Aula 6

March 22, 2020

Aula de 23 de Março

Vamos continuar explorando as possibilidade da recursão. Para isso, a ideia é resolver um problema clássico, a da séria de fibonacci.

A ideia é bem simples, a base da série, $F_1 e F_2$ valem ambos 1. Para seguir em frente e encontrar F_3, basta somar F_1 e F_2. Só que isso é válido para qualquer F_n, ou seja $F_n = F_{n-1} + F_{n-2}$.

Mas, antes de pensar em resolver o problema, vamos a boa prática de hoje. Será que dá para imaginar qual seria o resultado esperado, possibilitando a criação de testes automatizados, mesmo antes de começar?

A resposta é simples e é positiva, mas para isso é bem importante conhecer a assinatura da função que vamos usar. Pensando nisso, se queremos que uma função devolva o n-ésimo número de Fibonacci, precisamos de uma função que receba um parâmetro inteiro n. Que tal?

function fibo(n)

Conhecendo a função podemos pensar em testes para ela como?

fibo(1) == 1

fibo(3) == 2

fibo(10) == 55

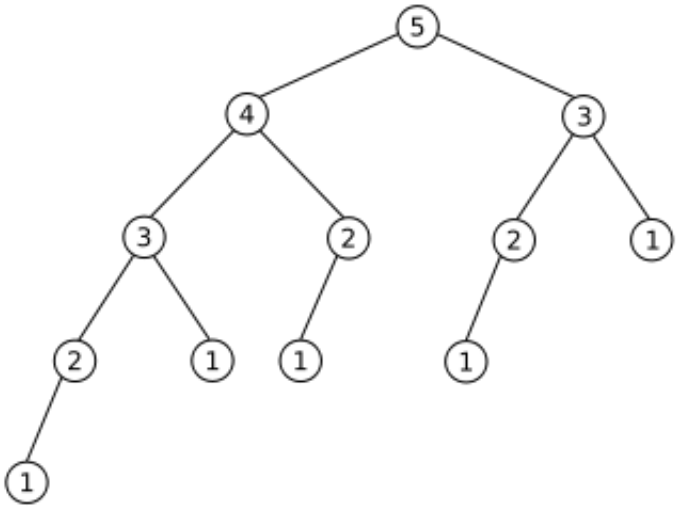
Conhecendo alguns resultados podemos escrever testes automatizados:

```
In [ ]: funtion testafibo()
          if fibo(1) == 1
            println("Deu certo para 1")
          end
          if fibo(3) == 2
            println("Deu certo para 2")
          end
          if fibo(10) == 55
            println("Deu certo para 10")
          end
          println("Final dos testes")
        end
```

A função de testes acima verifica se a função **fibo()** devolve o resultado correto para três casos. Mas, ela tem um defeito, ela imprime mensagens demais, o que pode ser ruim. Considerando isso, vamos ver o primeiro fundamento importante com relação a testes automatizados.

0.1 Se o teste passou, ele deve indicar apenas isso

Levando em conta o que foi escrito acima, podemos mudar o nosso teste para:



fibo5.png

```
In [ ]: funtion testafibo()
          if fibo(1) != 1
            println("Não deu certo para 1")
          end
          if fibo(3) != 2
            println("Não deu certo para 2")
          end
          if fibo(10) != 55
            println("Não eu certo para 10")
          end
          println("Final dos testes")
        end
```

Agora de posse da nossa função de testes, podemos escrever a nossa função de Fibonacci

```
In [ ]: function fibo(n)
```

É interessante notar que apesar de ser um dos exemplos clássicos de uso de recursão, o algoritmo acima é extremamente ineficiente. A razão é simples, cada vez que é feita a chamada, toda os valores de Fibonacci são recalculados para os valores de n e $n-1$.Veja o exemplo para 5 abaixo.

Vamos ver agora um outro problema, o cálculo do maior divisor comum entre dois números através do algoritmo de Euclides. Basicamente ele diz que o MDC de dois números a e b, é igual ao MDC de b e r, onde $r = a\%b$. Quando esse resto for zero, chegamos a solução, que é b.

Vamos a um exemplo abaixo:

```
In [26]: a = 348
         b = 156
         r = a % b
         println("Os números são: ", a, " ", b, " e o resto é: ", r)
         a = b
```

```
        b = r
        r = a % b
        println("Os números são: ", a, " ", b, " e o resto é: ", r)
        a = b
        b = r
        r = a % b
        println("Os números são: ", a, " ", b, " e o resto é: ", r)
```

```
Os números são: 348 156 e o resto é: 36
Os números são: 156 36 e o resto é: 12
Os números são: 36 12 e o resto é: 0
```

Inspirado no exemplo acima, podemos começar escrevendo uns testes para a nossa função, lembrando sempre que so devemos apontar os casos de erro ou o final dos testes.

Vamos usar o seguinte cabeçalho: **function MDC(a, b)**

```
In [ ]: # Aqui vamos escrever os testes da função MDC(a, b)
```

```
In [ ]: # Aqui vamos escrever a função MDC(a, b)
```

Para acabar a aula, vamos ver que recursão apesar de ser simples, não é tão simples, pois exige um bom esforço mental. Qual é o resultado da função abaixo se ela é chamada com 4?

```
In [ ]: function misterio(n)
            if n <= 2
                return n
            else
                return misterio(n - 1) + n * misterio(n -2)
            end
        end
```