first we have to transfer top two disks to second peg (temporary peg) otherwise we can not move the larger disk as shown in Figure-1.1. Again to do so we have to transfer smaller disk to third peg (destination peg). But for this case we have to take this peg as temporary peg. Then move the medium disk to second peg. After passing the smaller disk from third peg to second peg, only then we can move larger disk to third peg. We have now two disks to be transferred to the third peg as shown in Figure-1.2. Next move the smaller disk to first peg. Then move the medium disk to third peg and finally the smaller one to the third peg as shown in figure –1.3.

1)

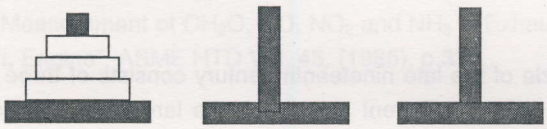

Figure-1.1: Initial position with three disks in source peg

2)

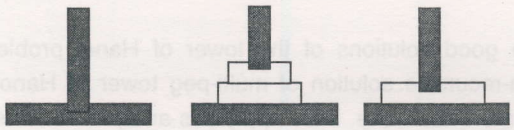

Figure-1.2: After three movements.

3)

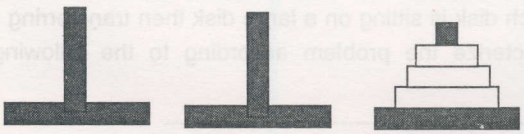

Figure-1.3: After complete the game (seven movements).

So we need seven moves. These moves are given below:

i) Move smaller disk from first peg to third peg.
ii) Move medium disk from first peg to second peg.
iii) Move smaller disk from third peg to second peg.
iv) Move larger disk from first peg to third peg.
v) Move smaller disk from second peg to first peg.
vi) Move medium disk from second peg to third peg.
vii) Move smaller disk from first peg to third peg.

II. Representing a big Number

One critical issue is to represent a large number in any tower of hanoi algorithm. We can solve this problem by using exponential. For example, if we represent 1125899906842624 then it is not possible to show it by any variable in C, C++ or pascal language. But we can easily represent it by 2^{50}. Next section discusses some mathematics and shows that any movement of tower of Hanoi can be represented as the summations of one number of this form 2^x and the other number of the same form with a multiplication. The mathematics discussed in the next section use this notation to represent any movement number.

III. Mathematics Behind Tower of Hanoi

1) If total disks are n then n-th disk will move one time, (n-1)-th disk will move two times and (n-2)-th disk will move four times and so on. We know that the total number of movement is equal to $2^n - 1$.

So,

$1+2+4+8+\ldots+2^{n-1} = 2^n - 1$

or, $2^0 + 2^1 + 2^2 + 2^3 + \ldots + 2^{n-m} + \ldots + 2^{n-1} = 2^n - 1$

Here m is a number indicating m-th disk from top. So the total movement of m-th disk is 2^{n-m}. The condition to start one particular disk's (m-th) movement is to remove m-1 disks from peg1 to peg 2 or peg 3. So any particular disk m will start moving at the move number 2^{m-1}.

2) A disk may start its journey from first pole to second pole or third pole. It depends on the position of the disk from last disk. If its position is odd it starts its journey moving third pole first otherwise second pole. If we write 1 to 2^{n-1} consecutive integers then we can specify the movement number of n-th disk, which is 2^{n-1}. It divides the movement number into two subgroups (1 to $2^{n-1}-1$ and $2^{n-1}+1$ to $2^n - 1$)
In this two subgroups, the movement of (n-1)-th disk can be easily specified as 2^{n-2} and $2^{n-2} + 2^{n-1}$, because first subgroup denotes the movement of (n-1) disks from 1-st peg to

2-nd peg and second subgroups denotes the same thing from 2-nd peg to 3-rd peg. In this way we can specify all other disks' movement.

The top disk will move peg 3 first where peg 2 is used as temporary peg. In first subgroup there is only one move for (n-1)-th disk that is peg1 to peg 2. In second subgroup there is only one move for (n-1)-th disk that is peg 2 to peg 3. In this way we can find details of all other disks that is written in (iv) of the following observations.

3) The following observations are made in this study. Let total disks is n and m-th disk denotes one particular disk where $1<=m<=n$.

(i) Total movement of the m-th disk is equal to 2^{n-m}.

(ii) The m-th disk starts its journey from 2^{m-1}.

(iii) The difference between two movements of that particular disk is equal to: 2^m.

(iv) If the position of a particular disk is odd from bottom it starts with its journey from peg1 to peg3. Otherwise peg1 to peg2.

If it goes to peg3 first then it continues its journey in this sequence:

Peg 1 to peg 3
Peg 3 to peg 2
Peg 2 to peg 1
Peg 1 to peg 3

If it goes to peg2 first then it continues its journey in this sequence:

Peg 1 to peg 2
Peg 2 to peg 3
Peg 3 to peg 1
Peg 1 to peg 2

4) One interesting thing is that if we create a full binary search tree with 1 to 2^n-1 consecutive numbers then each level will represent the movement of one disk. The explanation of this similarity is based on the formula derived in next section. That is: between two consecutive movements of a particular disk there exists only one movement of another disk that is bigger than that particular disk. This formula is true for all disks except the bottom most disk as there is no disk that is bigger than that. this is also true for a uniform set of disks. Uniform set of disks with m disks ($1<=m<n$, where n is the number of total disks) is the set containing the top most disk that means first disk, second disk, third disk,.....(m-1)-th disk, m-th disk. For example: uniform set of disks with 3 disks is the set of disks containing first, second, and third disk. In binary search tree the first node of any level x (with number 2^{x-1}) right children construct a set of consecutive integers from 1 to 2^{x-1}-1, if we visit the right children in in-order rule. The number 2^{x-1} indicates the first movement of x-th disk and the right children indicates the movement of the uniform set of disks with (x-1) disks. In this way if we apply this method we can get the next movement of x-th disk in the same level of the binary tree. For example: top level (i.e. root) indicates the bottom most disk's movement and the bottom most leaves indicate the top most disk's movement. For example, let n=4 whose corresponding binary tree structure is shown in Figure-2. Then the number of movement

need to complete the Tower of Hanoi is equal to 2^4-1, or 15. If we make a complete binary search tree then the top level represent the movement of bottom disk. And in the same way the leave indicates the movement of top disk.

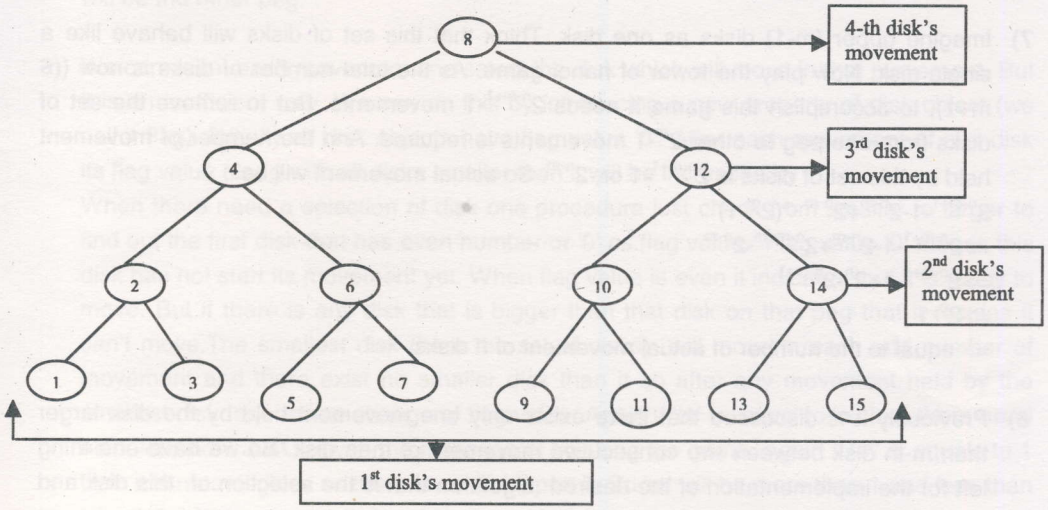

Figure-2: A complete binary search tree with 1 to 15 consecutive integers.

5) Previous discussion says about the starting movement of any disk along with the difference between its two consecutive moves. So, any particular disk's (let m) p-th movement can be expressed as $2^{m-1}+2^{m}*(p-1)$

6) Before starting any particular disk's (let m) movement, all disks smaller than it must be removed from source peg to the peg other than the peg that will first visited by m-th disk. We can treat this as a tower of hanoi problem with (m-1) disks. After removing m-th disk we get (m-1) disks pilled up in a peg. Before removing (m+1)-th disk we have to remove (m-1) disks to the peg where m-th disk is placed. So, again a small tower of hanoi with (m-1) is needed. As (m+1)-th disk is removed to a blank peg, all disks smaller than (m+1)-th disk now must be pilled up on it in order to create a blank peg for (m+2)-th disk. From the above discussion, it is clear that there exist two small tower of hanoi game with (m-1) disks between two consecutive movement of m-th disk.
This two tower of hanoi games need total $2^{m-1}-1+2^{m-1}-1$ movement. But the difference between two consecutive movement of m-th disk is 2^m. So, there must exist 2^m-1 movements.
Now look at the calculation:
$2^m-1=2^{m-1}-1+2^{m-1}+x$

or, x =1

Here x indicates the number of movements held by the disks that is larger than m-th disk between two consecutive movement of it. We get x as 1. So there exist only one movement that is held by disk larger than m-th disk between its two consecutive movement. The choice of this disk and its present peg along with destination peg need some explanation.

7) Imagine upper (m-1) disks as one disk. Think that this set of disks will behave like a single disk. Now play the tower of hanoi game. As the total number of disks is now (n-m+1), to accomplish this game it needs 2^{n-m+1}-1 movements. But to remove the set of disks from one peg to other 2^m-1 movements is required. And the number of movement held by this set of disks is 2^{n-m+1}-1 or, 2^{n-m}. So actual movement will be:

$2^{n-m+1}-1-2^{n-m}+2^{n-m} *(2^m-1)$

$=>2^{n-m+1}-1-2^{n-m}+2^{n-m+m}-2^{n-m}$

$=>2^{n-m+1}-1+2^n-2^{n-m+1}$

$=>2^n-1$

=> equal to the number of actual movement of n disks.

8) Previously it is discussed that there exists only one movement held by the disk larger than m-th disk between two consecutive movement of m-th disk. So we have one thing left for the implementation of the desired algorithm that is the selection of this disk and its present location and its destination. As we think (m-1) disks as a single disk, we can think m-th disk as 2-nd disk and (m+1)-th disk as 3-rd disk and so on. This transformation can be done by the following equation:

New position = old position - m + 1

Now we can think it as a simple tower of hanoi with (n-m+1) disks. So the disk choice becomes simple. And we have to find out the movement of 2-nd disk's movement.

IV. Description of the Proposed Algorithm

1) Now think about algorithm implementation. In this paper we will discuss about an object oriented approach. We treat each disk as an object. The set of (m-1) disk is treated as an object. We will maintain three peg object to tell the situation of each peg after any movement. This peg object can be used for visual implementation of this game. To do this we have to either sort this peg object after each insertion or treat those as three stacks object. In our algorithm we have done it as three peg objects as simple storage of disks. Initially all these three peg objects will be blank except source peg. All disks are pilled on that disk. When all disks are on destination peg then the game is over. This may be used as an indicator that indicates the termination of the game. But the algorithm shown in this paper does not use this technique.

2) Now think about disk objects. They have a property (we call it present location) which tells the present location of its. Initially all disk object have this property with source peg value. They also have another property (we call it next_location) which tell the peg that is next visited by that disk object. What will be the next_location value of each disk object initially ? The answer of this question is discussed in paragraph 2 & 3 of Section-III. After each movement of any disk its present_location will be next_location and next_location will be the other peg.

3) We can count each movement and select the disk which will move in that movement. But it is an inefficient way. We can do this by introducing a new property of disk object (we call it flag). Initially all disk objects have flag value '0'. After each movement of any disk its flag value along with all disks smaller than it will be increased by 1.
When there need a selection of disk one procedure just check from smaller to larger to find out the first disk that has even number or '0' as flag value. Flag value '0' means this disk has not start its movement yet. When flag value is even it indicates that it is ready to move. But if there is any disk that is bigger than that disk on that peg that it resides it can't move.The smallest disk (here the set of disks) will move in each odd number of movement and there exist no smaller disk than it so after any movement held by the smallest disk requires no change of flag value. Again if 'm' is equal to 'n' then there need no calculation. Just it gives the message of finishing the game. Again if 'm' is equals to 1 then it is meaningless as there is no disk upon it. So 'm' will be more than 1 and less than 'n'.

v) Proposed Algorithm

The algorithm has some procedures. The main procedure (name TOH()) is given below:

```
## TOH()
input n; input m;
check if m<2 & m>=n
    then exit() giving a message
    else
        initialize_peg() // initialize three peg objects
        initialize_disk()  // initialize (n-m+1) disks object
      for i ← 1 to 2^(n-m-1)
      move 1-st disk or set of disks
      update_disk(set)
      update_peg(set)
          show_peg()
      move 2-nd or mth disk
      update_disk(m)
      update_peg(m)
      show_peg()
      move 1-st disk or set of disks
```

```
        update_disk(set)
        update_peg(set)
        a ← select_disk()
        if(a==NULL)
              break;
        else
                move a
                update_disk(a)
                update_peg(a)
          next i
     show_peg()
END  // end of TOW()
```

To initialize the peg objects initialize_peg() is written. It is given below:

```
## initialize_peg()
pile all (n-m+1) disks on source peg
make other pegs blank
END // end of initialize_peg()
```

To initialize the disk objects initialize_disk() is written. It is given below:

```
## Initialize_disk()
1) Give all disks'  present_location as source peg
2) Give all disks'  next_location as destination peg or the other peg according to the rule
given  paragraph 2 & 3 of Section-III.
3)  Give all disks' flag value '0'
END //  end of initialize_disk()
```

The algorithm show_peg is used to display the status of each peg . The algorithm is given below:

```
## show_peg()
1)show the content of source peg in a sorting order
2)show the content of destination peg in a sorting order
3)show the content of the other peg in a sorting order
END // end of show_peg()
```

The algorithm of update_disk() is given below:

```
## update_disk(a)
1)   present_location = next_location
next_location = other than disks previous value of present_location and next_location.
If (a!=set&&a>m) then increase flag values of all disks' object including a-th disk that is
smaller than a-th disk but larger than m-th disk by 1
END // end of update_disk()
```

The algorithm of update_peg() is given below:

update_peg(a)
put a disk from a.present_location peg to a.next_location peg
END //end of update_peg()

The algorithm of select_disk() is given below:
select_disk()
Find the smallest disk object 'b' from (m+1)-th to n-th which has flag value even or '0'
If no such 'b' is present then b NULL
Return b
END // End of select_disk()

VI. Discussion

The main target of this paper is to introduce a new algorithm for finding out any particular disk's movement along with the description of the three pegs after each movement. The mathematics used to derive the algorithm is straightforward and simple. From this mathematics it is easy to predict what will happen if one or more disk is added in or subtracted from the set of disks (i.e (m-1)disks) or from disk below m-th disk. It turns (m-1) peg as one single peg. So it needs to compute less to describe each peg after each movement of m-th disk.

References

[1]. Horowitz, Ellis and Sahni, Sartaj, Computer Algorithms, The McGraw-Hill Companies, Inc., New York 1992.pp.101-110.

[2]. Kenneth H Rosen, Discrete Mathematics and its Application, The McGrawHill Companies, Inc., New York 1999 4-th edition.

[3]. M. Lu, "Towers of Hanoi graphs," *International Journal of Computer Mathematics*, vol. 19, no. 1, pp. 23-38, 1986.

[4]. Nell Dale, Susan c. Lilly, Pascal Plus Data Structures Algorithms and Advanced Programming, Tata McGraw-Hill companies, New Delhi 1985.

[5]. Thomas H. Cormen, Charles E. Leiserson, Ronald L.Rivest, Introduction to Algorithm, Prentice hall, New Delhi, 1999.

[6]. X.-M. Lu and T.~S. Dillon, "Nonrecursive solution to parallel multipeg Towers of Hanoi: a decomposition approach," *Mathematical and Computer Modelling*, vol. 24, no. 3, pp. 29-35, 1996.

[7]. X.-M. Lu and T.~S. Dillon, "Parallelism for multipeg Towers of Hanoi," *Mathematical and Computer Modelling*, vol. 21, no.3, pp. 3-17, 1995.

[8]. X.-M. Lu and T.~S. Dillon, "A note on parallelism for the Towers of Hanoi," *Mathematical and Computer Modelling*, vol. 20, no. 3, pp. 1-6, 1994.

[9]. X.-M. Lu and T.~S. Dillon, ``Parallel paradigms for the multipeg Towers of Hanoi.'' *Research Report*, November 1991.

[10]. X.-M. Lu, "An iterative solution for the 4-peg Towers of Hanoi," *The Computer Journal*, vol. 32, no. 2, pp. 187-189, 1989.

[11]. X.-M. Lu, "Towers of Hanoi problem with arbitrary k>=3 pegs," *International Journal of Computer Mathematics*, vol. 24, no. 1, pp. 39-54, 1988.

[12]. Yedidyah Langsam, M. J.Augensten, Aaron M. Tenenbaum, Data Structures Using C And C++, Prentice hall, New Jersey 07458.