Prova 2 - Estrutura de Dados

Turma 1

30 de outubro de 2025

Instruções:

- Tempo de duração da prova é de duas horas.

Dica para o sucesso na prova:

- JUSTIFIQUE TODAS AS QUESTÕES
- Estruturar a fase de estudo do problema ANTES de começar a fazer o algoritmo ou programa; Identificar entradas; identificar a saída esperada; RESOLVER o problema; identificar como validar a solução; codificar;testar.

1 Listas Encadeadas

1. **(0.5 PONTO)** Considere o uso de uma **lista encadeada simples** como estrutura de dados. Analise as afirmativas abaixo e escolha a alternativa correta sobre as vantagens e desvantagens dessa estrutura:

   (a) Listas encadeadas permitem alocação dinâmica de memória, sendo eficientes no uso de espaço quando comparadas a vetores.

   (b) Inserir ou remover elementos no início da lista encadeada é mais eficiente do que em vetores, pois não é necessário deslocar elementos.

   (c) A busca por um elemento em uma lista encadeada é, em geral, mais eficiente do que em vetores devido à sua estrutura sequencial.

   (d) Listas encadeadas exigem armazenamento adicional para os ponteiros, o que pode aumentar o consumo de memória em comparação a vetores.

   (e) Em listas encadeadas, o acesso direto a elementos é rápido, permitindo consultas em tempo constante $O(1)$.

   Alternativas:

   A) Somente as afirmativas 1, 2 e 3 estão corretas.

   B) Somente as afirmativas 1, 2 e 4 estão corretas.

   C) Somente as afirmativas 2, 4 e 5 estão corretas.

   D) Somente as afirmativas 1, 3 e 5 estão corretas.

   E) Todas as afirmativas estão corretas.

2. **(0.5 PONTO)** Considere uma **lista duplamente encadeada**, onde cada nó contém um ponteiro para o próximo elemento e outro para o elemento anterior. Analise as afirmativas abaixo sobre a complexidade de suas operações e escolha a alternativa correta:

   (a) Inserir um elemento no início da lista possui complexidade $O(1)$.

   (b) Inserir um elemento no final da lista possui complexidade $O(1)$, desde que seja mantido um ponteiro para o último nó.

(c) Remover um elemento de uma posição específica na lista possui complexidade $O(1)$, independentemente da posição.

(d) A busca por um elemento na lista possui complexidade $O(n)$ no pior caso.

(e) Inserir um elemento em uma posição intermediária na lista requer encontrar a posição primeiro, o que pode adicionar uma complexidade de $O(n)$ no pior caso.

Alternativas:

A) Somente as afirmativas 1, 2 e 4 estão corretas.

B) Somente as afirmativas 1, 3 e 5 estão corretas.

C) Somente as afirmativas 1, 2, 4 e 5 estão corretas.

D) Somente as afirmativas 2, 3 e 4 estão corretas.

E) Todas as afirmativas estão corretas.

3. **(0.5 PONTOS)** Considere a implementação parcial da classe `ListaDuplamenteEncadeadaCircular` em C++ apresentada abaixo:

```cpp
class ListaDuplamenteEncadeadaCircular {
public:
    Node *head;

    ListaDuplamenteEncadeadaCircular() {
        head = new Node();
        head->prox = head;
        head->ant = head;
    }

    void inserirInicio(int x) {
        Node *new_no = new Node();
        Node *antigo_no = head->prox;

        head->prox = new_no;
        new_no->prox = antigo_no;
        new_no->ant = antigo_no->ant;
        antigo_no->ant = new_no;
    }

    void inserirFinal() {
        // Complete
    }

    bool buscar(int x) {
        Node *p;
        p = head->prox;
        while (p != head) {
            if (p->dado == x) {
                return true;
            }
            p = p->prox;
        }
        return false;
    }

    void removerLista(int x) {
        Node *p;
        p = head->prox;
        while (p != head) {
            if (p->dado == x) {
                break;
```

```
43              }
44          }
45          if (p != head) {
46              p->prox->ant = p->ant;
47              p->ant->prox = p->prox;
48              delete p;
49          }
50      }
51
52      void imprimir() {
53          Node *p = head->prox;
54          while (p != head) {
55              cout << p->dado << endl;
56          }
57      }
58  };
```

Com base no código acima, responda às seguintes perguntas:

(a) Complete o método `inserirFinal()` de forma que ele insira corretamente um elemento no final da lista.

(b) Suponha que a lista já contém n elementos e que o método `buscar(int x)` seja chamado. Qual é a complexidade de tempo do método? Justifique sua resposta.

(c) Explique o que aconteceria se um nó não for encontrado no método `removerLista(int x)`. O que poderia ser melhorado para tornar o método mais eficiente?

(d) O método `imprimir()` apresenta um problema no código fornecido. Identifique o problema e explique como corrigi-lo.

4. **(0.5 PONTO)** Considere as operações de inserção e remoção de elementos em uma estrutura de dados **Pilha**. Assinale a alternativa correta:

(a) A operação `push()` em uma **Pilha** insere um elemento no topo da pilha e tem complexidade $O(n)$, pois todos os elementos devem ser deslocados para abrir espaço.

(b) A operação `pop()` em uma **Pilha** remove o elemento do topo e tem complexidade $O(1)$, pois o acesso ao topo da pilha é direto e não requer deslocamento de elementos.

(c) A operação `push()` em uma **Pilha** insere um elemento no fundo da pilha, o que faz com que a complexidade seja $O(1)$.

(d) A operação `pop()` em uma **Pilha** remove o elemento do fundo da pilha e tem complexidade $O(n)$, pois todos os elementos precisam ser rearranjados.

(e) A operação `push()` em uma **Pilha** é utilizada para remover um elemento do topo da pilha.

5. **(1 PONTO)** Considere as operações de inserção e remoção de elementos em uma estrutura de dados **Fila**. Assinale a alternativa correta:

   (a) A operação `enqueue()` em uma **Fila** insere um elemento no final da fila e tem complexidade $O(n)$, pois todos os elementos precisam ser deslocados para abrir espaço.

   (b) A operação `dequeue()` em uma **Fila** remove o elemento da frente da fila e tem complexidade $O(1)$, pois o acesso ao primeiro elemento da fila é direto e não requer deslocamento de elementos.

   (c) A operação `enqueue()` em uma **Fila** insere um elemento no início da fila, o que faz com que a complexidade seja $O(1)$.

   (d) A operação `dequeue()` em uma **Fila** remove o elemento do final da fila e tem complexidade $O(n)$, pois todos os elementos precisam ser rearranjados.

   (e) A operação `enqueue()` em uma **Fila** é utilizada para remover um elemento da fila.

6. **(1 PONTO)** Considere as estruturas de dados **Fila**, **Pilha** e **Lista Encadeada**, suas operações e características. Assinale a alternativa correta:

   (a) A **Fila** é uma estrutura de dados onde os elementos são inseridos e removidos do topo, seguindo o princípio de Last In, First Out (LIFO).

   (b) A **Pilha** segue o princípio de First In, First Out (FIFO), onde a inserção é feita no final e a remoção no início.

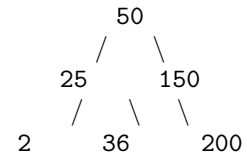
   (c) A **Lista Encadeada** permite inserções e remoções eficientes em qualquer posição (início, meio ou fim), mas não oferece a flexibilidade de acesso aleatório como os arrays.

   (d) A **Pilha** pode ser implementada apenas com listas encadeadas e não pode ser implementada com arrays.

   (e) Na **Fila**, a inserção é feita no início e a remoção é feita no final, seguindo o princípio de Last In, First Out (LIFO).

2 Árvores

1. **(2 PONTO)** Considere uma árvore binária de busca (BST) na qual a altura é definida como o número de arestas no caminho mais longo da raiz até uma folha. Considere a seguinte árvore binária de busca:



   (a) Insira os valores `32, 18, 10, 6, 42, 116, 350, 430, 7` na árvore, um por vez, seguindo as regras de uma árvore binária de busca. Desenhe a árvore resultante após cada inserção.

   (b) Calcule a altura da árvore antes e depois da inserção de todos os valores.

   (c) Liste o caminho da raiz até a folha mais distante na árvore final.

2. **(1 PONTO)** Considere a árvore binária de busca final obtida após as inserções da questão anterior. Imprima os nós da árvore nas seguintes ordens de travessia:

   (a) Pré-ordem (raiz, subárvore esquerda, subárvore direita).

   (b) Em ordem (subárvore esquerda, raiz, subárvore direita).

   (c) Pós-ordem (subárvore esquerda, subárvore direita, raiz).

3. **(1 PONTO)** Discuta por que uma árvore binária de busca balanceada (como uma árvore AVL) é mais eficiente em termos de busca em comparação a uma árvore binária de busca desbalanceada. Demonstre a complexidade em relação a altura das árvores.

4. **(1 PONTO)** Considere as estruturas de dados **Árvore**, **Árvore de Busca Binária** e **Árvore AVL**, suas propriedades e operações. Assinale as alternativas corretas:

   (a) Uma **Árvore de Busca Binária** (ABB) é uma árvore binária onde, para cada nó, todos os elementos à esquerda são menores e todos os elementos à direita são maiores que o valor armazenado nesse nó.

   (b) Uma **Árvore AVL** é uma árvore binária de busca auto-balanceada, onde a diferença de altura entre as subárvores esquerda e direita de qualquer nó não pode ser maior que 2.

   (c) Em uma **Árvore de Busca Binária**, as operações de busca, inserção e remoção têm complexidade $O(\log n)$ no melhor caso, mas podem ter complexidade $O(n)$ no pior caso, quando a árvore está desbalanceada.

   (d) A **Árvore AVL** pode ser desbalanceada em sua estrutura, mas sempre realiza operações de balanceamento após inserções ou remoções para manter a propriedade de balanceamento de altura.

   (e) A principal vantagem da **Árvore AVL** em relação a uma **Árvore de Busca Binária** simples é que ela mantém a árvore balanceada, garantindo que as operações de busca, inserção e remoção tenham complexidade $O(\log n)$ no pior caso.

   (f) Uma **Árvore de Busca Binária** pode ter uma altura de $O(n)$ no pior caso, o que ocorre quando a árvore se torna degenerada, formando uma lista encadeada.

   (g) A **Árvore AVL** é mais eficiente do que a **Árvore de Busca Binária** simples em termos de complexidade de tempo nas operações de inserção e remoção, pois mantém a árvore balanceada.

5. **(2 PONTOS)** As Árvores AVL são árvores binárias de busca balanceadas. Esse balanceamento é mantido por meio de rotações que ocorrem após inserções ou remoções que desbalanceiam a árvore.

   (a) Implemente, em linguagem de sua preferência, as operações de rotação simples à esquerda, rotação simples à direita, rotação dupla à esquerda e rotação dupla à direita em uma árvore AVL.

   (b) Explique o papel das rotações no balanceamento da árvore AVL. Em que situações cada tipo de rotação é necessária? Ilustre com exemplos.

   (c) Dada a sequência de inserções: $35, 2, 25, 3, 50, 23, -5, 0, 45, 30, 22, 39, 41$, represente a árvore AVL resultante após todas as rotações necessárias. Apresente a estrutura final da árvore.

   (d) Com base na propriedade de balanceamento da árvore AVL, analise e justifique as complexidades de tempo das operações de inserção, remoção e busca.

   (e) Suponha que os elementos $15, 20, 4, 40, 50, 3, 1, 0, 2$ sejam inseridos sequencialmente em uma árvore AVL inicialmente vazia. Descreva a sequência de rotações que ocorre ao longo do processo de inserção.