ID1019: Towers of Hanoi

Alexander Lundqvist

Spring Term 2023

Introduction

In this task we will implement a program that can solve the classic puzzle **Towers of Hanoi** in the least amount of moves for a puzzle with three pegs and `n` number of discs. The rules of the game is that one has to move the stack of discs on the leftmost peg to the rightmost peg but it is not allowed to stack a larger disc on top of a smaller one. We are given a skeleton code in the form:

```
def hanoi(0, _, _, _) do ... end
def hanoi(n, from, aux, to) do
  move tower of size n-1 .... ++
  [ move one disc ... ] ++
  move tower of size n-1 ....
end
```

Result

The resulting program uses a recursive solution to find the minimum number of moves required to solve the puzzle for `n` number of discs. This is the main part of the program.

```
defp solve(0, _, _, _) do [] end
defp solve(n, left, middle, right) do
  solve(n-1, left, right, middle) ++
  [{:move, left, right}] ++
  solve(n-1, middle, left, right)
end
```

First we have the base case for the recursion. When we try to solve for 0 discs we only return an empty list that will be appended to the final list of moves. This is because the list appending doesn't accept nil.

```
  defp solve(0, _, _, _) do [] end
```

This call to `solve/4` is used to solve the sub problem of moving all discs except the largest to the middle peg so we can move the largest disc from the leftmost peg to the rightmost. We do this by treating the middle peg as the right peg and vice versa.

```
solve(n-1, left, right, middle) ++
```

This line represents moving the largest disc to the end peg. When traveling down the recursion tree this will be the endpoint where a move is added to the list.

```
[{:move, left, right}] ++
```

Finally, the function calls `solve/4` and treats the middle peg as the left peg. This step is necessary to move all the $n-1$ discs from the middle peg to the right peg, now with the largest disc at the bottom.

```
solve(n-1, middle, left, right)
```

When we execute the program we can specify the amount of discs we want to solve for. The following is a printout for 3 discs which when compared to the printout in the instructions verifies that the program works as intended.

```
iex(2)> Hanoi.run
Welcome to the Towers of Hanoi puzzle solver!
The puzzle currently has 3 pegs.
Enter the number of discs you want to solve for: 3
[
  {:move, :left, :right},
  {:move, :left, :middle},
  {:move, :right, :middle},
  {:move, :left, :right},
  {:move, :middle, :left},
  {:move, :middle, :right},
  {:move, :left, :right}
]
```

We can also solve for even greater amount of discs. Here is a sample when trying to solve for 4 discs.

```
iex(3)> Hanoi.run
Welcome to the Towers of Hanoi puzzle solver!
The puzzle currently has 3 pegs.
Enter the number of discs you want to solve for: 4
[
  {:move, :left, :middle},
```

```
  {:move, :left, :right},
  {:move, :middle, :right},
  {:move, :left, :middle},
  {:move, :right, :left},
  {:move, :right, :middle},
  {:move, :left, :middle},
  {:move, :left, :right},
  {:move, :middle, :right},
  {:move, :middle, :left},
  {:move, :right, :left},
  {:move, :middle, :right},
  {:move, :left, :middle},
  {:move, :left, :right},
  {:move, :middle, :right}
]
```

As the list grows exponentially for each added disc, we can deduct that the total amount of moves follows the formula $moves = 2^{disks} - 1$. This gives us $2^{10} - 1 = 1023$ as the answer to how many moves it would take for 10 discs. The entire code is available at this github repository.