Trabalho 1

Introdução

O objectivo deste trabalho é fundamentalmente que os alunos exercitem os conhecimentos lecionados na disciplina relativamente à implementação de tipos de dados abstratos (descritos por especificações algébricas) e pratiquem a manipulação de listas ligadas.

O trabalho deve ser realizado em grupos de dois e é cotado para 10 valores, valendo 50% da nota da componente de projecto. Há um bónus de 2.5 valores para um conjunto de tarefas adicionais.

Problema

Um espaço vetorial é uma coleção de objetos chamados **vetores** que podem ser somados e multiplicados por escalares. Um espaço vetorial muito popular é o dos vetores Euclideanos ilustrado na figura abaixo. Frequentemente estes espaços estão equipados também com um produto interno. Apesar dos espaços vetoriais mais populares serem definidos sobre os reais, um espaço de vetores pode ser definido sobre qualquer corpo.

Figura 1: Vetores Euclideanos com 2 dimensões: $v = \langle v_1, v_2 \rangle$. À esquerda é ilustrada a soma dos vetores v e w e à direita a multiplicação de um vetor w pelo escalar 2.

Os espaços vetoriais são os objetos de estudo da Álgebra Linear pelo que este semestre irão certamente aprender muitas coisas sobre estes. Em AED, e neste trabalho em particular, estamos interessados nos vetores como valores de um tipo de dados abstrato Vetor que permite resolver problemas em áreas tão diferentes como a computação científica, o processamento de imagens e a extração de informação de textos.

Porque se pretende usar vetores definidos sobre um qualquer corpo, especificou-se este ADT através de uma especificação parametrizada Vector[Field]. O parâmetro Field define um tipo de dados abstrato cujos valores são os elementos de um corpo. As especificações Vector[Field] e Field são apresentadas em anexo e estão também incluídas no material fornecido.

Em muitas aplicações deste ADT são usados vetores de grandes dimensões e com muito poucas componentes não nulas, designados por **vetores esparsos**. Um exemplo de um vetor esparso num espaço vetorial definido sobre os reais é

$$\langle 0.0, 0.0, 2.3, 0.0, 0.0, 5.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 \rangle$$

Este vetor de dimensão 20, tem apenas as componentes 3 e 6 não nulas (correspondente a apenas 10%). Assim, é importante dispor de implementações que representem e manipulem de forma eficiente este tipo de vetores.

O trabalho consiste em programar uma classe genérica SparseVector<F> que seja uma implementação mutável para o tipo de dados abstrato Vector[Field] baseada em listas ligadas, eficiente para representar vetores esparsos. Para isso cada nó da lista deve corresponder a uma componente não nula do vetor e deve portanto guardar o índice da componente do vetor e o seu valor. É possível que venha a concluir que é conveniente ter os nós da lista ordenados de alguma forma.

- O API fornecido pela classe SparseVector<F> deve ser compatível com o refinamento definido em VectorToSparseVector.rfn.
- A classe SparseVector<F> deve ainda implementar Iterable<Pair<Integer,F>> fornecendo um iterador que dá, um a um, todos os índices e valores dos componentes não nulos do vetor por ordem crescente do índice. Isto significa que, no exemplo do vector acima, o iterador vai primeiro dar um objeto Pair correspondente a $(3, 2.3)$ e depois um objeto Pair correspondente a $(6, 5.1)$.
- A classe SparseVector<F> deve implementar o interface Cloneable com um método **public** SparseVector<F> clone() que devolve um vetor que é uma cópia (profunda) do vetor original.
- A classe SparseVector<F> deve ter uma implementação apropriada do método **public boolean** equals(Object o).
- A classe SparseVector<F> deve incluir na sua documentação informação relativa ao tempo de execução assintótico de cada operação (que deve obviamente procurar minimizar).

Note que a classe SparseVector<F> deve poder ser usada pelo programa cliente fornecido sem qualquer modificação do mesmo. Este programa, definido na classe SparseVectorRandomTestWithDouble, constrói um vetor esparso de números reais e chama aleatoriamente as várias operações do ADT sobre este vetor. Para ganhar confiança relativamente à correção da sua implementação deve fazer a execução várias vezes deste programa sujeita à monitorização suportada pela ferramenta ConGu (ou seja, deve correr o programa com Run As > ConGu Monitoring). O trabalho será maioritariamente avaliado de acordo com a correção da implementação fornecida, a qual será aferida automaticamente recorrendo a esta ferramenta.

Tarefas adicionais. Os 2.5 valores de bónus são atribuídos à implementação de uma pequena aplicação de extração de informação de documentos de texto baseada em vetores com o tipo SparseVector<FDouble>.

Face a uma pesquisa definida por um conjunto de palavras (como as suportadas pelo Google), a aplicação tem de identificar, numa coleção de documentos de texto previamente processados, qual o documento ou documentos mais relevantes. Para facilitar esta tarefa é fornecido um esqueleto da classe que define a aplicação (SearchDocs) e documentos de texto para testar esta aplicação.

Considera-se que se tem uma sequência $w = w_1 w_2 \dots w_N$ com as palavras relevantes para responder às pesquisas. A informação contida em cada documento d de um conjunto D é representada por um vector v^d de dimensão N onde v_i^d representa o "peso" da palavra w_i no documento d. O valor deste peso é calculado da seguinte forma:

$$v_i^d = f_i^d * \log \frac{|D|}{|\{d' \in D : f_i^{d'} > 0\}|}$$

onde f_i^d representa a frequência da palavra w_i no documento d e $|C|$ representa o número de elementos num conjunto C.

Uma pesquisa, definida por um conjunto de palavras Q, é também representada por um vector q de dimensão N onde q_i é 1.0 se a palavra w_i pertence a Q e 0.0 no caso contrário.

Para determinar os documentos em D mais relevantes face à pesquisa Q calcula-se o coseno entre o vector q e cada um dos vectores v^d. Este valor pode ser calculado da seguinte forma:

$$cos\ \theta^d = \frac{v^d \cdot q}{||q||\ ||v^d||}$$

onde $\cdot$ representa o produto interno e $||v|| = \sqrt{\sum_{i=1}^{N} v_i^2}$. Selecionam-se os documentos d com $cos\ \theta^d$ mais elevado, acima de um determinar limiar pré-definido.

Entrega

Não esqueça que:

- Todas as classes a entregar devem estar convenientemente documentadas através de comentários em *javadoc*.
- O código que desenvolver deve seguir, de forma coerente, um estilo de codificação (consulte o documento sobre o estilo na página da disciplina assim como o livro *The elements of Java Style*).

O quê? Um zip com as classes Java desenvolvidas.

Quando? Até às 11:59 de 19 de Abril de 2015.

Como? No moodle.

```
specification Vector[Field]
  sorts
    Vector[Field]
  constructors
    newVector: int Field -->? Vector[Field];
    set: Vector[Field] int Field -->? Vector[Field];
  observers
    dim: Vector[Field] --> int;
    get: Vector[Field] int  -->? Field;
    sumV: Vector[Field] Vector[Field] --> Vector[Field];
    scalarProd: Vector[Field] Field  --> Vector[Field];
    dotProd: Vector[Field] Vector[Field] -->? Field;
  others
    zero: Vector[Field] --> Field;
    density: Vector[Field] --> int;
    symVector: Vector[Field] --> Vector[Field];
    domains
    i: int; f: Field; V, V1: Vector[Field];
    newVector(i,f) if i>0;
    set(V, i, f) if i>0 and i<= dim(V);
    get(V, i) if i>0 and i<= dim(V);
    dotProd(V, V1) if dim(V)=dim(V1) and zero(V)=zero(V1);
  axioms
    i, j, n: int; z, f, f1, f2: Field; V, V1: Vector[Field];

    dim(newVector(n,f)) = n;
    dim(set(V, i, f)) = dim(V);

    get(newVector(n,z), j) = z;
    get(set(V, i, f), j) = f when i=j else get(V, j);

    scalarProd(newVector(n,z), f) = newVector(n,z);
    scalarProd(set(V, i, f1), f) = set(scalarProd(V, f), i, prod(f1,f));

    sumV(newVector(n,z), V) = V;
    sumV(set(V1, i, f1), V) = set(sumV(V1, V), i, sum(f1,get(V, i)));


    dotProd(newVector(n,z), V) = z;
    dotProd(set(V1, i, f1), V) = sum(prod(f1,get(V,i)), dotProd(V1,V))
        if get(V1,i) = zero(V);

    zero(newVector(n,z)) = z;
    zero(set(V, i, f)) = zero(V);

    //axioms that capture that the zero(V) is indeed the zero of the field
    sum(zero(V),f) = f;
    zero(V) = sum(f1,f2) if f2 = sym(f1);

    density(newVector(n,f)) = 0;
    density(set(V, i, f)) = 1 + density(V)
        if get(V,i) = zero(V) and f!=zero(V);

    symVector(newVector(n,f)) = newVector(n,f);
    symVector(set(V, i, f)) = set(symVector(V), i, sym(f));

    set(V, i, f) = V if f = zero(V) and get(V, i) = zero(V);
    set(set(V, i, f), j, f1) = set(V, i, f1) when i = j
         else set(set(V, j, f1), i, f);
end specification
```

```
specification
  sorts
    Field
  others
    sum: Field Field --> Field;
    prod: Field Field --> Field;
    sym: Field  --> Field;
    inv: Field  -->? Field;
  axioms
    f, f1, f2, f3: Field;

    //Associativity of addition and multiplication
    sum(sum(f1,f2),f3) = sum(f1, sum(f2,f3));
    prod(prod(f1,f2),f3) = prod(f1, prod(f2,f3));

    //Commutativity of addition and multiplication
    sum(f1,f2) = sum(f2, f1);
    prod(f1,f2) = prod(f2, f1);

    //multiplication distributes over addition
    prod(sum(f1,f2),f3) = sum(prod(f1,f3), prod(f2,f3));

end specification
```