UniSENAI

Trilha 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:** Discuta 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.

---

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.

Exemplo de Lista Simplesmente Encadeada (Em Pseudocódigo):

```
class Node:
  data  // Dados do nó
  next  // Ponteiro para o próximo nó

class LinkedList:
  head = None  // Inicializa a lista vazia

  function insertAtBeginning(value):
    newNode = new Node(value)
    newNode.next = head  // O novo nó aponta para o antigo head
    head = newNode       // O head agora é o novo nó

  function insertAtEnd(value):
    newNode = new Node(value)
    if head is None:
      head = newNode
    else:
      temp = head
      while temp.next is not None:
        temp = temp.next
      temp.next = newNode
```

---

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):

```
class Stack:
  stack = []

  function push(value):
    stack.append(value)

  function pop():
    if stack is not empty:
      return stack.pop()
    else:
      return None  // Pilha vazia

  function top():
    if stack is not empty:
      return stack[-1]  // Último elemento da pilha
    else:
      return None
```

---

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):

```
class Queue:
  queue = []

  function enqueue(value):
    queue.append(value)

  function dequeue():
    if queue is not empty:
      return queue.pop(0)  // Remove o primeiro elemento
    else:
      return None  // Fila vazia
```

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.