Estruturas de Dados 2018/2019 – Época Normal (25 de Junho de 2019)

Versão A

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) Expressão booleana para: o conteúdo da string a é igual ao conteúdo da string b?

b) Criar um array v com espaço para k valores *true* ou $false$:

c) Ir buscar o número de colunas de uma matriz `m[][]`:

d) Ir buscar a letra do meio de uma string s (assuma que tem tamanho ímpar):

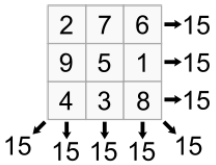
e) Desligar o antepenúltimo bit de um inteiro b (colocar a 0) :

Grupo 2 - Implementação de uma classe simples

2.1. Implemente uma **classe** `Matrix` que representa uma **matriz quadrada de inteiros**, com atributos públicos para representar a matriz em si, bem como o tamanho do lado. Inclua um construtor que receba o tamanho n e cria uma matriz de $n \times n$ (todas as células devem conter o número zero, inicialmente).

2.2. Implemente um **método** `clone()` da classe `Matrix` que cria e devolve uma novo objecto `Matrix` que é uma cópia da matriz original (mesmas dimensões e mesmo conteúdo). Note que depois de criada a cópia, mexer numa das matrizes não deve implicar mudanças na outra matriz.

2.3. Implemente um **método** `magic()` que devolve *true* se a matriz for mágica e $false$ caso contrário. Uma matriz é mágica se a soma dos números de uma qualquer linha, coluna ou diagonal for a mesma. Por exemplo, a matriz seguinte é mágica:



Se necessitar, pode criar métodos auxiliares.

Grupo 3 - Tipos Abstratos de Dados (TADs)

3.1. Explique o que é um **interface** e quais são as diferenças em relação a uma **classe**.

3.2. Suponha que usa como base um array ordenado para representar **conjuntos de números inteiros**. Indique a **complexidade temporal** (notação $\mathcal{O}$) do **melhor algoritmo que consegue imaginar** para a implementação de alguns métodos. Suponha que o conjunto tem n inteiros entre 1 e k e **justifique sucintamente** as respostas.

a) Devolver quantidade de inteiros no conjunto:

b) Verificar se um inteiro está no conjunto:

c) Inserir um inteiro no conjunto:

3.3. Explique como poderia obter uma **implementação mais eficiente** para verificar se o inteiro está no conjunto, indicando a nova complexidade e outras potenciais vantagens e desvantagens em comparação com o array ordenado.

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.

$4n \in \square\ (n)$ $\quad 3n \in \square\ (n^3)$ $\quad 2^n \in \square\ (n^4)$ $\quad n^2 \in \square\ (n \log n)$ $\quad \log_2 8 \in \square\ (42)$ $\quad n^n \in \square\ (n!)$

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 sempre $\mathcal{O}(n!)$ resultará em 0 pontos).

a)

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

b)

```
for (int i=24; i<42; i++)
    for (int j=1; j<n/2; j++)
        for (int k=0; k<n-2; k++)
            count++;
```

c)

```
int f1(int n) {
    if (n<1) return 0;
    return 1 + f1(n/2);
}
```

d)

```
int f2(int n) {
    if (n<1) return 0;
    return 1 + f2(n/2) + f2(n/2);
}
```

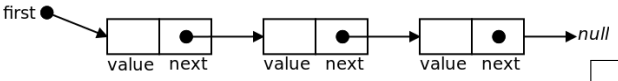
4.3. Justifique a sua resposta para a alínea (*d*) da pergunta anterior, **escrevendo a recorrência e desenhando a respectiva árvore de recorrência**, indicando porque dá origem à complexidade indicada.

4.4. Previsão de Tempo de Execução.
Complete a tabela seguinte usando a informação já preenchida em cada linha. Nos tempos, use uma previsão baseada no tempo já preenchido, tendo em conta a proporção entre n_1 e n_2 (ms = milisegundos).

Programa	Complexidade	Nome Comum	Tempo para $n_1 = 10$	Tempo para $n_2 = 20$
A		constante	10ms	
B		linear		50ms
C	$\Theta(n^2)$		20ms	
D		cúbica		40ms
E	$\Theta(2^n)$		10ms	

Grupo 5 - Listas, Pilhas e Filas

5.1. Considere a 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` (com *getters* e *setters*).



a) Implemente um **método** `contains(x)` devolve *true* se o elemento x está na lista e $false$ caso contrário. Não pode usar qualquer outro método da classe `SinglyLinkedList<T>`. Indique a **complexidade temporal** do método que descreveu.

b) Implemente 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>`. Indique a **complexidade temporal** do método que descreveu.

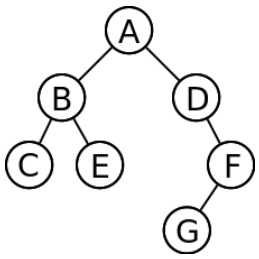
c) Implemente um **método** `rotate(`k`)` que devolve uma nova lista que é igual à lista inicial mas rodada (*"shiftada"*) k posições para a direita. Por exemplo, uma chamada a `rotate(2)` da lista $1 \to 2 \to 3 \to 4 \to 5 \to 6$ resultaria numa nova lista $5 \to 6 \to 1 \to 2 \to 3 \to 4$. Não pode usar qualquer outro método da classe `SinglyLinkedList<T>`. Indique a **complexidade temporal** do método que implementou.

5.2. Indique, justificando, uma **vantagem** e uma **desvantagem** de uma lista ligada em relação a um array.

5.3. Explique (não é preciso código) como **remover o antepenúltimo elemento de uma lista duplamente ligada**. Indique a **complexidade temporal** do método que descreveu e compare-o com a complexidade de fazer a mesma operação numa lista ligada simples.

Grupo 6 - Árvores

6.1. Considerando a **árvore da figura** seguinte (de altura 3), responda às seguintes alíneas:



a) Os nós da árvore em **preorder:**

b) Os nós da árvore em **inorder:**

c) Os nós da árvore em **postorder:**

6.2. Quantos nós tem no mínimo uma árvore de altura 6? E no máximo?

6.3. 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) Implemente **um método** `leaves()` da classe `BTree<T>` que devolve o **número de folhas** da árvore (sem usar a API das árvores). Pode criar métodos auxiliares. Indique a **complexidade temporal** do método que implementou.

b) Implemente **um método** `bfs()` da classe `BTree<T>` que devolve um array contendo os elementos da árvore numa **pesquisa em largura**. Por exemplo, para a árvore da figura 6.1, deveria ser devolvido um array $[A, B, D, C, E, F, G]$. Pode usar as APIs de listas, pilhas e filas (não há problema se não souber de cor os nomes dos métodos, desde que use nomes indicativos da operação que pretende). Indique a **complexidade temporal** do método que implementou.

Grupo 7 - Árvores Binárias de Pesquisa

7.1. Suponha que quer inserir (por esta ordem) os seguintes 6 números numa árvore binária de pesquisa: $22, 42, 13, 7, 39, 15$.

Para as duas alíneas ao lado, **desenhe a árvore** (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 **removermos o 22**?

7.2. Em que nó fica guardado o **elemento máximo** numa árvore binária de pesquisa? Justifique sucintamente.

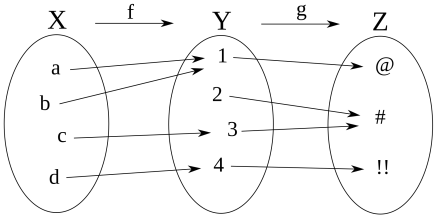
7.3. Suponha que quer inserir os números de 1 a n numa árvore binária de pesquisa. **Qual a mínima altura possível da árvore?** Justifique a sua resposta e indique também por qual ordem deve inserir os elementos para garantir essa altura mínima.

7.4. A **composição de funções** é uma operação que pega em duas funções f e g e produz uma função $h = g \circ f$ tal que $h(x) = g(f(x))$, ou seja, g é aplicada ao resultado de aplicar f a x.

Suponha que para representar uma função qualquer $f : K \to V$ usa um **TAD Dicionário** `BSTMap<K,V>` que mapeia chaves do tipo K em valores do tipo V implementado com uma **árvore binária de pesquisa** (que está **sempre equilibrada**), contendo os métodos *put*, *get*, *keys* e *size* habituais.

Implemente um método estático `BSTMap<X,Z> compose(BSTMap<X,Y> f, BSTMap<Y,Z> g)` que devolve um dicionário com a composição do mapa f com o mapa g. Uma chave do tipo X é mapeada num tipo Z correspondendo a aplicar primeiro f para obter um valor do tipo Y e depois usar esse valor como chave no mapa g. Pode assumir que os tipos X, Y e X são todos comparáveis. Indique a **complexidade temporal** do método que implementou.

Exemplo de composição:



$h = g \circ f$ é a composição de f com g. Por exemplo, h('c') = '#'.

Grupo 8 - Filas de Prioridade e Heaps

8.1. Imagine que tem uma **minHeap** descrita pelo seguinte array | 2 | 3 | 7 | 8 | 6 | 9 |

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 **adicionar um 1**?

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

8.2. Explique como poderia usar uma *minHeap* para **ordenar um conjunto de** n **números**. Indique que operações usaria e qual seria a **complexidade temporal** do método que descreveu.

8.3. (pergunta de valorização) Explique como poderia implementar uma **min-max-heap**, uma estrutura de dados que suporte inserção de elementos em $\mathcal{O}(\log n)$, remoção quer do mínimo quer do máximo em $\mathcal{O}(\log n)$ e leitura (sem remoção) quer do mínimo quer do máximo em $\mathcal{O}(1)$.

(pode usar o resto desta página para continuar respostas que necessitem de mais espaço para além do que foi dado)

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

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