Trabalho Prático 1 - G08_08

Representação interna dos polinómios

- No nosso trabalho, criámos dois novos tipos para a representação de polinómios. Um tipo para monómios (`Nomial`), e um tipo para polinómios (`Polynomial`), que consiste numa lista de monómios.
- Um monómio (`Nomial`) é do tipo `(Int, [(Char, Int)])`, ou seja, é constituído por um um par em que o primeiro elemento é o coeficiente do monómio representado por um inteiro.
- E em que o segundo elemento do par é uma lista de variáveis e expoentes, suportando assim, múltiplas variáveis. Essa lista de váriaveis é constituída por pares do tipo `(Char,Int)` em que o primeiro elemento representa uma variável (por exemplo: 'x', 'y' ou 'z') e que a segunda variável representa o expoente da variável.
- Decidimos representar a estrutura principal do monómio como um par porque todos os monómios podem ter apenas, no máximo, um coeficiente (`Int`) e variáveis (`Char`) com expoente (`Int`), tendo um tamanho fixo de dois componentes que podem ser facilmente separados.
- De forma a podermos representar monómios do tipo `2*x^2*y^3` em vez de termos uma estrutura do tipo `(Char, Int)` para as variáveis e os seus expoentes, representámos essa estrutura como uma lista de pares `[(Char, Int)]` para podermos representar múltiplas variáveis juntas dos seus expoentes associados.
- Os polinómios (`Polynomial`) são um conjunto de monómios somados uns aos outros pelo que escolhemos representá-los pelo tipo `[Nomial]`, uma lista de monómios.

Funcionalidades

Todas as funções principais entregam resultados na forma normalizada e no formato de `String`. E além disso, todas as funções principais recebem argumentos do tipo `String` que são convertidos para o tipo `Polynomial` para poderem ser afetuadas as operações.

Adicionar polinómios

- Uma das funcionalidades é a adição de polinómios através da função `add` que adiciona dois polinómios, monómio a monómio, recursivamente.
- A função verifica, para cada monómio de um polinómio, se as variáveis e os expoentes de cada monómio do outro polinómio são iguais e, se forem, soma os coeficientes e anexa o resultado ao polinómio resultante.
- Se não encontrar nenhum monómio com as mesmas variáveis, o monómio é anexado ao polinómio resultante.

Multiplicar polinómios

- Temos também a multiplicação de dois polinómios através da função `multiply`. A função multiplica polinómios, monómio a monómio, através de funções auxiliares: multiplica coeficientes, junta variáveis se forem diferentes e soma expoentes se as variáveis forem iguais.

Derivar polinómios em ordem a uma variável

- Para a derivação, implementámos derivação de polinómios em ordem a uma variável à escolha através da função `derivateInOrderTo`. Esta função, recebe como argumentos um polinómio (`String`), e um `Char` que representa a variável pela qual o queremos derivar, e através de funções auxiliares deriva monómio a monómio recursivamente.
- Por exemplo, derivando um monómio com apenas uma variável em ordem a `x`, caso o monómio não tenha a variável `x` será 0, se tiver a variável então multiplica-se o coeficiente pelo expoente e subtrai-se o expoente em 1.
- Caso tenha mais do que uma variável, temos de ver recursivamente as variáveis com outro caso base. Se a variável analisada for do tipo `x` multiplica-se o coeficiente e subtrai-se ao expoente novamente, mas se não for, em vez de ficar as variáveis e expoentes ficarem a 0, a variável e expoente ficam iguais e continua-se a percorrer o monómio recursivamente para as restantes variáveis.

Normalizar polinómios para a forma normal

A função `normalize` recebe um polinómio e retorna-o, em string, na forma mormal. Para tal, realiza algumas operações, em ordem, com o uso de funções auxiliares:

1. Remove variáveis de monómios com expoente 0.
2. Remove monómios com coeficiente 0.
3. Soma, em cada monómio, variáveis iguais. Ex: `2*x^2*x^3` passa para `2*x^5`.
4. Ordena, em cada monómio, as variáveis alfabeticamente, com selection sort.
5. Ordena, em cada monómio, as variáveis por ordem do expoente, com insertion sort.
6. Soma monómios com as mesmas variáveis e expoentes.
7. Ordena os monómios alfabeticamente, em relação à primeira variável, com selection sort.
8. Ordena os monómios, em relação ao expoente da primeira variável, com insertion sort.

No final, o polinómio resultante é retornado no formato de `String`.

Passar output para string

Todos os resultados das funções anteriores são do tipo `Polynomial` mas depois são passadas para o tipo `String` para uma leitura mais fácil. Essa transformação é executada através da função `stringify`.

- A função verifica que se o polinómio for uma lista vazia então é o valor 0.
- Para cada monómio passa para string os coeficientes e se forem `1` ou `-1` oculta-os.
- Para cada monómio une os expoentes e as variáveis por um sinal de `^` que representa o sinal de elevado, e os outros valores por um sinal `*` que representa a multiplicação.
- Depois junta os vários monómios já em string por strings com sinais de `+` ou de `-` consoante se o coeficiente é positivo ou negativo.

Receber input de string

Para converter a string de input para polinómio usámos a função `stringParsing` e utilizámos a seguinte estratégia:

1. Removemos todos os espaços da string.
2. Trocamos todas as instâncias de `-` para `+-`, isto facilita a divisão da string.

3. Dividimos a string nos caracteres `+`, ficando com uma lista de strings que representa cada monómio.
    - Ex: `"2*x^2+3*y-1"` passa a `["2*x^2", "3*y", "-1"]`.
4. Separamos o coeficiente e as variáveis de cada monómio, por um espaço. Se um monómio apenas tiver coeficiente, um `_` é adicionado na posição em que seria representado a variável e o expoente para facilitar passos seguintes.
5. Dividimos as strings nos espaços, passando de:
    - `["2*x^2", "3*y", "-1"]` para `[["2*", "x^2"], ["3*", "y"], ["-1*", "_"]]`
6. Finalmente, criamos cada monómio, convertendo para `Int` e somando coeficientes, se necessário, de `2*3` para `6`, por exemplo. Analisamos as variáveis, separando nos caracteres `*` e depois nos `^`, guardando numa lista de pares `[(Char, Int)]`. E se, em vez das variáveis e coeficientes tiver um `_`, é passado uma lista vazia `[]`. Sendo asssim a `String`:
    - `[["2*", "x^2"], ["3*", "y"], ["-1*", "_"]]` passa a `[(2, [('x', 2)]), (1, [('y', 1)]), (-1, [])]`

Exemplos de utilização:

Todos os inputs necessitam de ter coeficientes e variáveis separados por `*`.
As variáveis e coeficientes necessitam de ser separadas por `^`.
As variáveis têm de ser do tipo `Char` e os coeficientes do tipo `Int`.
Coeficientes podem ser positivos e negativos. Expoentes têm de ser positivos.

Alguns exemplos de utilização mais complexos que utilizam todas as capacidades das funcionalidades implementadas são os seguintes:

Adicionar:

```
Tipo: add :: String -> String -> String
Input: add "7*y^2 + 3*y*x + 5*z" "0*x^2 - 2*y + 5*z + y - 3*y^2"
Resultado: "4*y^2 + 3*x*y - y + 10*z"
```

Multiplicar:

```
Tipo: multiply :: String -> String -> String
Input: multiply "2*y^2 + 4*y*x^2 + 3*z" "0*x^2 - 9*y + 3*z + y - 3*y^2"
Resultado: "-6*y^4 - 16*y^3 - 12*y^3*x^2 + 12*x^2*y*z - 32*x^2*y^2 - 3*y^2*z +
9*z^2 - 24*y*z"
```

Derivar:

```
Tipo: derivateInOrderTo :: String -> Char -> String
Input: derivateInOrderTo  "-6*y^4 - 16*y^3 - 12*y^3*x^2 + 12*x^2*y*z - 32*x^2*y^2
- 3*y^2*z + 9*z^2 - 24*y*z" 'x'
Resultado: "-24*y^3*x - 64*y^2*x + 24*x*y*z"
```

Normalizar:

```
Tipo: normalize :: String -> String
Input: normalize "0*x -3*x + 2*z^3*x^2 -2*x - 3*y^2"
Resultado: "2*z^3*x^2 - 3*y^2 - 5*x"
```

Membros:

- André Morais - up202005303
- Lucas Sousa - up202004682