82-20-1

TOWERS OF HANOI AND GENERALIZED TOWERS OF HANOI

Paul Cull and Earl F. Ecklund, Jr.

Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

"Towers of Hanoi and Generalized Towers of Hanoi"

Paul Cull and E. F. Ecklund, Jr.
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

ABSTRACT

We investigate the time and space used by algorithms which solve the Towers of Hanoi problem. We show that any algorithm which solves the problem for n disks must use at least $0(2^n)$ time and $n + 0(1)$ bits of space. We obtain an algorithm which simultaneously attains these lower bounds.

For the generalized problem with t towers and n disks, we show that the solution is unique (up to re-naming towers) if and only if $n<t$ or $n = \binom{k}{t-2}$, for some $k > t-2$.

M.R. Categories: 05-04, 68C25, 68E99.

0. Introduction

The Towers of Hanoi problem is often used as an example of a problem which can be neatly solved by a recursive algorithm, and as an example of a problem which requires exponential time for its solution. In the Towers of Hanoi problem one is given three towers, called usually A, B, C, and n disks of different sizes. Initially the disks are stacked on tower A in order of size (disk n on the bottom, disk 1 on the top). The problem is to move the stack of disks to tower C, moving the disks one at a time in such a way that a disk is never stacked on top of a smaller disk. A solution to this problem is then a sequence of moves which satisfy these rules. An extra constraint is that the sequence of moves should be as short as possible. An algorithm solves the Towers of Hanoi problem, if when the algorithm is given as input n, the number of disks, and the names of the towers, then the algorithm produces the shortest sequence of moves which conforms to the above rules.

In section 1 of this paper we will investigate a variety of algorithms which solve the Towers of Hanoi problem and finally produce an algorithm which, in a certain sense, is the best possible.

The Towers of Hanoi problem can be generalized so that t towers ($t \geq 3$) may be used. When there are three towers the minimal sequence of moves needed to solve the problem is unique. In section 2 we investigate this situation for n disks and t towers and show that the solution is unique (up to relabeling of the towers) if and only if $n<t$ or $n = \binom{k}{t-2}$ for some $k > t-2$.

1. Towers of Hanoi Algorithms

For any problem, there will be an infinity of algorithms which solve the problem. How do we decide which is the "best" algorithm? There are a number of possible ways to compare algorithms. We will concentrate on two measures: time and space. We would like to say that one algorithm is faster, uses less time, than another algorithm if when we run the two algorithms on a computer the faster one will finish first. Unfortunately to make this a fair test we would have to keep a number of conditions constant. We would have to code the two algorithms in the same programming language, compile the two programs using the same compiler, and run the two programs under the same operating system on the same computer, and have no interference with either program while it is running. Even if we could practically satisfy all these conditions, we might be chagrined to find that algorithm A is faster under conditions C, but that algorithm B is faster under conditions D. To avoid this unhappy situation we will only calculate time to order. For our purposes two functions of n, f(n) and g(n), have the same order if $\lim_{n \to \infty} f(n)/g(n) = \text{constant} \neq 0$. We symbolize this relation by $f(n) = O(g(n))$, read f(n) is order g(n). Thus we will consider two algorithms to take the same time if their running times have the same order. In particular, we do not distinguish between algorithms whose running times are constant multiples of one another.

If we find that algorithm A has a time order which is strictly less than algorithm B, then we can be confident that for any large enough problem algorithm A will run faster than algorithm B, regardless of the actual conditions. On the other hand if algorithms A and B have the same time order, then we will not predict which one will be faster under a given set of actual conditions.

The space used by an algorithm is the number of bits of storage (memory) space the algorithm uses. We expect the space to be an increasing function of the size of the problem. Since we have chosen bits as our unit, we can be a bit more exact about space than we can be about time. We can distinguish an algorithm which uses 3n bits from an algorithm which uses 2n bits. But we will not distinguish an algorithm which uses 3n + 7 bits

from an algorithm which uses 3n + 1 bits, because we can hide a constant number of bits in the control structure of the algorithm.

So we will say that we have the "best" algorithm for a problem if we can show that the algorithm has minimal time order, and uses minimal space to within an additive constant.

It is not clear that such a best algorithm must exist. In some problems there is a time-space trade-off, a faster algorithm requires more space. We will demonstrate that this sort of trade-off does not exist in the Towers of Hanoi problem by eventually presenting an algorithm which achieves simultaneously minimal time and minimal space.

A solution to the Towers of Hanoi problem is given by the following recursive algorithm, which appears in a number of textbooks:

```
PROCEDURE HANOI(A,B,C,n)
   IF n=1 THEN move the top disk from tower A to tower C
          ELSE HANOI(A,C,B,n-1)
               move the top disk from tower A to tower C
               HANOI(B,A,C,n-1).
```

Is this the best algorithm for the problem? We will show that this algorithm has minimum time complexity, but does not have minimum space complexity.

Proposition: The Towers of Hanoi problem has time complexity $O(2^n)$.

Proof: One can easily show by induction that the above recursive algorithm correctly solves the problem. A difference equation for the running time, T(n), for this algorithm with n disks is:

$$T(n) = 2T(n-1) + C$$

since the procedure with n disks makes 2 calls on the procedure with n-1 disks, and the testing and movement of one disk is assumed to take constant time C. If we assume the initial condition T(1) = C, then the equation has the solution $T(n) = C(2^n-1)$, which establishes the required upper bound or the time complexity of the problem.

A lower bound can be established by considering the largest disk. It must be moved at least once. But to move the largest disk requires that the n-1 smaller disks must all be moved to a single tower, since none can remain

on top of the largest disk and one of the three towers must be vacant to permit a move of the largest disk. When the largest disk is moved to the target tower, the n-1 smaller disks must be sitting on a single tower and to complete the solution of the problem these n-1 disks must then be moved to the target tower. These two observations lead to the difference equation $M(n) \geq 2M(n-1) + 1$ for the number of moves M(n) needed to solve the Towers of Hanoi. Since one move is required to move one disk, we obtain $M(n) \geq 2^n-1$. Since each move requires at least constant time we have established the lower bound on time complexity.

Since the upper bound and lower bound are equal to order, we have established the $0(2^n)$ time complexity of the problem. ▨

Proposition: Any algorithm which solves the Towers of Hanoi problem must use at least n + 0(1) bits of storage.

Proof: Since the algorithm must produce 2^n-1 moves to solve the problem, the algorithm must be able to distinguish 2^n different situations. If the algorithm did not distinguish this many situations then the algorithm would halt in the same number of moves after each of the two nondistinguished situations, which would result in an error in at least one of the cases.

The number of situations distinguished by an algorithm is equal to the number of storage situations times the number of internal situations within the algorithm. Since the algorithm has a fixed finite size it can have only a constant number of different internal situations. The number of storage situations (states) is 2 to the number of storage bits. Thus $C \cdot 2^{SPACE} \geq 2^n$, and so $SPACE \geq n - \log C = n + 0(1)$. ▨

At this stage we know that any algorithm which solves the Towers of Hanoi problem must use at least $0(2^n)$ time and n + 0(1) space. We know that the recursive algorithm uses minimal time, but we do not yet know about its space usage. If the recursive algorithm uses more than minimal space, then we are faced with several possibilities: one, that the minimal space is only a lower bound and is not attainable by any algorithm; two, that minimal time can only be achieved by an algorithm which uses more than minimal space; three, that there is some other algorithm which attains both minimal time and minimal space.

We would like to say something about the space complexity of the recursive algorithm, but the data structure used by the program is not yet specified. The towers could each be represented by an array with n locations, and each location would need at most log n bits. So an array data structure with 0(n log n) bits would suffice. Alternately, each tower could be represented by a stack. Each stack location would need log n bits, so again this is a 0(n log n) bit structure. Actually a savings would be made. Since only n disks have to be represented, the stack structure needs only n locations versus the 3n locations used by the array structure. Another possible structure is an array in which the i^{th} element holds the name of the tower on which the i^{th} disk is located. This structure uses only 0(n) bits. Yet another possiblity is to not represent the towers, but to output the moves in the form FROM __ TO __. Thus we could use no storage for the towers. But the recursive algorithm still requires space for its recursive stack. At most n stack frames will be active at any time and each frame will use a constant number of bits for the names of the towers and log n bits for the number of disks. So the recursive algorithm will use 0(n log n) bits whether or not the towers are actually represented. We summarize these considerations by the following proposition.

Proposition: The recursive algorithm HANOI correctly solves the Towers of Hanoi problem and uses $0(2^n)$ time and 0(n log n) space.

Let us next consider an iterative algorithm for the Towers of Hanoi problem.

```
PROCEDURE HANOI ITERATIVE(A,B,C,n)
   IF n mod 2 = 0 THEN MOVE[1]:= A TO B
                  ELSE MOVE[1]:= A TO C
   K:= 1
   WHILE n > 1 DO
      n:= n-1; K:= 2*K
      IF n mod 2 = 0 THEN MOVE[K]:= A TO B
                          L1:= C; L2:= A; L3:= B
                     ELSE MOVE[K]:= A TO C
                          L1:= B; L2:= C; L3:= A
      FOR I:= 1 TO K-1 DO
         CASE MOVE[I] OF
              A TO B : MOVE[K+I]:= L1 TO L2
              A TO C : MOVE[K+I]:= L1 TO L3
              B TO A : MOVE[K+I]:= L2 TO L1
              B TO C : MOVE[K+I]:= L2 TO L3
              C TO A : MOVE[K+I]:= L3 TO L1
              C TO B : MOVE[K+I]:= L3 TO L2
```

Proposition: The HANOI ITERATIVE algorithm correctly solves the Towers of Hanoi problem and uses $0(2^n)$ time and $0(2^n)$ space.

Proof: This iterative algorithm produces the same moves as the recursive algorithm. For n=1 the iterative algorithm produces the move from A to C the same as the recursive algorithm. When n=2 in the WHILE loop the iterative algorithm will have produced the same moves as HANOI(A,C,B,n-1). Next n will be set equal to 1, the ELSE part of the IF will be executed and the next move will be A to C. Then the previous moves will be recopied with A replaced by B, B replaced by C, and C replaced by A, giving the same moves as HANOI(B,A,C,n-1). So the iterative algorithm produces the same moves as the recursive algorithm.

Since the iterative algorithm takes constant time to produce each move and there are 2^n-1 moves, the algorithm takes $0(2^n)$ time. The algorithm uses $0(2^n)$ space since it stores each move. ▨

Although it is easy to show that the HANOI ITERATIVE algorithm is correct and has minimal time order, it is clear that this algorithm uses much too much space since we have seen that the recursive HANOI algorithm uses only O(n log n) space. To obtain an iterative algorithm which uses as little space as the recursive algorithm we should try to design an iterative algorithm which directly simulates the recursive algorithm. The next algorithm RECURSIVE SIM is similar to an algorithm given by Tenenbaum and Augenstein [7], but we have chosen to explicitly keep track of the stack counter because this will aid us in finding an algorithm using even less space.

```
PROCEDURE RECURSIVE SIM (A,B,C,n)
   I:= 1
   L1[1]:= A; L2[1]:= C; L3[1]:= B
   NUM[1]:= n-1 ; PAR[1]:= 1
   WHILE I ≥ 1 DO
     IF NUM[I] > 1
        THEN L1[I+1]:= L1[I]
             L2[I+1]:= L3[I]
             L3[I+1]:= L2[I]
             NUM[I+1]:= NUM[I] - 1
             PAR[I+1]:= 1
             I:= I+1
        ELSE MOVE FROM L1[I] TO L3[I]
             WHILE PAR[I] = 2 DO
                   I:= I-1
             IF I ≥ 1 THEN MOVE FROM L1[I] TO L2[I]
                           PAR[I]:= 2
                           TEMP:= L1[I]
                           L1[I]:= L3[I]
                           L3[I]:= L2[I]
                           L2[I]:= TEMP
```

The RECURSIVE SIM algorithm simulates the recursive algorithm HANOI for $n \geq 2$ disks. The names of the towers are stored in the three parameters L1, L2, L3, the number of disks in a recursive call is stored in NUM, and the value of PAR indicates whether a call is the first or second of a pair of recursive calls.

RECURSIVE SIM sets up the parameters for the call HANOI (A,C,B,n-1). When the last move for this call is made, the array will contain the parameters for calls with 1 through n-2 disks, where each of these calls will have PAR=2. The array will still contain the parameters for the (A,C,B,n-1) call with PAR=1. The inner WHILE loop will pop each of the calls with PAR=2, leaving the array counter pointing at the (A,C,B,n-1) call. Since I will be 1 at this point the IF condition is satisfied and the MOVE FROM L1[I] TO L2[I] accomplishes the MOVE FROM A TO C of the recursive algorithm HANOI. The following assignment statements set up the call (B,A,C,n-1) with PAR=2. So when the moves for this call are completed all of the calls in the array will have PAR=2, and the inner WHILE loop will pop all of these calls setting I to 0. Then the IF condition will be false, so no operations are carried out, and the outer WHILE condition will be false so the algorithm will terminate.

Proposition: The RECURSIVE SIM algorithm correctly solves the Towers of Hanoi problem, and uses $0(2^n)$ time and $0(n \log n)$ space.

Proof: Correctness follows since this algorithm simulates the recursive algorithm which we have proved correct. The major space usage is the array. Since each time I is incremented the corresponding NUM[I] is decremented and since NUM[I] never falls below 1, there are at most n-1 locations ever used in the array. The four parameters L1, L2, L3, and PAR use only a constant amount of space, but NUM must store a number as large as n-1 so it uses $0(\log n)$ bits. Thus the array uses $0(n \log n)$ bits.

We now have to argue about time usage. Most of the operations deal with constant sized operands so these operations will take constant time. The exceptional operations are incrementing, decrementing, assigning, and comparing numbers which may have $0(\log n)$ bits. A difference equation for the time is:

$$T(n) = 2T(n-1) + C \log n$$

where T(n) is the time to solve a problem with n disks and C log n is the time for manipulating the numbers with O(log n) bits. We obtain this equation because the algorithm does some manipulation of O(log n) then carries out the algorithm for n-1 disks, then after some more manipulation of O(log n), the algorithm again carries out the algorithm for n-1 disks. One can easily verify that this equation has the solution $T(n) = a2^n + C\sum_{i=0}^{n-1} 2^i \log(n-i)$. The summation in this solution can be put into the form $\sum_{j=1}^{n} 2^{n-j} \log j$ which is less than $2^n \sum_{j=1}^{\infty} (\log j)/2^j$. By the ratio test this infinite series converges. So $T(n) = a2^n + O(2^n) = O(2^n)$. ∎

This algorithm behaves as well as we can expect in time since we know that $O(2^n)$ time is required. Can any space be saved? Notice that storing the parameter NUM requires O(log n) space. Do we need to save NUM? NUM is used as a control variable so it seems necessary. But if we look at NUM[1] + 1 we get n. When NUM[I+1] is set, it is set equal to NUM[I] - 1, but then

NUM[I+1] + I + 1 = NUM[I] - 1 + I + 1

= NUM[I] + I = n.

Thus the information we need about NUM is stored in I and n. So if we replace the test on NUM[I] = 1 with a test on I = n-1, we can dispense with storing NUM and improve the space complexity from O(n log n) to O(n). This replacement does not increase the time complexity of any step in the algorithm, so the time complexity remains $O(2^n)$.

Our new procedure is

```
PROCEDURE NEW SIM (A,B,C,n)
   I: = 1
   L1[1]:= A ; L2[1]:= C ; L3[1]:= B
   PAR[1]:= 1
   WHILE I ≥ 1 DO
     IF I ≠ n-1
        THEN L1[I+1]:= L1[I]
             L2[I+1]:= L3[I]
             L3[I+1]:= L2[I]
             PAR[I+1]:= 1
             I:= I+1
        ELSE MOVE FROM L1[I] TO L3[I]
             WHILE PAR[I] = 2 DO
                  I:= I-1
             IF I ≥ 1 THEN MOVE FROM L1[I] TO L2[I]
                           PAR[I]:= 2
                           TEMP:= L1[I]
                           L1[I]:= L3[I]
                           L3[I]:= L2[I]
                           L2[I]:= TEMP
```

From the above observation we have:

Proposition: NEW SIM solves the Towers of Hanoi problem and uses $0(2^n)$ time and $0(n)$ space.

Buneman and Levy [2] give the following iterative algorithm for the Towers of Hanoi problem:

```
MOVE SMALLEST DISK ONE TOWER CLOCKWISE;
DO A DISK (OTHER THAN THE SMALLEST) CAN BE MOVED →
   MOVE THAT DISK;
   MOVE THE SMALLEST DISK ONE TOWER CLOCKWISE
OD
```

Although they can argue that this algorithm does produce the minimal sequence of moves to solve the Towers of Hanoi, their algorithm is too incomplete to calculate its time and space complexity. How does their algorithm determine if a disk can be moved? The specification of some sort of data structure is obviously necessary to complete the description of their algorithm.

If one could create a data structure so that the algorithm could determine in constant time whether a disk can be moved, locate and move that disk in constant time, and locate and move the smallest disk in constant time, then one would have an $0(2^n)$ time algorithm, since the algorithm only makes 2^n-1 moves. Such a data structure can be constructed. Each tower can be represented by a stack which contains, in order, the disks which are on that tower. No move is possible and the algorithm is finished, if the smallest disk, 1, is on some stack and the other two stacks are empty. Otherwise, there are two stacks which do not contain the smallest disk. If one of these two stacks is empty, then move the top disk from the other stack onto the empty stack. If both these stacks are nonempty then take the smaller of the disks on the top of these two stacks and place it on the other stack. If this comparison and move could be done in constant time then this algorithm would take $0(2^n)$ time. But since it takes 0(log n) bits to represent a disk, it would seem more reasonable to assume that the time for the comparison and move is proportional to the number of bits in the smaller of the two disks.

To analyze the time complexity we make use of the following fact.
Fact: In the minimal sequence of moves for the Towers of Hanoi problem for n disks the disk i is moved 2^{n-i} times.

Proof: The recursive algorithm gives the minimal moves. If n=1 the disk 1 is moved once. Following the recursive algorithm, the n^{th} disk is moved once, which is 2^{n-n}. Any other disk is moved twice the number of times it is moved in the Towers of Hanoi problem with n-1 disks. Thus the i^{th} disk is moved $2\cdot(2^{n-1-i}) = 2^{n-i}$ times. ☐

Since the i^{th} disk is moved 2^{n-i} times and this move takes time proportional to log i, the algorithm will take time proportional to $\sum_{i=1}^{n} 2^{n-i} \log i$. This sum is less than $2^n \sum_{i=1}^{\infty} \log i/2^i$. Since by the ratio test the infinite series converges, the time taken by the algorithm is $O(2^n)$.

The data structure which we are using takes O(n log n) space, since there are n disks to be represented and at least half of them must be represented using log n bits.

We summarize these observations in the following proposition.

Proposition: The Buneman and Levy Towers of Hanoi algorithm with each tower represented by a stack uses $O(2^n)$ time and O(n log n) space.

Instead of representing each tower by a stack we could consider keeping an array, indexed by the disks, which contains the name of the tower which contains the disk. More formally, let DISK be an n element array, with each element being able to contain the name of a tower. DISK[i] will contain the name of the tower which contains disk i. Initially all array elements will contain the name of the starting tower. At the end of the algorithm all array elements will contain the name of the target tower. To move disk i to tower A, use the assignment statement: DISK[i]:= A. Since there are a fixed finite number of towers each array element needs only a constant number of bits, and this data structure will use only O(n) bits.

How can we determine using this data structure if a disk can be moved? If no disks can be moved then all the disks are on the same tower, so for all i DISK[i] equals DISK[1]. Otherwise for at least one i, DISK[i] does not equal DISK[1]. The smallest such i will be a disk which is on the top of its tower and this disk can be moved to the tower which is neither DISK[1] nor DISK[i], since this tower does not contain a disk smaller than i. Thus searching for the disk to move and moving it can be accomplished by the following loop:

```
FOR i = 1 TO n DO
  IF DISK[i] ≠ DISK[1]
      THEN DISK[i]:= {A,B,C} - {DISK[1]} - {DISK[i]}
           RETURN
```

The loop terminates when it has accomplished the move.

We have to estimate how long this searching takes. Above we demonstrated the fact that disk i is moved 2^{n-i} times. So this loop should take time proportional to i, 2^{n-i} times. Thus the total time will be proportional to $\sum_{i=1}^{n} i\,2^{n-i}$ which is less than $2^n \sum_{i=1}^{\infty} i/2^i$. Since by the ratio test the infinite series converges the total time is $0(2^n)$. We summerize the results about this algorithm using this data structure in the following proposition.

Proposition: The Buneman and Levy Towers of Hanoi algorithm with the disks being represented by an array which contains the name of the tower on which the disk resides, uses $0(2^n)$ time and $0(n)$ space.

We have given several algorithms for the Towers of Hanoi problem. We have not yet achieved a minimal space algorithm. To motivate the design of our minimal space algorithm we will look at the sequence of 31 moves needed to solve the problem with 5 disks. This sequence is shown in Table 1. Several patterns emerge from this example: every odd numbered move involves only disk 1; each move $4\ell + 2$ involves only disk 2; each move 4ℓ involves the same towers as the preceding move $4\ell - 3$, with the orientation occasionally reversed.

In the following the towers will be named 1, 2, and 3. In the algorithm we need to refer to the tower we are working on, which we do by using a two bit variable T which will suffice to hold 3 distinct values. T will be incremented and decremented with the understanding that 3 + 1 is 1 and 1 - 1 is 3.

TOWER 1	TOWER 2	TOWER 3		MOVE	DISK	FROM	TO
12345	-	-	0	00000	1	1	3
2345	-	1	1	00001	2	1	2
345	2	1	2	00010	1	3	2
345	12	-	3	00011	3	1	3
45	12	3	4	00100	1	2	1
145	2	3	5	00101	2	2	3
145	-	23	6	00110	1	1	3
45	-	123	7	00111	4	1	2
5	4	123	8	01000	1	3	2
5	14	23	9	01001	2	3	1
25	14	3	10	01010	1	2	1
125	4	3	11	01011	3	3	2
125	34	-	12	01100	1	1	3
25	34	1	13	01101	2	1	2
5	234	1	14	01110	1	3	2
5	1234	-	15	01111	5	1	3
-	1234	5	16	10000	1	2	1
1	234	5	17	10001	2	2	3
1	34	25	18	10010	1	1	3
-	34	125	19	10011	3	2	1
3	4	125	20	10100	1	3	2
3	14	25	21	10101	2	3	1
23	14	5	22	10110	1	2	1
123	4	5	23	10111	4	2	3
123	-	45	24	11000	1	1	3
23	-	145	25	11001	2	1	2
3	2	145	26	11010	1	3	2
3	12	45	27	11011	3	1	3
-	12	345	28	11100	1	2	1
1	2	345	29	11101	2	2	3
1	-	2345	30	11110	1	1	3
-	-	12345	31	11111			

Table 1. Towers of Hanoi Solution for 5 disks.

In order to exploit these patterns for a minimal space solution, we find it beneficial to inspect the binary representation of the move counter. Note that the 8th and 24th moves are reverse orientation moves. This leads us to note that the reverse moves occur if the increment of the move counter causes a carry into the 2^k position for k odd, and greater than 1. Hence, we arrive at the following low space iterative algorithm for the Towers of Hanoi problem:

```
PROCEDURE TOWERS
  MOVE COUNTER := 0        (* MOVE COUNTER has n-2 bits *)
  T:= 1                    (* starting tower *)
  P:= n MOD 2              (* only the last bit of n is stored *)
  WHILE TRUE DO
     IF odd (P)
       THEN MOVE DISK 1 FROM T TO T-1
            MOVE DISK 2 FROM T TO T+1
            MOVE DISK 1 FROM T-1 TO T+1
       ELSE MOVE DISK 1 FROM T TO T+1
            MOVE DISK 2 FROM T TO T-1
            MOVE DISK 1 FROM T+1 TO T-1
     IF ALL BITS OF MOVE COUNTER = 1 THEN RETURN
     IF POSITION OF RIGHTMOST 0 BIT IN MOVE COUNTER IS ODD, > 1
       THEN IF odd (P)
               THEN MOVE TOP DISK FROM T-1 TO T
                    T:= T+1
               ELSE MOVE TOP DISK FROM T+1 TO T
                    T:= T-1
       ELSE IF odd (P)
               THEN MOVE TOP DISK FROM T TO T-1
                    T:= T+1
               ELSE MOVE TOP DISK FROM T TO T+1
                    T:= T-1
     INCREMENT MOVE COUNTER
```

Proposition: Algorithm TOWERS correctly solves the Towers of Hanoi problem for $n \geq 2$ disks, by giving the sequence of moves to move n disks from tower 1 to tower 3.

Proof: We prove this proposition by induction using the following inductive hypothesis.

> Inductive Hypothesis: If MOVE COUNTER is initialized as an array of n-2 bits, and each bit is set to 0, then the WHILE loop correctly moves $n \geq 2$ disks from tower T_o to the correct target tower, and at the conclusion of the loop T will contain the correct value. We distinguish four cases where the values of the target tower and final value of T are as follows:

	Target Tower	Final Value of T
a) P even n odd	$T_o + 1$	$T_o - 1$
b) P odd n even	$T_o + 1$	T_o
c) P even n even	$T_o - 1$	T_o
d) P odd n odd	$T_o - 1$	$T_o + 1$

For the base case, n=2, the behavior of the algorithm will depend on whether or not P is odd. If P is odd the THEN part of the first IF is executed and two disks are correctly moved from T_o to $T_o + 1$. If P is even the ELSE part is executed and two disks are correctly moved from T_o to $T_o - 1$. The condition of the next IF is true since MOVE COUNTER has no bits and hence all its bits are 1's, so the loop terminates.

For this base case we are in case (b) or (c) of the hypothesis and we have verified that the algorithm moves the required number of disks to the proper target tower. In these cases we want the final value of T to be the same as its initial value, but no instruction has changed T so it still contains its initial value.

If n > 2 then at some point in the algorithm the MOVE COUNTER will have only its high order bit equal to 0; all of its other bits will equal 1. We use the inductive hypothesis to calculate the state of the process when this condition occurs, since the loop would terminate at this point if it had been started with n-1. The various situations are displayed in the following table:

	P even n odd n-1 even	P odd n even n-1 odd	P even n even n-1 odd	P odd n odd n-1 even
n-1 RINGS HAVE BEEN MOVED TO	$T_o - 1$	$T_o - 1$	$T_o + 1$	$T_o + 1$
T contains	T_o	$T_o + 1$	$T_o - 1$	T_o
NEXT MOVE	$T \rightarrow T + 1$ $T_o \rightarrow T_o + 1$	$T - 1 \rightarrow T$ $T_o \rightarrow T_o + 1$	$T + 1 \rightarrow T$ $T_o \rightarrow T_o - 1$	$T \rightarrow T - 1$ $T_o \rightarrow T_o - 1$
NEXT VALUE OF T	$T_o - 1$	$T_o - 1$	$T_o + 1$	$T_o + 1$

Since only one bit in MOVE COUNTER is 0 and the position of this zero has the same parity as n, we can calculate the next move which we record in in the third row of the table, and we can calculate the new value of T which we record in the fourth row of the table.

Next MOVE COUNTER will be incremented so that it contains a single 1 as its leftmost bit and all of its remaining bits will be 0. Now the algorithm will behave as if the leftmost bit did not exist, that is, as if it were dealing with only n-1 disks.

So in case (a), n-1 rings will be moved from the current value of T, i.e., $T_o - 1$, to $(T_o - 1) - 1$ which is $T_o + 1$ as required by the inductive hypothesis, and the value of T will be unchanged resulting in T contining $T_o - 1$ as required.

In case (b), n-1 rings will be moved from $T_o - 1$ to $(T_o - 1) - 1$ which is $T_o + 1$ as required, and T will contain $(T_o - 1) + 1$ which is T_o as required.

In case (c), n-1 rings will be moved from $T_o + 1$ to $(T_o + 1) + 1$ which is $T_o - 1$ as required, and T will contain $(T_o + 1) - 1$ which is T_o as required.

In case (d), n-1 rings will be moved from $T_o + 1$ to $(T_o + 1) + 1$ which is $T_o - 1$ as required and T will contain $T_o + 1$ as required.

So all conditions are verified and the algorithm performs correctly. ▨

Proposition: The algorithm TOWERS uses $0(2^n)$ time and n+1 bits space.

Proof: For space, MOVE COUNTER has n-2 bits, T has 2 bits and P has 1 bit, and since these are the only variables the algorithm uses n+1 bits of storage.

For time, the algorithm goes through the WHILE loop $2^{n-2} - 1$ times. On all except the last repeat, 4 moves are made. On the last repeat, only 3 moves are made. This correctly totals to $2^n - 1$ moves. The only nonconstant time operations are those that interrogate or change the move counter. Since the move counter has n-2 bits these operations could take $0(n)$ time, giving a $0(n2^n)$ algorithm. But the operations on move counter only have to progress until the rightmost 0 is encountered. The move counter takes on each of the values 0 through $2^{n-2} - 1$ exactly once during the course of the algorithm. For half these numbers the rightmost 0 will be in the first position. For one quarter of these numbers the rightmost 0 will be in the second position. For $\frac{1}{2^i}$ of these numbers the rightmost 0 will be in position i, $i \leq n-2$. One number will contain no 0. Thus the running time of the algorithm will be $C_1 2^{n-2}(\sum_{i=1}^{n-2} i/2^i) + C_2 n$ which is less than $C_1 2^{n-2} \sum_{i=0}^{\infty} i/2^i + C_2 n$ and since the series converges this is $C_3 2^{n-2} + C_2 n$ which is $0(2^n)$ as claimed. ▨

We summarize the results of this section in the following theorem.

Theorem: Any algorithm which solves the Towers of Hanoi problem for n disks must use at least $0(2^n)$ time and $n + 0(1)$ bits of storage. The algorithm TOWERS solves the problem and simultaneously uses minimum time and minimum space.

2. Generalized Towers of Hanoi

In the generalized Towers of Hanoi problem, one is given t towers ($t \geq 3$) and n disks. Initially the n disks are stacked on tower 1, and one is to move them to tower t moving one disk at a time between any two towers in such a way that a disk is never stacked on top of a smaller disk.

Finding the minimum number of moves for n disks on t towers, M(n,t) appeared as problem 353 in The American Mathematical Monthly (1939). J. S. Frame [1] and B. M. Stewart [2] published solutions in which each derived the following formula:

$$M(n,t) = \{\sum_{n-0}^{s-1} 2^i \cdot \binom{t-3+i}{t-3}\} + 2^s \cdot \{n-\binom{t-3+s}{t-2}\}$$

where s is such that $\binom{t-3+s}{t-2} \leq n < \binom{t-2+s}{t-2}$. Table 2 enumerates some of the values for M(n,t) for small n and t.

Each disk is moved some power of 2 times - this can be deduced from the recursive nature of the problem. One interpretation of the coefficient of 2^i is that it counts the number of disks which are moved exactly 2^i times.

Considering table 2, we see that there is another interpretation for the coefficients of 2^i. Inspection of M(n,t) for fixed t (starting at the assumed boundary of M(0,t) = 0) indicates that there is $1 = \binom{t-3}{t-3}$ increment of size $1 = 2^0$ to arrive at the value of M(1,t); $t-2 = \binom{t-2}{t-3}$ increment of size $2 = 2^1$ provide the values of M(2,t), M(3,t)...M(t-1,t); and so forth. That is, there are $\binom{t-3+i}{t-3}$ increments of size 2^i in the values of M(n,t) as n steps through the $\binom{t-3+i}{t-3}$ length interval from $\binom{t-3+i}{t-2}$ to $\binom{t-2+i}{t-2}$.

We will consider moving n disks by partitioning the stack of n disks into two sub-stacks of n_1 and n_2 disks ($n_1+n_2 = n$), then move n_1 disks from tower 1 to tower 2 using all t towers, followed by moving the n_2 remaining disks from tower 1 to tower t using the t-1 towers other than tower 2, and concluding by moving the n_1 disks from tower 2 to tower t. For a correctly chosen partitioning, $M(n,t) = M(n_1,t) + M(n_2,t-1) + M(n_1,t)$.

Three Towers:

Disks	1	2	3	4	5	6	7	8	9	10
Moves	1	3	7	15	31	63	127	255	511	1023

Four Towers:

Disks	1	2	3	4	5	6	7	8	9	10
Moves	1	3	5	9	13	17	25	33	41	49
Disks	11	12	13	14	15	16	17	18	19	20
Moves	65	81	97	113	129	161	193	225	257	289

Five Towers:

Disks	1	2	3	4	5	6	7	8	9	10
Moves	1	3	5	7	11	15	19	23	27	31
Disks	11	12	13	14	15	16	17	18	19	20
Moves	39	47	55	63	71	79	87	95	103	111

Six Towers:

Disks	1	2	3	4	5	6	7	8	9	10
Moves	1	3	5	7	9	13	17	21	25	29
Disks	11	12	13	14	15	16	17	18	19	20
Moves	33	37	41	45	49	57	65	73	81	89

Table 2. Certain values of M(n,t).

We can change the partition from $n=n_1+n_2$ to $n=n_1'+n_2'$ where where $n_1' = n_1+1$ and $n_2' = n_2-1$.

$M(n,t) = 2.M(n_1,t) + M(n_2,t) = 2.M(n_1',t) + M(n_2',t)$ only if the n_1 is in the i-th increment interval for t towers and n_2 is in the (i+1)-st increment interval for t-1 towers, i.e. only if $M(n_1',t) = M(n_1,t) + 2^i$

and $M(n_2',t) = M(n_2,t) - 2^{i+1}$.

Note that $n \geq \binom{t-3+s}{t-3} = \binom{t-4+s}{t-2} + \binom{t-4+s}{t-3}$; that the increment interval for t towers between $\binom{t-4+s}{t-2}$ and $\binom{t-3+s}{t-2}$ disks has increment size 2^{s-1}; and that the increment interval for t-1 towers between $\binom{t-4+s}{t-3}$ and $\binom{t-3+s}{t-3}$ has increment size 2^s. Therefore, we must partition n such that $n_1 \geq \binom{t-4+s}{t-2}$ and $n_2 \geq \binom{t-4+s}{t-3}$. Suppose $n = \binom{t-3+s}{t-2} + i$, $(i > 0)$, then each of the partitions of n:

$$n_1 = \binom{t-4+s}{t-2} + i, \quad n_2 = \binom{t-4+s}{t-3} \ ;$$

$$n_1 = \binom{t-4+s}{t-2} + i-1, \quad n_2 = \binom{t-4+s}{t-3} + 1 \ ;$$

$$\vdots \qquad\qquad \vdots$$

$$n_1 = \binom{t-4+s}{t-2}, \quad n_2 = \binom{t-4+s}{t-3} + i \ ;$$

yields a strategy involving the minimum number of moves, M(n,t). Considering the end points of this interval of partitions, we obtain

$$M(n,t) = 2.M(\binom{t-4+s}{t-2},t) + M(n-\binom{t-4+s}{t-2},t-1) \qquad (1)$$

$$= 2.M(n-\binom{t-4+s}{t-3},t) + M(\binom{t-4+s}{t-3},t-1), \qquad (2)$$

Theorem: The strategy to accomplish a partitioning solution to the generalized Towers of Hanoi problem in M(n,t) moves is unique if and only if 1) $n = \binom{t-2+s}{t-2}$ for some $s \geq 1$ or 2) $n < t-1$.

Proof: The proof is by double induction, we induct on t, and for fixed t we induct on n.

If t=3, we are in the Towers of Hanoi problem for which it is well known that the solution is unique.

Assume that $t_0 > 3$ and that the theorem is true for $3 \leq t < t_0$.

Base Step: $n \leq t_0-1$; In this case, $M(n,t_0) = 2\,t_0-1$ and the sequence of moves is accomplished by spreading the top n-1 disks on the t_0-2 towers from 2 to t_0-1, one disk per tower, then moving disk n to tower t, concluding by collecting the top n-1 disks (in reverse order) on to tower t. This is clearly unique up to re-labelling of the towers.

Assume the theorem is true for t_0 towers with $1 \leq n < n_0$ disks.

If $\binom{t_0-3+s}{t_0-2} < n < \binom{t_0-3+s}{t_0-2}$

then equations (1) and (2) give non-unique partitions which solve the problem in an optimal number of moves, hence, $n_0 = \binom{t_0-3+s}{t_0-2}$ is a necessary condition for a unique solution.

If $n_0 = \binom{t_0-3+s}{t_0-2}$ then since $n_0 > t_0$ one must partition n_0 to achieve $M(n_0,t_0)$ moves. Furthermore any partitioning other than $n_1 = \binom{t_0-4+s}{t_0-2}$, $n_2 = \binom{t_0-4+s}{t_0-3}$ requires more than $M(n_0,t_0)$ moves. By the induction hypothesis on n_0, there is a unique strategy to accomplish $M(n_1,t_0)$ moves to place disks 1 through n_1 on tower t_2 and by the induction hypothesis on t_0 there is a unique strategy to move the n_2 disks, n_1+1 through n from tower 1 to tower t using t_0-1 towers. Hence, $n_0 = \binom{t_0-3+s}{t_0-2}$ is a sufficient condition for a unique solution in $M(n_0,t_0)$ moves. By induction, on n the theorem is true for all n on t_0 towers, and by induction on t the theorem is true for all t. Hence, the theorem is proven. ∎

In closing we would like to mention two problems which remain for the generalized Towers of Hanoi problem. First, when there is not a unique solution to the problem, one may wish to enumerate the number of inequivalent solutions. This seems to require a well-thought-out

definition of the equivalency of various move patterns. For example, if n=4 and t=4, one might ask if (1,4) (1,2) (4,2) (1,3)... is equivalent to (1,4), (1,2), (1,3), (4,2),... In either case, we have the disks distributed on the towers in the pattern 4 12 3 - when disk 4 is moved to tower 4.

The second, and more serious concern, is to prove that the partitioning solution to the problem is the minimal solution to the problem. In the Monthly, following Frame's and Stewart's solutions, the editor [1] noted that they each had assumed a lemma, to wit: If the smallest n_{t-1} disks are placed on tower t-1, the next n_{t-2} disks are placed on tower t-2, and so on until the largest disk is placed with one move on tower t; then for a suitable partition of $n = 1+n_2+n_3+\ldots+n_{t-1}$. This strategy requires as small a number of moves as any other. It is clear that this is equivalent to partitioning $n = n_1+n_2$ where $n_1=n_{t-1}$ and $n_2=1+n_2+\ldots+n_{t-2}$ and then applying the minimal solutions to the (n_1,t) case and the $(n_2, t-1)$ case appropriately. In the cases where the partitioned strategy is not unique, there also may be non-partitioned strategies which are not unique, for example, if n=7, t=5, and M(7,5)=19. Then S_1 = (1,5) (1,4) (1,3) (1,2) (3,2) (4,2) (5,2),(1,3),(1,4),(1,5)
(4,5) (3,5) (2,1) (2,3) (2,4) (2,5) (4,5) (3,5) (1,5); and

S_2 = (1,5) (1,4) (1,3) (1,2) (5,3) (4,2) (1,5) (1,4) (5,4) (1,5)
(4,1) (4,5),(1,5),(3,1),(2,4),(2,5),(3,5),(4,5) (1,5)

are 19 move solutions. Solution S_1 is a partitioning solution and has the disk distributed as

7 1234 5 6 ____

When disk 7 is moved to tower 5, while solution S_2 is not a partitioning solution in any sense, and has the disks distributed as

7 24 13 56 ____

when disk 7 is moved to tower 5.

Wood [6] cites further derivations for M(n,t) yet acknowledges that the proofs "while mathematically precise are incomplete". We would very much like to see the Lemma proven.

References:

[1] _____, Editorial Note, The American Mathematical Monthly, V48 (1941), p. 219.

[2] Peter Buneman and Leon Levy, The Towers of Hanoi Problem, Information Processing Letters, V10 (1980), pp. 243-244.

[3] J. S. Frame, Solution to Problem 3918, The American Mathematical Monthly, V48 (1941), pp. 216-217.

[4] B. M. Stewart, Problem 3918, The American Mathematical Monthly, V46 (1939), p. 363.

[5] B. M. Stewart, Solution to Problem 3918, The American Mathematical Monthly, V48 (1941) pp. 217-219.

[6] Derick Wood, The Towers of Brahma and Hanoi Revisited, Computer Science Technical Report No. 80-CS-23, McMaster University, 1980.

[7] Aaron Tenenbaum and Moshe Augenstein, Data Structures Using PASCAL, Prentice-Hall, Englewood Cliffs, NJ (1981) pp. 149-154.