UniSENAI

Trilha de Estrutura de Dados

Módulo 04: Estrutura de Dados não 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 Não Lineares

Estruturas de dados não lineares são aquelas em que os elementos não estão organizados em uma sequência única, mas sim em forma hierárquica ou em rede, permitindo múltiplos caminhos entre os dados.

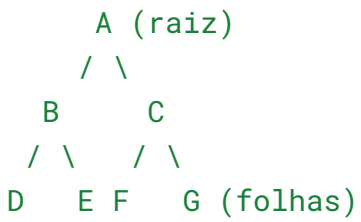
---

1. Árvores

Conceito e Terminologias:
Uma **árvore** é uma estrutura hierárquica composta por **nós**, onde o primeiro nó é chamado de **raiz**, e cada nó pode ter filhos, formando uma relação de hierarquia. Diferentes tipos de árvores possuem variações em suas propriedades.

- **Raiz**: O nó no topo da árvore.
- **Folhas**: Nós que não possuem filhos.
- **Altura**: O número máximo de níveis da árvore (da raiz à folha mais distante).
- **Profundidade**: Número de arestas da raiz até o nó.
- **Nível**: Posição do nó em relação à raiz (raiz está no nível 0 ou 1, dependendo da convenção).

Exemplo Visual:



---

2. Árvores Binárias

Uma **árvore binária** é uma árvore onde cada nó tem no máximo dois filhos, denominados **filho esquerdo** e **filho direito**.

- **Percurso em Árvores Binárias**: O percurso de uma árvore é a maneira de visitar cada nó de forma sistemática. Existem três tipos principais:
  - **In-Order**: Primeiro visita a subárvore esquerda, depois o nó atual, e por último a subárvore direita.
  - **Pre-Order**: Primeiro visita o nó atual, depois a subárvore esquerda, e por último a subárvore direita.
  - **Post-Order**: Primeiro visita a subárvore esquerda, depois a subárvore direita, e por último o nó atual.

Exemplo de Percurso In-Order:

```
Para a árvore:
        1
       / \
      2   3
     / \
    4   5
In-order: 4, 2, 5, 1, 3
```

---

3. Árvores Binárias de Busca (BST)

Uma **árvore binária de busca (BST)** é uma árvore binária onde, para cada nó, todos os nós na subárvore à esquerda têm valores menores, e todos os nós na subárvore à direita têm valores maiores.

- **Operações**:
  - **Inserção**: Um novo valor é inserido de acordo com a propriedade da BST.
  - **Remoção**: O nó a ser removido e substituído de forma que a propriedade da BST seja mantida.
  - **Busca**: A busca é realizada comparando o valor do nó com o valor procurado, indo para a esquerda ou direita até encontrar o valor ou atingir uma folha.

Árvores binárias de busca permitem realizar operações de busca, inserção e remoção de forma mais eficiente do que estruturas lineares, especialmente quando bem balanceadas.

---

4. Árvores Balanceadas (AVL e Red-Black)

Árvores AVL e **Red-Black** são tipos de árvores binárias de busca que mantêm a árvore balanceada, de forma que o tempo de busca, inserção e remoção seja garantido como O(log n).

- **Árvore AVL**: A altura das subárvores de cada nó pode diferir em no máximo 1. Se uma inserção ou remoção desequilibrar a árvore, rotações são usadas para restaurar o equilíbrio.
- **Árvore Red-Black**: Mantém uma propriedade de cores para os nós (vermelho e preto) e equilibra a árvore durante as operações de inserção e remoção.

---

5. Heaps e Árvores de Segmento

- **Heaps**: Uma estrutura especial de árvore binária usada principalmente para implementar filas de prioridade. Em um **max-heap**, o valor de cada nó é maior que os valores de seus filhos.

- **Árvores de Segmento**: Estruturas que permitem calcular funções como somas ou mínimos em intervalos dinâmicos de uma sequência de números. Úteis em problemas de intervalos, como no cálculo de frequências em uma faixa específica.

Observação: Heaps e árvores de segmento são estruturas mais especializadas, normalmente abordadas em problemas de otimização, filas de prioridade e algoritmos avançados.

---

6. Grafos

Conceitos e Representações: Um **grafo** é um conjunto de **vértices (nós)** conectados por **arestas**. Os grafos são usados para modelar relações e conexões, como redes de transporte, mapas e circuitos.

- **Representações**:
  - **Lista de Adjacência**: Cada nó é associado a uma lista de nós conectados a ele.
  - **Matriz de Adjacência**: Uma matriz 2D onde cada posição indica a existência de uma aresta entre dois nós.

Tipos de Grafos:

- **Simples**: Não possui laços ou arestas múltiplas.
- **Direcionado**: As arestas têm uma direção, do nó de origem para o de destino.
- **Ponderado**: As arestas possuem pesos associados.
- **Não Ponderado**: As arestas não possuem peso.

Exemplo Visual de Grafo	Lista de Adjacência
1 --- 2 | | 3 --- 4	1: 2, 3 2: 1, 4 3: 1, 4 4: 2, 3

Grafos são amplamente utilizados em sistemas de rotas, redes sociais, redes de computadores, dependências de tarefas e algoritmos de recomendação.

---

7. Algoritmos de Busca em Grafos

- **DFS (Depth-First Search)**: Explora o grafo começando por um vértice e vai o mais fundo possível antes de retroceder. usa **pilha** (explícita ou recursão)
- **BFS (Breadth-First Search)**: Explora o grafo camada por camada, visitando todos os vértices de um nível antes de passar para o próximo. usa **fila**

---

8. Algoritmos para Caminhos Mínimos

- **Dijkstra**: Encontra o caminho mais curto de um nó a todos os outros nós em um grafo com arestas de pesos não negativos.
- **Floyd-Warshall**: Calcula os caminhos mínimos entre todos os pares de nós em um grafo.

Nota

O estudo das estruturas de dados não lineares é fundamental para resolver problemas complexos que envolvem hierarquia (árvores) e conectividade (grafos). A implementação desses conceitos permite construir algoritmos eficientes para buscas, ordenações, e otimizações em diversos cenários.

Lista de Exercícios de Fixação

1. **Árvores Binárias**:
    - Implemente uma árvore binária e crie as funções para realizar os percursos *in-order*, *pre-order* e *post-order*.
    - Dado um conjunto de números, construa uma árvore binária de busca (BST) e implemente funções para inserir, buscar e remover elementos da árvore.
2. **Árvores AVL**:
    - Implemente uma árvore AVL com as operações de inserção e remoção, garantindo que a árvore permaneça balanceada após cada operação.
3. **Heaps**:
    - Implemente um **max-heap** e escreva funções para inserir um novo elemento e remover o maior elemento.
    - Use um heap para implementar uma fila de prioridades que sempre retorna o maior valor.
4. **Grafos**:
    - Crie um grafo simples utilizando uma lista de adjacência e implemente os algoritmos de busca **DFS** e **BFS**.
    - Modifique o grafo para ser direcionado e implemente o algoritmo de Dijkstra para encontrar o caminho mais curto entre dois nós.
5. **Caminhos Mínimos**:
    - Implemente o algoritmo de **Dijkstra** para um grafo ponderado e calcule o caminho mais curto de um vértice para os demais.
    - Utilize o algoritmo de **Floyd-Warshall** para calcular o caminho mais curto entre todos os pares de nós em um grafo.