A NOTE ON ITERATIVE HANOI

Bertrand Meyer

Department of Computer Science
University of California
Santa Barbara, California 93106 (USA)

A recent note in *SIGPLAN Notices* [1] presents an iterative solution to the well-known Tower of Hanoi problem. I would like to mention that an essentially equivalent solution was published several years ago in a book co-authored by Claude R. Baudoin and myself [2, pages 376-378]. If I recall correctly, the idea for this solution came during a late night discussion with Patrick Greussay; I have to admit, however, that although all the persons concerned were younger by seven or eight years than they are now, none of us was still in fourth grade, as reported for the discoverer of the solution published in [1].

The way we explained our solution in our book is, I think, a little easier to understand than [1]. We assume that the reader is familiar with the recursive version of the algorithm (if not, learn French and buy [2], it's a bargain). Let the call

Hanoi (n, x, y, z)

mean "transfer n disks from peg x to peg y, using peg z as intermediate storage". The body of the procedure is

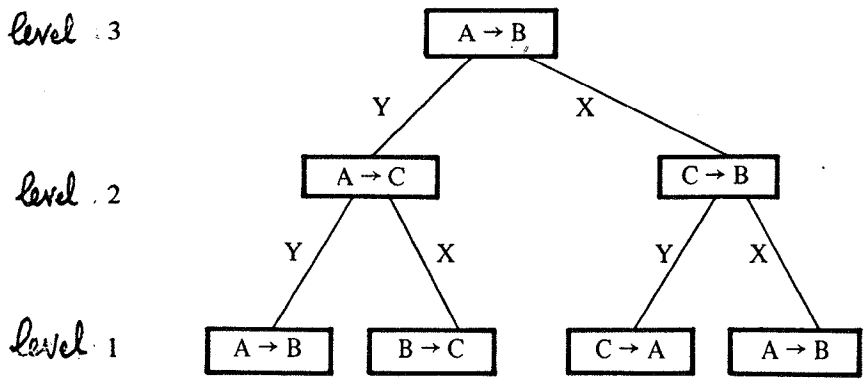
if $n > 0$ **then**

Hanoi (n-1, x, z, y) ; move (x, y) ; Hanoi (n-1, z, y, x)

end if

As with any recursive procedure, a tree can be associated with an execution of the procedure; here it is a binary tree. Execution of the procedure can be viewed as an inorder traversal of this binary tree, where the action to be performed when visiting a node labeled $x \rightarrow y$ is *move (x, y)*.

The first recursive call swaps arguments y and z; the second one swaps x and z. We label each edge of the tree "Y" or "X" depending on which of these swaps is to be executed along the edge (see figure).



We define the **level** of a node of this binary tree to be 1 if the node is a leaf, 2 if it is the parent of a leaf, etc. Let S_n be the sequence of the levels of the nodes encountered during the traversal. It follows directly from the recursive structure of the algorithm that S_n has $2^{n-1}-1$ elements and satisfies:

S_0 = empty ;

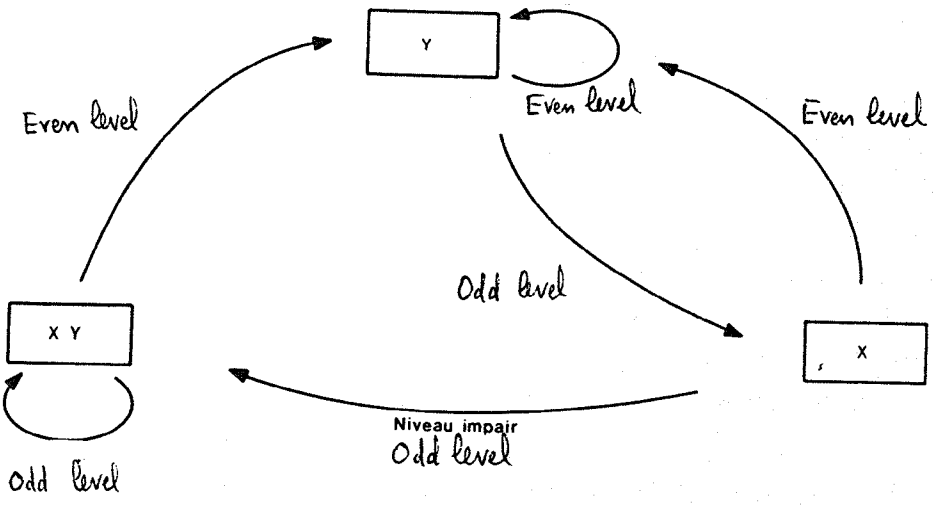
$S_i = S_{i-1}\ i\ S_{i-1}$ $\qquad (i > 0)$

Thus S_1 = <1>, S_2 = <1 2 1>, S_3 = <1 2 1 3 1 2 1>, etc.

An interesting remark about these recurrence equations is that they also apply to the sequence C whose n-th element C_n is the sequence of indexes (counted from the rightmost position) of the rightmost ones in the binary representations of the numbers 1, 2, 3, ..., 2^n-1. This is readily seen from the fact that the above equations directly correspond to the way the sequence of binary representations of the numbers 1, 2, 3, ..., 2^n-1 may be constructed: first 1, 2, ..., $2^{n-1}-1$, then 2^{n-1} (i.e. binary 10...0, with $n-1$ zeros), then the initial ones again, each with a 1 added at the n-th position from the right in its binary representation. Thus the sequences S_n and C_n are identical.

Coming back to the binary tree representing the execution of the procedure, it follows from the definition of the procedure that, during the traversal, leaves and interior nodes are alternatively visited. Let us consider what happens when the algorithm visits a node E which is at an even-numbered level and try to understand how it got there. E may only be an interior node; thus the preceding one was a leaf, say L. If L is a left child, then E is its parent and the swap to perform is a "Y"; otherwise, to go from L to E, the algorithm goes up the tree x times, performing x swaps of the "X" type, then up one "Y". But E is at an even level, L is at an odd level, so the difference between their levels in the tree, which is $x+1$, must be odd; thus x is even, which means that the "X" swaps cancel out and all that has to be done is one "Y" swap.

For interior nodes on odd-numbered levels, the same line of reasoning shows the swap to be "X" if the previous node was on an even level, and "XY" ("X" followed by "Y") otherwise. We thus obtain the following transition diagram:



The combination of the previous ideas leads to the following program, adapted from [2] (readers not familiar with Dijkstra's guarded commands may mentally replace the symbols → and □ by "**then**" and "**else if**", respectively).

```
-- move n disks from A to B, using C as intermediate storage
var x, y, z: PEG;
    current_even, previous_even: boolean;
x := C ; y := B ; z := A ;
previous_even := "n is even" ;
for i := 1 to 2^n-1 do
    current_even := "the rightmost 1 in the binary representation of i
                     has an even index" ;
    if
        current_even → swap (y, z) □
        not current_even and previous_even → swap (x, z) □
        not current_even and not previous_even →
            swap (x, z) ; swap (y, z)
    end if ;
    move (x, y) ; previous_even := current_even
end do
```

References

[1] Herbert Mayer and Don Perkins: Towers Of Hanoi Revisited, A Nonrecursive Surprise; *SIGPLAN Notices*, 19, 2, February 1984, pages 80-84.

[2] Bertrand Meyer and Claude Baudoin: *Méthodes de Programmation* (Programming Methods), Eyrolles, Paris, 1978 (new printing, 1984), 661 pages.