**Programação Funcional e em Lógica**

**Relatório do Módulo 1 – Programação Funcional**

**Realizado por: Frederico Lopes (upXXXXXXX) e Pedro Pacheco (up201806824)**

**Descrição e casos teste das funções implementadas**

fibRec :: (Integral a) => a -> a

Implementação recursiva pouco eficiente em que para cada termo que não 0 ou 1 é feita uma chamada recursiva para calcular esse termo. O mesmo termo pode e em princípio será calculado várias vezes.

Caso de teste:

- fibRec 50

fibLista :: (Integral a) => a -> a

Implementação com lista semelhante ao que acontece em programação dinâmica noutros tipos de linguagens. Cada termo só é calculado uma vez e este fica "guardado" na lista para ser usado no cálculo dos tempos seguintes. Só são calculados os termos necessários para o cálculo da sequência de Fibonacci.

Caso de teste:

- fibRec 500

fibListaInfinita :: (Integral a) => a -> a

Implementação com lista infinita em que se toma partido da lazy evaluation de Haskell. Uma lista infinita é gerada, que soma (zipWith) os elementos da lista com a cauda dos elementos da mesma lista criando assim uma lista com os resultados das somas.

Caso de teste:

- fibRec 1000

scanner :: String -> BigNumber

String é passada para Int e de seguida são feitas sucessivas divisões das quais é guardado o módulo para guardar no BigNumber. A função tem várias guardas para garantir que também é possível ler números negativos.

Casos de teste:

- scanner "1000"

- scanner "-1000"

output :: BigNumber -> String

Nesta função é simplesmente feita uma chamada da função "show" para todos os elementos do BigNumber através da função concatMap.

Casos de teste:

- scanner [-1,0,0,0]

- scanner [1,9,8,7]

somaBN :: BigNumber -> BigNumber -> BigNumber

Nesta função é feita uma soma sucessiva dos algarismos menos significativos e se existir overflow este é passado para o algarismo seguinte. Foi necessário criar várias guardas para os casos de overflow para não ficarem escritos 0 (zeros) no BigNumber (ex: somaBN [1,5] [1,0] seria [0,2,5] sem as guardas).

Estas somas sucessivas são conseguidas a partir de chamadas recursivas para aceder aos vários elementos do BigNumber.

Casos de teste:

- somaBN [5] [2,7]

- somaBN [5] [27]

- somaBN [9,9] [5,5]

subBN :: BigNumber -> BigNumber -> BigNumber

A função de subtração é algo semelhante à de adição da parte de ser uma subtração sucessiva, ainda que as operações realizadas entre os algarismos sejam diferentes.

Casos de teste:

-

-

-

simpleMul :: Int -> Int -> Int

A função simpleMul recebe dois números inteiros e faz uma multiplicação simples. É chamada na função mulBN com o propósito de multiplicar dois números presentes em duas listas distintas, cada uma representativa de um BigNumber.

Casos de teste:

- simpleMul 2 3

- simpleMul 12 35

- simpleMul 342 555

mulBN :: BigNumber -> BigNumber -> BigNumber

A função mulBN começa por colocar numa lista revertida a multiplicação simples, usando a função simpleMul, dos algarismos das duas listas representativas de BigNumbers distintos e revertidos. De seguida divide a lista anterior a meio e guarda as duas metades num tuplo. Posteriormente adiciona um 0 no ínicio da lista guardada na segunda posição do tuplo e um 0 no fim da lista guardada na primeira posição do tuplo. Ambos os zeros são adicionados por motivos da soma que se segue posterior à multiplicação no algoritmo da multiplicação. No final, são somados os dois números resultantes da multiplicação, devolvendo o resultado final da multiplicação.

Casos de teste:

-

-

-

divBN:

fibRecBN :: Int -> BigNumber

Semelhante a fibRec com a diferença que a operação de soma entre termos é feita com somaBN devido aos termos serem BigNumbers.

Casos de teste:

-

-

-

fibListaBN :: Int -> BigNumber

Semelhante a fibLista com a diferença que a operação de soma entre termos é feita com somaBN devido aos termos serem BigNumbers.

Casos de teste:

-

-

-

fibListaInfinitaBN :: Int -> BigNumber

Semelhante a fibListaInfinita com a diferença que a operação de soma entre termos é feita com somaBN devido aos termos serem BigNumbers.

Casos de teste:

-

-

-

O desenvolvimento das funções da alínea 2 (dois) tiveram por base estratégias de #explicar estratégias de desenvolvimento#

4)?????????????????

Int: [-9223372036854775808, 9223372036854775807]

Integer: ?

BigNumber: O limite de algarismos é o limite maximo de tamanho de listas em Haskell.