The Colour Towers of Hanoi: A Generalization

M. C. Er

Department of Computing Science, The University of Wollongong, P.O. Box 1144, Wollongong, N.S.W. 2500 Australia

The colour Towers of Hanoi problem is proposed. In this variant, discs are coloured white and black. The white and black discs are required to move in the clockwise and the counterclockwise directions, respectively, subject to the usual constraints of the standard problem. Initially, the discs are stacked on the pegs randomly without violating the constraints. The objective is to move them to a specified peg in increasing order with the largest disc at the bottom. A recursive solution to the problem and the unerlying strategies are presented.

INTRODUCTION

The Towers of Hanoi problem and its variants have attracted a good deal of attention in the recent literature. An earlier version of the recursive solution to the Towers of Hanoi problem is discussed by Dijkstra;[1] and a slightly different form is described by Hayes.[2] Various iterative solutions to this problem have been discovered by Buneman and Levy,[3] Dijkstra,[1] Er,[4] Hayes[2] and Walsh.[5] A variant, known as the cyclic Towers of Hanoi problem, is proposed by Atkinson[6] who also presents a recursive solution to it. A generalization to the Towers of Hanoi problem has recently been proposed by Er;[7] and a recursive solution to this generalized problem has also been discovered.

In this paper yet another variant of the Towers of Hanoi problem is proposed. In this variant, every disc is painted either white or black; and it is required that all white discs are moved in one direction and all black discs are moved in the opposite direction so that they end up on a specified peg subject to the usual constraints of the standard problem. This colour Towers of Hanoi problem seems to be extremely complex; however we show that a surprisingly simple recursive solution is indeed possible.

THE PROBLEM

The colour Towers of Hanoi problem may be described simply as follows. Three pegs are arranged in a circle; and thus the clockwise and the counterclockwise directions may be defined in the usual sense when viewed from the top. There are n discs of different sizes, each of which is coloured white or black, initially stacked on these three pegs with the larger discs below the smaller. The objective is to move these discs around until they are stacked on a specified peg subject to the following constraints.

(i) Only one of the top discs may be moved at a time.
(ii) No disc may ever stack on a disc smaller than itself.
(iii) White and black discs may be moved to their neighbouring pegs only in the clockwise and the counterclockwise directions, respectively.

STRATEGY

In problem solving, the most difficult and essential task is to discover a strategy or a set of strategies which will win under all circumstances. We therefore first of all describe a set of strategies which will lead to a feasible solution before spelling out the details of an algorithm.

One important observation is that a disc dominates all smaller discs, owing to the constraints on disc moves. In other words, when a disc is moved from a source peg to a target peg, all smaller discs have to be stacked on a spare peg before the move can take place without violating any of the rules. This important observation immediately leads us to a recursive solution. At any moment, the largest disc among all subtowers is located and its immediate destination is computed. Then all smaller discs must be cleared to a dictated spare peg. This subproblem obviously could be solved recursively. And this strategy forms a basic framework for developing a simple recursive solution to the colour problem.

Of course, if all discs are already stacked on a specified peg, we have trivially done. However, in a general case, it is more likely that all discs are spread on all pegs. So, at each decision point, we select the largest disc among the given towers and determine its movement. If the largest disc is already on the target peg it should move to, we recursively solve the subproblem by excluding the largest disc from consideration, i.e. all smaller discs are to be moved to the same target peg. If, however, the source peg occupied by the largest disc and the target peg it should move to are different, then we have four cases to be considered.

Case (i): The largest disc is white and the target peg is clockwise from the source peg. In this case, all smaller discs could be cleared to the spare peg in the counterclockwise direction from the source peg; and the largest disc is then moved to the target peg; and finally the smaller discs are moved to the same target peg.

Case (ii): The largest disc is black and the target peg is counterclockwise from the source peg. The case is similar to Case (i), but the black largest disc should be moved counterclockwise to the target peg instead.

Case (iii): The largest disc is white and the target peg is counterclockwise from the source peg. In this situation, the largest white disc cannot be moved to the target peg in one step. However, the situation can be converted to an instance of Case (i) by moving all smaller discs to the target peg and then moving the largest white disc to the spare peg. After such moves, the

CCC-0010-4620/84/0027-0080$01.50

largest white disc is on a peg such that the target peg is clockwise from it.

Case (iv): The largest disc is black and the target peg is clockwise from the source peg. This situation is similar to Case (iii) as the largest black disc requires more than one step to move to the target peg. We thus apply the same strategy to convert the situation to an instance of Case (ii).

The subproblem of moving all smaller discs to the target peg is similar to the original problem and thus can be solved recursively by treating those smaller discs as subtowers to be moved.

The outline is made precise in the form of an algorithm described in the next section.

RECURSIVE ALGORITHM

Here, we adopt a convention that white discs are represented by positive integers and black discs are represented by negative integers. Further, we assume that the pegs are numbered 1, 2 and 3 so that the clockwise and the counterclockwise directions are defined by $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$ and $1 \rightarrow 3 \rightarrow 2 \rightarrow 1$, respectively.

Assuming the following type definitions:

```
peg       = 1 . . 3;
colourdisc = −maxdisc . . maxdisc;
tomove    = 0 . . maxdisc;
subtowers = array [peg] of tomove;
discstack = array [1 . . maxdisc] of colourdisc;
```

we may declare the following global variables:

```
towers       : array [peg] of discstack;
stackpointers : array [peg] of tomove;
```

The recursive solution may be described by the following Pascal program.

```
procedure ColourTowers (var n: subtowers; target: peg);
  { When called with the number of discs in each
    subtower, indicated by the variable parameter n, this
    procedure moves them to the target peg. The total
    number of discs in all subtowers is invariant before
    and after an activation of this procedure, but not so
    during its course of execution. }
var source, spare: peg;
    bigdisc      : colourdisc;
begin
  if n[1] + n[2] + n[3] < > n[target] then
    begin
    source := ChooseMaxPeg(n);
    bigdisc := LargestDisc(source, n[source]);
    n[source] := n[source] − 1;
    if target = source then
      begin
      ColourTowers (n, target);
      n[source] := n[source] + 1
      end
    else begin
      spare := 6 − target − source;
      if IsWhite (bigdisc) = IsClockwise (source, target)
                                                  then
        begin
        ColourTowers (n, spare);
        MoveDisc (source, target);
        ColourTowers (n, target);
        n[target] := n[target] + 1
        end
      else begin
        ColourTowers (n, target);
        MoveDisc (source, spare);
        n[spare] := n[spare] + 1;
        ColourTowers (n, target)
        end
      end
    end
end { ColourTowers };
```

The function *ChooseMaxPeg* locates a peg where the largest disc among the given subtowers resides. Its details may be described as follows.

```
function ChooseMaxPeg (n: subtowers): peg;
  { Given the number of discs in each subtower, n, this
    function returns the peg where the largest disc among
    them resides as a result. }
var maxpeg, p: peg;
begin
  maxpeg := 1;
  for p := 2 to 3 do
    if abs(LargestDisc(p, n[p]))
        > abs(LargestDisc(maxpeg, n[maxpeg]))
      then maxpeg := p;
  ChooseMaxPeg := maxpeg
end { ChooseMaxPeg };
```

Note that the function *abs* returns the absolute value of its argument.

The function *LargestDisc* locates the largest disc of a given subtower on a specified peg, and may be described as follows.

```
function LargestDisc(p: peg; height: tomove): colourdisc;
  { When called with the height of a subtower on a peg p,
    this function computes the largest disc in the sub-
    tower by a simple offset (height) from its top disc. }
begin
  if height = 0 then LargestDisc := 0
    else LargestDisc := towers[p][stackpointers[p] −
                                              height + 1]
end { LargestDisc };
```

Here, we assume that *stackpointers*[*p*] always points to the top disc of the tower on peg *p*. When the tower is empty, *LargestDisc* returns the 'null disc' of value zero as a result.

The function *IsWhite* determines whether or not a given disc is white as follows:

```
function IsWhite (d: colourdisc): boolean;
begin IsWhite := d > 0
end { IsWhite };
```

The function *IsClockwise* tests whether a given target peg is clockwise from a given source peg by using a simple arithmetic calculation as follows.

```
function IsClockwise (source, target: peg): boolean;
begin
  IsClockwise := (target − source = 1) or (target − source
                                                  = −2)
end { IsClockwise };
```

Finally, we note that the procedure *MoveDisc* calls the standard stack manipulation routines, *Pop* and *Push* in turn to update the global array towers.

```
procedure MoveDisc (source, target: peg);
  { This procedure calls the routine Pop to pop a disc
    from the source peg, and then calls the routine Push
    to push the disc onto the target peg. }
begin
  Push (target, Pop(source))
end { MoveDisc };
```

CORRECTNESS

To prove that the procedure *ColourTowers* is correct, we need to verify all possible cases.

First of all, we observe that *ColourTowers* is trivially correct if all towers are empty. Now, we may assume that *ColourTowers* is correct for m discs randomly distributed on the pegs but satisfying the constraints. Then we prove by induction that the procedure is also correct for $(m + 1)$ discs randomly distributed on the pegs. Also, we must show that the total number of discs registered in the variable parameter n is invariant before and after an activation of *ColourTowers* and that n is updated correctly.

When *ColourTowers* is called with $(m + 1)$ discs and a target peg they should move to, it first of all computes the largest disc among the three towers and the source peg on which this disc resides. There are three cases to be considered given the source peg of the largest disc and the target peg it should move to.

Case A: The target peg is the source peg. In other words, the largest disc is already on the target peg, and thus need not be moved. *ColourTowers* is then called recursively to move the m smaller discs to the target peg, that, by assumption, is correct. The largest disc is subtracted from and added back to the source peg before and after such a recursive call respectively. Hence n is updated correctly, and the total number of discs is invariant before and after the activation of *ColourTowers*.

Case B: The largest disc could be moved from the source peg to the target peg in one step. Cases (i) and (ii) discussed in Section 3 satisfy this condition; namely, the largest disc is white and is to move to the target peg clockwise, and the largest disc is black and is to move to the target peg counterclockwise. In either case, *ColourTowers* is called recursively to move m smaller discs to a spare peg; then the largest disc is moved to the target peg in one step; and finally *ColourTowers* is called recursively again to move m smaller discs to the target peg. By assumption, the two recursive calls are correct; thus the above sequence of disc moves will move $(m + 1)$ discs to the target peg. Further, we see that the largest disc is subtracted from the source peg before the first recursive call, and is added to the target peg after the second recursive call. Since the two recursive calls maintain m as an invariant, so the total number of discs is invariant. Also by assumption, the two recursive calls update n correctly, thus *ColourTowers* will update n correctly in this case.

Case C: The largest disc can only be moved from the source peg to the target peg in two steps. This condition covers Cases (iii) and (iv) discussed in Section 3. Namely, the largest white disc is to move to the target peg counterclockwise; and the largest black disc is to move to the target peg clockwise. *ColourTowers* converts this case to an instance of Case B by calling itself recursively to move the m smaller discs to the target peg, and then moving the largest disc to the spare peg from where the target peg could be reached in one step. Since, by assumption, the recursive call maintains m as an invariant and updates n correctly, and that the largest disc is subtracted from the source peg and added to the spare peg, *ColourTowers* thus updates n with $(m + 1)$ discs correctly and maintains the total number of discs as an invariant up to this point. Finally, a recursive call to *ColourTowers* is carried out to move $(m + 1)$ discs to the target peg, and such a call must fall into Case B. By the arguments of Case B, we thus prove that Case C is also correct.

As only one of these cases must be true during each activation of *ColourTowers*, by induction, we thus prove that *ColourTowers* is correct.

REMARKS

The colour Towers of Hanoi problem at first sight seems rather complex. However, by discovering a crucial strategy that larger discs dominate smaller discs and analysing all possible cases carefully, we have shown that a simple recursive solution to the problem is feasible.

It is also interesting to discover an iterative solution to the colour problem and to analyse the average-case performance. It seems that the colour Towers of Hanoi remains a challenging problem to human players who have no pencil and paper to work with.

Acknowledgement

This research was supported by RGC under grant 05-143-105.

REFERENCES

1. E. W. Dijkstra, *A Short Introduction to the Art of Programming*, EWD316 (1971).
2. P. J. Hayes, A note on the Towers of Hanoi problem. *The Computer Journal* **20**, 282–285 (1977).
3. P. Buneman and L. Levy, The Towers of Hanoi problem. *Information Processing Letters* **10**, 243–244 (1980).
4. M. C. Er, A representation approach to the Tower of Hanoi problem. *The Computer Journal* **25**, 442–447 (1982).
5. T. R. Walsh, The Towers of Hanoi revisited: moving the rings by counting the moves. *Information Processing Letters* **15**, 64–67 (1982).
6. M. D. Atkinson, The cyclic Towers of Hanoi. *Information Processing Letters* **13**, 118–119 (1981).
7. M. C. Er, The generalised Towers of Hanoi problem. *Information Processing Letters*, in press.

Received January 1983