Algoritmos e Estruturas de Dados
Exame de Época Normal
Departamento de Informática, Universidade Nova de Lisboa
16 de Janeiro de 2012

Atenção: O Anexo ao exame contém interfaces e classes que lhe poderão ser úteis.

1. Considere a árvore AVL apresentada na Figura 1. Em cada nó está apenas representada a chave do mesmo.

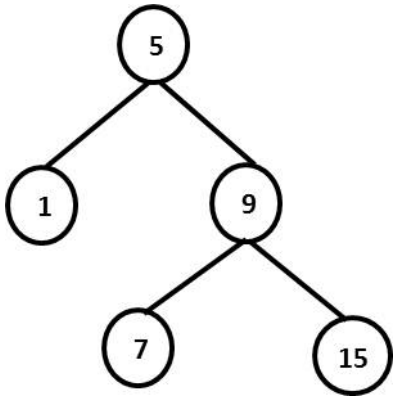

Figura 1

   a) Desenhe a árvore AVL resultante de inserir um elemento com a chave 10 na árvore apresentada;
   b) Desenhe a árvore AVL resultante de remover o elemento com a chave 5 da árvore apresentada na Figura 1. Atenção que a remoção deve ser feita da **árvore original,** representada na Figura 1.

2. Considere a Classe BinarySearchTree apresentada nas aulas. Desenvolva um construtor para a mesma classe que receba, como parâmetro de entrada, uma árvore BinarySearchTree existente, e crie uma nova árvore, em tudo igual à recebida. O construtor deve criar novos nós para a árvore que representa, que serão cópias dos nós da árvore original. Calcule a complexidade temporal do método que desenvolver, no caso esperado, justificando. **Atenção**: não é permitido usar o método `clone()`.

```
public class BinarySearchTree<K extends Comparable<K>, V>
    implements OrderedDictionary<K,V> {

   //A raiz da árvore
   protected BSTNode<K,V> root;

   //Número de entradas da árvore
   protected int currentSize;

   public BinarySearchTree(BinarySearchTree<K,V> otherTree);

}
```

3. Implemente a classe `FusionIterator<K>` que deverá receber dois iteradores de sequências de chaves ordenadas e permitir a iteração ordenada do conjunto das duas sequências originais. A classe irá implementar o interface `Iterator<K>` que conhece das aulas de AED e deve assumir que os elementos manipulados pelo iterador implementam o interface `Comparable<K>`.
   Exemplo: Considere duas sequências ordenadas de números inteiros, S_1 e S_2 com os seguintes elementos: S_1 = {0, 2} e S_2 = {1, 3, 4}. Os elementos devolvidos pelo FusionIterator das sequências S_1 e S_2 sairão pela seguinte ordem: 0, 1, 2, 3, 4.

   Defina as variáveis de instância da classe e desenvolva todos os métodos necessários para a completa implementação da mesma, inclusive métodos auxiliares de que necessite. Calcule as complexidades temporais dos métodos que implementar, no melhor caso, no pior caso e no caso esperado, justificando. Para este cálculo assuma que todos os métodos dos iteradores passados no construtor do FusionIterator são constantes, assim como o método compareTo associado ao tipo K.

```
public class FusionIterator<K extends Comparable<K>>
                              implements Iterator<K> {

   //Construtor da classe, recebe dois iteradores de elementos
   //comparáveis
   public FusionIterator(Iterator<K> it1, Iterator<K> it2);

   //Devolve true se a iteração ainda não tiver chegado ao fim
   public boolean hasNext();

   //Devolve o próximo elemento na iteração. Deve utilizar o método
   //compareTo do Interface Comparable para comparar os elementos
   //manipulados
   public K next() throws NoSuchElementException;

   //Reinicia a iteração. Depois da chamada ao método rewind(), o método
   //next() deve devolver o primeiro elemento da iteração
   public void rewind();
}
```

4. Aproximam-se as eleições para o conselho de gestão do clube de Bairro "Os amigos da Troika". É preciso organizar a votação de forma a podermos saber o número de votos atribuídos a cada sócio. O clube valoriza os sócios mais antigos atribuindo, a cada sócio, tantos votos quanto o número de anos de filiação do mesmo no clube e adicionando um (1) a esse número. Relativamente a cada sócio sabemos sempre o seu nome e o ano em que se filiou no clube. Se soubermos qual o ano em que estamos, é simples saber o número de votos de um sócio. Por exemplo, o "Ti Manel" entrou para o clube em 1982. Isto significa que em 2012, ele terá direito a 31 votos. O "Toni Rodinhas" só tem direito a 11 votos porque entrou para o Clube em 2002. Já a bebé dele, a "Sissi" que nasceu no primeiro dia de 2012, no feriado, já é sócia, com direito a 1 voto **(continua na página 3)**.

(continuação da página 2 – pergunta 4)

É preciso desenvolver um sistema de suporte às eleições do clube. Pensámos num tipo abstrato de dados para suportar a resolução do problema, e pedimos-lhe que pense na melhor implementação para o mesmo. O TAD em causa é apresentado no interface Java abaixo:

```
public interface TroikaFriends {

   //Devolve o ano corrente
   int currentYear();

   //Adiciona um novo sócio ao clube, verificando que não existe já um
   //sócio com o mesmo nome. O ano de filiação do sócio é o ano em curso
   void addPartner(String partnerName);

   //Adiciona um novo sócio ao clube, verificando que não existe já um
   //sócio com o mesmo nome. O ano de filiação do sócio é registryYear
   //Requires: registryYear <= currentYear()
   void addPartner(String partnerName, int registryYear)
        throws InvalidYearException;

   //Actualiza o ano em curso para o ano seguinte ao corrente
   void newYear();

   //Devolve o número de votos atribuídos ao sócio partnerName
   int partnerVotes(String partnerName);

   //Devolve o número de sócios do clube
   int numberOfPartners();

   //Devolve a soma de todos os votos atribuídos ao total de sócios
   //do clube
   int numberOfCurrentVotes();
}
```

Parta do princípio de que o construtor da classe que implementar o interface receberá, como parâmetro, o ano corrente.

Baseando-se na descrição apresentada e no interface fornecido, explicite detalhadamente as estruturas de dados mais adequadas para implementar o mesmo, descreva brevemente como implementaria as sete operações e calcule as suas complexidades temporais, no caso esperado, justificando.

5. Pedimos-lhe que considere a implementação de um Projeto de Revista Online, pensando no Sistema de Gestão de conteúdos. Este sistema irá gerir os autores dos textos publicados na revista, assim como a informação associada aos próprios textos. Os textos em si estarão disponíveis num repositório pelo que, para se ler um determinado texto, utilizamos o URL onde este está localizado. Assim, o sistema irá fazer a gestão dos *Autores* dos textos, cuja informação inclui Código de autor, Nome de autor, Email e Número de telefone. Relativamente aos *Textos*, estes têm um Código de Texto, Título, um Autor e um URL. Assuma que não existem nomes de Autores repetidos e que os títulos dos textos de um autor são únicos. O Sistema deverá implementar as seguintes operações:
   a) *Inserir um Autor*, recebendo a seguinte informação: Código de Autor, Nome de Autor, Email e Número de telefone. A inserção só tem sucesso se o código do autor ainda não existir no sistema;
   b) *Inserir um Texto*, que é composto de: Código de Texto, Título, Código de Autor do Texto e URL. A Inserção só tem sucesso se o código de autor já existir no sistema e se o código do texto não existir no sistema;
   c) *Aceder ao URL de um texto*, dando o código do texto. Esta operação só tem sucesso se o código do texto existir no Sistema;
   d) *Remover Texto,* dando o código do texto. Esta operação só tem sucesso se o código do texto existir no sistema;
   e) *Listar os Títulos de todos os textos de um determinado autor*, dando o **nome do autor**. A listagem deve ser apresentada por ordem alfabética do título do texto. Esta operação só tem sucesso se o autor em causa fizer parte do sistema.

   Espera-se que o número de autores a contribuir para a revista seja da ordem dos milhares. O número de textos armazenados será da ordem das centenas de milhar. Como esta é uma revista disponível na Internet, é possível atualizar o seu conteúdo diariamente, sabendo-se que um autor poderá contribuir diariamente para a mesma, sendo guardados todos os textos dos últimos 4 anos.

   Com base nesta especificação, explicite detalhadamente as estruturas de dados mais adequadas para implementar esta aplicação, descreva sumariamente os algoritmos para efetuar as cinco operações (enumeradas de (a) a (e)) e calcule (justificando) as suas complexidades temporais, no caso esperado.

Anexo – Interfaces e Classes de Apoio

```
public interface Entry<K,V>{
   K getKey( );
   V getValue( );
}

public interface Comparable<T>{
   int compareTo( T object );
}

public interface Iterator<E> {
    boolean hasNext( );
    E next( ) throws NoSuchElementException;
    void rewind( );
}

public interface Dictionary<K,V>{
   boolean isEmpty( );
   int size( );
   Iterator<Entry<K,V>> iterator( );
   V find( K key );
   V insert( K key, V value );
   V remove( K key );
}
```

```
public interface OrderedDictionary<K extends Comparable<K>, V>
        extends Dictionary<K,V>{

   Entry<K,V> minEntry( ) throws EmptyDictionaryException;

   Entry<K,V> maxEntry( ) throws EmptyDictionaryException;
}


class BSTNode<K,V>{

   .........
   public BSTNode( K key, V value, BSTNode<K,V> left,
                    BSTNode<K,V> right );
   public BSTNode( K key, V value );
   public EntryClass<K,V> getEntry( );
   public K getKey( );
   public V getValue( );
   public BSTNode<K,V> getLeft( );
   public BSTNode<K,V> getRight( );
   public void setEntry( EntryClass<K,V> newEntry );
   public void setEntry( K newKey, V newValue );
   public void setKey( K newKey );
   public void setValue( V newValue );
   public void setLeft( BSTNode<K,V> newLeft );
   public void setRight( BSTNode<K,V> newRight );
   public boolean isLeaf( );
}
```