Tower of Hanoi

The Tower of Hanoi puzzle was invented by the French mathematician Edouard Lucas in 1883. We are given a tower of eight disks (initially four in the applet below), initially stacked in increasing size on one of three pegs. The objective is to transfer the entire tower to one of the other pegs (the rightmost one in the applet below), moving only one disk at a time and never a larger one onto a smaller.

The puzzle is well known to students of Computer Science since it appears in virtually any introductory text on data structures or algorithms. Its solution touches on two important topics discussed later on:

- recursive functions and stacks
- recurrence relations

The applet has several controls that allow one to select the number of disks and observe the solution in a Fast or Slow manner. To solve the puzzle drag disks from one peg to another following the rules. You can drop a disk on to a peg when its center is sufficiently close to the center of the peg. The applet expects you to move disks from the leftmost peg to the rightmost peg.

Recursive solution

Let call the three pegs Src (Source), Aux (Auxiliary) and Dst (Destination). To better understand and appreciate the following solution you should try solving the puzzle for small number of disks, say, 2,3, and, perhaps, 4. However one solves the problem, sooner or later the bottom disks will have to be moved from Src to Dst. At this point in time all the remaining disks will have to be stacked in decreasing size order on Aux. After moving the bottom disk from Src to Dst these disks will have to be moved from Aux to Dst. Therefore, for a given number N of disks, the problem appears to be solved if we know how to accomplish the following tasks:

1. Move the top N-1 disks from Src to Aux (using Dst as an intermediary peg)
2. Move the bottom disks from Src to Dst
3. Move N-1 disks from Aux to Dst (using Src as an intermediary peg)

Assume there is a function Solve with four arguments - number of disks and three pegs (source, intermediary and destination - in this order). Then the body of the function might look like

```
Solve(N, Src, Aux, Dst)
    if N is 0 exit
    Solve(N-1, Src, Dst, Aux)
```

```
Move from Src to Dst
Solve(N-1, Aux, Src, Dst)
```

This actually serves as the definition of the function Solve. The function is recursive in that it calls itself repeatedly with decreasing values of N until a terminating condition (in our case N=0) has been met. To me the sheer simplicity of the solution is breathtaking. For N=3 it translates into

1. Move from Src to Dst
2. Move from Src to Aux
3. Move from Dst to Aux
4. Move from Src to Dst
5. Move from Aux to Src
6. Move from Aux to Dst
7. Move from Src to Dst

Of course "Move" means moving the topmost disk. For N=4 we get the following sequence

1. Move from Src to Aux
2. Move from Src to Dst
3. Move from Aux to Dst
4. Move from Src to Aux
5. Move from Dst to Src
6. Move from Dst to Aux
7. Move from Src to Aux
8. Move from Src to Dst
9. Move from Aux to Dst
10. Move from Aux to Src
11. Move from Dst to Src
12. Move from Aux to Dst
13. Move from Src to Aux
14. Move from Src to Dst
15. Move from Aux to Dst

Recurrence relations

Let T_N be the minimum number of moves needed to solve the puzzle with N disks. From the previous section $T_3=7$ and $T_4=15$. One can easily convince oneself that $T_2=3$ and $T_1=1$. A trained mathematician would also note that $T_0=0$. Now let us try to derive a general formula.

The recursive solution above involves moving twice (N-1) disks from one peg to another and making one additional move in between. It then follows that

$$T_N \leq T_{N-1}+1+T_{N-1} = 2T_{N-1}+1$$

The inequality suggests that it might be possible to move N disks with fewer than $2T_{N-1}+1$ moves. Which is actually not the case. Indeed, when the time comes to move the bottom disk (N-1) disks will have been moved from Src to Aux in at least T_{N-1} moves. Since we are trying to use as few steps as possible, we may assume that that portion of the task took exactly T_{N-1} moves. It takes just one move to move the biggest disk from Src to Dst. One then needs exactly T_{N-1} more steps to finish the task. Therefore the minimum number of moves needed to solve the puzzle with N disks equals $T_{N-1} + 1 + T_{N-1} = 2T_{N-1} + 1$ moves.

In other words,

$$T_N = 2T_{N-1} + 1$$

Thus we can define the quantity T_N as

$$T_0 = 0$$
$$T_N = 2T_{N-1} + 1 \text{ for } N>0$$

Thus we may compute $T_1 = 2T_0 + 1 = 1$, $T_2 = 2T_1 + 1= 3$, $T_3 = 2T_2 + 1 = 7$ and so on sequentially.

The above expression is known as a recurrence relation which, as you might have noticed, is but a recursive function. T_N is defined in terms of only one of its preceding values. Other recurrence relations may be more complicated, for example, f(N) = 2f(N-1) + 3f(N-2). Recurrence relations appear under various guises in numerous branches of Mathematics and applications.

Returning to the definition of T_N, define $S_N = T_N + 1$. Then $S_0 = 1$ and $S_N = T_N + 1 = (2T_{N-1} + 1) + 1 = 2T_{N-1} + 2 = 2(T_{N-1} + 1) = 2S_{N-1}$. Which is to say that S_N could be defined as

$$S_0 = 1$$
$$S_N = 2S_{N-1} \text{ for } N>0$$

The latter is solved easily in the closed (non-recurrent) form $S_N=2^N$. Wherefrom

$$T_N = 2^N-1 \text{ for } N \geq 0.$$