FATEC JAHU

AVALIAÇÃO P2 – ESTRUTURA DE DADOS – 26/06/2025

Prof. Me. Tiago Antonio da Silva

INSTRUÇÕES

I. Leia a prova com atenção e em silêncio.
II. A prova é **individual** e com consulta apenas ao seu pendrive.
III. Ao término da prova, compacte seus arquivos JavaScript em um zip com seu nome completo e chame o professor para recolhimento da prova.
IV. A prova se inicia às 9h30m e terminará às 13h00m. Não serão aceitos envios tardios. Tempo mínimo de permanência: 1h30m.
V. COMENTE TODO O CÓDIGO EXPLICANDO O FUNCIONAMENTO DE CADA FUNCIONALIDADE.
VI. Dúvidas? Retorne ao primeiro item.

QUESTÕES

Questão 1: Ordenando um Conjunto de Dados

Use a classe **`Sorter`** para ordenar o seguinte conjunto de números: **`[64, 34, 25, 12, 22, 11, 90]`**.

a) Implemente o código para ordenar o array usando o método **`Sorter.bubbleSort`**.

b) Implemente o código para ordenar o mesmo array usando o método **`Sorter.mergeSort`**.

c) Imprima no console o resultado de ambas as ordenações. Qual método você esperaria que fosse mais rápido para um array com 1 milhão de números e por quê?

Questão 2: Encontrando Elementos

Dado o array **ordenado `[2, 5, 8, 12, 16, 23, 38, 56, 72, 91]`**, utilize a classe **`Buscas`** para encontrar o elemento 23.

a) Escreva o código que usa **`Buscas.sequencial`** para encontrar o índice do número 23.

b) Escreva o código que usa **`Buscas.binaria`** para encontrar o índice do mesmo número.

c) Imprima os resultados e explique por que a busca binária é mais eficiente para este cenário.

Questão 3: Construindo e Percorrendo uma Árvore Binária

Usando a classe **`BinaryTree`**, construa uma árvore a partir dos seguintes valores, inseridos nesta ordem: **`[50, 30, 70, 20, 40, 60, 80]`**.

a) Instancie uma **`BinaryTree`**.

b) Insira cada um dos números na árvore.

c) Utilize o método **`preOrder`** para percorrer a árvore e imprima a sequência de valores no console. O resultado reflete a ordem de inserção ou a estrutura da árvore?

Questão 4: Comparando `BinaryTree` vs. `AVLTree`

Demonstre a principal diferença entre uma árvore de busca binária comum e uma árvore AVL.

a) Crie uma instância da **`BinaryTree`** e insira a seguinte sequência de números (que já está ordenada): **`[10, 20, 30, 40, 50, 60]`**.

b) Crie uma instância da **`AVLTree`** e insira a mesma sequência de números.

c) Após as inserções, acesse e imprima a raiz de cada árvore (**`tree.root`**) para comparar suas estruturas.

d) Descreva a diferença observada na estrutura das duas árvores. Por que a estrutura da **`AVLTree`** é superior em termos de desempenho de busca?

Questão 5: Modelando uma Rede de Contatos

Use a classe **`Grafo`** para modelar uma pequena rede de amizades.

a) Crie uma instância da classe **`Grafo`**.

b) Adicione os seguintes vértices: 'Alice', 'Bruno', 'Carlos', 'Diana'.

c) Adicione as seguintes arestas (amizades): ('Alice', 'Bruno'), ('Bruno', 'Carlos'), ('Carlos', 'Diana'), ('Alice', 'Diana').

d) Use o método **`imprimirGrafo()`** para mostrar a lista de adjacência da rede.

e) Agora, simule que 'Carlos' e 'Diana' deixaram de ser amigos. Remova a aresta entre eles e imprima o grafo novamente para confirmar a alteração.

Questão 6: Análise de Desempenho do Quick Sort

A implementação de **`Sorter.quickSort`** usa o último elemento como pivô. Isso pode levar a um desempenho ruim em certos cenários.

a) Crie um array com 10.000 números aleatórios.

b) Crie um segundo array com 10.000 números já ordenados (de 1 a 10.000).

c) Use **`console.time('label')`** e **`console.timeEnd('label')`** para medir o tempo de execução do **`Sorter.quickSort`** em cada um dos arrays.

d) Imprima os tempos e explique por que há uma diferença tão grande de desempenho.

Questão 7: O Custo do Auto-Balanceamento

Uma **`AVLTree`** garante buscas rápidas, mas o processo de inserção tem um custo adicional (as rotações). Meça e compare esse custo.

a) Gere um array com 20.000 números aleatórios.

b) Meça o tempo total para inserir todos os números do array em uma instância de **`BinaryTree`**.

c) Meça o tempo total para inserir os mesmos números em uma instância de **`AVLTree`**.

d) Compare os tempos de inserção. Em que tipo de aplicação vale a pena pagar o "preço" extra na inserção da **`AVLTree`**?

Questão 8: Busca por Interpolação vs. Binária

A **`Buscas.interpolacao`** pode ser mais rápida que a **`Buscas.binaria`**, mas apenas sob condições ideais.

a) Crie um array grande (ex: 500.000 elementos) com números **uniformemente distribuídos** (ex: [2, 4, 6, 8, ...]).

b) Crie um segundo array grande com números **não uniformemente distribuídos** (ex: [1, 2, 3, 100, 150, 200, 500000, 500001, ...]).

c) Para cada array, meça o tempo de busca por um elemento próximo ao final usando **`Buscas.binaria`** e **`Buscas.interpolacao`**.

d) Analise os resultados. Em qual cenário a busca por interpolação teve melhor desempenho e por quê?

Questão 9: Implementando Travessia em Grafo (BFS)

A classe **`Grafo`** não possui um método de travessia para encontrar caminhos. Sua tarefa é adicionar essa funcionalidade.

a) Adicione um novo método à classe **`Grafo`** chamado **`breadthFirstSearch(inicio, fim)`** que implemente o algoritmo de Busca em Largura (BFS) para encontrar o caminho mais curto entre dois vértices. O método deve retornar um array contendo a sequência de vértices do caminho (ex: ['A', 'B', 'C']).

b) Crie uma instância de **`Grafo`** para representar um mapa simples (ex: cidades vizinhas).

c) Use o seu novo método para encontrar e imprimir o caminho mais curto entre duas cidades não adjacentes.

Dica: Você precisará de uma fila para controlar os vértices a serem visitados e de uma forma de rastrear o "pai" de cada nó para reconstruir o caminho no final.

Questão 10: Sistema de Indexação e Busca em Larga Escala

Simule e compare duas estratégias para um sistema que precisa processar um grande volume de dados e depois realizar múltiplas buscas.

a) Gere um array com 1 milhão de números inteiros aleatórios (dados).

b) Gere um segundo array com 1.000 números aleatórios para servirem como os termos de busca (buscas).

c) **Estratégia 1: Ordenação e Busca Binária** a. Meça o tempo para ordenar o array dados usando **`Sorter.mergeSort`**. b. Em seguida, meça o tempo total para buscar cada um dos 1.000 elementos do array buscas dentro do array dados (já ordenado) usando **`Buscas.binaria`**.

d) **Estratégia 2: Árvore AVL** a. Meça o tempo para inserir todos os 1 milhão de elementos do array dados em uma **`AVLTree`**. b. Em seguida, meça o tempo total para buscar cada um dos 1.000 elementos do array buscas na **`AVLTree`**. (Nota: você precisará adicionar um método de busca na **`AVLTree`**. Pode adaptar o método **`search`** da **`BinaryTree`**).

e) **Análise Final:** Some os tempos de "preparação" e "busca" de cada estratégia. Qual delas foi mais eficiente para o problema completo? Discuta os prós e contras de cada abordagem.