Algoritmos e Estruturas de Dados 2021/22
Primeiro Teste – 13 de novembro de 2021
Departamento de Informática, Universidade Nova de Lisboa

Número:
Nome:
Sala:
Assinatura:

Atenção:

- Os Anexos ao teste poderão ser-lhe úteis.

Regras:

- 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. Implemente um método recursivo que, dado um vetor de números inteiros completamente preenchido, devolve a posição da primeira ocorrência de um valor inferior a um determinado número inteiro denominado `target`, passado como parâmetro ao método. O método deve devolver -1, caso não exista nenhum valor inferior a `target`.
   `firstBelow` (apresentado abaixo) deve constituir o acesso público ao método, podendo ser necessário implementar métodos auxiliares para concretizar a recursão. Determine ainda a complexidade temporal do método no pior caso, **justificando**.

   Nota: Implemente a solução em código Java. **Apenas as soluções recursivas serão avaliadas.**

```
public static int firstBelow( int[] v, int target ) {
```

2. Considere a classe de lista duplamente ligada `DoubleList<E>` apresentada nas aulas. A classe, tal como foi estudada, contém um único construtor, que cria uma lista vazia (ver abaixo). Implemente um segundo construtor para a mesma classe que recebe, como parâmetro, um nó do tipo `DoubleListNode<E>` que deverá ser considerado a referência para a cabeça (`head`) da lista. O construtor deverá depois inicializar os restantes atributos da lista, recorrendo à referência dada e avançando na lista até atingir o final. Considere que o parâmetro recebido pelo construtor poderá ou não ser null. Implemente este construtor com a melhor complexidade temporal possível. Deve ainda determinar a complexidade temporal do construtor no melhor e no pior caso, **justificando**.

```
package dataStructures;

public class DoubleList<E> implements List<E>{

   // Node at the head of the list.
   protected DoubleListNode<E> head;
   // Node at the tail of the list.
   protected DoubleListNode<E> tail;
   // Number of elements in the list.
   protected int currentSize;

   // Constructor of empty list
   public DoubleList( )
    {
        head = null;
        tail = null;
        currentSize = 0;
    }
```

```
   public DoubleList( DoubleListNode<E> h ) {









    }
```

```
. . .
}
```

Número:

Nome:

Grupo II – Análise de algoritmos

Considere o método apresentado abaixo.

```
Requires: a > 0, b > 0
int operation( int a, int b ) {
        if (b == 1) return a;
        int result = operation( a+a , b/2 );
        if ( b%2 != 0 )
             result += a;
        return result;
}
```

Determine a complexidade temporal do método apresentado, no melhor caso, no pior caso, e no caso esperado, **justificando**.

Melhor caso - O(1) -> quando b=1 pois apenas existe uma chamada recursiva
Caso esperado -> como b = 2^(chamRec), portanto chamRec = n = log2(b) => O(1) + O(log2(b))
Pior caso - O(log2(b)) quando b é o maior número inteiro

O(log2(n)) ?????

porque é que ele meteu o n e não o b????

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 4, 5 e 6 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, explicando o desenrolar da operação completa.

O Tipo Abstrato de Dados (TAD) `Comentator` define os dados e operações associadas a um comentador de televisão, permitindo o acesso público a vários atributos do mesmo. Este TAD é utilizado por um canal de televisão para organizar debates televisivos, baseados no TAD Debate, que irá incluir vários atributos relacionados com o mesmo, como a data, o tema, o nome do moderador e o conjunto de comentadores que irão participar no debate. Para que um comentador possa participar num debate específico, a sua participação tem de ser confirmada, participação esta que inclui uma intervenção individual de uma determinada duração (em minutos). Este valor contribui para a definição da duração mínima total do debate.

Apresentam-se abaixo os interfaces definidos para o problema. Estes interfaces fazem parte de um mesmo pacote de resolução deste problema. O interface `DebateUpdate` (não público) contém o método de confirmação da participação de um comentador no debate e o interface `Debate` contém os métodos de acesso aos dados do debate.

```
public interface Comentator {

  // Nome do comentador
  String getName();
  // Idade do comentador
  int getAge();
  // Empregador atual do comentador
  String getAffiliation();
}

public interface Debate {

  // Para simplificar, a data do debate é uma cadeia de carateres.
  String getDate();

  //Tema do debate
  String getTitle();

  //Nome do(a) moderador(a) do debate
  String getModeratorName();

  // Participantes confirmados do debate: comentadores ordenados por momento
  // da confirmação, do mais antigo para o mais recente
  Iterator<Comentator> getParticipants();

  // Duração mínima total do debate (em minutos), soma das intervenções individuais
  // dos participantes, definidas no momento da confirmação da participação
  int getMinTotalTime();
}

interface DebateUpdate extends Debate {

  // confirmação da participação de c no debate com uma
  // intervenção individual de timeIntervention (em minutos)
  void confirm(Comentator c, int timeIntervention);
}
```

Reflita sobre a melhor implementação para o Interface **`DebateUpdate`** (que inclui o interface **`Debate`**) e responda às seguintes questões **(separadamente). Antes de começar a responder, leia todas as perguntas até ao fim:**

1. Proponha as variáveis de instância necessárias e respetivos tipos para apoiar a implementação dos métodos **`getTitle()`**, **`getModeratorName()`** e **`getDate()`.** Descreva a função destas variáveis.

2. Proponha uma Estrutura de Dados, de entre as apresentadas nas aulas e que fazem parte do pacote **`dataStructures`**, para apoiar a implementação do método **`getParticipants()`**. A sua resposta deve incluir, **justificando**:
   a) **Tipo Abstrato de Dados (TAD) genérico;**
   b) **Estrutura de Dados (ED) genérica que deverá ser 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.**

Pode continuar a pergunta na página seguinte

Continuação da pergunta anterior

3. Proponha ainda, se achar conveniente, variáveis de instância adicionais, para apoiar a implementação do método **`getMinTotalTime()`** e justifique a sua necessidade.

4. Com base nas EDs e variáveis de instância propostas em 1, 2 e 3, descreva brevemente como implementaria a operação **`confirm()`** do interface `DebateUpdate`. Estude ainda a complexidade temporal desta operação no melhor caso, no pior caso e no caso esperado, **justificando**.

5. Com base nas EDs e variáveis de instância propostas em 1, 2 e 3, descreva brevemente como implementaria a operação **`getMinTotalTime()`.** Esta operação deverá ter **complexidade temporal constante.**

6. Considerando as EDs e variáveis de instância propostas em 1, 2 e 3, descreva brevemente como implementaria a operação **`generatePressRelease(Debate d)`**, operação externa aos interfaces apresentados (e que irá fazer parte de uma aplicação que poderá importar o pacote associado ao debate e o pacote `dataStructures`), considerando que:
   - **`generatePressRelease()`** deverá imprimir uma press release sobre o debate que deverá incluir os dados individuais do debate, incluindo a sua duração mínima, e por cada participante no debate, os dados do participante. Os dados dos participantes no debate devem ser impressos por ordem do momento de confirmação da participação no debate, iniciando na confirmação mais antiga e terminando na mais recente;
   - a operação **`getParticipants()`** deverá devolver um iterador da ED proposta em 2;
   - a execução da operação **`getMinTotalTime()`** deve ter complexidade constante, em todos os casos.

   Estude ainda a **complexidade temporal de `generatePressRelease()`** no melhor caso, no pior caso e no caso esperado, **justificando**.

Pode continuar a pergunta na página seguinte

Continuação da pergunta anterior
(se não necessitar deste espaço pode usar para rascunho)

Anexo A - Recorrências

Recorrência 1

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

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

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

Recorrência 2a)

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

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

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

Recorrência 2b)

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

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 );
}
```

```
class DoubleListNode<E> implements Serializable {
   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 );
}
```