18.8 Towers of Hanoi

Earlier in this chapter we studied methods that can be easily implemented both recursively and iteratively. Now, we present a problem whose recursive solution demonstrates the elegance of recursion, and whose iterative solution may not be as apparent.

The **Towers of Hanoi** is one of the classic problems every budding computer scientist must grapple with. Legend has it that in a temple in the Far East, priests are attempting to move a stack of golden disks from one diamond peg to another (Fig. 18.10). The initial stack has 64 disks threaded onto one peg and arranged from bottom to top by decreasing size. The priests are attempting to move the stack from one peg to another under the constraints that exactly one disk is moved at a time and at no time may a larger disk be placed above a smaller disk. Three pegs are provided, one being used for temporarily holding disks. Supposedly, the world will end when the priests complete their task, so there's little incentive for us to facilitate their efforts.

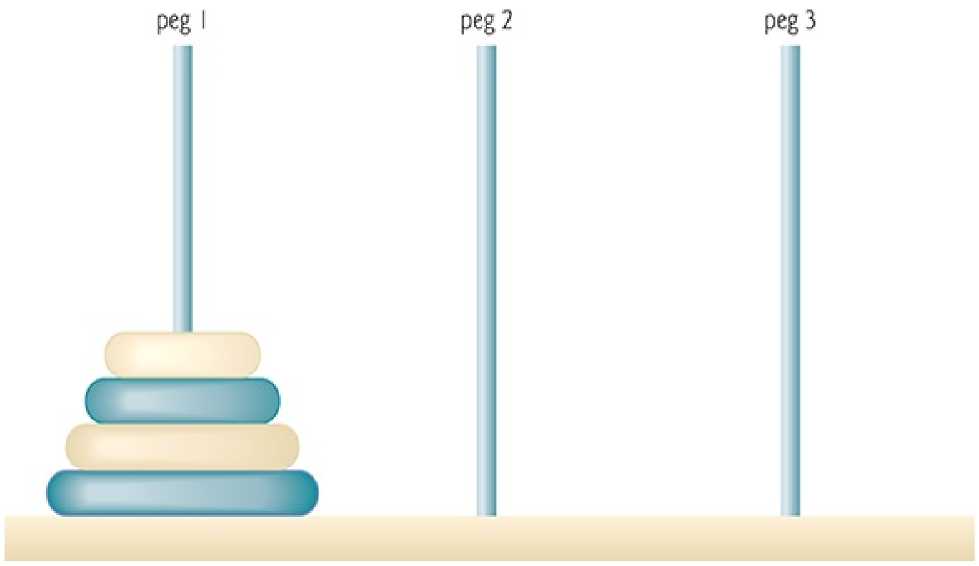

Fig. 18.10

Towers of Hanoi for the case with four disks.

Let’s assume that the priests are attempting to move the disks from peg 1 to peg 3. We wish to develop an algorithm that prints the precise sequence of peg-to-peg disk transfers.

If we try to find an iterative solution, we’ll likely find ourselves hopelessly “knotted up” in managing the disks. Instead, attacking this problem recursively quickly yields a solution. Moving *n* disks can be viewed in terms of moving only $n - 1$ disks (hence the recursion) as follows:

1. Move $n - 1$ disks from peg 1 to peg 2, using peg 3 as a temporary holding area.
2. Move the last disk (the largest) from peg 1 to peg 3.
3. Move $n - 1$ disks from peg 2 to peg 3, using peg 1 as a temporary holding area.

The process ends when the last task involves moving $n = 1$ disk (i.e., the base case). This task is accomplished by moving the disk, without using a temporary holding area.

In Fig. 18.11, method `solveTowers` (lines 5–22) solves the Towers of Hanoi, given the total number of disks (in this case `3`), the starting peg, the ending peg, and the temporary holding peg as parameters.

```
 1 // Fig. 18.11: TowersOfHanoi.java
 2 // Towers of Hanoi solution with a recursive met
 3 public class TowersOfHanoi {
 4    // recursively move disks between towers
 5    public static void solveTowers(int disks, int
 6       int destinationPeg, int tempPeg) {
 7       // base case -- only one disk to move
 8       if (disks == 1) {
 9          System.out.printf("%n%d --> %d", source
10          return;
11       }
12
13       // recursion step -- move (disk - 1) disks
14       // to tempPeg using destinationPeg
15       solveTowers(disks - 1, sourcePeg, tempPeg,
16
17       // move last disk from sourcePeg to destin
18       System.out.printf("%n%d --> %d", sourcePeg
19
20       // move (disks - 1) disks from tempPeg to
21       solveTowers(disks - 1, tempPeg, destinatio
22    }
23
24    public static void main(String[] args) {
25       int startPeg = 1; // value 1 used to indic
26       int endPeg = 3; // value 3 used to indicat
27       int tempPeg = 2; // value 2 used to indica
28       int totalDisks = 3; // number of disks
29
30       // initial nonrecursive call: move all dis
31       solveTowers(totalDisks, startPeg, endPeg,
32    }
33 }
```

```
1 --> 3
1 --> 2
3 --> 2
1 --> 3
2 --> 1
2 --> 3
1 --> 3
```

Fig. 18.11

Towers of Hanoi solution with a recursive method.

The base case (lines 8–11) occurs when only one disk needs to be moved from the starting peg to the ending peg. The recursion step (lines 15–21) moves `disks - 1` disks (line 15) from the first peg (`sourcePeg`) to the temporary holding peg (`tempPeg`). When all but one of the disks have been moved to the temporary peg, line 18 moves the largest disk from the start peg to the destination peg. Line 21 finishes the rest of the moves by calling the method `solveTowers` to recursively move `disks - 1` disks from the temporary peg (`tempPeg`) to the destination peg (`destinationPeg`), this time using the first peg (`sourcePeg`) as the temporary peg.

Line 31 in `main` calls the recursive `solveTowers` method, which outputs the steps to the command prompt.