Programação Funcional e em Lógica

Project developed for PFL unit

Project Team

- Rui Pedro Mendes Moreira **up201906355**
- Sérgio Rodrigues da Gama **up201906690**

Test Cases

- There are tests for all the main BigNumber and Fibonacci modules functions in the Tests.hs file. The tests for `output and scanner` were done with `IO Monad` by comparing some expected values with the ones computed by those functions and returning PASSED or FAILED to the screen.

- All the other tests were made with the help of QuickTest module, making a property function for the Fibonacci functions and somaBN, subBn, mulBN and divBn, from BigNumber module. Finally, in order to execute all of these tests with one function call, the main IO function is responsible for running the tests.

Functions Descriptions

Fibonacci Functions

- **1.1 fibRec**: This function takes one argument, which is the index of the Fibonacci sequence we want to calculate. It is implemented using a `naive recursive strategy`, calculating the value fib(n), by adding fib(n-1) + fib(n-2), aside from the base cases which are fib(0) = 0 and fib(1) = 1;

- **1.2 fibLista**: This function is based in fibRec, however, taking advantage of a `dynamic programming` approach. It saves the calculated results in a list that it is passed to the next function calls, retrieving the values needed from that list and preventing multiple computations of the same value. In order to achieve this, fibLista() uses an external auxiliary function `fibListaAux()`.

- **1.3 fibListaInfinita**: In contrary to the last function that uses finite lists, this one takes advantage of `infinit lists`. Due to the lazy computation feature of Haskell, the list is only calculated upon what it is requested.

- **1.4 fibRecBN**, **1.5 fibListaBN** and **1.6 fibListaInfinitaBN**: This group of functions work exactly the same way the ones previously explained, but with the integration of BigNumbers.

BigNumber Functions

Brief BigNumber module function description, see beneath for more details

- **2.2 scanner**: String to BigNumber conversion.

- **2.3 output**: BigNumber to String conversion.

- **2.4 somaBN**: Addition function for BigNumbers, returning the result in a BigNumber.

- **2.5 subBN**: Subtraction fucntion for BigNumbers.
- **2.6 mulBN**: Multiplication function for BigNumbers.
- **2.7 divBN**: Division function for BigNumbers, returning both the remainder and the quotient in BigNumber form.
- **3 safeDivBN**: Divides one BigNumber by another, by making use of `Maybe` Preludes data to solve division by 0 problem, returning Nothing in that case.

BigNumber Util functions

- **bnToInt**: BigNumber to Int conversion.
- **intToBN**: Int to BigNumber conversion.
- **bnFractionalToInt**: Converts a tuple of BigNumbers retrieved from divBN, into an Int.
- **intToList**: Int to [Int] conversion.
- **listToInt**: [Int] to Int conversion.
- **stringToNumbers**: Converts a String into an array of Int's. When the String has non digit characters, digitToInt() prelude function, outputs the matching error.
- **numbersToString**: Converts an array of Int's into a String (an array of Char's).
- **trimString**: Removes all the trailling '0' characters from the beginning of the String, until a different one is found.
- **trimInts**: Removes all the trailling 0's from an array of Int's from the beginning of the array, until it is found a none 0 Int.
- **biggerBN**: Compares two BigNumbers (a and b) and returns 0, if a > b, 1 if b > a, 2 if a == b, 4 if both numbers are Zeros
- **notBN**: Negates a BigNumber, by changing its sign (Bool). Zero prevails Zero.
- **somaBNaux**: Sums two numbers in a form of lists of Ints and returns a list of Ints with the result.
- **subBNaux**: Subtracts two numbers in a form of lists of Ints and returns a list of Ints with the result.
- **mulBNaux**: Multiplies two numbers in a form of lists of Ints and returns a list of Ints with the result.
- **divBNaux**: Divides one number by another, both in a form of lists of Ints and returns a list of Ints with the result.
- **divAuxEmptyToZero**: Converts a tuple of lists of Ints into a tuple of correspondant BigNumbers.

Strategies used on implementig BigNumber functions

Data Definition

- The BigNumbers (Numbers in which the size is only limited by the computer memory) definition was created with the intent of simplifying the further development of the functions needed to manipulate them. Therefore, we define a constructor BN as being a list of digits and a Bool dictating the sign `(True: positive, False: negative)`. However, 0 dos not has sign, so we added a constructor (Zero) to represent it.

Main BigNumber Manipulation Functions

- **scanner**: When the String is in the correct format following this Regular Expression `(+|-)(0|1|2|3|4|5|6|7|8|9)* + (+|-|ε)(0)*`, otherwise an error is shown. The convertion is done firstly by checking the sign and atributting the correspondant Bool in the BigNumber (True -> positive, False -> negative) and then converting the numbers int a list of Ints using stringToNumbers() util function. When one or multiple 0 are inputted, we call the Zero constructor of BigNumber data. It also removes the trailling zeros from the String using `trimString()` util function.

- **output**: When the BigNumber it is Zero, returns "0", otherwise it checks for the number signal and parses into "+" or "-", followed by the concatenation of the digits previously converted and trimmed with the composed function made by `trimString . numbersToString` functions.

- **somaBN**: somaBN is composed by the 6 cases that can occour in addition operation:

    - Both numbers are Zero -> returns Zero
    - Only one number is Zero -> returns the other number
    - When both (a and b) are none Zero numbers:
    - (-a) + (-b) <=> -a - b <=> -(a+b) -> returns the sum of a + b, through `somaBNaux()` function, and its sign to False (meaning a negative number)
    - (+a) + (+b) <=> a + b -> returns the sum of a + b, through `somaBNaux()` function, and its sign to True (meaning a positive number)
    - (+a) + (-b) <=> a - b -> returns the subtraction of a - b, by calling the subBN() function with a (positive) and b (positive, opposite of the inputted)
    - (-a) + (+b) <=> b - a -> returns the subtraction of b - a, by calling the subBN() function with b (positive) and a (positive, opposite of the inputted)

- **subBN**: subBN is composed by the 7 cases that can occour in subtraction operation:

    - Both numbers are Zero -> returns Zero
    - The first number is Zero -> returns the second number negated
    - The second number is Zero -> returns the other number
    - When both (a and b) are none Zero numbers:
    - (+a) - (+b) <=> a - b -> returns the subtraction of a - b, through `subBNaux()` function, and its sign is given upon the value returned by `biggerBN()` function
    - (-a) - (-b) <=> b - a -> returns the sum of b - a, by calling the subBN function with a (positive) and b (positive)
    - (+a) - (-b) <=> a + b -> returns the subtraction of a + b, by calling the somaBN() function with a (positive) and b (positive)
    - (-a) - (+b) <=> -(a + b) -> returns the subtraction of a + b, by calling the somaBN() function with a (positive) and b (positive), followed by the negation of the BigNumber through `notBN()` function.

- **mulBN**: mulBN basicly calls `mulBNaux()` with both numbers digits list and returns a BigNumber with that result and the correspondat sign.
- **divBN**: Like mulBN, divBN calls `divBNaux()` with both numbers digits list and returns a tuple of two BigNumbers with the result from divBNaux(), that is previously converted from ([Int],[Int]) to a tuple of BigNumbers using `divAuxEmptyToZero()` function.

Auxiliary BigNumber Calculations Functions

- **somaBNaux**:
- **subBNaux**:
- **mulBNaux**:
- **divBNaux**:

Function Comparisons (answer to topic 4)

We used `:set +s` in ghci to see the execution time of the Fibonacci module functions for Integrals and for BigNumbers, to observate the differences, we tested the execution for the same 3 values 5 times and made the average of those trials.

Values tested: 15, 25, 30, 50 and 500

Note: the higher values were not tested in the naive strategies, because of the computation time too big. Execution time is in seconds.

Integrals

fibRec

	1	2	3	4	5	Average
15	0.01	0.01	0.01	0.01	0.01	0.01
25	0.41	0.42	0.49	0.40	0.40	0.424
30	4.58	5.16	5.22	5.40	4.98	5.068

fibLista

	1	2	3	4	5	Average
15	0	0	0	0	0	0
25	0	0	0	0	0	0
30	0	0	0	0	0	0
50	0.01	0	0	0.01	0.01	0.006
500	0.06	0.02	0.02	0.02	0.02	0.028

fibListaInfinita

	1	2	3	4	5	Average
15	0	0	0	0	0	0
25	0	0	0	0	0	0
30	0	0	0	0	0	0
50	0.01	0	0	0.01	0	0.004
500	0.01	0.01	0.01	0.01	0.01	0.01

BigNumber

fibRecBN

	1	2	3	4	5	Average
15	0.02	0.02	0.02	0.02	0.02	0.02
25	1.71	1.69	1.86	1.72	1.67	1.73
30	18.3	20.38	20.36	20.62	20.1	19.952

fibListaBN

	1	2	3	4	5	Average
15	0	0	0	0	0	0
25	0	0	0	0	0	0
30	0	0	0	0	0	0
50	0.01	0.01	0.01	0	0	0.006
500	0.11	0.11	0.10	0.10	0.11	0.106

fibListaInfinitaBN

	1	2	3	4	5	Average
15	0	0	0	0	0	0
25	0	0	0	0	0	0
30	0	0	0	0	0	0
50	0.01	0	0	0	0	0.002
500	0.09	0.10	0.10	0.10	0.10	0.098

Conclusion

- After all these tests we observe that the execution times of naive functions are greater than the rest of the strategies used. Moreover, the use of the dynaic programming strategie with infinit list seems to be faster than the dynamic programming approach with finite lists.
- The Fibonacci functions with BigNumber integration are in general slower than the ones implemented with Integers (Int and Integers).