Task 4 Tower of Hanoi

Adrian Jonsson Sjödin

Spring Term 2023

Introduction

This weeks task was to implement a program able to solve the popular Tower of Hanoi puzzle. To do this we once again use recursion to break down the problem into subparts.

Method

The strategy to use is described in the task description but I will go through it here again while explaining the code seen in code overview 1.

We start with the base case of when we the number of disks we need to move are zero. For that case we simply return an empty list. After this we have the recursive steps which basically works as a recipe and tells how to solve solve the problem.

We start with asking how many disks do you have (`numb_of_disks`), which peg are the rings on now (`a`), which peg will you use as your temporary peg (`b`), and which peg do you want to move them to `c`? If there are one or more rings the program will use the recipe to solve the problem step by step.

1. Move all the disks except the last one to the temporary peg (`hanoi(numb_of_disks, a, c, b)`) using the same recipe.
2. Move the last ring from the original peg to the target peg (`[{:move, a, c}]`)
3. Move the other rings from the temporary peg to the target peg using the same recipe (`hanoi(numb_of_disks, b, a, c)`)

Each time the recipe, it is a new and smaller version of the problem since one disk has already been moved. Eventually the program will run out of disks to move which means the problem is solved.

We get the instruction of how to move our disks by concatenating together the lists returned by each step of the recipe.

Worth mentioning is that the tuple {`:move, a, c`} means {`:move, source_peg, target_peg`}, they are parameters and not fixed values "a" and "c". I realize that the names can be confusing but what is important to remember is that the first argument for `hanoi/4` is the number of disks we want to move, the second is the source peg, the third is the temporary peg, and the last is the target peg.

Code Overview 1: `hanoi/4`

```
defmodule Hanoi do
    def hanoi(0, _, _, _) do [] end
    def hanoi(numb_of_disks, a, b, c) do
        hanoi(numb_of_disks-1, a, c, b) ++
        [{:move, a, c}] ++
        hanoi(numb_of_disks-1, b, a, c)
    end
end
```

Result

Running the command `Hanoi.hanoi(3, :a, :b, :c)` we get the output seen in code overview 2, which matches with the output given in the task instructions.

Code Overview 2: `hanoi(3, :a, :b, :c)`

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

In figure 1-3 we see that the sequence of moves given in code overview 2 does solve the puzzle and also the three different sections. Moving all disks except the last one to the temporary peg, then moving the last one to the target peg, and lastly moving the other rings from the temporary peg to the target peg. This is exactly like we described how the recipe would work in the Method section above.

Lastly to calculate how many moves it takes to solve a tower of size n once can use the formula $2^n - 1$. Using this formula we get that it would require 1023 moves to move a tower ten disks high from peg A to peg C.

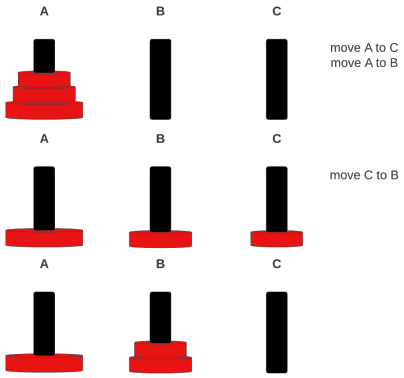

Figure 1: First section. Moving a tower of two to peg B.

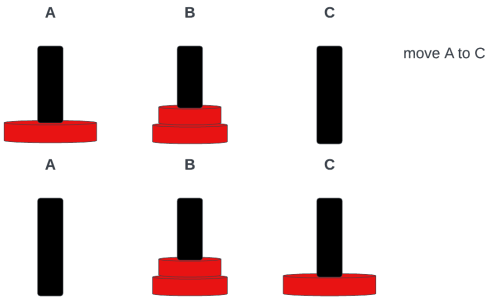

Figure 2: Second section. Moving the last disk to peg C.

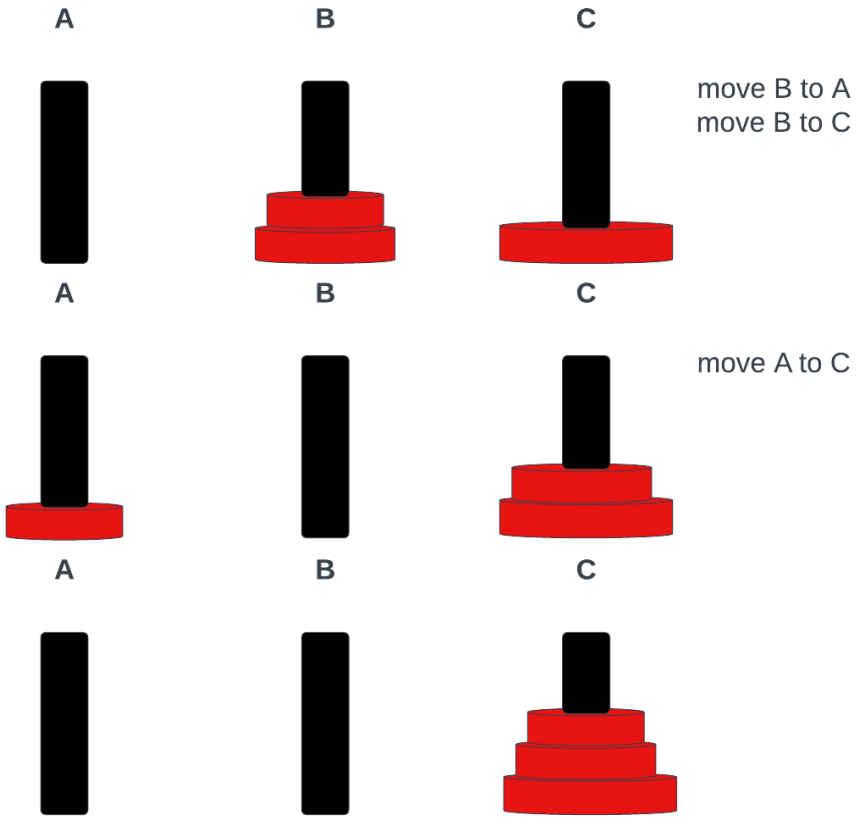

Figure 3: Third section. Move the tower of two from peg B to peg C.

Discussion

There's not really much to discuss when it comes to this task. The way to solve it is described above in the method section and the code for the `hanoi/4` function is just four lines of code. The hard part was, as in the earlier tasks, the recursive part. But having grasped that it was basically just writing down the recipe that we want the program to follow.

You can find the code for this task, and earlier tasks at my GitHub. I have also included a function there that implements the formula for calculating the number of moves required to move a tower.