Algoritmos e Estruturas de Dados 2021/22
Segundo Teste – 11 de janeiro de 2022
Departamento de Informática, Universidade Nova de Lisboa

Número:
Nome:
Sala:
Assinatura:

Atenção:

- Os Anexos ao teste poderão ser-lhe úteis.

Regras:

- **NOVA REGRA: Todos os estudantes devem ficar na sala até o final do período do teste**
- O teste tem a duração de 2 horas.
- Não são permitidos portáteis nem telemóveis.
- O teste é sem consulta.
- Não há esclarecimento de dúvidas. Se suspeitar que o enunciado tem algum erro, deve avisar o docente.
- O teste pode ser resolvido a lápis.
- Na mesa pode ter consigo caneta, lápis e borracha.
- **NÃO Pode desagrafar o teste.**
- Qualquer aluno envolvido numa fraude reprova na disciplina.

Pode usar este espaço para rascunho

Número:
Nome:

Grupo I – Programação em AED

1. Considere uma extensão da implementação da Árvore Binária de Pesquisa (ABP) genérica dada nas aulas de AED onde, em cada nó da árvore, é possível aceder ao seu antecessor (classe `BSTWithParent<K,V>`). Esta implementação é apoiada por uma extensão da classe nó genérico de ABP (`BSTNode<K,V>`) que inclui uma referência para o mesmo antecessor (denominado de "parent") e que é apresentada abaixo (`BSTNodeWithParent<K,V>`).

   Neste contexto, pedimos-lhe que implemente o método recursivo `createNodeWithParent()` que apoia a implementação de um construtor especial para a classe `BSTWithParent<K,V>`. O construtor recebe como parâmetro uma ABP (`BinarySearchTree<K,V>`) e vai gerar uma nova `BSTWithParent` a partir da mesma, criando, em cada novo nó da nova árvore, a referência para os seus filhos e para o antecessor. Não deve haver partilha de nós entre as duas árvores.

   O método **`createNodeWithParent()`** que irá desenvolver recebe, como parâmetros, um `BSTNode<K,V>` referente ao nó corrente e um `BSTNodeWithParent<K,V>` que permite o acesso ao antecessor do nó, já criado anteriormente. O mesmo método vai depois devolver o novo nó da classe `BSTNodeWithParent<K,V>` que conterá os mesmos dados que o nó corrente e que deverá incluir uma referência para o antecessor. **Apenas os métodos recursivos serão avaliados**. Deve ainda calcular a complexidade temporal do construtor no melhor caso, no pior caso e no caso esperado, **justificando**.

   Ajuda: Considere o exemplo da Figura 5 onde, relativamente ao nó com chave 2, o seu filho esquerdo é o nó com chave 1, o filho direito é o nó com chave 4, e o antecessor será o nó com chave 8.

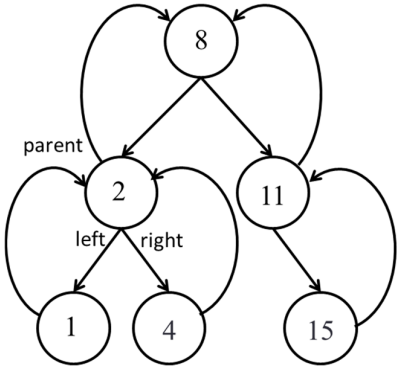

Figura 5

```
// ----------------------------Início da Classe BSTNodeWithParent
package dataStructures;
class BSTNodeWithParent<K,V> extends BSTNode<K,V>{

   // Referência para o nó antecessor.
   private BSTNodeWithParent<K,V> parent;

   // Construtor da classe que recebe a chave e o valor da Entrada a associar ao nó, os
   // filhos esquerdo (left) e direito (right), e o antecessor (parent)
   public BSTNodeWithParent( K key, V value, BSTNodeWithParent<K,V> left,
                             BSTNodeWithParent<K,V> right,  BSTNodeWithParent<K,V> parent){

      super(key, value, left, right);
      this.parent = parent;
   }
// --------------------Classe BSTNodeWithParent continua na página seguinte
```

```
   // Método de acesso ao antecessor do nó corrente.
   public BSTNodeWithParent<K,V> getParent( ){
      return parent;
   }

   // Método modificador do nó corrente.
   public void setParent( BSTNodeWithParent<K,V> newParent ){
      this.parent  = newParent;
   }

}
// ----------------------------Fim da classe BSTNodeWithParent

// ---------------------------- Início da Classe BSTWithParent
package dataStructures;
public class BSTWithParent<K extends Comparable<K>, V> extends BinarySearchTree<K,V>{

   //Raiz da Árvore.
   protected BSTNodeWithParent<K,V> root;

   ...

   // Construtor recebe como parâmetro uma BinarySearchTree<K,V>  e cria uma BSTWithParent a
   // partir da mesma, criando, em cada novo nó da nova árvore, a referência para os filhos e
   // para o antecessor. Não deve haver partilha de nós entre as duas árvores.
   public BSTWithParent(BinarySearchTree<K,V> tree) {
      currentSize= tree.size();
      root=createNodeWithParent(tree.root, null);
   }

   // Cria e devolve novo nó BSTNodeWithParent<K,V> que conterá os mesmos dados que node
   // e que deverá incluir uma referência para o antecessor parent.
   protected BSTNodeWithParent<K,V> createNodeWithParent(BSTNode<K,V> node,
                                                         BSTNodeWithParent<K,V> parent){









   }

   . . .
}

// ----------------------------Fim da classe BSTWithParent
```

2. Considere o Tipo Abstrato de Dados Music, apresentado abaixo como interface Java e que representa uma música que poderá fazer parte de uma Playlist.

```
public interface Music {
   // Devolve título da Música (que se considera único)
   String getMusicTitle();
   //Devolve identificador único da Música
   String getMusicId();
   // Devolve o ano de publicação da Música
   int getMusicPublicationYear();
   //Devolve o artista que executa a Música
   String getMusicArtist();
   //Executa a Música
   void playMusic();
}
```

Implemente a classe `PlayListIterator` que recebe, no construtor, um Iterador bidireccional de Músicas, que inclui todas as músicas na PlayList e um Iterador de número inteiros, que, em cada iteração, devolve o número de ordem, no primeiro iterador, da próxima música a ser tocada. O terceiro parâmetro no construtor será o número de Músicas na PlayList. Tenha em conta que o iterador que contém a ordem de execução das músicas poderá conter músicas repetidas, poderá estar vazio ou poderá não executar algumas das músicas na playlist. Determine a complexidade temporal dos métodos `desenvolvidos` no pior caso, **justificando**. Para este efeito poderá assumir que o conteúdo de cada iterador estará guardado em lista duplamente ligada (`DoubleList` do pacote `DataStructures`).

Exemplo:
Sequência de Músicas: {“Sacrifice”, The Weeknd; “Oh My God”, Adele; “Happier Than Ever”, Billie Eilish}
Ordem de execução das Músicas: { 3, 3, 1, 3, 3}
Sequência de execução: “Happier Than Ever” toca 2 vezes, “Sacrifice” uma vez e “Happier Than Ever” toca mais duas vezes. “Oh My God” nunca chega a tocar.
Embora a PlayList contenha apenas 3 músicas, pretende-se tocar 5 músicas, com uma delas a ser repetida 4 vezes.

```
// ----------------------------Início da Classe PlayListIterator
import dataStructures.TwoWayIterator;
import dataStructures.Iterator;
import dataStructures.NoSuchElementException;

public class PlayListIterator implements Iterator<Music> {
   //variáveis de instância




```

// ----------------classe PlayListIterator continua na página seguinte

```
   public PlayListIterator(TwoWayIterator<Music> playList, Iterator<Integer> order,
                           int numberMusics) {









   }
   public boolean hasNext() {




   }
   public void rewind() {









   }
   public Music next() throws NoSuchElementException {









   }
   //-----------------------------Eventuais métodos auxiliares
```

```
} // -----------------------------Fim da Classe PlayListIterator
```

Pode usar esta folha para responder ao grupo I, se o espaço não chegar. Se o fizer, faça referência a esta folha no espaço da pergunta nas páginas anteriores.

Pode usar esta folha para responder ao grupo I, se o espaço não chegar. Se o fizer, faça referência a esta folha no espaço da pergunta nas páginas anteriores.

Número:

Nome:

Grupo II – Manipulação de Estruturas de Dados

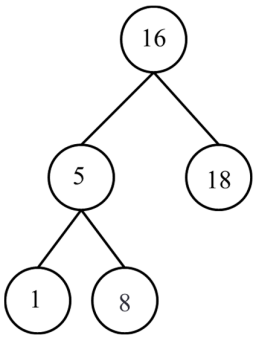

Figura 1

1. Considere a Árvore AVL apresentada na Figura 1. O valor apresentado dentro do nó de cada árvore refere a chave do mesmo nó. Tendo em conta estes dados:
   a) Desenhe a árvore binária de pesquisa resultante de remover o nó com chave 16 na árvore da Figura sem efetuar rotações. **Deve utilizar os algoritmos apresentados nas aulas teóricas para o efeito.**
   b) 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.**

Resposta 1. a)

Resposta 1.b)

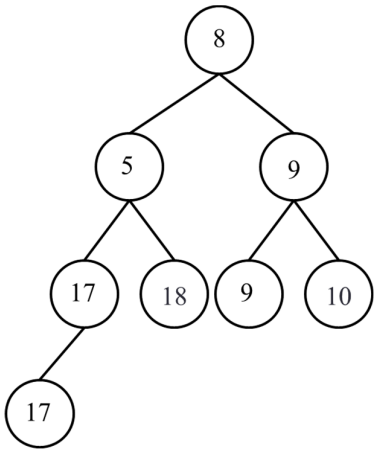

Figura 2

Figura 3

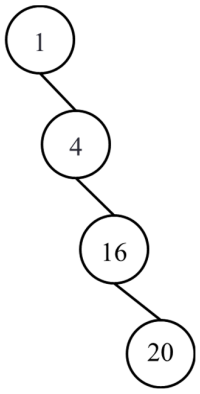

Figura 4

2. Considere as Árvores Binárias apresentadas nas Figuras 2, 3 e 4. O valor apresentado dentro do nó de cada árvore é referente à chave (ou prioridade) do mesmo nó. Tendo em conta estes dados, e as definições estudadas nas aulas teóricas, indique:
   a) Quais destas árvores constituem Árvores Binárias de Pesquisa? Justifique a sua resposta.

   b) Quais destas árvores constituem Árvores Equilibradas? Justifique também esta resposta.

   c) Quais destas árvores constituem Heaps por mínimo (definido nas aulas Teóricas de AED como uma Árvore Binária Completa com prioridade)? Justifique a resposta.

Número:
Nome:

Grupo III – Tipos Abstratos de Dados e Estruturas de Dados

Neste grupo 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. LEIA TODA A PERGUNTA ATÉ AO FIM ANTES DE COMEÇAR A RESPONDER.
Nas suas respostas às questões 2, 3 e 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 na pergunta 1, explicando o desenrolar da operação completa.

Pedimos que conceba uma solução para um sistema de gestão do armazenamento e acesso de Músicas, que irá utilizar o Tipo Abstrato de Dados (TAD) Music da pergunta I.2. Não devem ser consideradas, neste grupo de perguntas, as questões relacionadas com o tratamento de playlists do Grupo I. Este problema tem então como objetivo o armazenamento de Músicas, facilitando o seu acesso através do identificador da Música para inserção, remoção e pesquisa. O sistema inclui também uma componente de gestão das Músicas mais recentes, devendo ter sempre preparada a Listagem de todas as Músicas que foram publicadas no ano que está a decorrer, por ordem do seu Título, que assumimos que é único.

Assim, o sistema deverá permitir as seguintes operações:

- ***Op1:*** *Inserir música no sistema,* recebendo os dados da música: identificador interno, título, artista, e ano de publicação. A operação só terá sucesso se o identificador interno não existir no sistema;
- ***Op2:*** *Remover música do sistema,* recebendo o identificador da música. A operação só terá sucesso se o identificador da música já existir no sistema;
- ***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;
- ***Op4:*** *Listar músicas mais recentes*. A operação só terá sucesso se existir pelo menos uma Música no sistema que tenha sido publicada no ano que está a decorrer. A listagem deverá incluir todas as músicas publicadas no ano que decorre, ordenadas alfabeticamente pelo título da música;
- ***Op5:*** *Mudança de Ano*. Esta operação é executada automaticamente por reconhecimento, pelo sistema, de alteração do ano.

Espera-se que o **número de músicas seja da ordem dos milhares**.

Com base nesta especificação, reflita sobre a melhor implementação para a resolução do problema e responda (**separadamente**) às seguintes questões:

1. Proponha **Estruturas de Dados** (EDs) e **variáveis de instância** para apoiar a implementação das operações ***Op1, Op2, Op3***, ***Op4 e Op5***. 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. Deverá **sempre** incluir as variáveis de instância a considerar na implementação do TAD Music.

   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 associados ao par `Entry<K,V>`) e respetivo significado;**
   d) **Nome atribuído às 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.

Resposta à pergunta III.1

2. Considerando a sua resposta em 1, descreva brevemente como implementaria ***Op1*** *(Inserir música no sistema),* considerando que esta operação deve preparar as EDs para a execução de ***Op4***. Estude ainda a complexidade temporal da operação, no caso esperado, **justificando**.

3. Considerando a sua resposta em 1, descreva brevemente como implementaria ***Op2*** *(Remover música do sistema),* considerando que esta operação pode ter implicações na execução de ***Op4.*** Estude ainda a complexidade temporal de ***Op2***, no caso esperado, **justificando**.

4. Considerando a sua resposta em 1, descreva brevemente como implementaria ***Op4*** *(Listar músicas mais recentes)*. Estude ainda a complexidade temporal de ***Op4***, no caso esperado, justificando.

Pode usar este espaço como rascunho

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 TwoWayIterator<E>
   extends Iterator<E>{
    boolean hasPrevious( );
    E previous( ) throws NoSuchElementException;
    void fullForward( );
}

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 DoubleListNode<E> {
   public DoubleListNode( E theElement,
                  DoubleListNode<E> thePrevious,
                  DoubleListNode<E> theNext );
   public DoubleListNode( E theElement );
   public E getElement( );
   public DoubleListNode<E> getPrevious( );
   public DoubleListNode<E> getNext( );
   public void setElement( E newElement );
   public void setPrevious
               ( DoubleListNode<E> newPrevious );
   public void setNext
               ( DoubleListNode<E> newNext );
}

public class DoubleList<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 DoubleListNode<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
                      ( DoubleListNode<E> node );
   public E remove( int position )
            throws InvalidPositionException;
   protected DoubleListNode<E> findNode
                                   ( E element );
   public boolean remove( E element );
   public void append( DoubleList<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( );
}
```