Recursão

Funções Recursivas

A recursividade é muito utilizada na matemática e em programação para determinadas linguagens (ex: Prolog, Haskell, C, etc).

Uma função é dita recursiva, se ela é definida em termos dela mesma.

- Exemplos de definição matemática recursiva

4.1 Fatorial: n!

Definição não recursiva:
n! = n * (n-1) * (n-2) * ... * 2 * 1, para n > 0
0! = 1

Ex: 4! = 4 * 3 * 2 * 1 = 24

Definição recursiva:
n! = n * (n-1)! , para n > 0
0! = 1

Ex: 4! = 4 * 3! = 4 * 3 * 2! = 4 * 3 * 2 * 1! = 4 * 3 * 2 * 1 * 0! = 24

Em Haskell:

a) Definição de fatorial com guardas:

```
fatorialg :: Int -> Int
fatorialg n
 | n==0 = 1
 | otherwise = n * fatorialg (n-1)

> fatorialg 4
24
```

b) Definição de fatorial com casamento de padrão:

```
fatorial :: Int -> Int
fatorial 0 = 1
fatorial n = n * fatorial (n-1)

> fatorial 4
24
```

Obs: o fatorial de 0 é nosso caso base: fatorial 0 = 1

4.2 Sequência de Fibonacci

Definição recursiva

Recursão

Fn: enésimo elemento na sequência

Fn = 1, se n=1 ou F1 = 1
Fn = 1, se n=2 ou F2 = 1
Fn = Fn-2 + Fn-1, se n > 2

F3 = F1 + F2 = 1 + 1 = 2

F5 = F3 + F4 = (2) + (F2 + F3) = 2 + (1 + 2) = 5

Sequência: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Em Haskell:

a) Definição de Fibonacci com guardas:

```
fibog :: Int -> Int

fibog n
 | n == 1 = 1
 | n == 2 = 1
 | otherwise = fibog (n - 2) + fibog (n - 1)

> fibog 1
1

> fibog 4
3
```

b) Definição de Fibonacci com casamento de padrões:

```
fibo :: Int -> Int

fibo 1 = 1
fibo 2 = 1
fibo n = fibo (n - 2) + fibo (n - 1)

> fibo 1
1

> fibo 4
3
```

4.3 Máximo Divisor de 2 números (Algoritmo de Euclides)

Definição recursiva:
mdc(m,n) = m, se n = 0
mdc(m,n) = mdc(n, m mod n), se n > 0

Ex: mdc (18,12) = ?
Múltiplos de 18: {18, 9, 6, 3, 2, 1}
Múltiplos de 12: {12, 6, 4, 3, 2, 1}
Portanto, mdc (18,12) = 6

Pelo algoritmo:
mdc (18,12) = ? Como n > 0, temos 18 mod 12 = 6
mdc (12,6) = ? Como n > 0, temos 12 mod 6 = 0
mdc (6,0) = 6 pois n = 0.

Assim: mdc(18,12)=mdc(12,6)=mdc(6,0)=6

Em Haskell:

a) Definição de mdc com guardas:

```
mdcg :: (Int,Int) -> Int
mdcg (m,n)
 | n == 0 = m
 | otherwise = mdcg (n, (mod m n))

> mdcg (18,12)
6
```

b) Definição de mdc com casamento de padrões:

```
mdc :: (Int,Int) -> Int
mdc (m,0)  = m
mdc (m,n) = mdc (n, (mod m n))

> mdc (18,12)
6
```

4.4 Binomial (n k): combinação de n elementos em grupos de tamanho k

Definição recursiva:
binomial (n,k) = 1, se k = 0 ou binomial (n,0) = 1,
binomial (n,k) = 1, se k = n ou binomial (n,n) = 1,
binomial (n,k) = binomial (n-1,k) + binomial (n-1,k-1), se 0 < k < n
obs: binomial (n,k) não é definido se k>n

Pelo algoritmo:
binomial (4,2) = ? Como 0 < k < n, temos que binomial (4,2) = binomial (3,2) + binomial (3,1)
binomial (3,2) = ? Como 0 < k < n, temos que binomial (3,2) = binomial (2,2) + binomial (2,1)
binomial (2,2) = 1, pois k = n
binomial (2,1) = ? Como 0 < k < n, temos que binomial (2,1) = binomial (1,1) + binomial (1,0)
binomial (1,1) = 1, pois k = n
binomial (1,0) = 1, pois k = 0
Portanto, binomial (2,1) = binomial (1,1) + binomial (1,0) = 1 + 1 = 2
Portanto, binomial (3,2) = binomial (2,2) + binomial (2,1) = 1 + 2 = 3
binomial (3,1) = ? Como 0 < k < n, temos que binomial (3,1) = binomial (2,1) + binomial (2,0)
binomial (2,1) = 2 obs: já calculado anteriormente
binomial (2,0) = 1, pois k = 0
Portanto, binomial (3,1) = binomial (2,1) + binomial

(2,0) = 2 + 1 = 3

Portanto, binomial (4,2) = binomial (3,2) + binomial (3,1) = 3 + 3 = 6

Resultado: binomial (4,2) = 6

Em Haskell:

a) Definição de binomial com guardas:

```
binomialg :: (Int,Int) -> Int
binomialg (n,0)  = 1
binomialg (n,k)
 | k == 0 = 1
 | k == n = 1
 | otherwise = binomialg (n-1,k) + binomialg (n-1,k-1)

> binomialg (4,2)
6
```

b) Definição de binomial com casamento de padrões:

```
binomial :: (Int,Int) -> Int
binomial (n,0)  = 1
binomial (n,k) = if (k == n) then 1 else binomial (n-1,k) +
binomial (n-1,k-1)
```

Obs: binomial (m,m) = 1 não funciona em Haskell. Só se pode utilizar a variável b uma vez como argumento de entrada

> binomial (4,2)
6