Árvores Binárias

Uma árvore binária é uma estrutura de dados hierárquica usada para representar conjuntos de objetos e suas relações em forma de uma estrutura de grafos acíclicos e conectados, essencial para problemas com hierarquia, como diretórios de arquivos, organogramas, e buscas de dados.

As árvores são compostas por nós (elementos) e arestas (conexões entre nós). Características importantes incluem:

- Raiz: nó principal, localizado no topo, sem pai.
- Pai e Filho: nós antecessor e sucessor imediato, respectivamente.
- Nó Folha: nó terminal, sem filhos.
- Nó Interno: nó não terminal, que possui pelo menos um filho.
- Caminho: sequência de nós conectados por arestas.

Essas propriedades permitem que as árvores representem estruturas não lineares e sejam aplicadas na modelagem de problemas hierárquicos em diferentes contextos, sempre garantindo um caminho único entre a raiz e qualquer outro nó.

Grau do nó e subárvores

Em uma árvore, cada nó pode ser a raiz de uma nova subárvore composta por ele e todos os nós abaixo dele. O grau de um nó refere-se ao número de subárvores que ele possui. Por exemplo, se um nó possui duas subárvores, ele terá grau 2. Além disso, nós folhas, que não possuem filhos, têm grau zero. Esse conceito ajuda a entender a estrutura da árvore e os relacionamentos entre seus elementos.

Altura e nível da árvore

Os nós de uma árvore são organizados em níveis. O nível de um nó é determinado pela quantidade de nós no caminho que o separa da raiz, sendo a raiz o nível 0. Já a altura ou profundidade da árvore representa o comprimento do caminho mais longo da raiz até uma de suas folhas, indicando o número total de níveis.

Tipos de árvore

Existem diferentes tipos de árvores que atendem a necessidades específicas:

- Árvore binária de busca: Ideal para buscas rápidas de dados ordenados.
- Árvores AVL e rubro-negras: São árvores de busca balanceadas que mantêm o equilíbrio dos nós, permitindo operações eficientes.
- Árvores B, B+, B*: Usadas em bancos de dados para busca rápida e eficiente em grandes volumes de dados.
- Quadtree e Octree: Aplicadas em processamento de imagens e modelagem tridimensional.

Árvore binária

A árvore binária é uma estrutura na qual cada nó pode ter até duas subárvores: uma à esquerda e outra à direita. Ela é amplamente utilizada em situações onde é necessário tomar decisões binárias, como algoritmos de compressão de dados, busca em bancos de dados, compiladores e representação de expressões matemáticas. Existem três tipos principais de árvores binárias:

- Estritamente binária: Cada nó possui ou zero ou dois filhos.
- Binária cheia: Todos os nós folha estão no mesmo nível, e cada nível possui exatamente o dobro de nós do anterior.
- Quase completa: A diferença de altura entre as subárvores de qualquer nó é no máximo 1.

Tipos de implementação

Para implementar uma árvore binária, há duas abordagens principais:

1. Array (alocação estática): Usa um array para organizar os elementos da árvore, mais eficiente em memória quando a árvore é completa.
2. Lista encadeada (alocação dinâmica): Usa ponteiros para conectar os nós, sendo mais flexível e adequada para árvores com estrutura variável.

Independentemente da implementação, operações como criação, inserção, remoção, busca e destruição de uma árvore são básicas e permitem o gerenciamento eficiente de dados.

Implementação de Árvores Binárias com Array (Heap)

Uma das formas de implementar árvores binárias é utilizando um array como estrutura de dados, conhecida como heap. Este método é particularmente útil para simular árvores binárias completas ou quase completas, com exceção do último nível, que pode estar parcialmente preenchido.

Para utilizar um array como uma heap, é preciso determinar previamente o número máximo de elementos da árvore, pois isso define o tamanho fixo do array. Essa abordagem requer que a estrutura da árvore seja conhecida de antemão, o que pode ser uma limitação em casos onde o número de elementos pode variar.

A grande vantagem do array é a forma eficiente de acessar os elementos da árvore usando fórmulas de indexação:

- FILHO_ESQ(PAI) = 2 * PAI + 1: Retorna o índice do filho à esquerda do nó localizado na posição do índice PAI.
- FILHO_DIR(PAI) = 2 * PAI + 2: Retorna o índice do filho à direita do nó na posição PAI.

Com essas fórmulas, podemos navegar pela árvore de forma direta, permitindo operações rápidas de acesso e busca, especialmente em estruturas onde os elementos são conhecidos previamente e a busca é a operação principal. Essa implementação é comum em aplicações que exigem hierarquia fixa e desempenho otimizado para buscas, como em algoritmos de ordenação (heapsort) e manipulação de prioridades em filas de prioridade. Segue abaixo, ilustrado pela Figura 1, um exemplo da utilização da implementação de árvore binária com array.

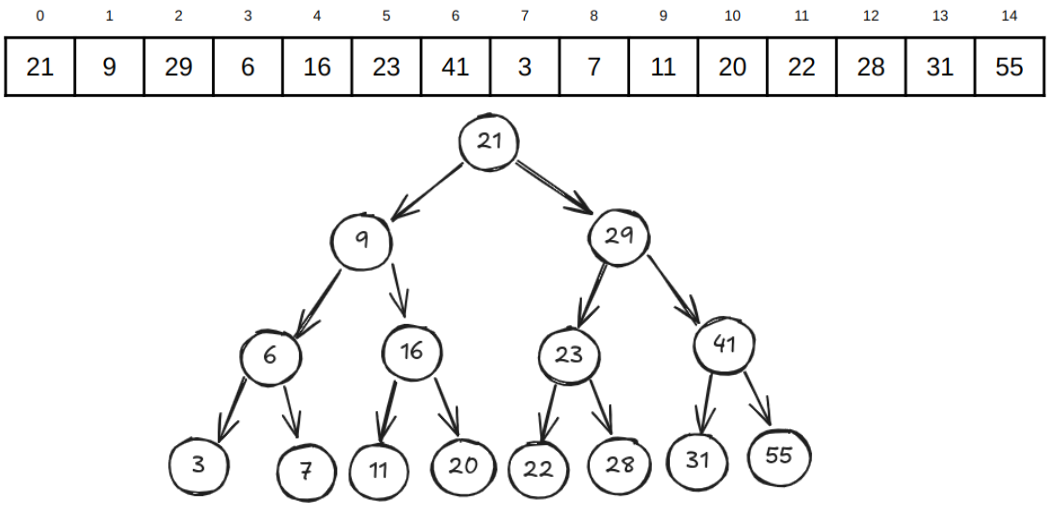

Figura 1 - Árvores Binárias com Array (Heap)

Percorrendo uma Árvore Binária

Uma operação fundamental em árvores binárias é o percurso de todos os nós para realizar uma ação específica, como imprimir o valor do nó ou modificá-lo. Diferentes métodos de percurso possibilitam visitar os nós em ordens distintas, sem uma sequência "natural" para seguir. Três abordagens clássicas de percurso incluem:

- Pré-ordem: visitar a raiz, depois o filho à esquerda, seguido pelo filho à direita.
- Em-ordem: visitar primeiro o filho à esquerda, depois a raiz e, finalmente, o filho à direita.
- Pós-ordem: visitar primeiro o filho à esquerda, seguido pelo filho à direita, e, por último, a raiz.

Essas variações se distinguem pela ordem de visitação de cada nó, sendo facilmente implementadas usando algoritmos recursivos, que facilitam a compreensão do processo.

Percurso Pré-ordem

A função de percurso pré-ordem recebe a árvore (ou a raiz) como parâmetro e verifica se é válida e não está vazia. No percurso, o nó é visitado primeiro, com o valor impresso, seguido por uma chamada recursiva para a subárvore da esquerda e, depois, para a direita. Esse fluxo é mostrado na Figura 12.23. A recursividade é essencial para percorrer a árvore inteira, e o endereço do nó é passado como parâmetro para permitir que a função acesse cada nó corretamente.

- Exemplo da árvore da Figura 1 percorrida em pré-ordem:
  21, 9, 6, 3, 7, 16, 11, 20, 29, 23, 22, 28, 41, 31, 55

Percurso Em-ordem

No percurso em-ordem, o processo começa na subárvore da esquerda, seguido pela raiz, e finaliza com o filho à direita. A função, ilustrada na Figura 12.25, também recebe a raiz como parâmetro e realiza verificações iniciais semelhantes. Esse percurso é bastante útil para exibir os nós em ordem crescente em árvores de busca binária, pois visita os nós na sequência ordenada. A implementação recursiva permite essa ordem precisa de visitação dos nós.

- Exemplo da árvore da Figura 1 percorrida em-ordem:
  3, 6, 7, 9, 11, 16, 20, 21, 22, 23, 28, 29, 31, 41,55

Percurso Pós-ordem

No percurso pós-ordem, a visita ocorre na seguinte sequência: subárvore à esquerda, subárvore à direita e, por fim, a raiz. A função para esse percurso está representada na Figura 12.27 e inclui verificações para validar e garantir que a árvore não esteja vazia. Esse tipo de percurso é comum em algoritmos que processam ou limpam nós após acessar suas subárvores, como na liberação de memória.

Esses percursos são fundamentais em muitas aplicações e são ilustrados nas Figuras 12.24, 12.26 e 12.28, respectivamente, para auxiliar na visualização de como cada abordagem explora a árvore e acessa os nós em diferentes ordens.

- Exemplo da árvore da Figura 1 percorrida em pós-ordem:
  3, 7, 6, 11, 20, 16, 9, 22, 28, 23, 31, 55, 41, 29,21

Árvore Binária de Busca Balanceada

Uma árvore binária de busca balanceada é uma estrutura de dados onde a altura das subárvores de cada nó difere no máximo em uma unidade, mantendo um equilíbrio que facilita operações como busca, inserção e remoção. O fator de balanceamento (fb) de um nó é a diferença entre as alturas das subárvores esquerda e direita. Manter essa estrutura balanceada é essencial para garantir a eficiência das operações, pois o custo delas depende diretamente do equilíbrio da árvore:

- O(log N) para uma árvore balanceada, permitindo operações rápidas e eficientes;

- O(N) em uma árvore desbalanceada, o que pode resultar em tempo de execução lento e ineficaz.

Para resolver o problema de balanceamento, é necessário ajustar a estrutura da árvore a cada inserção e remoção. Diversos tipos de árvores foram desenvolvidos para atender a diferentes necessidades de aplicações, sendo a Árvore AVL um dos modelos mais conhecidos para esse fim.

Árvore AVL

A árvore AVL é uma árvore binária balanceada em altura, desenvolvida por Adelson-Velskii e Landis em 1962. Este tipo de árvore se reestrutura localmente, ou seja, reequilibra apenas a área impactada pelas operações de inserção e remoção, utilizando rotações simples e duplas para manter o fator de balanceamento de cada nó entre +1, 0 e -1. Através dessas rotações, a árvore AVL mantém-se quase completa, mantendo a complexidade de operações em O(log N), mesmo após mudanças na estrutura. O fator de balanceamento é calculado subtraindo a altura da subárvore direita da altura da subárvore esquerda. Caso a diferença ultrapasse o limite de +1 ou -1, ocorre o processo de rebalanceamento, essencial para manter a árvore AVL.

Rotações

As rotações são operações essenciais para balancear uma árvore AVL. Elas reorganizam a estrutura para evitar desbalanceamentos, mantendo a árvore mais eficiente. Existem dois tipos de rotações:

1. Rotações Simples

- **Rotação LL:** É uma rotação à direita, realizada quando um novo nó é inserido na subárvore esquerda do filho esquerdo de um nó desbalanceado (A). Neste caso, o nó intermediário (B) substitui o nó desbalanceado (A) como raiz local, e A se torna a subárvore direita de B, como ilustrado na Figura 2.

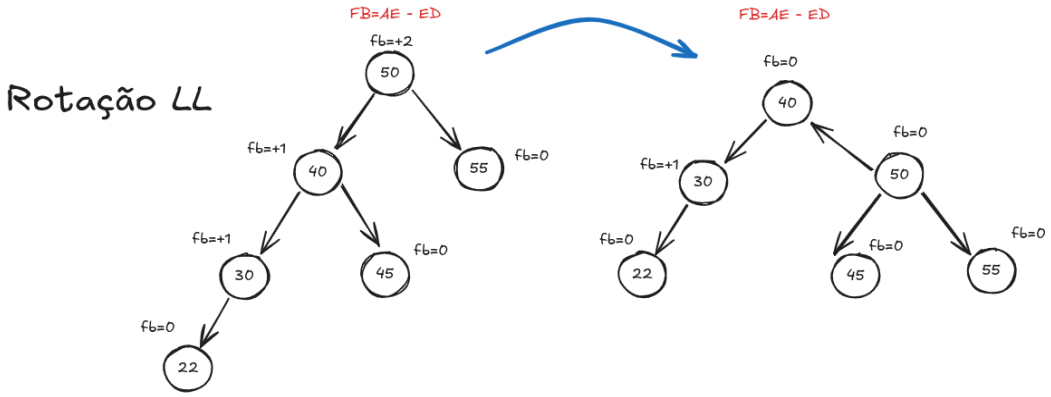

Figura 2 - Rotação LL

- **Rotação RR:** É uma rotação à esquerda, feita quando um novo nó é inserido na subárvore direita do filho direito de um nó desbalanceado (A). Nesse caso, o nó intermediário (B) substitui A, que passa a ser a subárvore esquerda de B, como ilustrado na Figura 3.

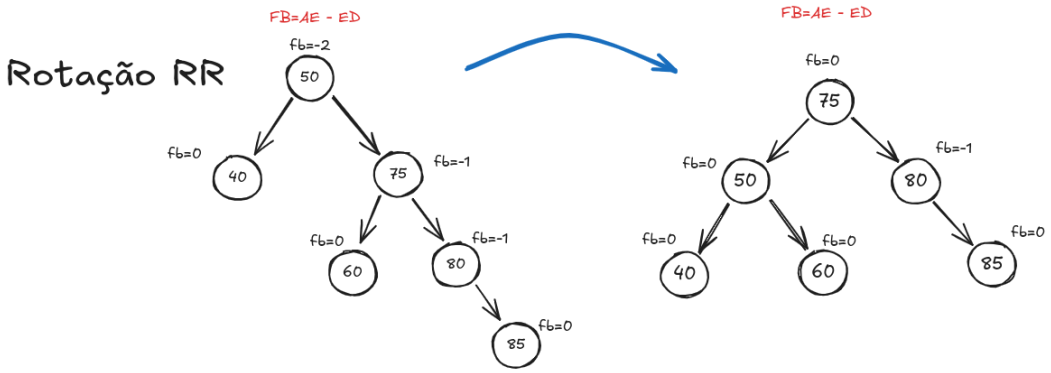

Figura 3 - Rotação RR

2. Rotações Duplas

- **Rotação LR**: Uma rotação dupla à direita, ocorre quando um nó é inserido na subárvore direita do filho esquerdo de A, criando uma configuração de desbalanceamento. A rotação LR pode ser realizada em duas etapas: primeiro uma rotação à esquerda (RR) no filho esquerdo de A, seguida de uma rotação à direita (LL) em A, como ilustrado na Figura 4.

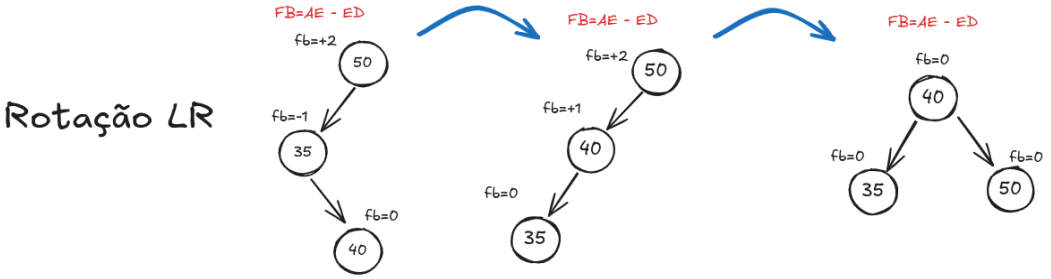

Figura 4 - Rotação LR

- **Rotação RL:** Uma rotação dupla à esquerda, ocorre quando um nó é inserido na subárvore esquerda do filho direito de A, invertendo a inclinação. Nesse caso, realiza-se uma rotação dupla, aplicando uma rotação à direita (LL) no filho direito de A e depois uma rotação à esquerda (RR) em A, como ilustrado na Figura 5.

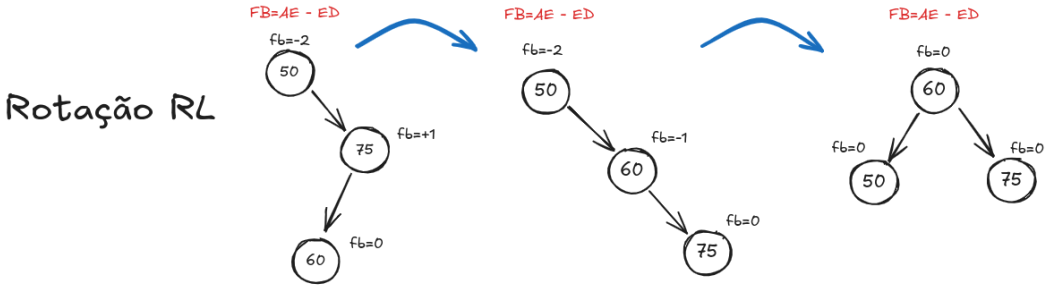

Figura 5 - Rotação RL

Cada tipo de rotação é aplicado no ancestral mais próximo do nó recém-inserido cujo fator de balanceamento ultrapassou +2 ou -2. A sequência e o tipo de rotação aplicada variam de acordo com o local de inserção e o padrão de desbalanceamento, mas, em qualquer caso, o objetivo é retornar o fator de balanceamento de cada nó ao intervalo de +1, 0 ou -1, preservando a eficiência da árvore AVL.

Árvore Rubro-Negra

A árvore rubro-negra (ou Red-Black Tree) é uma estrutura de árvore binária de busca balanceada que, além de organizar os nós em ordem, utiliza uma propriedade de coloração para manter o balanceamento da árvore sem que a altura de suas subárvores varie significativamente. Cada nó em uma árvore rubro-negra possui uma cor, que pode ser vermelha ou preta, e a coloração é essencial para garantir que o tempo de operações como busca, inserção e exclusão seja ***O(log n)***, onde ***n*** é o número de nós.

As regras de coloração e balanceamento tornam as árvores rubro-negras mais flexíveis em comparação com as árvores AVL, pois elas permitem que a árvore fique "aproximadamente balanceada", ao invés de rigorosamente balanceada. Isso faz com que as árvores rubro-negras sejam mais eficientes em cenários onde inserções e exclusões frequentes ocorrem, como em sistemas de bancos de dados e gerenciamento de memória.

Motivos para Colorir os Vértices

A coloração dos nós em vermelho e preto é utilizada para manter a árvore balanceada de maneira mais eficiente. As regras de coloração da árvore rubro-negra são as seguintes:

- Cada nó é ou vermelho ou preto.
- A raiz da árvore é sempre preta.
- Nós folha (nós nulos) são considerados pretos.
- Se um nó é vermelho, seus filhos devem ser pretos (ou seja, não podem existir dois nós vermelhos consecutivos em um caminho).
- Todo caminho da raiz até uma folha nula deve conter o mesmo número de nós pretos.

Essas regras garantem que, mesmo quando novos nós são inseridos ou removidos, a árvore permanece aproximadamente balanceada, mantendo a altura da árvore próxima de ***log n***, o que é crucial para a eficiência das operações.

Quando e Por Que Realizar Rotações e Recolorações

Quando um novo nó é inserido ou um nó é removido, a árvore rubro-negra pode violar uma ou mais de suas regras de coloração, necessitando ajustes para restaurar as propriedades da árvore. As operações de **rotação** e **recoloração** são usadas para corrigir essas violações.

- **Rotações:** Rotações são necessárias para corrigir o balanceamento estrutural da árvore quando ocorrem desbalanceamentos que não podem ser resolvidos apenas com recoloração. Existem dois tipos de rotações:
    - **Rotação à Esquerda:** Movimenta um nó para a posição de seu filho direito, enquanto o filho direito do nó torna-se seu pai.
    - **Rotação à Direita:** Move um nó para a posição de seu filho esquerdo, promovendo o filho esquerdo do nó como seu novo pai.

As rotações são geralmente aplicadas em pares ou combinadas com recolorações para restaurar o balanceamento.

- **Recolorações:** A recoloração altera as cores dos nós sem modificar a estrutura, o que pode resolver problemas quando duas regras de coloração são violadas, especialmente a regra que impede dois nós vermelhos consecutivos. A recoloração é frequentemente utilizada como o primeiro passo ao inserir um nó vermelho, e é seguida por rotações, caso o desbalanceamento persista.

Quando e Por Que a Árvore Rubro-Negra Pode Ficar Desbalanceada

A árvore rubro-negra pode ficar desbalanceada nas seguintes situações:

1. **Inserção de um Nó:** Quando um novo nó é adicionado, ele é inicialmente colorido de vermelho. Se o pai desse nó recém-inserido também for vermelho, uma violação da regra "não pode haver dois nós vermelhos consecutivos" ocorre, e é necessário um ajuste. Dependendo da cor do tio do novo nó, pode ser necessário realizar recolorações ou rotações para restaurar o balanceamento da árvore.

2. **Remoção de um Nó:** Ao remover um nó, principalmente se ele era preto, uma violação na contagem de pretos em um caminho pode acontecer. Para resolver isso, é necessário ajustar a coloração dos nós e, possivelmente, realizar rotações para garantir que a árvore continue a obedecer às regras de balanceamento.

Esses ajustes mantêm a árvore aproximadamente balanceada, garantindo que ela nunca fique excessivamente desbalanceada e que a altura permaneça dentro do limite de $2\log(n + 1)$, o que preserva o tempo de busca, inserção e exclusão em $O(\log n)$.