PFL-2021-G4_04

Members

André Flores - 201907001

Sara Marinha - 201906805

Functions

fibRec

Calculate n-th fibonacci number using recursion.

fibLista

Calculate n-th fibonacci number using a finite list comprehension where each number is the sum of the two previous numbers.

fibHelper

Calculate n-th fibonacci number using a list accumulator and recursion where with each recursive call the new fibonacci number is appended to the list until it is sufficiently long, then last value is returned.

fibLista'

Calculate n-th fibonacci number by calling fibHelper with accumulator list starting at [0, 1].

fibListaInfinita

Calculate n-th fibonacci number using an infinite list comprehension where each number is the sum of the two previous numbers.

fibRecBN

Calculate n-th fibonacci number using recursion using BigNumber for argument and return value.

fibListaBN

Calculate n-th fibonacci number using a finite list comprehension where each number is the sum of the two previous numbers using BigNumber for argument and return value.

fibListaInfinitaBN

Calculate n-th fibonacci number using an infinite list comprehension where each number is the sum of the two previous numbers using BigNumber for argument and return value.

isNegative

Checks whether the BigNumber is negative.

isPositive

Checks whether the BigNumber is positive.

minusBNHelper

Calculates the additive inverse of a given BigNumber.

removeLeadingZeros

Removes leading zeros of a BigNumber.

minusBN

Calculates the additive inverse of a given BigNumber after removing leading zeros.

firstBN

Return the leftmost digit of a BigNumber.

tailBN

Return the given BigNumber without its leftmost element.

paddBN

Padds the first argument with leading zeros until it has the same length as the second argument.

scannerHelper

Converts a string into BigNumber, can only handle positive numbers.

scanner

Converts a string into a BigNumber.

output

Converts BigNumber into a String.

somaPosBN

Add two positive BigNumbers using manual addition algorithm.

somaBN

Add any two BigNumbers.

orderedSubPosBN

Subtract two positive BigNumbers as long as the left is greater than the right using manual subtraction algorithm.

lengthBN

Return the length (number of digits) of a BigNumber.

greaterEqualLength

Compare two BigNumbers of equal length and check if the left is greater than the right.

greaterBN

Compare two BigNumbers and check if the left is greater than the right.

subBNHelper

Subtract any two positive BigNumbers.

subBN

Subtract any two BigNumbers.

eBN

Equivalent to x * 10 ^ n.

baseMulBN

Multiply two BigNumbers in the format x * 10 ^ ex and y * 10 ^ ey if x and y are single digit.

digitOrdBN

Get the n-th digit of a BigNumber, counting from the left and starting at 0.

mulListBN

Return a list of the results of the smaller multiplications that are made in multiplying bigger numbers using a manual multiplication algorithm.

mulPosBN

Multiply two positive BigNumbers by using foldl with somaBN to add the results of mulListBN.

mulBN

Multiply any two BigNumbers.

equalsBN

Check if two BigNumbers are equal.

greaterOrEqualsBN

Compare two BigNumbers to check if left is greater than or equal to the right.

lesserOrEqualsBN

Compare two BigNumbers to check if left is lesser than or equal to the right.

naiveDivBN

Divide BigNumbers using trial and error with multiplication.

divBNHelper

Divide BigNumbers using a manual division algorithm.

divBN

Whole division of two BigNumbers by calling divBNHelper with accumulators starting at 0.

bigNumberToInt

Convert a BigNumber (BN xs x) into an Int, by adding x (already an Int) to 10 times the result of bigNumberToInt xs until xs is empty.

safeDivBN

Check if trying to divide by 0, if so return Nothing, otherwise call divBN.

Test Cases

```
--fibRec :: (Integral a) => a -> a
fibRec 0
fibRec 1
fibRec 20

--fibLista :: (Integral a) => a -> a
fibLista 0
fibLista 1
fibLista 20

--fibHelper :: Integral a => Int -> Int -> [a] -> a
fibHelper 5 2 [0,1]
fibHelper 8 5 [0,1,1,2,3]
fibHelper 8 3 [0,1,1]

--fibLista' :: Integral a => Int -> a
fibLista' 8
fibLista' 5

--fibListaInfinita :: (Integral a) => a -> a
fibListaInfinita 8
fibListaInfinita 5

--fibRecBN :: BigNumber -> BigNumber
output (fibRecBN (scanner "8"))
output (fibRecBN (scanner "5"))

--fibListaBN :: BigNumber -> BigNumber
output (fibListaBN (scanner "8"))
output (fibListaBN (scanner "5"))

--fibListaInfinitaBN :: BigNumber -> BigNumber
output (fibListaInfinitaBN (scanner "8"))
output (fibListaInfinitaBN (scanner "5"))

--isNegative :: BigNumber -> Bool
isNegative (scanner "12")
isNegative (scanner "0")
isNegative (scanner "-12")

--isPositive :: BigNumber -> Bool
isPositive (scanner "12")
isPositive (scanner "0")
isPositive (scanner "-12")

--minusBNHelper :: BigNumber -> BigNumber
output (minusBNHelper (scanner "-12"))
output (minusBNHelper (scanner "12"))

--removeLeadingZeros :: BigNumber -> BigNumber
output (removeLeadingZeros (scanner "000012"))
output (removeLeadingZeros (scanner "100002"))
output (removeLeadingZeros (scanner "120000"))

--minusBN :: BigNumber -> BigNumber
output (minusBN (scanner "-000012"))
output (minusBN (scanner "000012"))
output (minusBN (scanner "12"))

--firstBN :: BigNumber -> Int
firstBN (scanner "-12")
firstBN (scanner "12")

--tailBN :: BigNumber -> BigNumber
output (tailBN (scanner "1245"))
output (tailBN (scanner "12"))

--paddBN :: BigNumber -> BigNumber -> BigNumber
output (paddBN (scanner "12") (scanner "123"))
output (paddBN (scanner "12") (scanner "12367"))
output (paddBN (scanner "12") (scanner "1"))

--scannerHelper :: String  -> BigNumber
output (scannerHelper "12")

--scanner :: String -> BigNumber
output (scanner "-12")
output (scanner "12")
```

```
--somaPosBN :: BigNumber -> BigNumber -> BigNumber
output (somaPosBN (scanner "1") (scanner "0"))
output (somaPosBN (scanner "1") (scanner "1"))
output (somaPosBN (scanner "0") (scanner "1"))

--somaBN :: BigNumber -> BigNumber -> BigNumber
output (somaBN (scanner "1") (scanner "0"))
output (somaBN (scanner "1") (scanner "1"))
output (somaBN (scanner "0") (scanner "1"))
output (somaBN (scanner "123") (scanner "-123"))
output (somaBN (scanner "-123") (scanner "-123"))
output (somaBN (scanner "-123") (scanner "123"))

--orderedSubPosBN :: BigNumber -> BigNumber -> BigNumber
output (orderedSubPosBN (scanner "1") (scanner "0"))
output (orderedSubPosBN (scanner "1") (scanner "1"))

--lengthBN :: BigNumber -> Int
lengthBN (scanner "1234567890")

--greaterEqualLengthBN :: BigNumber -> BigNumber -> Bool
greaterEqualLengthBN (scanner "9") (scanner "1")
greaterEqualLengthBN (scanner "9") (scanner "9")
greaterEqualLengthBN (scanner "1") (scanner "9")

--greaterBN :: BigNumber -> BigNumber -> Bool
greaterBN (scanner "1") (scanner "10")
greaterBN (scanner "1") (scanner "1")
greaterBN (scanner "10") (scanner "1")

--subBNHelper :: BigNumber -> BigNumber -> BigNumber
output (subBNHelper (scanner "2") (scanner "1"))
output (subBNHelper (scanner "1") (scanner "2"))

--subBN :: BigNumber -> BigNumber -> BigNumber
output (subBN (scanner "1") (scanner "0"))
output (subBN (scanner "1") (scanner "1"))
output (subBN (scanner "0") (scanner "1"))
output (subBN (scanner "123") (scanner "-123"))
output (subBN (scanner "-123") (scanner "-123"))
output (subBN (scanner "-123") (scanner "123"))

--eBN :: BigNumber -> Int -> BigNumber
output ((scanner "1") `eBN` 1)
output ((scanner "-1") `eBN` 2)
output ((scanner "0") `eBN` 100)

--baseMulBN :: BigNumber -> BigNumber -> Int -> Int -> BigNumber
output (baseMulBN (scanner "1") (scanner "3") 0 0)
output (baseMulBN (scanner "9") (scanner "9") 1 0)

--digitOrdBN :: BigNumber -> Int -> BigNumber
digitOrdBN (scanner "123") 0
digitOrdBN (scanner "123") 1
```

```
--mulListBN :: BigNumber -> BigNumber -> [BigNumber]
mulListBN (scanner "20") (scanner "13")

--mulPosBN :: BigNumber -> BigNumber -> BigNumber
output (mulBN (scanner "1") (scanner "1"))
output (mulBN (scanner "1") (scanner "0"))
output (mulBN (scanner "0") (scanner "1"))
output (mulBN (scanner "123") (scanner "123"))

--mulBN :: BigNumber -> BigNumber -> BigNumber
output (mulBN (scanner "1") (scanner "1"))
output (mulBN (scanner "1") (scanner "0"))
output (mulBN (scanner "0") (scanner "1"))
output (mulBN (scanner "123") (scanner "123"))
output (mulBN (scanner "-123") (scanner "123"))
output (mulBN (scanner "123") (scanner "-123"))
output (mulBN (scanner "-123") (scanner "-123"))

--equalsBN :: BigNumber -> BigNumber -> Bool
equalsBN (scanner "1") (scanner "10")
equalsBN (scanner "1") (scanner "1")
equalsBN (scanner "10") (scanner "1")

--greaterOrEqualsBN :: BigNumber -> BigNumber -> Bool
greaterOrEqualsBN (scanner "1") (scanner "10")
greaterOrEqualsBN (scanner "1") (scanner "1")
greaterOrEqualsBN (scanner "10") (scanner "1")

--lesserOrEqualsBN :: BigNumber -> BigNumber -> Bool
lesserOrEqualsBN (scanner "1") (scanner "10")
lesserOrEqualsBN (scanner "1") (scanner "1")
lesserOrEqualsBN (scanner "10") (scanner "1")

--naiveDivBN :: BigNumber -> BigNumber -> BigNumber -> (BigNumber, BigNumber)
naiveDivBN (scanner "12") (scanner "3") (scanner "1")
naiveDivBN (scanner "9") (scanner "3") (scanner "1")

--divBN :: BigNumber -> BigNumber -> (BigNumber, BigNumber)
divBN (scanner "10") (scanner "1")
divBN (scanner "10") (scanner "10")
divBN (scanner "10") (scanner "100")
divBN (scanner "77") (scanner "4")

--bigNumberToInt :: BigNumber -> Int
bigNumberToInt (scanner "0")
bigNumberToInt (scanner "10")
bigNumberToInt (scanner "-10")

--safeDivBN :: BigNumber -> BigNumber -> Maybe (BigNumber, BigNumber)
safeDivBN (scanner "1") (scanner "0")
safeDivBN (scanner "1") (scanner "1")
```

Strategies

BigNumber

We used a *data* definition instead of a *type* one since it allowed us to recursively define a new type and to use pattern matching. BigNumber is defined like an inversed list where the "head" would be its last digit to facilitate the algorithms we used.

scanner

Checks and handles whether the argument string represents a positive or negative number. Reverses string to facilitate storing the number in a BigNumber format.

output

Converts a BigNumber into a string by converting its last digit to its representative character and appending that value to the result of a recursive call to output with the rest of the BigNumber. If a number is negative, a '-' is inserted at the beginning.

somaBN and subBN

Add or subtract two BigNumbers using a manual addition/subtraction algorithm. Addition starts by adding their last two digits, if value exceeds 9 then a carry happens to the next digit of the first BigNumber (xs). Subtraction subtracts the last digits of the argument BigNumbers, if the value would be negative carry 10 from the last digit of the first BigNumber (xs). Both construct the result with this value and the result of a recursive call. The functions are complimentary and may call each other to handle negative numbers.

mulBN

Uses a manual multiplication algorithm. Adds the values of a list of simpler multiplications of all permutations of digit pairs between the two BigNumbers with appropriate order of magnitude.

divBN

Follows manual division algorithm. Use the quotient and remainder as accumulators starting at 0. Take the remainder, multiply it by 10 and add the first digit of the dividend, then naively divide (using multiplication) that value by the quotient and store the results appropriately in the accumulators. Execute recursive calls with the remaining digits of the dividend and the quotient until dividend is Empty, then return a quotient and remainder pair.

Fibonacci Comparison

We attempted to calculate the first number where overflow would occur for each type.

Int

We used this code:

```
head (filter (\(x, y) -> x < 0) (zip (map fibListaInfinita [0::Int .. ]) [0 ..]))
```

returning this:

```
(-6246583658587674878,93)
```

Which indicated to us the largest fibonacci number that could be represented as an *Int* was the 92nd, 7540113804746346429.

Integer and BigNumber

We attempted the same with both *Integer* and *BigNumber*, but this time using a timeout of 1 minute because reaching a result was taking too long. Code:

```
--Integer
timeout (60*10^6) (print (fst (head (filter (\(x, y) -> x < 0) (zip (map
fibListaInfinita [0::Integer ..]) [0 ..] )))))

--BigNumber
lista = zeroBN : [last lista `somaBN` oneBN]

x = head (filter (\(x, y) -> not (x `greaterOrEqualsBN` zeroBN)) (zip (map
fibListaInfinitaBN lista) [0 ..]))

z = timeout (60*10^6) (print (fst x))
```

We arrived at the conclusion that the largest number both of these types can represent is only limited by the available memory in the computer where they are being stored, allowing them to be arbitrarily large.