Árvores Binárias Balanceadas

Introdução

Em um curso de estrutura de dados, exploramos diversas formas de organizar e acessar informações. As árvores binárias balanceadas emergem como uma solução elegante para otimizar a eficiência em cenários onde a busca, inserção e remoção de dados são operações frequentes.

O que são Árvores Binárias Balanceadas?

Uma árvore binária é uma estrutura de dados hierárquica na qual cada nó possui no máximo dois filhos (esquerdo e direito). A característica distintiva das árvores binárias balanceadas é que a altura das subárvores esquerda e direita de qualquer nó difere em no máximo 1. Essa propriedade garante que a árvore não se torne "inclinada" para um lado, mantendo um formato relativamente compacto.

Vantagens em Relação a Listas

Comparadas a listas, as árvores binárias balanceadas oferecem vantagens significativas em termos de desempenho:

- **Busca Eficiente:** Em listas, a busca por um elemento pode exigir percorrer todas as posições de armazenamento de dados, no pior caso (elemento buscado não consta da lista). Em árvores binárias balanceadas, a busca é consideravelmente mais rápida, pois a cada posição visitada, metade dos dados restantes é descartada.
- **Inserção e Remoção Otimizadas:** Inserir ou remover elementos em listas pode ser custoso, especialmente no início ou meio, devido à necessidade de deslocar elementos subsequentes (no caso de implementação em arrays). Em árvores balanceadas, essas operações são realizadas de forma mais eficiente, mantendo a estrutura equilibrada.

Aplicações no Mundo Real

A versatilidade das árvores binárias balanceadas as torna valiosas em diversas aplicações:

- **Bancos de Dados:** Índices em bancos de dados frequentemente utilizam árvores balanceadas (como B-trees) para acelerar a busca por registros específicos.
- **Compiladores:** A tabela de símbolos, que armazena informações sobre variáveis e funções, pode ser implementada com árvores balanceadas para buscas rápidas durante a compilação.
- **Sistemas Operacionais:** O gerenciamento de memória pode empregar árvores balanceadas para rastrear blocos de memória livres e alocados de forma eficiente.

- **Jogos:** Em jogos que exigem ordenação de elementos (como placares), árvores balanceadas podem ser utilizadas para manter a lista de pontuações sempre organizada.

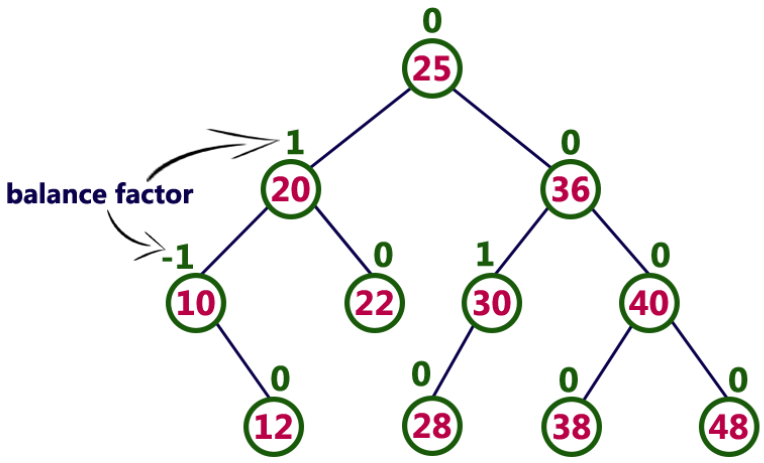
Conclusão

As árvores binárias balanceadas representam uma ferramenta poderosa no arsenal de um programador. Ao equilibrar a estrutura da árvore, elas proporcionam operações de busca, inserção e remoção eficientes, superando as limitações das listas em muitos cenários. Seu impacto se estende por diversas áreas da computação, consolidando sua importância no desenvolvimento de software moderno.

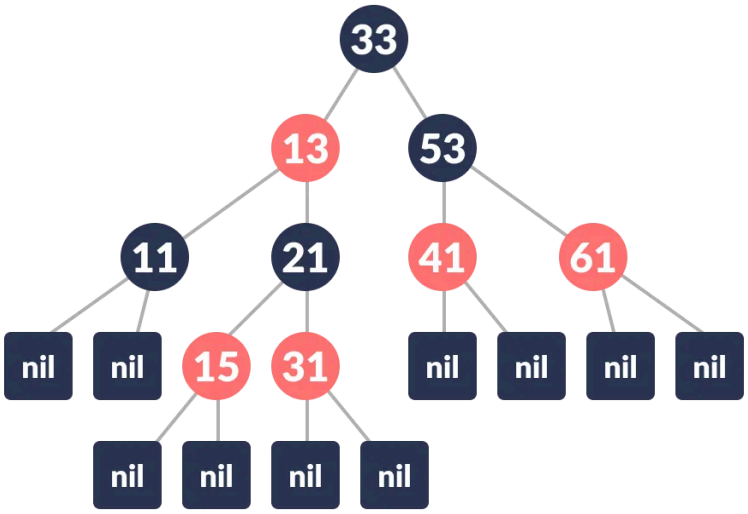
Tipos de árvores binárias balanceadas

Existem diversos tipos de árvores binárias balanceadas, cada uma com suas próprias características e mecanismos de balanceamento. Alguns dos tipos mais comuns incluem:

- **Árvores AVL (Adelson-Velsky e Landis):** As árvores AVL foram uma das primeiras estruturas de dados de árvores balanceadas a serem inventadas. Elas mantêm o equilíbrio garantindo que a diferença de altura entre as subárvores esquerda e direita de qualquer nó seja no máximo 1. O balanceamento é realizado por meio de rotações simples e duplas.

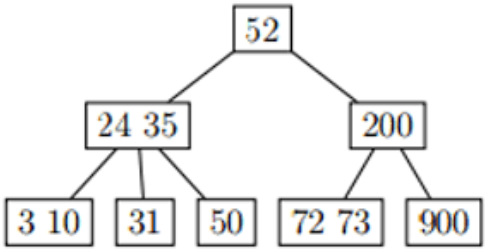


- **Árvores Vermelho-Preto:** As árvores vermelho-preto utilizam um bit adicional em cada nó para indicar sua "cor" (vermelho ou preto). Elas seguem um conjunto de regras que garantem o balanceamento, como: a raiz é sempre preta, todos os caminhos da raiz até uma folha possuem o mesmo número de nós pretos, e nenhum caminho pode ter dois nós vermelhos consecutivos.

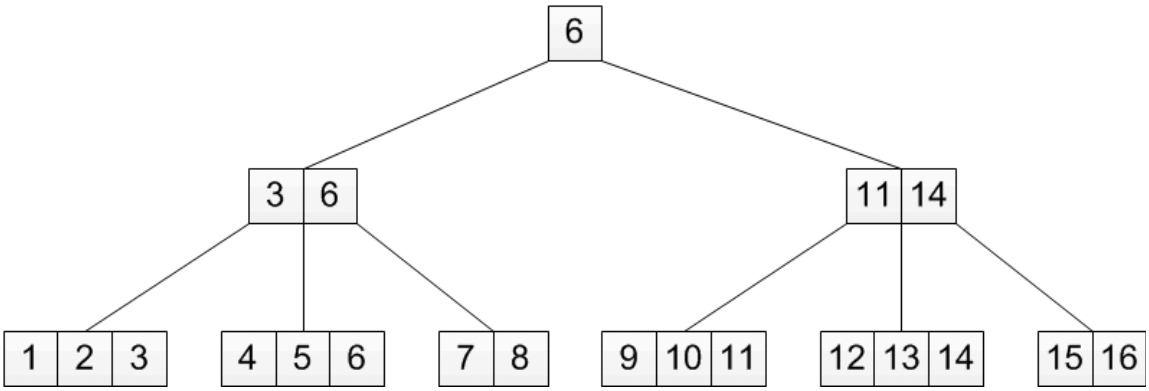


- **Árvores B:** As árvores B são um tipo de árvore balanceada otimizada para sistemas de armazenamento em disco. Elas permitem que cada nó armazene múltiplas chaves e filhos, reduzindo o número de acessos ao disco necessários para realizar operações.

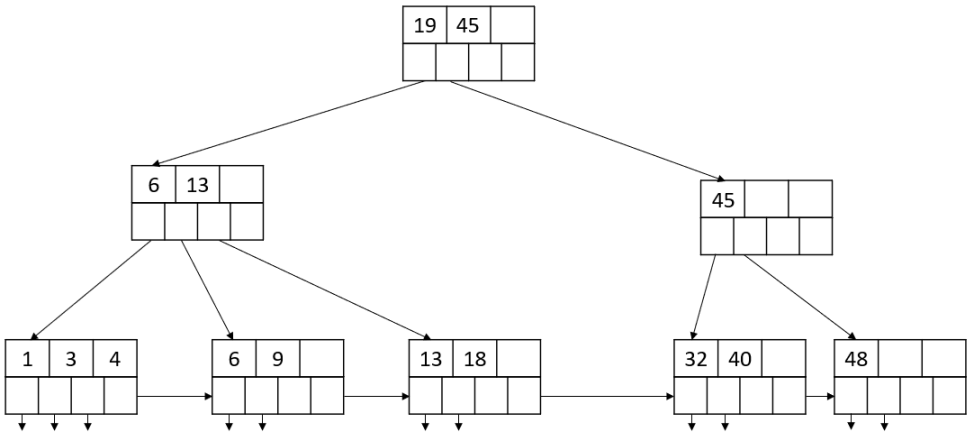
Árvore B com ordem 3



Árvore B com ordem 4



- **Árvores B+:** As árvores B+ são uma variação das árvores B que armazenam todas as chaves nas folhas, e as folhas são ligadas em uma lista encadeada, facilitando a realização de varreduras sequenciais.



Cada tipo de árvore balanceada possui suas vantagens e desvantagens, e a escolha da estrutura mais adequada depende das necessidades específicas da aplicação.

Árvores AVL: Equilíbrio e Eficiência na Busca

Um Pouco de História

As árvores AVL, nomeadas em homenagem a seus criadores, Georgy Adelson-Velsky e Evgenii Landis, foram uma das primeiras estruturas de dados de árvore balanceada a serem inventadas, surgindo em 1962. Elas se destacam por sua capacidade de manter um equilíbrio dinâmico, garantindo operações de busca, inserção e remoção eficientes, mesmo em cenários onde os dados inseridos não estão uniformemente distribuídos.

Características de uma Árvore AVL

Uma árvore AVL é uma árvore binária de busca que satisfaz a seguinte propriedade fundamental: para qualquer nó na árvore, a diferença de altura entre sua subárvore esquerda e sua subárvore direita (conhecida como fator de balanceamento) é no máximo 1. Essa restrição garante que a árvore não se torne excessivamente "inclinada" para um lado, mantendo uma altura relativamente baixa e, consequentemente, operações de busca rápidas.

Algoritmos Básicos de Manipulação

- **Cálculo do Fator de Balanceamento**

```
funcao calcularFatorBalanceamento(no):
```

```
    se no eh nulo:
        retorna 0
    alturaEsquerda = calcularAltura(no.esquerda)
    alturaDireita = calcularAltura(no.direita)
    retorna alturaDireita - alturaEsquerda
```

- **Rotações**

As rotações são operações fundamentais para reequilibrar a árvore após inserções ou remoções que causam desbalanceamento. Existem dois tipos principais de rotações:

- **Rotação Simples à Esquerda**

```
funcao rotacaoSimplesEsquerda(no):
    novoNoRaiz = no.direita
    no.direita = novoNoRaiz.esquerda
    novoNoRaiz.esquerda = no
    atualizarAltura(no)
    atualizarAltura(novoNoRaiz)
    retorna novoNoRaiz
```

- **Rotação Simples à Direita**

```
funcao rotacaoSimplesDireita(no):
    novoNoRaiz = no.esquerda
    no.esquerda = novoNoRaiz.direita
    novoNoRaiz.direita = no
    atualizarAltura(no)
    atualizarAltura(novoNoRaiz)
```

```
    retorna novoNoRaiz
```

- **Rotação Dupla à Esquerda**

```
funcao rotacaoDuplaEsquerda(no):

    no.direita = rotacaoSimplesDireita(no.direita)

    retorna rotacaoSimplesEsquerda(no)
```

- **Rotação Dupla à Direita**

```
funcao rotacaoDuplaDireita(no):

    no.esquerda = rotacaoSimplesEsquerda(no.esquerda)

    retorna rotacaoSimplesDireita(no)
```

- **Cálculo do Fator de Balanceamento em Árvore AVL**

```
funcao calcularFatorBalanceamento(no):

    se no for nulo:

        retornar 0  // Nó nulo tem altura 0, logo fator de
balanceamento 0

    alturaEsquerda = calcularAltura(no.esquerda)

    alturaDireita = calcularAltura(no.direita)

    retornar alturaDireita - alturaEsquerda
```

- **Inserção**

A inserção em uma árvore AVL segue o mesmo princípio da inserção em uma árvore binária de busca comum, mas após cada inserção, o fator de balanceamento de cada nó no caminho de inserção é verificado. Se algum nó ficar desbalanceado, as rotações apropriadas são aplicadas para restaurar o equilíbrio.

```
funcao inserir(raiz, chave):
    # Inserção normal em uma árvore binária de busca
    se raiz for nulo:
        retornar novoNo(chave)
    se chave < raiz.chave:
        raiz.esquerda = inserir(raiz.esquerda, chave)
    senao se chave > raiz.chave:
        raiz.direita = inserir(raiz.direita, chave)
    senao:
        retornar raiz

    # Atualizar a altura do nó ancestral
    atualizarAltura(raiz)

    # Verificar o balanceamento e realizar rotações se necessário
    balanceamento = calcularFatorBalanceamento(raiz)

    # Caso Esquerda-Esquerda
    se balanceamento > 1 e chave < raiz.esquerda.chave:
        retornar rotacaoDireita(raiz)

    # Caso Direita-Direita
    se balanceamento < -1 e chave > raiz.direita.chave:
        retornar rotacaoEsquerda(raiz)
```

```
    # Caso Esquerda-Direita
    se balanceamento > 1 e chave > raiz.esquerda.chave:
        raiz.esquerda = rotacaoEsquerda(raiz.esquerda)
        retornar rotacaoDireita(raiz)

    # Caso Direita-Esquerda
    se balanceamento < -1 e chave < raiz.direita.chave:
        raiz.direita = rotacaoDireita(raiz.direita)
        retornar rotacaoEsquerda(raiz)

    retornar raiz
```

- **Remoção**

A remoção em uma árvore AVL também segue o princípio da remoção em uma árvore binária de busca, mas após a remoção, o fator de balanceamento de cada nó no caminho de remoção é verificado e, se necessário, as rotações são aplicadas para reequilibrar a árvore.

```
funcao remover(raiz, chave):
    # 1. Remoção padrão em uma árvore binária de busca
    se raiz for nulo:
        retornar raiz  # Valor não encontrado

    se chave < raiz.chave:
        raiz.esquerda = remover(raiz.esquerda, chave)
    senao se chave > raiz.chave:
```

```
        raiz.direita = remover(raiz.direita, chave)
    senao:  # Nó encontrado
        # Caso 1: Nó folha ou com um filho
        se raiz.esquerda for nulo ou raiz.direita for nulo:
            temp = raiz.esquerda se raiz.esquerda nao for nulo senao
raiz.direita
            se temp for nulo:  # Nó folha
                temp = raiz
                raiz = nulo
            senao:  # Nó com um filho
                raiz = temp
        senao:  # Caso 2: Nó com dois filhos
            temp = encontrarMinimo(raiz.direita)  # Encontra o
sucessor
                                                  # inorder
            raiz.chave = temp.chave
            raiz.direita = remover(raiz.direita, temp.chave)

    # Se a árvore tinha apenas um nó, retorna nulo
    se raiz for nulo:
        retornar raiz

    # 2. Atualizar altura e rebalancear
    atualizarAltura(raiz)
    balanceamento = calcularFatorBalanceamento(raiz)
```

```
    # Casos de rebalanceamento (semelhantes à inserção)

    # Caso Esquerda-Esquerda

    se balanceamento > 1 e calcularFatorBalanceamento(raiz.esquerda)
>= 0:

        retornar rotacaoDireita(raiz)



    # Caso Esquerda-Direita

    se balanceamento > 1 e calcularFatorBalanceamento(raiz.esquerda)
< 0:

        raiz.esquerda = rotacaoEsquerda(raiz.esquerda)

        retornar rotacaoDireita(raiz)



    # Caso Direita-Direita

    se balanceamento < -1 e calcularFatorBalanceamento(raiz.direita)
<= 0:

        retornar rotacaoEsquerda(raiz)



    # Caso Direita-Esquerda

    se balanceamento < -1 e calcularFatorBalanceamento(raiz.direita)
> 0:

        raiz.direita = rotacaoDireita(raiz.direita)

        retornar rotacaoEsquerda(raiz)



    retornar raiz



# Função auxiliar para encontrar o nó com o valor mínimo em uma
subárvore
```

```
funcao encontrarMinimo(no):

    atual = no

    enquanto atual.esquerda nao for nulo:

        atual = atual.esquerda

    retornar atual
```

Conclusão

As árvores AVL são uma estrutura de dados poderosa e eficiente para armazenar e acessar dados de forma ordenada. Sua capacidade de se manter balanceada garante um desempenho otimizado em operações de busca, inserção e remoção, tornando-as uma escolha valiosa em diversas aplicações.