Towers of Hanoi

Programming II - Elixir Version

Spring Term 2023

Introduction

In this exercise you will see how recursive thinking can help when solving a problem. A small puzzle that you might have seen is quite tricky to solve using brute force. Using a recursive strategy however, the puzzle is surprisingly simple.

The towers

The classical puzzle that we will use is *the Towers of Hanoi*. It consists of three pegs with a set of discs piled up on one. The puzzle is to move the discs, one by one, from their initial position to the last peg. The problem is that you're only allowed to place a smaller disc on a larger (or as the first disc on the peg). Figure 1 show a puzzle with four discs in their original position.

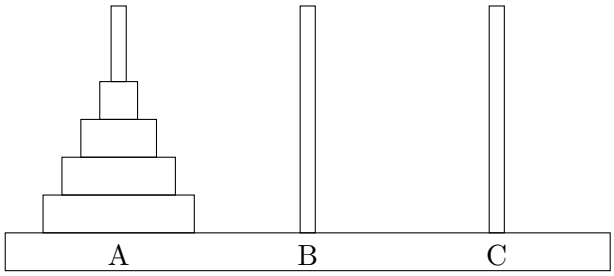

Figure 1: The Towers of Hanoi

If the puzzle only has four discs it's rather simple. One solution would be to move the smallest disc to peg B, then ... hmm. I'm sure there is a simple solution but is there a strategy that work for towers of five, six or seven discs?

Recursive strategy

The strategy that we will use is surprisingly simple. If we have a tower of size four, we first move the upper three discs to peg A, then move the last disc to peg C and then move the three discs from peg B to C.

This is cheating you say, we should move the discs one by one and can not take three discs in one go. You're of course right so let's modify the solution.

Moving the largest disc was not a problem, the violation was how we moved a tower of three discs from peg A to B and then over to C. What if we instead of moving all in one go , we did like this: move the upper two to C, the third one to B and then the upper two from C to B.

Not allowed you say, we're moving two discs in one go.

Ok, moving the larger disc was not a problem, the violations was how we moved a tower of two discs from ....Hmm, move the smallest to B, then the second to C and the smallest to B. That will solve the first problem. After having moved the third disc to C we can use peg A as the extra peg and move the smallest to A, the second one to C and then the smallest to C.

Problem solved.... or almost. We have now moved a tower of three from A to C and should when the fourth disc is placed on peg B move the tower on top of it. We know how to move a tower of three if we have one extra peg to use and since the disc that is left on peg A is larger than any disc in our small tower we can use peg A as this extra peg.

The revolutionary insight is that if we can move a tower of four disc from one peg to another, using a third one as a temporary peg. Then we can solve the towers of Hanoi with a tower of five discs. In general it means that a solution for a tower of n discs in the ticket to solving a tower of $n+1$ discs. Since it is trivial to solve the problem for a tower of 0 discs we thus have a solution for a tower of 1 disc and this is the ticket to solve 2 discs etc.

The sequence

Let's implement a function that returns the sequence of moves necessary to solve towers of Hanoi of a given size n. The pegs are called `:a`, `:b` and `:c` and a move is represented by a tuple {`:move, from, to`} meaning that the uppermost disc on the *from peg* is moved to the *to peg*.

The function takes four arguments: the size of the tower, the *from peg*, an *auxiliary peg* and the *to peg* where the tower should be placed. We could for example call the function like this:

```
hanoi(3, :a, :b, :c)
```

and expect to have the result:

```
[
  {:move, :a, :c},
  {:move, :a, :b},
  {:move, :c, :b},
  {:move, :a, :c},
  {:move, :b, :a},
  {:move, :b, :c},
  {:move, :a, :c}
  ]
```

Check that the sequence of moves actually solves the tower of Hanoi with a tower of size 3. Also identify the three different sections; the first three moves will move a tower of 2 to peg *:b*, the fourth the last disc to peg *:c* and the last three will take the tower of 2 and place it on top of it.

When we implement a recursive function we first need to get the base case right. In this puzzle the base case is solving the towers of Hanoi for the size of 0. The number of moves we should then do is och course zero and is independent of the pegs so the base case should look like this:

```
def hanoi(0, _, _, _) do ... end
```

No what about the recursive case? It is made up from three sequences: moving a smaller tower from *from* to an *auxiliary* peg (using the third peg as the auxiliary), moving the last disc from *from* to *to* and moving the smaller tower from the *auxiliary* peg to *to* (using the em from peg as the auxiliary). Hmm, something like this:

```
def hanoi(n, from, aux, to) do
  move tower of size  n-1 ....  ++
  [ move one disc ... ] ++
  move tower of size n-1 ....
end
```

Generate a sequence for size: 1, 2 and 3 to see that it works. What does the sequence look like for size 4? How many moves do you need to solve towers of Hanoi for size 10?

Write a small report where you explain the problem and how a recursive solution works. Include the answers to the questions above.