PFL

> Descrição de vários casos de teste para todas as funções

Os casos de teste estão descritos no ficheiro **tests.hs** e podem ser vistos através da execução da função **testes**. Em todos os casos é mostrado, explicitamente, o input utilizado e o output associado. Para o bom funcionamento da função e para permitir a execução de todos os testes de uma só vez, não foram descritos, na função testes, os casos de erro que as funções estão preparadas para lidar mas podem ser entendidos na seguinte tabela:

Função	Erro e descrição
scanner	Esta função reporta um erro, através da função auxiliar *charToInt*, caso um caracter da *String* não seja um dígito positivo, excetuando o primeiro caracter que pode ser negativo.
output	Esta função reporta um erro, através da função auxiliar *intToChar*, caso um elemento da lista não seja um dígito positivo, excetuando o primeiro elemento que pode ser negativo.
somaBN subBN mulBN divBN safeDivBN	Estas funções reportam um erro caso alguns dos argumentos não seja um BigNumber válido. Esta verificação é feita através da função *verificaBN*.
divBN	Esta função reporta um erro caso o divisor seja o BigNumber correspondente ao número 0.

> Explicação sucinta do funcionamento de cada função

BigNumber.hs

Função	Descrição
scanner	Esta função realiza a conversão de String para BigNumber recorrendo a duas funções auxiliares *charToInt* e *scanner'*.
scanner'	Esta função converte a cabeça da lista para inteiro, com a ajuda da segunda função auxiliar *charToInt*, chamando posteriormente a si mesma de forma recursiva até converter todos os elementos da lista.
charToInt	Esta função faz a associação entre o valor do caracter com o valor numérico através do uso de *pattern matching*. Caso receba um caracter que não represente um dígito positivo, esta retorna erro.
output	Esta função realiza a conversão de BigNumber para string recorrendo a duas funções auxiliares *output'* e *intToChar*.
output'	Esta função converte a cabeça da lista para um caracter, com a ajuda da segunda função auxiliar *intToChar*, concatenando-a com a chamada recursiva a si mesma até converter todos os elementos da lista.
intToChar	Esta função faz a associação entre o valor númerico com o valor do caracter através do uso de *pattern matching*. Caso receba no BigNumber algum valor que não seja um dígito postivo, esta retorna erro.
verificaBN	Função auxiliar para verificar se o número recebido é um BigNumber válido. Verifica se a cabeça da lista está entre -9 e 9. Posteriormente chama uma função auxiliar *verificaBNAux* para verificar o resto da lista.
verificaBNAux	Função auxiliar que verifica se os elementos da cauda da lista são BigNumbers, ou seja, estão entre o intervalo de 0 a 9.
removeZero	Função auxiliar para eliminar os zeros à esquerda.
maiorQue	Esta função verifica qual a lista maior entre as duas que recebe, retorna True se a primeira for maior.
adicionaSinal	Esta função adiciona o sinal negativo ao BigNumber.
somaBN	Esta função calcula a soma entre 2 BigNumbers. Utiliza 3 funções auxiliares *somaBNAux*, *somaBNAux1* e *subtrairBNAbs* de forma a calcular a soma de forma correta para todos os casos possíveis.
somaBNAux	Esta função soma as cabeças das duas listas recebidas e chama-se recursivamente para os restos das listas, transportando eventuais carrys. É usada na soma entre dois números negativos ou entre dois números positivos.
somaBNAux1	Esta função é usada para a soma de dois BigNumbers com sinais opostos. Recebe no primeiro argumento o número com maior valor absoluto visto que o resultado final terá o sinal desse. Chama a função *subtrairBNAbs* para fazer a diferença entre os dois números.
subtrairBNAbs	Esta função faz a diferença entre dois valores absolutos de BigNumbers, sendo o primeiro argumento o maior dos dois. É usada para a soma entre BigNumbers de sinais simétricos.
subBN	Esta função calcula a subtração através da soma entre primeiro número e o simétrico do segundo.

Função	Descrição
mulBN	Esta função calcula a multiplicação entre 2 BigNumbers. Utiliza 3 funções auxiliares *mulBNAux*, *mulBNAux1* e *escalaBN*.
mulBNAux	Função auxiliar, que recorre à chamada recursiva de modo a multiplicar cada elemento do primeiro BigNumber recebido pelo segundo BigNumber. Para cada número do primeiro BigNumber, exceto o primeiro, acrescentamos posteriormente um 0 à cabeça do segundo de modo a realizar a soma final corretamente, uma vez que o 0 ficará à direita após o reverter da lista.
mulBNAux1	Esta função corrige possíveis elementos da lista que tenham mais que um algarismo devido às multiplicações anteriores, transportando eventuais carrys para o resto da lista.
escalaBN	Função auxiliar que realiza a multiplicação de um número pelo BigNumber.
divBN	Esta função calcula a divisão entre 2 BigNumbers. Utiliza uma função auxiliar *divBNAux*.
divBNAux	Esta função vai subtraindo o dividendo pelo divisor, acrescentando uma unidade ao quociente sempre que acontece a subtração, preservando o resto no segundo elemento do par.
safeDivBN	Calcula a divisão de dois BigNumbers e deteta a divisão por 0, através do uso de um *monad* do tipo Maybe.

Fib.hs

Função	Descrição
fibRec	Esta função calcula o enésimo número de *Fibonacci* de forma recursiva.
fibLista	Esta função calcula o enésimo número de *Fibonacci* recorrendo a programação dinâmica.
fibListaInfinita	Esta função calcula o enésimo número de *Fibonacci* recorrendo a listas infinitas.
fibRecBN	Esta função calcula número de *Fibonacci*, do BigNumber recebido, de forma recursiva.
fibListaBN	Esta função calcula número de *Fibonacci*, do BigNumber recebido, recorrendo a programação dinâmica.
(@@)	Esta função permite obter o enésimo elemento de uma lista, usando um BigNumber como argumento.
listRange	Esta função cria uma lista de BigNumber desde o primeiro argumento até ao segundo argumento recebido.
fibListaInfinitaBN	Esta função calcula número de *Fibonacci*, do BigNumber recebido, recorrendo a listas infinitas.

> Estratégias utilizadas na implementação das funções da alínea 2

Na alínea 2, decidimos representar os BigNumber como uma lista de Int usando o type. Assim, de forma a garantir que só trabalhávamos com BigNumbers nas funções implementadas, criamos uma função verificaBN, que só aceita números de 0 a 9 em cada elemento da lista e apenas o primeiro elemento pode ter valores negativos. Assim, chamamos esta função em todas as funções principais do nosso trabalho, exceto na *scanner* e no *output* que fazem a verificação enquanto convertem. Para facilitar os desenvolvimentos das funções foi seguida a sugestão de reverter previamente o BigNumber.

Para o cálculo da soma de dois BigNumbers a estratégia utilizada foi verificar os sinais de ambos os argumentos. Se tiverem o mesmo sinal vamos somando pela parte menos significativa e transportando os carrys, colocando o seu sinal no final. Se tiverem sinais opostos, fazemos a subtração do número com maior valor absoluto pelo que tem menor valor absoluto, utilizando uma estratégia similar à anterior com os carrys, e mantemos o sinal do maior.

Para o cálculo da subtração de dois BigNumbers somamos o primeiro BigNumber com o simétrico do segundo.

Para calcular a multiplicação entre dois BigNumbers, multiplicamos cada elemento do primeiro BigNumber recebido pelo segundo BigNumber, somando os resultados obtidos. Para isto, introduzimos um 0 à cabeça sempre que avançamos uma casa decimal, uma vez que ficaria à direita após o reverter da lista.

Para calcular a divisão entre dois BigNumber, a estratégia adotada foi ir subtraindo o dividendo pelo divisor enquanto for possível e ir guardando o quociente num contador. A divisão segura passa apenas por verificar se o divisor é 0.

> Exercício 4

Função	Número	Tempo médio de execução (s)
fibRec	30	3.404
fibRecBN	30	23.182
fibRec	5000	∞
fibRecBN	5000	∞

Função	Número	Tempo médio de execução (s)
fibLista	5000	0.15
fibListaBN	5000	51.505
fibListaInfinita	5000	0.01
fibListaInfinitaBN	5000	6.145

A partir do número 30, a função recursiva de Fibonnaci começa a ter valores de execução muito elevados, principalmente utilizando BigNumbers. O mesmo acontece para as restantes funções do Fibonacci de BigNumbers a partir do número 5000.

Em relação aos tipos das funções, quando é utilizado o Fibonacci com o argumento do tipo Int, a sua precisão não permite calcular o número correto de Fibonacci a partir de 92, enquanto que tanto os argumentos do tipo Integer e BigNumbers, continuam a calcular corretamente o valor de Fibonacci de 5000 por exemplo, sendo o segundo muito mais lento do que o primeiro.