

Implementação de uma Árvore AVL

Autor: Samuel Nascimento Santos
Joice da Rocha Silveira
Lorena Dias Freitas
Manuela de Oliveira Figueira
Rafaela Canguçu Souza

Agenda

O objetivo dessa apresentação é
Explicar a implementação da
Árvore AVL.

1. O que é uma Árvore AVL:

Árvore binária de busca
que se auto balanceia
automaticamente para
manter operações rápidas
e eficientes.

2. Inserção e balanceamento:

Inserir o nó e aplicar rotações se
a árvore ficar desequilibrada.

3. Implementação na prática:

Aplicação da árvore AVL
no sistema de pedidos.

4. Considerações Finais:

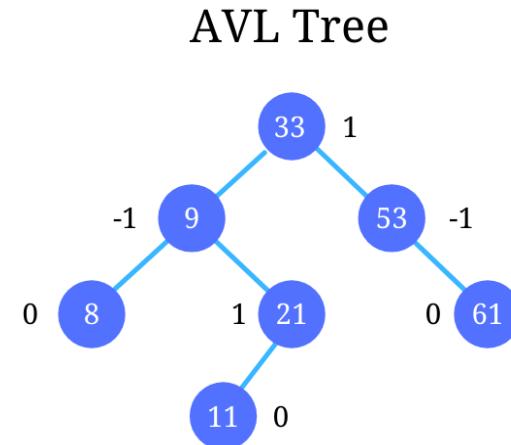
Eficiência da árvore de
pesquisa, desempenho e
como se adequa.

5. Referências:

Fonte de buscas e estudos
para entendimento da
árvore AVL.

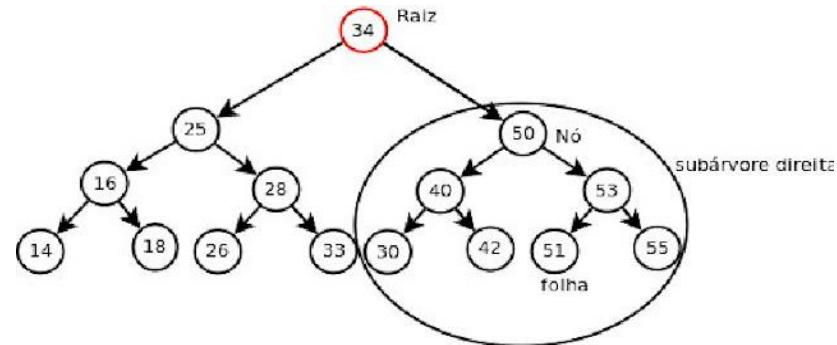
O que é uma Árvore AVL:

- É uma árvore binária de busca (BST).
- Mantém a altura equilibrada automaticamente.
- Quando desbalanceada, realiza rotações para corrigir.
- Garante que FB de cada nó esteja entre -1 e 1.
- Mantém operações eficientes: busca, inserção e remoção em $O(\log n)$.



Inserção e balanceamento:

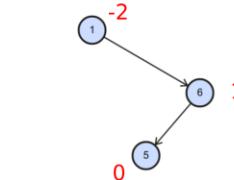
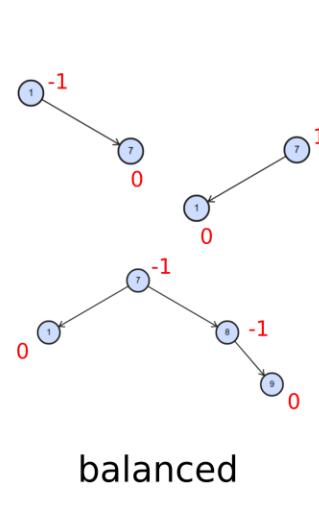
- Inserção:
 - Compare o valor com o nó atual
 - Vá para esquerda se for menor, para a direita se for maior
 - Insira um novo nó na posição correta, como em uma BST normal.



Inserção e balanceamento:

- Balanceamento:

- A árvore recalcula a altura de cada nó no caminho de volta até a raiz.
- Para cada nó, calcula o Fator Balanceamento (FB)
$$FB = \text{altura(esquerda)} - \text{altura(direita)}$$
- Se o FB de algum nó ficar fora do intervalo $[-1, +1]$, esse nó está desbalanceado



Implementação na prática:

- Árvore AVL no código:

```
# indexador_avl.py - Implementação da Árvore AVL

class No:
    """Nó da Árvore AVL. Armazena a chave e os dados (mapa)."""
    def __init__(self, chave, dados):
        self.chave = chave # Chave (e.g., código do item/pedido)
        self.dados = dados # Dados (o "mapa" - dicionário do item/pedido)
        self.altura = 1
        self.esquerda = None
        self.direita = None

class ArvoreAvl:
    """Implementação da Árvore AVL."""
    def __init__(self):
        self.raiz = None

    def _get_altura(self, no):
        """Retorna a altura do nó (0 se for None)."""
        if not no:
            return 0
        return no.altura

    def _get_fator_balanceamento(self, no):
        """Calcula o fator de balanceamento (Altura_Esquerda - Altura_Direita)."""
        if not no:
            return 0
        return self._get_altura(no.esquerda) - self._get_altura(no.direita)

    def _atualizar_altura(self, no):
        """Recalcula a altura de um nó."""
        if not no:
            return
        no.altura = 1 + max(self._get_altura(no.esquerda), self._get_altura(no.direita))
```

Implementação de uma Árvore AVL

Implementação na prática:

- Uso de balanceamento no código:

```
# 3. Calcula o balanceamento
balanceamento = self._get_fator_balanceamento(raiz)

# 4. Aplica rotações se estiver desbalanceado

# Caso Esquerda-Esquerda (LL)
if balanceamento > 1 and chave < raiz.esquerda.chave:
    return self._rotacao_direita(raiz)

# Caso Direita-Direita (RR)
if balanceamento < -1 and chave > raiz.direita.chave:
    return self._rotacao_esquerda(raiz)

# Caso Esquerda-Direita (LR)
if balanceamento > 1 and chave > raiz.esquerda.chave:
    raiz.esquerda = self._rotacao_esquerda(raiz.esquerda)
    return self._rotacao_direita(raiz)

# Caso Direita-Esquerda (RL)
if balanceamento < -1 and chave < raiz.direita.chave:
    raiz.direita = self._rotacao_direita(raiz.direita)
    return self._rotacao_esquerda(raiz)

return raiz
```

Implementação na prática:

- Uso de Funções Públicas na Árvore AVL:

```
def buscar(self, chave):
    """Busca um item pela chave e retorna o dicionário de dados (o mapa)."""
    return self._buscar_recurssivo(self.raiz, chave)

def _buscar_recurssivo(self, raiz, chave):
    """Função recursiva para busca."""
    if not raiz:
        return None
    if raiz.chave == chave:
        return raiz.dados
    elif chave < raiz.chave:
        return self._buscar_recurssivo(raiz.esquerda, chave)
    else:
        return self._buscar_recurssivo(raiz.direita, chave)

def atualizar_dados(self, chave, novo_mapa_dados):
    """Atualiza os dados de um nó existente."""
    no = self._encontrar_no(self.raiz, chave)
    if no:
        no.dados.update(novo_mapa_dados) # Atualiza o dicionário/mapa
        return True
    return False

def _encontrar_no(self, raiz, chave):
    """Encontra e retorna o objeto No."""
    if not raiz or raiz.chave == chave:
        return raiz
    elif chave < raiz.chave:
        return self._encontrar_no(raiz.esquerda, chave)
    else:
        return self._encontrar_no(raiz.direita, chave)

def percorrer_em_ordem(self):
    """Retorna todos os 'dados' (mapas) em ordem de chave."""
    resultado = []
    self.percorrer_em_ordem_recurssivo(self.raiz, resultado)
```

Implementação na prática:

- Rotação na Árvore AVL:

```
# --- Rotações ---
def _rotacao_direita(self, y):
    """Rotação simples à direita."""
    x = y.esquerda
    T2 = x.direita
    x.direita = y
    y.esquerda = T2
    self._atualizar_altura(y)
    self._atualizar_altura(x)
    return x

def _rotacao_esquerda(self, x):
    """Rotação simples à esquerda."""
    y = x.direita
    T1 = y.esquerda
    y.esquerda = x
    x.direita = T1
    self._atualizar_altura(x)
    self._atualizar_altura(y)
    return y
```

Considerações finais



- Estrutura eficiente para buscas
- Operações sempre平衡adas
- Desempenho $O(\log n)$
- Adequada para indexação de dados

Referências

- GEEKSFORGEEKS. AVL Tree. Disponível em:
<https://www.geeksforgeeks.org/avl-tree-data-structure/>
- AVL tree. Wikipedia: The Free Encyclopedia. Disponível em:
https://en.wikipedia.org/wiki/AVL_tree
- CORMEN, Thomas H. et al. *Algoritmos: teoria e prática*. 3. ed. Rio de Janeiro: Elsevier, 2012.