**Algoritmos e Estruturas de Dados Departamento de Informática, Universidade Nova de Lisboa**

**Segundo teste - 13 de dezembro de 2018**

**Atenção**: Os Anexos ao teste poderão ser-lhe úteis.

1. Considere as Árvores AVL apresentadas nas Figuras 1 e 2. O valor apresentado dentro do

nó de cada árvore refere a chave do mesmo nó. Tendo em conta estes dados, efetue as seguintes operações: a) Desenhe a árvore binária de pesquisa resultante de remover o no com chave 10 na

árvore da Figura 1, sem efetuar rotações. Considera que a árvore resultante é uma árvore AVL? Se a sua resposta for negativa, efetue a rotação necessária para transformá-la em AVL. Deve utilizar os algoritmos apresentados nas aulas teóricas para

o efeito. b) Desenhe agora a árvore binária de pesquisa resultante de inserir um nó com chave 10

na AVL apresentada na Figura 2, sem efetuar rotações. Considera que a árvore resultante é uma árvore AVL? Se a sua resposta for negativa, efetue a rotação necessária para transformá-la em AVL. Deve utilizar os algoritmos apresentados nas aulas teóricas para o efeito.

18

28

Fig. 1

Fig. *2*

2. Considere a classe BinarySearch Tree apresentada nas aulas teóricas de AED. Pedimos

The que implemente o método auxiliar (protected) recursivo insertNodeRec, descrito em baixo. Este método executa, recursivamente, a inserção de um nó folha newNode numa sub-árvore com raiz subTreeRoot. A inserção só deve ter sucesso se a chave da entrada contida em newNode ainda não existe na sub-árvore em questão. Implemente o método insertNodeRec (veja a página seguinte) de forma recursiva e calcule ainda a sua complexidade temporal no caso esperado, justificando. Apresenta-se abaixo também o método insertNode que faz a primeira chamada ao método recursivo, passando a raiz da árvore como parâmetro. NOTA: só serão avaliadas soluções **recursivas.**

public class BinarySearch Tree<K extends Comparable<K>, V>

implements OrderedDictionary<K,V>{

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

*Il* número de nós da árvore, protected int currentSize;

*//* Devolve true se a inserção foi bem sucedida e false se a chave da entrada *//* contida em newNode já existe na árvore.

protected boolean insertNode(BSTNode<K,V> newNode) { return insertNodeRec(this.root, newNode)

*//* requires: sub TreeRoot != null && newNode != null protected boolean insertNodeRec(BSTNode<K,V> subTreeRoot,

BSTNode<K,V> newNode) { *//* a implementar ...

3. Pedimos-lhe que implemente o lterador SumIterator que irá disponibilizar a iteração da

soma de sequências de números inteiros contidas em vários iteradores recebidos no construtor do Iterador. O parâmetro recebido no construtor é um iterador global que, em cada iteração (em cada chamada bem sucedida do método next()) devolve um iterador de números inteiros. Pretende-se que o iterador SumIterator calcule e devolva, em cada iteração, a soma da sequência de números inteiros contidos no iterador corrente, devolvendo esse valor como resultado do seu método next(). Deve implementar todos os métodos do iterador, tal como descrito abaixo (ver página seguinte) e incluir variáveis de instância que ache necessárias, assim como possíveis métodos auxiliares de que necessite. Deve ainda determinar a complexidade temporal de todos os métodos desenvolvidos, no pior caso, justificando. Para este efeito pode assumir que todos os iteradores estão guardados em listas duplamente ligadas. **Exempl**o: Se as sequências de números inteiros S*1* e S2 (ver abaixo) forem adicionadas, atra*v*és de dois iteradores separados, ao iterador global de iteradores, passado como parâmetro no construtor do SumIterator, o primeiro next() do iterador irá devolver o número 14. O segundo next() do SumIterator irá devolver o número 3. O SumIterator terá neste caso, apenas duas iterações, antes de terminar.

*51* = {5, 13, -4,0} S2 = {1, 1, 1}

*(*A pergunta 3 continua na página seguinte)

public class SumIterator implements Iterator<Integer>{

*//*Cria iterador das somas de iterador public SumIterator(Iterator<Iterator<Integer>> data);

*//*Inicializa a iteração. public void rewind();

*//D*evolve true se a iteração ainda não terminou. public boolean hasNext();

*//*devolve a soma dos valores contidos no iterador de números inteiros *// c*orrente. public Integer next() throws NoSuchelementException;

**4. Nesta pergunta deve analisar o problema apresentado e conceber uma** solução para o

**problema completo de acordo com os requisitos descritos. Deverá depois responder a cada uma das questões, de acordo com a solução que concebeu. Nas suas respostas às questões 4.2, 4.3 e 4.4, não deve desenvolver código em java, apenas fazer uma descrição da implementação das operações pedidas, de acordo com a sua escolha de estruturas de dados e variáveis de instância.**

Pedimos-lhe que apoie o desenvolvimento de um sistema de gestão de música online, da perspetiva da classificação da base de dados musical. O sistema disponibiliza milhares de músicas, identificadas de forma única por um identificador interno. Para além deste identificador, associa-se a seguinte informação a cada música: título, artista, duração, ano de publicação e URL. Para facilitar a implementação, assume-se para o exercício que o título da música também é único. Cada música é ainda classificada através de palavras chave, chamadas *tag*s. No ato da inserção na base de dados, a música é criada sem *tags,* que podem ser adicionadas (e removidas) mais tarde. Uma determinada *tag* pode ser classificar várias músicas. O sistema deve permitir a consulta da informação associada a cada música (inclusive a listagem das suas *tags,* ordenadas alfabeticamente) a partir do identificador único. Além disso, deve ser possível listar todas as músicas classificadas com uma determinada *ta*g (ordenadas pelo título da música). Assim, o sistema deverá permitir as seguintes operações: *Op1: Inserir música no sistema,* recebendo os dados da música: identificador interno,

título, artista, duração, ano de publicação e URL. A operação só terá sucesso se o

identificador interno não existir no sistema; *Op2: Adicionar tag a música,* recebendo o identificador da música e a *tag.* A operação só

terá sucesso se o identificador da música já existir no sistema e se a *tag* não estiver já

associada à música; *Op3: Remover associação de tag a música,* recebendo o identificador da música e *a tag.* A

operação só terá sucesso se o identificador da música já existir no sistema e se a *tag* estiver associada à música;

*(*A pergunta 4 continua na página seguinte)

*Op3: Consultar dados da música,* recebendo o identificador da música. A operação só terá

sucesso se o identificador da música já existir no sistema. Todos os dados da música

devem ser listados, incluindo as *tag*s (ordenadas alfabeticamente); *Op4: Listar músicas associadas a uma tag,* recebendo a t*ag.* A operação só terá sucesso se

a tag já existir no sistema e existir pelo menos uma música classificada com esta *tag.*

A listagem deverá ser ordenada alfabeticamente pelo título música. Espera-se que o número de músicas seja da ordem dos milhares. Uma música poderá ter centenas de *tags* associadas. Com base nesta especificação, reflita sobre a melhor implementação para a resolução do problema e responda (**separadamente**) às seguintes questões: 4.1 Proponha Estruturas de Dados (EDs) e **variáveis de inst**ância para apoiar a

implementação das operações *Op1, Op2, Op3 e Op4.* 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. Para cada Estrutura de Dados proposta, a sua resposta deve incluir, justificando: a) Tipo Abstrato de Dados (TAD) genérico; b) Estrutura de Dados (ED) genérica que é usada para implementar o TAD; c) Tipo de dados (do problema) a guardar dentro da ED (Tipo do Elemento (E); ou

**tipos associad**os ao par Entry<K,V>) e respetivo significado; d) 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 EDs e, nesse caso, deverá

ser descrito da mesma forma na sua resposta. 4.2 Considerando a sua resposta em 4.1, descreva brevemente como implementaria *Op1.*

Estude ainda a complexidade temporal de *Op1*, no caso esperado, justi**ficando.** 4.3 Considerando a sua resposta em 4.1, descreva brevemente como implementaria *Op2,*

considerando que esta operação deve preparar as EDs para a execução de *Op3 e Op4.*

Estude ainda a complexidade temporal de *Op2*, no caso esperado, justificando. 4.4 Considerando a sua resposta em 4.1, descreva brevemente como implementaria *Op4.*

Estude ainda a complexidade temporal de *Op4*, no caso esperado, justificando.

**4**

**Anexo A - Recorrências**

Recorrência 1

*n=*0

*n* =

1

**o*u***

b=

1

16T(n-1) + c ni

n>2

*T(n) = 1*

10fb")

*b*>1

com

a > 0, b> 1,

c>1

constantes

**Recorrência 2a)**

*n*=0

n=1

T(n) -

**ou**

IoT) + 0(1) n>1

n22

Tin) - J orlog *n)*

10(m)

*b*=1 6=2

com

a>0,

*b* = 1,2

constantes

Recorrência 2b)

n=1

n=0 *T*(*n)* = {

| B*T*(?) + 2*(n*) n>1

**ou**

*n >* 2

com

a>0, *b*>1.

c>1 constantes

*T(*n) =

(0(n)

*O(n*log*en)* (onlosed)

bcc *b=c* b>c

**Anexo B - Interfaces e Classes de Apoio**

public interface Comparable<T>{

int compareTo( T object );

class DListNode<E> implements Serializable {

public DListNode( E theElement, DListNode<E>

thePrevious, DListNode<E> theNext

);

public interface Stack<E> {

boolean isEmpty(); int size(); E top() throws EmptyStackexception; void push( E element ); E pop() throws EmptyStackexception;

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 interface Queue<E> {

boolean isEmpty(); int size(); *v*oid enqueue E element ); E dequeue() throws Empty Queue Exception;

public interface Iterator<E> {

boolean hasNext();

E next() throws NoSuchelementException; *v*oid rewind();

**mm T1**

public interface List<E> {

boolean is Empty(); 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 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 voi**d 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 )

**M**

public interface Entryck,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 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 is Leaf();