

Árvore AVL

Autor: Katarine Meira
Nicolle Khetlem
Luana Magalhães
Pedro Henrique Ferreira
Samily Sena
Tarscilla Almeida

Vitória da Conquista, 2025

Agenda

Este trabalho tem como objetivo, apresentar o desenvolvimento de uma árvore AVL, em um sistema de pedidos em Python para um restaurante fictício.

1. O que é uma árvore AVL:

conceito de árvore AVL como uma árvore binária de busca autobalanceada.

2. Balanceamento:

Explica o fator de balanceamento e como a AVL mantém a diferença de altura entre subárvores sempre menor ou igual a 1.

3. Rotação:

Mostra as rotações simples e duplas usadas para corrigir desbalanceamentos.

4. Implementação:

Apresenta a implementação da árvore AVL na prática.

5. Considerações Finais:

Destaca a importância das árvores AVL na garantia de desempenho consistente.

6. Referências:

Indica as fontes utilizadas para embasar o conteúdo apresentado sobre árvores AVL.

O que é uma árvore AVL?

Uma árvore AVL é um tipo de árvore binária de busca, possuindo a propriedade de autobalanceamento dinâmico, ela é uma estrutura de dados compostas de nós tendo as seguintes garantias:

1. Cada árvore possui um nó raiz (no topo)
2. O nó raiz tem zero, um ou dois filhos
3. Cada nó filho tem zero, um ou dois nós filhos.
4. Cada nó tem até dois filhos
5. Para cada nó, seus descendentes da esquerda são menores que o nó atual, que é menor que o descendentes da direita

O que é uma árvore AVL?

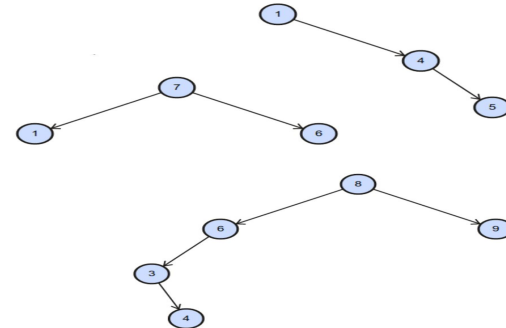
As árvores AVL têm uma garantia adicional:

1. A diferença entre as profundidade das camadas do lado direito e esquerdo não pode ser maior que um. Essa diferença é chamada de fator de balanço.
2. Para manter essa garantia, a implementação de uma AVL incluirá um algoritmo para balanceamento da árvore quando a adição de um elemento prejudicar essa garantia.



-

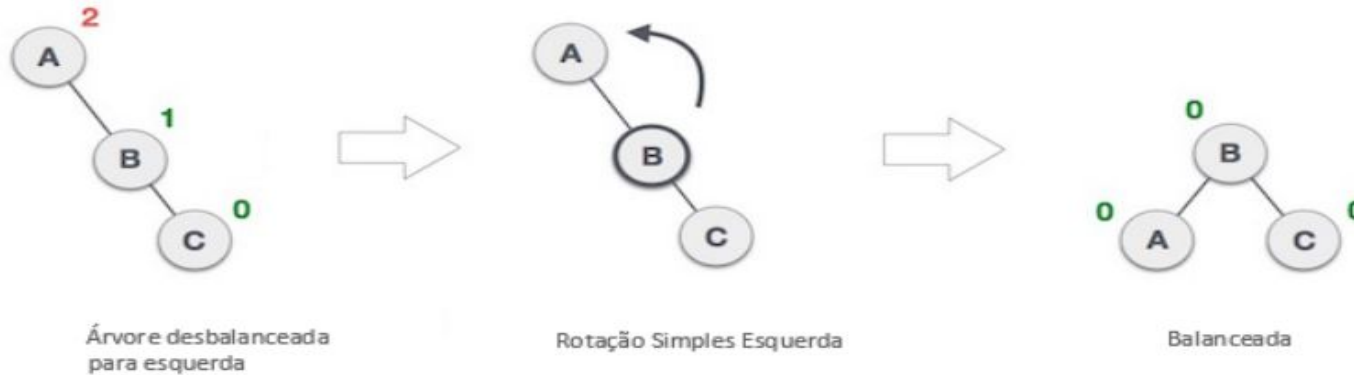
Árvore AVL



5

Rotação

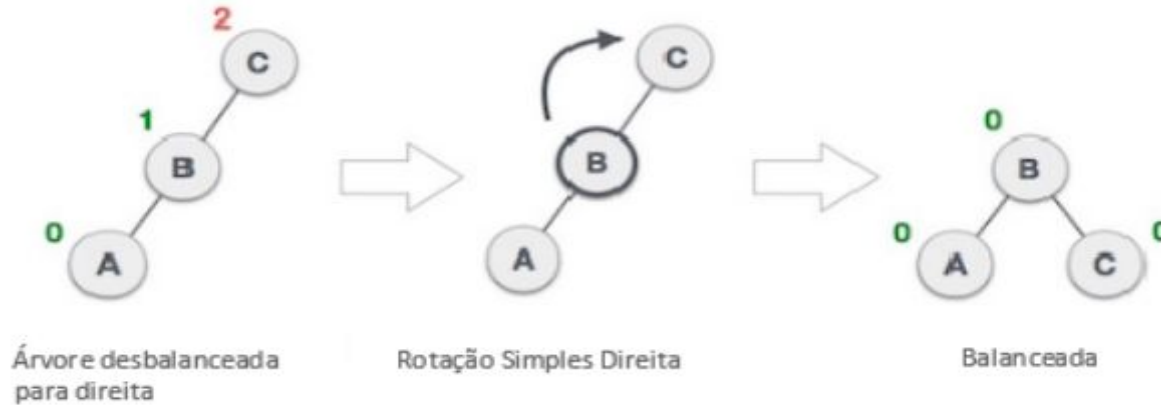
- **Simple à esquerda (rotação SE)**
 - Cada nó se move uma posição para a direita da posição atual.
- **Dupla à esquerda (rotação DE)**
 - São uma combinação de uma única rotação para a direita seguida de uma rotação para a esquerda.



Rotação

- **Simple à direita (rotação SD)**

- Cada nó se move uma posição para a direita da posição atual.



- **Dupla à direita (rotação DD)**

- São uma combinação de uma única rotação para a esquerda seguida de uma rotação para a direita.

Implementação:

- **Árvore AVL (Pedidos)**

```
def inserir(self, raiz, id_pedido, dados_pedido):
    if raiz is None:
        return NoAVL(id_pedido, dados_pedido)

    if id_pedido < raiz.id:
        raiz.esquerda = self.inserir(raiz.esquerda, id_pedido, dados_pedido)
    elif id_pedido > raiz.id:
        raiz.direita = self.inserir(raiz.direita, id_pedido, dados_pedido)
    else:
        raiz.dados = dados_pedido
        return raiz

    raiz.altura = 1 + max(self.altura(raiz.esquerda), self.altura(raiz.direita))
    bal = self.balanceamento(raiz)
```

```
    if bal > 1 and id_pedido < raiz.esquerda.id:
        return self.rotacao_direita(raiz)
    if bal < -1 and id_pedido > raiz.direita.id:
        return self.rotacao_esquerda(raiz)
    if bal > 1 and id_pedido > raiz.esquerda.id:
        raiz.esquerda = self.rotacao_esquerda(raiz.esquerda)
        return self.rotacao_direita(raiz)
    if bal < -1 and id_pedido < raiz.direita.id:
        raiz.direita = self.rotacao_direita(raiz.direita)
        return self.rotacao_esquerda(raiz)

    return raiz
```

A função se chama recursivamente até encontrar o local onde deve inserir.

Implementação:

- **Balanceamento**

```
def balanceamento(self, no):  
    if no is None:  
        return 0  
    return self.altura(no.esquerda) - self.altura(no.direita)
```

Depois de inserir um nó, a árvore verifica se algum nó ficou inclinado demais para um lado.

- **Rotação**

```
def rotacao_direita(self, y):  
    x = y.esquerda  
    t2 = x.direita  
    x.direita = y  
    y.esquerda = t2  
    y.altura = 1 + max(self.altura(y.esquerda), self.altura(y.direita))  
    x.altura = 1 + max(self.altura(x.esquerda), self.altura(x.direita))  
    return x  
  
def rotacao_esquerda(self, x):  
    y = x.direita  
    t2 = y.esquerda  
    y.esquerda = x  
    x.direita = t2  
    x.altura = 1 + max(self.altura(x.esquerda), self.altura(x.direita))  
    y.altura = 1 + max(self.altura(y.esquerda), self.altura(y.direita))  
    return y
```

Implementação:

- **Buscar**

```
def buscar(self, raiz, id_pedido):  
    if raiz is None:  
        return None  
    if id_pedido == raiz.id:  
        return raiz.dados  
    if id_pedido < raiz.id:  
        return self.buscar(raiz.esquerda, id_pedido)  
    return self.buscar(raiz.direita, id_pedido)
```

Procura um pedido percorrendo a árvore.

- **Carregar**

```
def carregar_pedidos(self):  
    dados = carregar_json(CAMINHO_JSON_PEDIDOS)  
  
    if not isinstance(dados, list):  
        dados = []  
  
    self.raiz = None  
    for pedido in dados:  
        self.raiz = self.inserir(self.raiz, pedido["id_pedido"], pedido["dados"])
```

reconstrói a árvore inteira a partir do JSON
sempre que o programa inicia.

Considerações finais



- **Garantia de Balanceamento Automático:** árvores AVL mantêm sua estrutura sempre equilibrada.
- **Desempenho Consistente:** Possuem complexidade de tempo $O(\log n)$.
- **Alta Eficiência em Consultas:** São vantajosas onde a busca é frequente, como em aplicações de banco de dados e estruturas de indexação.
- **Mecanismo de Rotações Robusto:** Rotação simples e dupla garantem o equilíbrio rapidamente após inserções e remoções.

Referências



PORTAL DO PROFESSOR – FCT/UNESP. Cadilag – Árvore AVL. Disponível em:
https://portaldoprofessor.fct.unesp.br/projetos/cadilag/apps/structs/arv_avl.php. Acesso em: 23 nov. 2025.

VIEIRA, Aldo. Simulador de Árvore AVL – UFSC. Disponível em:
<https://www.inf.ufsc.br/~aldo.vw/estruturas/simulador/AVL.html>. Acesso em: 23 nov. 2025.

GALLES, David. AVL Tree Visualization – University of San Francisco. Disponível em:
<https://www.cs.usfca.edu/~galles/visualization/AVLtree.html>. Acesso em: 23 nov. 2025.