Estruturas de Dados 2018/2019 – Junho de 2019

Exame Modelo Duração: 2h30m + 30m

Número mecanográfico:
Nome completo do estudante:

Grupo 1 - Fundamentos de Java

1.1. Escreva pequenos **excertos de código** para cada uma das seguintes alíneas:

a) Extrair o algarismo das centenas de um inteiro n:

b) Expressão booleana que indica se um número n tem exactamente 2 dígitos:

c) Criar um array v com espaço para k caracteres:

d) Ir buscar o número de linhas de uma matriz `m[][]`:

e) Ir buscar a última letra de uma string s:

f) Ir buscar o penúltimo bit de um inteiro b:

Grupo 2 - Implementação de uma classe simples

2.1. Escreva **código para definir uma classe** `Matrix` que representa representa uma matriz de inteiros. Inclua atributos para representar a matriz em si, bem como o número de linhas e de colunas. Adicione também um construtor que recebe o número de linhas e colunas, bem como um inteiro k e cria a respetiva matriz (inicialmente com todas as posições preenchidas com k).

2.2. Escreva um excerto de código para **criar um objecto** m contendo uma matriz de 3 linhas por 4 colunas contendo em todas as posições o inteiro 42:

2.3. Para comparar se dois objectos são iguais é normalmente usado o método **equals**. Por exemplo, se tivermos duas matrizes $m1$ e $m2$, `m1.equals(m2)` deveria devolver *true* se as matrizes são iguais (têm exactamente o mesmo conteúdo), ou *false* caso contrário. Escreva um método **equals** para a classe Matrix com esse comportamento (verifica se a matriz é igual a outra)

2.4. Escreva um **método** `transpose()` da classe Matrix que devolva uma nova matriz que é a transposta da original (trocar linhas com colunas). Por exemplo, a figura seguinte mostra uma matriz e como ficar a sua transposta:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

Grupo 3 - Tipos Abstratos de Dados (TADs)

3.1. Escreva um **interface** Java para representar um conjunto de palavras, contendo métodos para inserir e remover uma palavra, verificar se uma palavra está no conjunto e devolver o número de palavras do conjunto (use nomes indicativos da sua função para os métodos respetivos).

3.2. Suponha que quer criar uma classe X que **concretiza uma implementação** do interface anterior. Como deveria ser a primeira linha que declara essa classe?

3.3. Suponha que tem uma implementação do interface anterior que usa como base um array (desordenado) onde são guardadas as palavras (uma por cada posição). Seja n o número de palavras do conjunto. Indique a **melhor complexidade temporal** (notação $\mathcal{O}$) que conseguiria obter para os seguintes métodos (justificando sucintamente):

a) Devolver número de palavras no conjunto:

b) Verificar se uma palavra está no conjunto:

3.4. Explique como poderia obter uma implementação mais eficiente para a verificar se a palavra está no conjunto, indicando a nova complexidade e outras potenciais vantagens e desvantagens dessa outra implementação quando comparada com o array desordenado (ex: mais algum método é melhor ou pior?)

Grupo 4 - Complexidade Algorítmica

4.1. Para os pares de funções seguintes, escreva na caixa em branco $\mathbf{\Theta}$, $\mathcal{O}$ ou $\mathbf{\Omega}$ para tornar a afirmação verdadeira. Note que se as funções em causa tiverem uma relação Θ, não deverá escrever $\mathcal{O}$ ou Ω.

$n^4 = \square\ (4^n)$ $\quad 2^n = \square\ (2^{n+2})$ $\quad 42 = \square\ 24$ $\quad \log(\log n^2) = \square\ (\log n^2)$ $\quad \sqrt{n} = \square\ (\log_4 n)$

4.2. Descreva o pior caso em termos de tempo de execução para os seguintes pedaços de código, usando notação $\mathcal{O}$ para a variável n. Use o limite mais "apertado" possível (ex: indicar todos como sendo $\mathcal{O}(n!)$ resultará em 0 pontos).

a)

```
for (int i=0; i<n; i++)
    for (int j=0; j<i; j++)
        count++;
```

b)

```
for (int i=n/2; i>0; i--)
    for (int j=0; j<100; j++)
        count++;
```

c)

```
for (int i=1; i<n; i=i*2)
    count++;
```

d)

```
int f1(int n) {
    return f1(n-1) + 1;
}
```

e)

```
int f2(int n) {
    int count = 0;
    for (int i=0; i<n; i++) count++;
    return count + f2(n/2) + f2(n/2)
}
```

4.3. Justifique a sua resposta para a alínea (*e*) da pergunta anterior, **escrevendo a recorrência e desenhando a respectiva árvore de recorrência**, indicando porque dá origem à complexidade indicada. Indique um algoritmo que conheça cuja complexidade seja determinada pela mesma recorrência.

4.4. Previsão de Tempo de Execução.
Imagine que tem um programa que demora **1 segundo** para um dado input. Se o **tamanho do input triplicar**, quanto tempo estima que o programa demora para cada uma das seguintes complexidades?

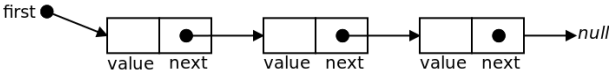
Se tiver complexidade **linear**, deverá demorar cerca de _____ segundos.

Se tiver complexidade **quadrática**, deverá demorar cerca de _____ segundos.

Se tiver complexidade **cúbica**, deverá demorar cerca de _____ segundos.

Grupo 5 - Listas, Pilhas e Filas

5.1. Considere uma implementação `SinglyLinkedList<T>` de uma lista ligada simples genérica tal como dada nas aulas com atributos `size` e first, e `Node<T>` representando um nó com atributos `value` e `next`.



a) Escreva **código para definir um método print()** que percorre a lista e imprime todos os seus valores (um por linha). Não pode usar qualquer outro método da classe `SinglyLinkedList<T>`.

b) Escreva **código para definir um método remove(i)** que remove o i-ésimo elemento da lista (elemento na posição i). As posições começam em 0. Se a posição i não existir, o método não deve fazer nada. Não pode usar qualquer outro método da classe `SinglyLinkedList<T>`.

c) Escreva **código para definir um método reverse(i)** que devolve uma nova lista que é igual à lista inicial invertida (ex: $1 \rightarrow 2 \rightarrow 3$ resultaria numa nova lista $3 \rightarrow 2 \rightarrow 1$). Pode destruir a lista original e usar a API das listas (ex: removeFirst, removeLast, addFirst, addLast). Indique a **complexidade temporal** do método que implementou.

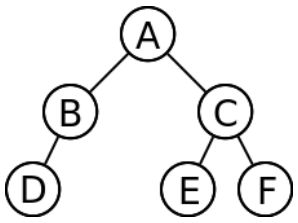
5.2. Indique, justificando, uma situação onde faz mais sentido usar uma lista ligada ao invés de um simples array.

5.3. Explique (não é preciso código) como **remover o primeiro elemento de uma lista circular** que apenas tenha uma referência para o último elemento da lista (e não para o primeiro). Não se esqueça de descrever como e em que atributos tinha de mexer e indique a **complexidade temporal** do método que descreveu.

5.4. Explique (não é preciso código) como **inverter o conteúdo de uma pilha**, sendo que para interagir com a pilha apenas pode usar a sua API (*push*, *pop* e *size*). Indique a **complexidade temporal** do método que descreveu.

Grupo 6 - Árvores

6.1. Considerando a **árvore da figura** seguinte, responda às seguintes alíneas:



a) Qual o nó **raíz**? **b)** Quais os nós **folha**?

c) Um par de nós **irmãos**? **d)** Qual a **altura** da árvore?

e) Os nós da árvore em **preorder:**

f) Os nós da árvore em **inorder:**

g) Os nós numa **pesquisa em largura:**

6.2. Consider uma implementação `BTree<T>` de uma árvore binária genérica tal como dada nas aulas com um atributo `root` a apontar para a raíz que é um `BNode<T>` representando um nó com atributos `value`, `left` e `right`.

a) Escreva **código para definir um método depth()** da classe `BTree<T>` que devolve a profundidade da árvore (sem usar a API das árvores).

b) Escreva **código para definir um método estático count(BTree<Integer> t)** que devolva a quantidade de nós que armazenam números pares.

6.3. Suponha que quer um método para **procurar um número numa árvore binária simples** (não é de pesquisa nem impõe nenhuma propriedade sobre a maneira como guarda os números). Indique, justificando, qual a melhor complexidade temporal que conseguiria para implementar esse método.

Grupo 7 - Árvores Binárias de Pesquisa

7.1. Vamos inserir (por esta ordem) os seguintes números numa árvore binária de pesquisa: 42, 20, 24, 50, 53, 11. Para as 3 alíneas de baixo **desenhe a árvore** correspondente (depois de completa a operação respetiva):

a) Como fica a **árvore original** depois de inseridos os 6 números?

b) Como fica a árvore (original) depois de **adicionarmos um 51**?

c) Como fica a árvore (original) depois de **removermos o 42**?

7.2. Suponha que quer inserir os elementos $1, 2, 3, 4, 5, 6, 7$ numa árvore binária de pesquisa. Qual a mínima altura possível da árvore? Indique por qual ordem deve inserir os elementos pra garantir essa altura mínima.

7.3. Em que nó fica guardado o elemento mínimo numa árvore binária de pesquisa? Justifique sucintamente.

7.4. Explique (por palavras) um algoritmo para **procurar um elemento numa árvore binária de pesquisa**. Indique, justificando, a **complexidade temporal** do método que descreveu supondo que a árvore está equilibrada.

7.5. Suponha que tem um **TAD Dicionário** `BTMap<K,V>` que mapeia chaves do tipo K em valores do tipo V implementado com uma árvore binária de pesquisa e que esta está sempre equilibrada. Suponha também que tem acesso a uma lista de n reviews de hotéis. Cada review atribui uma nota de 0 a 10 a um hotel, que é identificado pelo nome. Usando a API dos dicionários (*put*, *get*, *keys*, *size*) explique com algum detalhe (pode ser só por palavras) um algoritmo que descubra quais os hotéis com pelo menos 20 reviews e nota média superior a 8. Indique, justificando, a a **complexidade temporal** do método que descreveu.

Grupo 8 - Filas de Prioridade e Heaps

8.1. Imagine que tem uma **minHeap** descrita pelo seguinte array

3	4	5	8	7	10

Para as 3 alíneas de baixo **desenhe a árvore** (a heap) correspondente (sempre já com o invariante reposto):

a) Qual a heap representada pelo **array original**?

b) Como fica a heap (original) depois de **adicionarmos um 2**?

c) Como fica a heap (original) depois de **removermos o mínimo**?

8.2. Explique como poderia usar uma minHeap para guardar um conjunto (dinâmico) de números e ir sempre conseguindo **retirar o maior elemento** (ao invés de retirar o menor) em tempo logarítmico. Use simplesmente a API normal da minHeap, sem mudar a sua implementação.

8.3. Suponha que lê um conjunto de n números, inserindo-os um a um numa fila de prioridade implementada com uma *minHeap*. Depois retira-os um a um da heap e vai escrevendo os valores à medida que vão sendo removidos.

a) Qual a **ordem** em que aparecem os números no output?

b) Indique, justificando, a **complexidade temporal e espacial** (memória) de todo o processo descrito.

(pode usar esta página para a continuação de respostas que necessitem de mais espaço para além do que foi dado)

(pode usar esta página para a continuação de respostas que necessitem de mais espaço para além do que foi dado)