Algoritmos e Estruturas de Dados
Exame de Recurso
Departamento de Informática, Universidade Nova de Lisboa
9 de Janeiro de 2019

Atenção: Os Anexos ao exame poderão ser-lhe úteis.

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

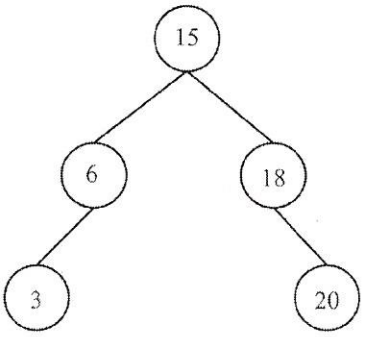

Figura 1

i) Desenhe a árvore AVL resultante de inserir o elemento com a chave 5 na árvore apresentada;

ii) Desenhe a árvore AVL resultante de remover o elemento com chave 15 na árvore apresentada na Figura 1. **Atenção** que a remoção deve ser feita na **árvore original,** representada na Figura.

1. b) Considere a tabela de dispersão aberta apresentada na Figura 2. Esta tabela constitui uma implementação do tipo abstrato de dados (TAD) Dicionário (`Dictionary<K,V>`), tal como apresentado nas aulas de AED. A tabela foi criada para conter no máximo 7 entradas (*maxSize*) e é constituída por um vetor de 7 posições (*arraySize*). Em cada posição do vetor, existe uma lista duplamente ligada, ordenada crescentemente pela chave das entradas, que guarda as colisões (entradas cujo resultado da aplicação da função de dispersão à chave da entrada é o mesmo). A função de dispersão da tabela apresentada é definida da seguinte forma:

```
// Returns the hash value of the specified key.
 protected int hash( int key )
 {
     return Math.abs( key ) % arraySize;
  }
```

Neste momento, a tabela contém seis (6) entradas (*currentSize*). As entradas são do tipo `Entry<Integer, Character>`. A posição de cada entrada é encontrada pela aplicação da

função de dispersão hash à chave da entrada. Assim, a entrada (9,'X') foi guardada na lista existente no índice 2 e a entrada (6,'13'), na lista pertencente ao índice 6.

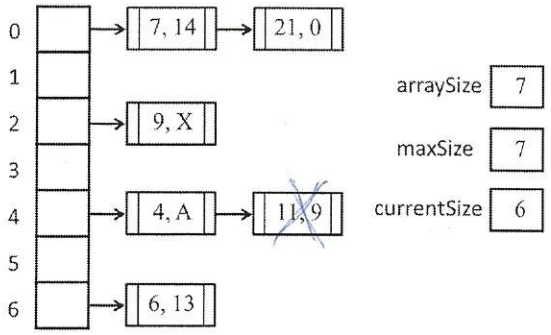

Figura 2

Dada a tabela de dispersão aberta apresentada na **Figura 2**, desenhe a tabela resultante de executar todas as operações pedidas abaixo, pela ordem apresentada (desenhe apenas a tabela final):

- Remoção da entrada com chave 13; 6
- Remoção da entrada com chave 11; 5
- Inserção da entrada (9,'9'). 5

Justifique o resultado de cada operação. Com a tabela, apresente também os valores finais das variáveis de instância arraySize, maxSize e currentSize.

2. Considere a Classe `BinarySearchTree` apresentada nas aulas (ver anexo B). Pedimos-lhe que desenvolva um método **recursivo** que conta, e devolve, o número de nós existentes na subárvore esquerda e na subárvore direita da raiz da árvore. Estes dois valores serão devolvidos numa `Entry<Integer,Integer>`. Apresenta-se abaixo o cabeçalho do método principal, mas poderá criar métodos auxiliares para desenvolver a sua solução. Calcule ainda a complexidade temporal do mesmo método, no pior caso e no caso esperado, justificando. **Apenas as soluções recursivas serão avaliadas.**

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

  // a raiz da árvore.
  protected BSTNodeB<K,V> root;
  // Número de entradas da árvore.
  protected int currentsize;
      ...

  // Método devolve o número de nós nas subárvores esquerda e direita da raiz,
  // numa Entry<Integer, Integer>
  protected Entry<Integer,Integer> subtreesSize(){

    ...
   }
      ...
}
```

(O exame continua na página 3)

3. Considere a implementação de lista duplamente ligada apresentada nas aulas de AED (Classe `DoublyLinkedList<E>`). Implemente um método para esta classe, denominado `removeAllElements()`, que permite a remoção de todos os nós da lista que contenham um determinado elemento, passado como argumento ao método.

Requisitos de implementação:

- O método deverá executar operações de remoção sobre a lista atual e não fazer novas inserções, nem criar listas adicionais.
- Poderá apoiar a sua implementação **apenas nos métodos de remoção disponíveis na classe `DoublyLinkedList`** (ver Anexo B), com algumas exceções. **Não deverá utilizar os seguintes métodos** na sua implementação:
  - `remove(int position)`,
  - `remove(E element)`.
- A sua implementação deverá ser estritamente linear (complexidade O(n)) no pior caso.

Apresenta-se abaixo o cabeçalho do método em questão. Para facilitar a implementação, poderá assumir que o tipo E é comparável.

```
public class DoublyLinkedList<E extends Comparable<E>>
             implements List<E>{

   // Nó cabeça da lista.
   protected DListNode<E> head;

   // Nó cauda da lista.
   protected DListNode<E> tail;

   // Número de elmentos da lista.
   protected int currentSize;

   // Remove todos os nós da lista que contenham element.
   public void removeAllElements(E element) {
   ...
   }

...
}
```

(O exame continua na página 4)

4. O Tipo Abstrato de Dados (TAD) `PortugueseDictionary` guarda palavras da Língua Portuguesa e o respetivo significado. O TAD inclui operações que permitem inserir e remover palavras, consultar e atualizar o significado das palavras e listar as palavras que se iniciam por uma letra específica, ordenadas alfabeticamente. O TAD é apresentado no interface abaixo:

```
public interface PortugueseDictionary {

   //Associa o significado (meaning) a word no dicionário.  Se word já
   //existir no dicionário, altera o significado e devolve o antigo.
   String addWord(String word, String meaning);

   //Remove word do dicionário.
   void removeWord(String word) throws WordDoesNotExist;

   //Devolve o significado de word, guardado no dicionario.
   String getMeaning(String word) throws WordDoesNotExist;

   //Devolve uma lista ordenada de todas as palavras guardadas no dicionário
   //que se iniciam por letter. Esta listagem deve ser efetuada por ordem
   //alfabética das palavras.
   Iterator<String> listWordsByInitial(char letter) throws NoWords;
}
```

Relativamente ao interface apresentado, reflita sobre a melhor implementação para o mesmo e responda às seguintes questões **(separadamente)**:

4.1 Proponha **Estruturas de Dados** (EDs) e **variáveis de instância** para apoiar a implementação de todas as operações que fazem parte do TAD.

Descreva **cada uma das EDs escolhidas individualmente**, incluindo sempre a seguinte informação:

- **Tipo Abstrato de Dados (TAD) genérico;**
- **Estrutura de Dados (ED) genérica que é usada para implementar o TAD;**
- **Tipo de dados (do problema) a guardar dentro da ED (Tipo do Elemento (E); ou tipos associados ao par `Entry<K,V>`) e respetivo significado;**
- **Nome atribuído à EDs.**

ATENÇÃO: Tenha em conta que, quando a escolha recai sobre um dicionário, ordenado ou não, será necessário escolher o tipo da Chave (K) e o tipo do Valor associado (V). Além disso, o tipo em V poderá, ele mesmo, constituir uma ED e, nesse caso, deverá ser descrito da mesma forma na sua resposta.

4.2. Descreva brevemente como implementaria todas as operações do TAD apresentado e calcule as suas complexidades temporais, no caso esperado, justificando. **Não deve desenvolver código em java, apenas fazer uma descrição da implementação das operações de acordo com a sua escolha de estruturas de dados e variáveis de instância.**

(O exame continua na página 5)

5. Nesta pergunta pedimos-lhe que considere o desenvolvimento de um sistema de mensagens de texto online. Os *Utilizadores* deste sistema criam conta utilizando uma String única que os define, a que chamaremos screenName. Além desta informação, o sistema irá guardar o nome completo do utilizador, a informação sobre se este está (atualmente) ou não online (se está a utilizar o sistema) e todas as mensagens que enviou. Uma mensagem é uma cadeia de caracteres (String). Para simplificar, iremos assumir que as mensagens têm (apenas) um remetente e um destinatário e que qualquer utilizador (que esteja online) pode enviar mensagens para qualquer utilizador que também esteja online. Além das operações usuais de criação de utilizadores e mensagens, o sistema deve permitir a consulta dos dados de cada utilizador (inclusive da sua thread de mensagens enviadas e recebidas, ordenada cronologicamente da mais recente para a mais antiga). Deverá ainda ser possível listar todos os utilizadores que estão online, ordenados alfabeticamente por screenName.

   Assim, o sistema deverá permitir as seguintes operações:

   a) *Criar utilizador*, recebendo a seguinte informação: screenName único e nome completo. O utilizador é criado como se estivesse offline (fora do sistema). A inserção só tem sucesso se o screenName ainda não existir no sistema;
   b) *Alterar estado do utilizador,* recebendo o screenName e o estado a tomar pelo utilizador (online ou offline). Esta operação só terá sucesso se o screenName já existir no sistema;
   c) *Enviar mensagem,* recebendo os screenNames do remetente e do destinatário da mensagem, assim como o seu texto. Esta operação só terá sucesso se ambos os screenNames forem válidos (existirem no sistema) e se ambos os utilizadores estiverem online;
   d) *Consultar dados do utilizador*, dando o seu screenName. Esta operação só terá sucesso se o screenName já existir no sistema e deve devolver os dados individuais do utilizador, assim como a listagem completa das mensagens que estão associadas ao utilizador (enviadas e recebidas por ele) ordenadas cronologicamente, da mais recente para a mais antiga;
   e) *Listar utilizadores que estão online*. A operação só terá sucesso se pelo menos um utilizador estiver a utilizar o sistema. Esta listagem deverá ser ordenada alfabeticamente por screenName.

   O sistema deverá ser pensado tendo em conta que podem existir **milhares de utilizadores** que podem enviar **muitos milhares de mensagens**. Não sabemos quantas mensagens poderá enviar e receber cada utilizador.
   Com base nesta especificação, reflita sobre a melhor implementação para a resolução do problema e responda (**separadamente**) às seguintes questões:

   5.1 Proponha **Estruturas de Dados** (EDs) e **variáveis de instância** para apoiar a implementação de todas as operações descritas. Se a sua tarefa for facilitada pela criação de Tipos Abstratos de Dados (TADs) do problema, auxiliares à sua solução, deverá também descrever as variáveis de instância e estruturas de dados associadas a estes na sua resposta.

   Descreva **cada uma das EDs escolhidas individualmente**, incluindo sempre a seguinte informação: **(CONTINUA NA PÁG. 6)**

- **Tipo Abstrato de Dados (TAD) genérico;**
- **Estrutura de Dados (ED) genérica que é usada para implementar o TAD;**
- **Tipo de dados (do problema) a guardar dentro da ED (Tipo do Elemento (E); ou tipos associados ao par `Entry<K,V>`) e respetivo significado;**
- **Nome atribuído à ED.**

ATENÇÃO: Tenha em conta que, quando a escolha recai sobre um dicionário, ordenado ou não, será necessário escolher o tipo da Chave (K) e o tipo do Valor associado (V). Além disso, o tipo em V poderá, ele mesmo, constituir uma ED e, nesse caso, deverá ser descrito da mesma forma na sua resposta.

5.2 Considerando a sua resposta em 5.1, descreva brevemente como implementaria **a)**, tendo em conta as necessidades de preparação das EDs para a execução das restantes operações. Estude ainda a complexidade temporal da operação, no caso esperado, **justificando**.

5.3 Considerando a sua resposta em 5.1, descreva brevemente como implementaria **b)**, tendo em conta as necessidades de preparação das EDs para a execução das restantes operações. Estude ainda a complexidade temporal da operação, no caso esperado, **justificando**.

5.4 Considerando a sua resposta em 5.1, descreva brevemente como implementaria **c)**, tendo em conta as necessidades de preparação das EDs para a execução das restantes operações. Estude ainda a complexidade temporal da operação, no caso esperado, **justificando**.

Anexo A - Recorrências

Recorrência 1

$$T(n) = \begin{cases} a & n = 0 \\ & \textbf{ou} \\ bT(n-1) + c & n \geq 1 \end{cases} \qquad \begin{matrix} n = 1 \\ \\ n \geq 2 \end{matrix}$$

$$T(n) = \begin{cases} O(n) & b = 1 \\ O(b^n) & b > 1 \end{cases}$$

com $a \geq 0, \quad b \geq 1, \quad c \geq 1$ **constantes**

Recorrência 2a)

$$T(n) = \begin{cases} a & n = 0 \\ & \textbf{ou} \\ bT(\frac{n}{2}) + O(1) & n \geq 1 \end{cases} \qquad \begin{matrix} n = 1 \\ \\ n \geq 2 \end{matrix}$$

$$T(n) = \begin{cases} O(\log n) & b = 1 \\ O(n) & b = 2 \end{cases}$$

com $a \geq 0, \quad b = 1, 2$ **constantes**

Recorrência 2b)

$$T(n) = \begin{cases} a & n = 0 \\ & \textbf{ou} \\ bT(\frac{n}{c}) + O(n) & n \geq 1 \end{cases} \qquad \begin{matrix} n = 1 \\ \\ n \geq 2 \end{matrix}$$

com $a \geq 0, \quad b \geq 1, \quad c > 1$ **constantes**

$$T(n) = \begin{cases} O(n) & b < c \\ O(n \log_c n) & b = c \\ O(n^{\log_c b}) & b > c \end{cases}$$

Anexo B – Interfaces e Classes de Apoio

```
public interface Comparable<T>{
   int compareTo( T object );
}
public interface Stack<E>{
   boolean isEmpty( );
   int size( );
   E top( ) throws EmptyStackException;
   void push( E element );
   E pop( ) throws EmptyStackException;
}
public interface Queue<E> {
   boolean isEmpty( );
   int size( );
   void enqueue( E element );
   E dequeue( ) throws EmptyQueueException;
}
public interface Iterator<E> {
    boolean hasNext( );
    E next( ) throws NoSuchElementException;
    void rewind( );
}
public interface List<E> {
    boolean isEmpty( );
    int size( );
    Iterator<E> iterator( );
    E getFirst( ) throws EmptyListException;
    E getLast( ) throws EmptyListException;
    E get( int position ) throws
                        InvalidPositionException;
    int find( E element );
    void addFirst( E element );
    void addLast( E element );
    void add( int position, E element )
       throws InvalidPositionException;
    E removeFirst( ) throws EmptyListException;
    E removeLast( ) throws EmptyListException;
    E remove( int position ) throws
                        InvalidPositionException;
    boolean remove( E element );
}
public interface Entry<K,V>{
   K getKey( );
   V getValue( );
}
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 DListNode<E> implements Serializable {
    public DListNode( E theElement, DListNode<E>
              thePrevious, DListNode<E> theNext );
    public DListNode( E theElement );
    public E getElement( );
    public DListNode<E> getPrevious( );
    public DListNode<E> getNext( );
    public void setElement( E newElement );
    public void setPrevious
                    ( DListNode<E> newPrevious );
    public void setNext( DListNode<E> newNext );
}
public class DoublyLinkedList<E>
                            implements List<E> {
    public boolean isEmpty( );
    public int size( );
    public Iterator<E> iterator( );
    public E getFirst( ) throws
                                EmptyListException;
    public E getLast( ) throws EmptyListException;
    protected DListNode<E> getNode
                                ( int position );
    public E get( int position ) throws
                        InvalidPositionException;
    public int find( E element );
    public void addFirst( E element );
    public void addLast( E element );
    protected void addMiddle
                    ( int position, E element );
    public void add( int position, E element )
                throws InvalidPositionException;
    protected void removeFirstNode( );
    public E removeFirst( ) throws
                                EmptyListException;
    protected void removeLastNode( );
    public E removeLast( ) throws
                                EmptyListException;
    protected void removeMiddleNode
                            ( DListNode<E> node );
    public E remove( int position )
                throws InvalidPositionException;
    protected DListNode<E> findNode( E element );
    public boolean remove( E element );
    public void append( DoublyLinkedList<E> list )
}
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( );
}
```