IEICE TRANS. INF. & SYST., VOL.E94–D, NO.2 FEBRUARY 2011

LETTER Special Section on Foundations of Computer Science — Mathematical Foundations and Applications of Algorithms and Computer Science —

An Optimal Algorithm for Solving the Towers of Hanoi Problem with the Least Storage Used

Yu-Kumg CHEN[†], **Chen-An FANG**[†], ***Nonmembers***, ***and*** **Fan-Chieh CHENG**[††a)], ***Student Member***

SUMMARY The Towers of Hanoi problem is a classical problem in puzzles, games, mathematics, data structures, and algorithms. In this letter, a least memory used algorithm is proposed by combining the source array and target array for comparing the sizes of disk and labeling the disks in the towers of Hanoi problem. As a result, the proposed algorithm reduces the space needed from $2n+2$ to $n+5$, where n represents the disks number.

key words: *Towers of Hanoi, data structure, algorithm, mathematic, game*

1. Introduction

The Towers of Hanoi problem has been an interesting subject over a century since Lucas [1] presented in 1883. In the Towers of Hanoi problem, there are three pegs, labeled 1, 2, and 3, and several numbered disks of different sizes, each with a hole in the center. Initially, all of the disks are on pole 1, with the largest disk on the bottom, then the next largest, and so on.

Figure 1 (a) shows the initial configuration of the Hanoi Tower, which is started with three disks numbered from smallest to largest. The object of the problem is to move all of the disks from peg 1 to peg 3. Note that peg 2 is used for temporary storage. The rules of the problem are:

1. Only one disk can be moved at a time.
2. No disk may ever be placed on top of a smaller disk.
3. Other than the prohibition of rule 2, the top disk on any peg can be moved to either of the other poles.

In 1971, Dijkstra [2] presented a recursive algorithm to solve the Towers of Hanoi problem. Considering the pegs to be in a ring and moving clockwise, Buneman [3] proposed an iterative program to solve the Towers of Hanoi problem in 1980. In 1981, Atkinson [4] used the circular approach to solve the Towers of Hanoi problem.

Based on the binary encoding method, Walsh [5] proposed another recursive algorithm to solve the Towers of Hanoi problem with the assumed n disks. All of these algorithms need $2^n - 1$ moves of disk. The parallel Towers of Hanoi problem is formed when the rule 1 is removed from the Towers of Hanoi problem.

Based on the four types of parallel moves, such as single move, exchanged move, consecutive move, and circular move, Wu and Chen [6] propose an algorithm to solve the parallel Towers of Hanoi problem in 1992. When n is odd, this algorithm can reduce the times of move to $3\times2^{(n-1)/2}-1$. Otherwise, it needs $2 \times 2^{n/2} - 1$ moves.

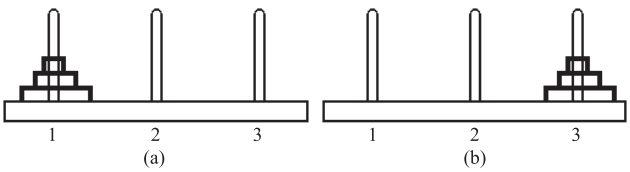

Fig. 1 Example of the Towers of Hanoi problem with three disks (a) the starting position (b) the final position.

All of these algorithms take a lot of memory to complete the Towers of Hanoi problem. For example, Buneman's method [3] takes $2n$ memory space, Atkinson's method [4] takes $O(2.733^n)$ memory space. In 1996, Chedid and Mogi [7] proposed a simple iterative algorithm that uses two one-dimensional arrays with size of n and two memory variables by applying the concept of the binary tree. On the other hand, $2n + 2$ memory is needed to solve the Towers of Hanoi problem in this algorithm. In this letter, a least memory used algorithm is proposed by combining the source array and target array for comparing the sizes of disk and labeling the disks in the towers of Hanoi problem. The proposed algorithm deducts the memory required from $2n + 2$ to $n + 5$.

2. Proposed Algorithm

Let i be the times of moving disk in solving the towers of Hanoi problem and let d be the disk number moved for each time i. For solving the towers of Hanoi problem with n equal to 3, the sequent varied values of d are: 1, 2, 1, 3, 1, 2, 1.

Figure 2 shows a binary tree with $n = 3$. The sequence of d moved can be obtained to solve the towers of Hanoi problem with the inorder traversal of the binary tree. In 1996, Chedid [7] modeled a simple mathematics formula to derive the sequence. We also use this mathematics formula to derive the sequence in the proposed algorithm.

A one-dimensional array FromToPeg[0, ..., $n - 1$] is used to store the source and target pegs with n elements. Besides, another one-dimensional array TopDiskNo[0, ..., 2] is used to store the top disk numbers of three pegs with 3 elements in the proposed algorithm. If there is not any disk on the peg, the top disk number may be marked $n + 1$. Then the proposed algorithm for solving the towers of Hanoi

Manuscript received March 27, 2010.

[†]The authors are with the Department of Electronic Engineering, Huafan University, Taipei, Taiwan.

[††]The author is with the Department of Electronic Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan.

a) E-mail: d9802108@mail.ntust.edu.tw

DOI: 10.1587/transinf.E94.D.240

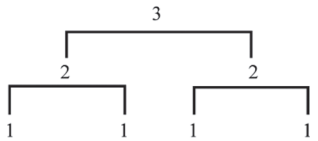

Fig. 2 Binary tree for deriving the value of d with $n = 3$.

```
Procedure Mark_Hanoi
Input: the disk number n
Output: the ways of disks moved
Step 1.   Set FromToPeg to {1, 1, 1, ..., 1}
Step 2.   Set TopDiskNo to {n+1, n+1, n+1}
Step 3.   For i := 1 to 2^n–1 do
            begin
Step 3.1.   Find the largest d (1≤d≤n) such that i mod 2^(d–1) = 0;
Step 3.2.   Display "FromToPeg[d-1]"; /* move disk d from peg FromToPeg[d-1]*/
Step 3.3.   Set TopDiskNo[FromToPeg[d-1]-1] := d;
Step 3.4.   if n is even then
              begin
Step 3.4.1.     if TopDiskNo[(FromToPeg[d-1]) mod 3] > d then
                  begin
                    Set TopDiskNo[(FromToPeg[d-1]) mod 3] := d;
                    Set TopDiskNo[FromToPeg[d-1]-1] :=n+1;
                    Set FromToPeg[d-1] := ((FromToPeg[d-1]) mod 3)+1;
                  end;
                else
                  begin
                    Set TopDiskNo[FromToPeg[d-1]+1 mod 3] := d;
                    Set TopDiskNo[FromToPeg[d-1]-1] :=n+1;
                    Set FromToPeg[d-1] := ((FromToPeg[d-1]) mod 3)+1;
                  end;
Step 3.4.2.     Display "-> FromToPeg[d]"; /*move disk d to peg FromToPeg[d]*/
              end;
            else /* n is odd */
              begin
Step 3.4.3.     if TopDiskNo[(FromToPeg[d-1]+1) mod 3] > d then
                  begin
                    Set TopDiskNo[(FromToPeg[d-1]+1) mod 3] := d;
                    Set TopDiskNo[FromToPeg[d-1]-1] :=n+1;
                    Set FromToPeg[d-1] :=((FromToPeg[d-1]+1) mod 3)+1;
                  end;
                else
                  begin
                    Set TopDiskNo[(FromToPeg[d-1]) mod 3] := d;
                    Set TopDiskNo[FromToPeg[d-1]-1] :=n+1;
                    Set FromToPeg[d-1] := ((FromToPeg[d-1]) mod 3)+1;
                  end;
Step 3.4.4.     Display "-> FromToPeg[d]"; /*move disk d to peg FromToPeg[d-1]*/
              end;
            end; /* for */
```

Fig. 3 Pseudo-code of the proposed algorithm.

problem is shown on Fig. 3.

We trace the executions in order to describe the correction by using a parameter n with odd value (3, 5, 7, etc.) or even value (2, 4, 6, etc.) in the proposed algorithm. When n is equal to 3, the array FromToPeg[0...2] is used to store the source and target pegs in the proposed algorithm.

All the elements of this array can be expressed as "1, 1, 1" by the initialization of 1. Another array TopDiskNo[0..2] is used to store the top disk numbers of three pegs. All the elements of this array can be expressed as "4, 4, 4" by the initialization of 4. Then the proposed algorithm runs a loop with $2^n - 1 = 7$ times. The sequential values of d for each times i in the loop are 1, 2, 1, 3, 1, 2, and 1. The value of d can be derived from the formula in step 3.1. If i is equal to 1, the value of d is 1. The element of TopDiskNo[0] is set to 1 in step 3.4. Since n is odd, the odd processing procedure is executed in step 3.4.3. Since the value of TopDiskNo[2] is 4 which is greater than d, the element of TopDiskNo[0] is set to $n + 1 = 4$ and the element of FromToPeg[0] is set as 3. The new elements of arrays TopDiskNo and FromToPeg are 4, 4, 1 and 3, 1, 1, respectively.

Table 1 Disks moved with $n = 3$.

FromToPeg		TopDiskNo		i	d	MOVE
Before	After	Before	After			
1,1,1	1,1,1	4,4,4	4,4,4	0	0	
[1],1,1	[3],1,1	[1],4,4	4,4,[1]	1	1	1→3
3,[1],1	3,[2],1	[2],4,1	4,[2],1	2	2	1→2
[3],2,1	[2],2,1	4,2,[1]	4,[1],4	3	1	3→2
2,2,[1]	2,2,[3]	[3],1,4	4,1,[3]	4	3	1→3
[2],2,3	[1],2,3	4,[1],3	[1],4,3	5	1	2→1
1,[2],3	1,[3],3	1,[2],3	1,4,[2]	6	2	2→3
[1],3,3	[3],3,3	[1],4,2	4,4,[1]	7	1	1→3

Table 2 Disks moved with $n = 4$.

FromToPeg		TopDiskNo		i	d	MOVE
Before	After	Before	After			
1,1,1,1	1,1,1,1	5,5,5	5,5,5	0	0	
[1],1,1,1	[2],1,1,1	[1],5,5	5,[1],5	1	1	1→2
2,[1],1,1	2,[3],1,1	[2],1,5	5,1,[2]	2	2	1→3
[2],3,1,1	[3],3,1,1	5,[1],2	5,5,[1]	3	1	2→3
3,3,[1],1	3,3,[2],1	[3],5,1	5,[3],1	4	3	1→2
[3],3,2,1	[1],3,2,1	5,3,[1]	[1],3,5	5	1	3→1
1,[3],2,1	1,[2],2,1	1,3,[2]	1,[2],5	6	2	3→2
[1],2,2,1	[2],2,2,1	[1],2,5	5,[1],5	7	1	1→2
2,2,2,[1]	2,2,2,[3]	[4],1,5	5,1,[4]	8	4	1→3
[2],2,2,3	[3],2,2,3	5,[1],4	5,5,[1]	9	1	2→3
3,[2],2,3	3,[1],2,3	5,[2],1	[2],5,1	10	2	2→1
[3],1,2,3	[1],1,2,3	2,5,[1]	[1],5,5	11	1	3→1
1,1,[2],3	1,1,[3],3	1,[3],5	1,5,[3]	12	3	2→3
[1],1,3,3	[2],1,3,3	[1],5,3	5,[1],3	13	1	1→2
2,[1],3,3	2,[3],3,3	[2],1,3	5,1,[2]	14	2	1→3
[2],3,3,3	[3],3,3,3	5,[1],2	5,5,[1]	15	1	2→3

Table 1 lists the results of arrays FromToPeg and TopDiskNo, variable d, and disk moved for each times i. The numbers with blocks of the array FromToPeg in the columns of "Before" and "After" fields are the moving numbers of the source and the target pegs, respectively. From the numbers with blocks of the array TopDiskNo in the columns of "Before" and "After" fields, we can see the changes of the disk numbers on the top of three pegs. In the case of i equal to 1, the disk 1 is moved from the peg 1 to peg 3.

When $n = 4$, the processing steps are the same as that of $n = 3$ before the step 3.4. All the elements in arrays FromToPeg[0..3] and TopDiskNo[0..2] are initialized to 1, 1, 1, 1 and 5, 5, 5, respectively. The sequential values of d for each times i in the loop are 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, and 1. Since n is even, the even processing procedure is executed in step 3.4.1. When d is equal to 1, the element of TopDiskNo[0] is set to $n + 1 = 5$ and the element of FromToPeg[0] is set as 2 because the value of TopDiskNo[1] is 5 which is greater than d. The new elements of arrays TopDiskNo and FromToPeg are 5, 1, 5 and 2, 1, 1, 1, respectively. Table 2 lists the results of arrays FromToPeg and TopDiskNo, variable d, and disk moved for

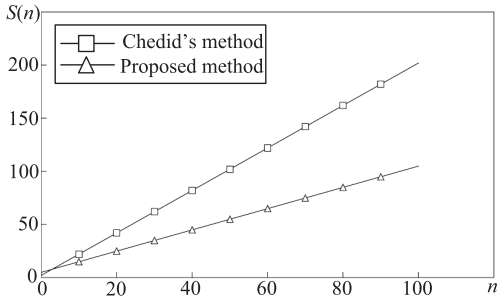

Fig. 4 Memory required for the proposed method and Chedid's method.

each times I with $n = 4$. The numbers with blocks of the array FromToPeg in the columns of "Before" and "After" fields are the moving numbers of the source and the target pegs, respectively. From the numbers with blocks of the array TopDiskNo in the columns of "Before" and "After" fields, we can see the changes of the disk numbers on the top of three pegs. In the case of i equal to 1, the disk 1 is moved from the peg 1 to peg 2.

3. Analysis

Let $S(n)$ be the memory required function of n. Since the method of Chedid [7] uses two one-dimensional arrays with n elements and two memory variables, its memory required can be expressed as

$$S(n) = 2n + 2. \quad (1)$$

Since there are one array with n elements, one array with three elements, and 2 memory variables used in the proposed algorithm, the memory needed can be written as

$$S(n) = n + 3 + 2 = n + 5. \quad (2)$$

Consider the coordinate system with the $S(n)$ and n axes shown in Fig. 4. The lines of the memory required for the proposed method and Chedid's method are shown in Fig. 4. From Fig. 4, the memory needed of the proposed method is always lower than that of the Chedid's method when n is greater than 3.

4. Conclusions

In this letter, we present an effective algorithm to reduce the memory required for solving the towers of Hanoi problem. Our main future work involves using the same concepts of the proposed algorithm to reduce the memory needed for solving the parallel Towers of Hanoi problem. The contribution of the proposed method is using an array with n elements to store the information of pegs moved. Thus, the memory required can be reduced from $2n + 2$ to $n + 5$.

References

[1] M. Gardner, Hexaflexagons and Other Mathematical Diversions: The First Scientific American Book of Puzzles and Games, University Of Chicago Press, 1988.
[2] E.W. Dijkstra, A. Short Introduction to the Art of Programming, EWD 316, 1971.
[3] P. Buneman and L. Levy, "The towers of hanoi problem," Inf. Process. Lett., vol.10, no.4/5, pp.243–244, 1980.
[4] M.D. Atkinson, "The cyclic towers of hanoi," Inf. Process. Lett., vol.13, no.3, pp.118–119, 1981.
[5] T.R. Walsh, "The towers of hanoi revisited: Moving the rings by counting the moves," Inf. Process. Lett., vol.15, no.2, pp.64–67, 1982.
[6] J.S. Wu and R.J. Chen, "The towers of hanoi problem with parallel moves," Inf. Process. Lett., vol.44, no.5, pp.241–243, 1992.
[7] F.B. Chedid and T. Mogi, "A simple iterative algorithm for the towers of hanoi problem," IEEE Trans. Educ., vol.39, no.2, pp.274–275, 1996.