UniSENAI

Trilha de Estrutura de Dados

Módulo 02: Estrutura de Dados Lineares

Instruções para a melhor prática de Estudo

1. **Leia atentamente todo o conteúdo:** Antes de iniciar qualquer atividade, faça uma leitura detalhada do material fornecido na trilha, compreendendo os conceitos e os exemplos apresentados.

2. **Não se limite ao material da trilha:** Utilize o material da trilha como base, mas busque outros materiais de apoio, como livros, artigos acadêmicos, vídeos, e blogs especializados. Isso enriquecerá o entendimento sobre o tema.

3. **Explore a literatura:** Consulte livros e publicações reconhecidas na área, buscando expandir seu conhecimento além do que foi apresentado. A literatura acadêmica oferece uma base sólida para a compreensão de temas complexos.

4. **Realize todas as atividades propostas:** Conclua cada uma das atividades práticas e teóricas, garantindo que você esteja aplicando o conhecimento adquirido de maneira ativa.

5. **Evite o uso de Inteligência Artificial para resolução de atividades:** Utilize suas próprias habilidades e conhecimentos para resolver os exercícios. O aprendizado vem do esforço e da prática.

6. **Participe de debates:** Discute os conteúdos estudados com professores, colegas e profissionais da área. O debate enriquece o entendimento e permite a troca de diferentes pontos de vista.

7. **Pratique regularmente:** Não deixe as atividades para a última hora. Pratique diariamente e revise o conteúdo com frequência para consolidar o aprendizado.

8. **Peça feedback:** Solicite o retorno dos professores sobre suas atividades e participe de discussões sobre os erros e acertos, utilizando o feedback para aprimorar suas habilidades.

Essas instruções são fundamentais para garantir um aprendizado profundo e eficaz ao longo das trilhas.

Estruturas de Dados Lineares

1. Vetores

Definição: Vetores, também conhecidos como arrays, são coleções de elementos de mesmo tipo, armazenados de forma contígua na memória. Cada elemento é acessado por um índice.

- **Alocação de Memória**: A memória é alocada de forma contínua, ou seja, os elementos ocupam endereços adjacentes na memória.
- **Operações**:
    - **Inserção**: A inserção de um elemento em um vetor é simples quando se trata de adicionar ao final. No entanto, para inserir em qualquer outra posição, pode ser necessário deslocar os elementos subsequentes, o que impacta o desempenho.
    - **Remoção**: Semelhante à inserção, remover um elemento de qualquer posição pode exigir o deslocamento dos elementos subsequentes para preencher o espaço vazio.
    - **Busca**: A busca em um vetor pode ser linear (O(n)) no caso de um vetor desordenado, ou binária (O(log n)) se o vetor estiver ordenado.

Quando usar vetores

- Quando o tamanho é fixo ou pouco variável
- Quando acesso rápido por índice é essencial
- Quando o custo de inserções/remoções não é crítico

Exemplo de vetor

```
let numbers = [10, 20, 30, 40, 50];

console.log(numbers[2]);

// Deve imprimir o número 30
```

---

2. Listas Simplesmente e Duplamente Encadeadas

Definição: Uma **lista encadeada** é uma estrutura de dados composta por nós, onde cada nó contém um valor e uma referência (ponteiro) para o próximo nó da lista.

- **Lista Simplesmente Encadeada**:
  Cada nó aponta apenas para o próximo nó na sequência. A inserção e a remoção podem ser feitas facilmente, pois os elementos não precisam ser deslocados, apenas os ponteiros são ajustados.
  Operações:
    - **Inserção**: Pode ocorrer no início, meio ou fim da lista.

    - **Remoção**: Exige a localização do nó anterior ao que será removido, para ajustar os ponteiros.
    - **Busca**: A busca é linear, pois os elementos precisam ser percorridos de nó em nó.
- **Lista Duplamente Encadeada**:
  Cada nó contém dois ponteiros: um para o próximo nó e outro para o nó anterior. Isso permite percorrer a lista em ambas as direções, facilitando a navegação e a remoção de elementos.
  Operações:
    - **Inserção**: Pode ocorrer no início, meio ou fim, com a atualização de dois ponteiros (anterior e próximo).
    - **Remoção**: Semelhante à lista simplesmente encadeada, mas com a atualização de dois ponteiros.
    - **Busca**: A busca é linear, semelhante à lista simplesmente encadeada.

Quando usar listas encadeadas

- Quando há muitas inserções e remoções
- Quando o tamanho da coleção varia bastante
- Quando acesso sequencial é suficiente

Exemplo de Lista Simplesmente Encadeada (Em Pseudocódigo):

```
Lista: 10 → 20 → 30 → null
Cada número é um nó.
Cada seta representa a referência para o próximo elemento.
```

Explicação curta:

- O primeiro elemento é o **head**
- Cada nó guarda:
- um valor
- uma referência para o próximo
- O último nó aponta para `null`

Inserção no início:

```
Antes: 10 → 20 → 30 → null
Inserir 5
Depois: 5 → 10 → 20 → 30 → null
```

Pseudocódigo funcional

Como podemos ver na imagem a seguir se trata de uma função para inserir um número ao início da lista:

```javascript
// Inserir no Início da Lista

let head = null; // Inicializa a lista vazia

function insertAtBeginning(value) {
    const newNode = {
        value: value,
        next: head
    };

    head = newNode;
}
```

📌 **O que acontece aqui:**

1. Criamos um novo nó
2. Ele aponta para o antigo `head`
3. Atualizamos o `head` para o novo nó

A seguir temos um exemplo de inserção de um número ao final da lista:

```javascript
// Inserir no Fim da Lista

function insertAtEnd(value) {
    const newNode = {
        value: value,
        next: null
    };

    if(head === null) {
        head = newNode;
    } else {
        let temp = head;

        while(temp.next !== null) {
            temp = temp.next;
        }

        temp.next = newNode;
    }
}
```

📌 **O que acontece aqui:**

- Cria um novo nó
- Se a lista estiver vazia, o novo nó vira o `head`
- Caso contrário, percorre a lista até encontrar o último nó
- Faz o último nó apontar para o novo nó

---

3. Pilhas (Stacks)

Definição: Uma **pilha (stack)** é uma estrutura de dados que segue o princípio **LIFO** (Last In, First Out), onde o último elemento a entrar é o primeiro a sair.

Operações:

- **Push**: Adiciona um elemento ao topo da pilha.
- **Pop**: Remove o elemento no topo da pilha.
- **Top/Peek**: Visualiza o elemento no topo sem removê-lo.

Aplicações:

- Controle de chamadas de funções.
- Desfazer operações em editores de texto.
- Avaliação de expressões aritméticas.

Exemplo de Pilha (Em Pseudocódigo):

Estrutura da Pilha

```
let stack = []; // Pilha vazia
```

📌 A pilha será representada por um vetor, onde:

- o **final do vetor** representa o topo da pilha

Inserir Elemento na Pilha (Push)

```
function push(value) {
    stack.push(value);
}
```

👉 Adiciona um elemento **no topo da pilha**
📈 Complexidade: **O(1)**

Remover Elemento da Pilha (Pop)

```
function pop() {
    if(stack.length === 0) {
        return null; // Pilha vazia
    }

    return stack.pop();
}
```

👉 Remove e retorna o elemento do topo da pilha
📈 Complexidade: **O(1)**

Consultar o Elemento do Topo (Top / Peek)

```
function top() {
    if(stack.length === 0) {
        return null; // Pilha vazia
    }

    return stack[stack.length - 1];
}
```

👉 Retorna o elemento do topo **sem removê-lo**
📈 Complexidade: **O(1)**

---

4. Filas (Queues)

Definição: Uma **fila (queue)** é uma estrutura de dados que segue o princípio **FIFO** (First In, First Out), onde o primeiro elemento a entrar é o primeiro a sair.

Tipos de Filas:

- **Fila Simples**: Elementos entram no final e saem pelo início.
- **Fila Circular**: O último elemento é conectado ao primeiro, formando um ciclo.
- **Fila com Prioridades**: Cada elemento tem uma prioridade e a saída é feita com base nela, não necessariamente na ordem de chegada.

Operações:

- **Enqueue**: Insere um elemento no final da fila.
- **Dequeue**: Remove o elemento do início da fila.

Exemplo de Fila (Em Pseudocódigo):

```
let queue = []; // Fila vazia
```

📌 A fila será representada por um vetor, onde:

- o **início do vetor** representa o primeiro elemento da fila
- o **final do vetor** representa o último elemento

Inserir Elemento na Fila (Enqueue)

```
function enqueue(value) {
    queue.push(value);
}
```

👉 Adiciona um elemento **no final da fila**
📈 Complexidade: **O(1)**

Remover Elemento da Fila (Dequeue)

```
function dequeue() {
    if(queue.length === 0) {
        return null; // Fila vazia
    }

    return queue.shift(); // Remove o primeiro elemento
}
```

👉 Remove e retorna o **primeiro elemento da fila**
📈 Complexidade: **O(n)** (há deslocamento dos elementos)

Observação: Pilhas e **filas** são **Tipos Abstratos de Dados**, que podem ser implementados usando vetores ou listas encadeadas.

Fechamento conectando com mundo real (Dev / Backend / Sistemas)

Estruturas de dados lineares são amplamente utilizadas em sistemas reais, como filas de processamento, controle de requisições, gerenciamento de histórico, execução de tarefas assíncronas e estruturas internas de frameworks e bancos de dados. Dominar essas estruturas é essencial para escrever código eficiente, escalável e fácil de manter.

Lista de Exercícios de Fixação

1. **Vetores**:
    - Crie um vetor que armazene 10 números inteiros e desenvolva uma função para buscar um número específico no vetor.
    - Implemente uma função para remover um elemento do vetor em uma posição específica.
2. **Lista Simplesmente Encadeada**:
    - Implemente uma lista simplesmente encadeada com as seguintes operações: inserir no início, inserir no final e remover de uma posição específica.
    - Modifique o código anterior para permitir a busca de um elemento por valor.
3. **Lista Duplamente Encadeada**:
    - Implemente uma lista duplamente encadeada com as operações de inserir no início e remover do final da lista.
    - Crie uma função que percorra a lista em ambas as direções, imprimindo os valores dos nós.
4. **Pilha (Stack)**:
    - Implemente uma pilha e adicione operações para verificar se a pilha está cheia ou vazia.
    - Utilize uma pilha para verificar se uma expressão aritmética contém parênteses balanceados (exemplo: `((1+2) * (3/4))`).
5. **Fila (Queue)**:
    - Crie uma fila e implemente as operações de `enqueue` e `dequeue`.
    - Modifique o código para implementar uma fila circular.
    - Desenvolva um programa que simule o atendimento de um banco utilizando uma fila simples.