Árvore binária de busca

A árvore binária de busca é uma estrutura de dados de árvore na qual cada nó possui, no máximo, dois filhos. Além disso, cada nó é maior do que todos os nós na subárvore esquerda é menor do que todos os nós na subárvore direita. Isso permite eficientes operações de busca, inserção e remoção de elementos. Essa estrutura é amplamente utilizada em muitos algoritmos de ordenação e em muitos problemas de busca em árvores.

Métodos:

1. **Inserção:** Para inserir um novo elemento na árvore, começamos na raiz e comparamos o valor do novo elemento com o valor na raiz. Se o valor for menor, vamos para a subárvore esquerda. Se for maior, vamos para a subárvore direita. Continuamos esse processo até encontrar uma folha vazia e inserir o novo elemento nesse ponto.

3. **Busca:** Para buscar um elemento específico na árvore, começamos na raiz e comparamos o valor procurado com o valor na raiz. Se o valor for igual, a busca é bem-sucedida. Se o valor procurado for menor, vamos para a subárvore esquerda. Se for maior, vamos para a subárvore direita. Continuamos esse processo até encontrar o elemento ou chegar a uma folha vazia, indicando que o elemento não está na árvore.

5. **Remoção:** Para remover um elemento da árvore, começamos a busca como na operação de busca acima. Uma vez que o elemento é encontrado, há três casos possíveis: o nó não tem filhos, o nó tem um filho ou o nó tem dois filhos. No primeiro caso, basta remover o nó. No segundo caso, podemos simplesmente substituir o nó pelo seu filho único. No terceiro caso, precisamos encontrar o nó sucessor (o menor elemento na subárvore direita) e substituir o nó removido pelo nó sucessor. Depois, removemos o nó sucessor.

Complexidade:

Busca: A complexidade de tempo para buscar um elemento em uma árvore binária de busca é O(log n), onde n é o número de nós na árvore. Isso se deve ao fato de que, em média, precisamos comparar o valor procurado com cerca de log n nós para encontrar o elemento.

Inserção: A complexidade de tempo para inserir um elemento em uma árvore binária de busca é também O(log n), já que precisamos percorrer um caminho na árvore que tem comprimento logarítmico em relação ao número de nós.

Remoção: A complexidade de tempo para remover um elemento de uma árvore binária de busca é O(log n), já que precisamos encontrar o nó que contém o elemento antes de removê-lo, e essa operação tem complexidade O(log n).

Se a árvore não estiver balanceada, as operações podem ter complexidade de tempo pior, como O(n) em casos extremos, o que torna a árvore binária de busca menos eficiente.

Árvore rubro negra

Uma árvore Rubro-Negra é uma forma de implementação de árvore binária de busca que possui uma propriedade adicional de balanceamento.

As seguintes regras são aplicadas para mantê-la balanceada:

1. Todos os nós são ou vermelhos ou pretos.
2. A raiz é sempre preta.
3. Todas as folhas são pretas (nós nil).
4. Se um nó é vermelho, então seus filhos são pretos.
5. Para todo nó, todos os caminhos até as folhas contêm o mesmo número de nós pretos.

Métodos:

Inserção:

1. Adicione o novo nó como se fosse na árvore binária de busca normal.
2. Pinte o novo nó de vermelho.

3. Enquanto o pai do nó atual é vermelho e o avô não é a raiz, faça:

    a. Se o pai do nó atual é filho esquerdo do avô, então defina o tio como o irmão direito do pai.
    b. Se não, defina o tio como o irmão esquerdo do pai.

c. Se o tio é vermelho, então pinte o pai e o tio de preto e o avô de vermelho. Defina o nó atual como o avô e repita o loop.

d. Se o tio é preto, então faça uma rotação esquerda ou direita dependendo se o nó é filho direito ou esquerdo do pai, respectivamente. Pinte o pai de preto e o avô de vermelho. Defina o nó atual como raiz.

4. Pinte a raiz de preto.

Remoção:

1. Encontre o nó a ser removido como na árvore binária de busca normal.

2. Se ambos os filhos são não-nulos, então encontre o sucessor do nó e substitua o nó pelo sucessor.

3. Se o filho é nulo, então defina o nó como o filho.

4. Se o nó é vermelho, então basta removê-lo.

5. Se o nó é preto, então faça as seguintes correções:

    a. Enquanto o nó não é raiz e o irmão é preto, faça:

        i. Se o filho esquerdo do irmão é vermelho, então faça uma rotação direita no irmão.

        ii. Se o filho direito do irmão é vermelho, então faça uma rotação esquerda no irmão e pinte o filho direito de preto.

    b. Se o nó não é raiz, então pinte o irmão de vermelho.
    c. Se o nó é raiz, então pinte o irmão de preto.

Busca:

A busca é realizada da mesma forma que na árvore binária de busca normal, comparando o valor procurado com o valor de cada nó e seguindo o caminho esquerdo ou direito apropriado até encontrar o nó desejado ou determinar que ele não está na árvore.

Complexidade:

A complexidade dos métodos na árvore Rubro-Negra depende do tamanho da árvore.

Inserção: A complexidade da inserção em uma árvore Rubro-Negra é de O(log n), onde n é o número de nós na árvore. Isso se deve à propriedade de auto-equilíbrio da árvore, que permite que a inserção seja realizada em uma profundidade razoável, independentemente do tamanho da árvore.

Remoção: A complexidade da remoção em uma árvore Rubro-Negra também é de O(log n), onde n é o número de nós na árvore. Isso se deve à propriedade de auto-equilíbrio da árvore, que permite que a remoção seja realizada em uma profundidade razoável, independentemente do tamanho da árvore.

Busca: A complexidade da busca em uma árvore Rubro-Negra é de O(log n), onde n é o número de nós na árvore. Isso se deve à propriedade de auto-equilíbrio da árvore, que permite que a busca seja realizada em uma profundidade razoável, independentemente do tamanho da árvore.

Árvore AVL

A Árvore AVL é uma estrutura de dados de árvore binária de busca equilibrada, onde o balanceamento é mantido pelo ajuste das alturas dos nós filhos da esquerda e da direita. Isso garante que a árvore tenha uma altura mínima, o que resulta em melhor desempenho em operações de busca, inserção e remoção. A árvore AVL é chamada de "equilibrada" porque a diferença de altura entre os dois filhos de um nó nunca será maior do que 1.

Altura:

A altura de uma árvore AVL é calculada como a profundidade máxima de seus nós, ou seja, o número de níveis de um nó raiz até a folha mais distante. A fórmula para calcular a altura de uma árvore AVL é:

altura(nó) = 1 + max(altura(nó esquerdo), altura(nó direito))

onde "altura(nó esquerdo)" e "altura(nó direito)" são as alturas dos filhos esquerdo e direito do nó atual, respectivamente. Se um nó não tiver filhos, sua altura é considerada como 0. A partir dessa fórmula, a altura da árvore inteira pode ser calculada a partir da raiz.

Inserção:

1. Adicione o novo nó na posição apropriada, como faria na árvore binária de busca.

2. Atualize a altura dos nós ao longo do caminho da raiz até o novo nó.

3. Verifique se a árvore está desequilibrada e, se sim, execute a rotação adequada para corrigir o desequilíbrio.

Remoção:

1. Remova o nó conforme faria na árvore binária de busca.

2. Atualize a altura dos nós ao longo do caminho da raiz até o nó pai do nó removido.

3. Verifique se a árvore está desequilibrada e, se sim, execute a rotação adequada para corrigir o desequilíbrio.

Busca:

1. Comece na raiz da árvore.

2. Se o valor procurado for menor do que o nó atual, siga para o nó esquerdo.

3. Se o valor procurado for maior do que o nó atual, siga para o nó direito.

4. Se o valor procurado for igual ao nó atual, retorne o nó.

5. Repita esses passos até encontrar o nó ou até chegar a uma folha (nó sem filhos).

Conceito de Rotação:

A necessidade de rotacionar um nó na árvore AVL é identificada comparando a altura da subárvore esquerda e da subárvore direita. Se a diferença de altura entre as duas subárvores for maior que um, então é necessário realizar uma rotação.

1. **Rotação simples à direita (RR)** - Ocorre quando a subárvore esquerda de um nó é mais pesada do que a subárvore direita. O nó é

rotacionado para a direita e a raiz da subárvore esquerda se torna a raiz da árvore.

2. **Rotação simples à esquerda (LL)** - Ocorre quando a subárvore direita de um nó é mais pesada do que a subárvore esquerda. O nó é rotacionado para a esquerda e a raiz da subárvore direita se torna a raiz da árvore.

3. **Rotação dupla à direita (RL)** - Ocorre quando a subárvore direita de um nó é mais pesada do que a subárvore esquerda e a subárvore direita da subárvore direita é mais pesada do que a subárvore esquerda. O nó é primeiro rotacionado para a esquerda e depois rotacionado para a direita.

4. **Rotação dupla à esquerda (LR)** - Ocorre quando a subárvore esquerda de um nó é mais pesada do que a subárvore direita e a subárvore esquerda da subárvore esquerda é mais pesada do que a subárvore direita. O nó é primeiro rotacionado para a direita e depois rotacionado para a esquerda.

Complexidade:

Busca: A complexidade da busca é O(log n), onde n é o número de nós na árvore. Isso significa que, para cada nível adicional que você precisa descer na árvore, o número de nós que você precisa examinar é reduzido pela metade.

Remoção: A remoção em uma árvore AVL é mais complexa do que a inserção ou busca, pois é necessário verificar se é necessário realizar uma rotação e, se for, realizá-la. A complexidade da remoção é O(log n), onde n é o número de nós na árvore.

Inserção: É necessário verificar se é necessário realizar uma rotação e, se for, realizá-la. A complexidade da inserção é O(log n), onde n é o número de nós na árvore.

A complexidade dos métodos da árvore AVL é geralmente O(log n), o que significa que eles são eficientes mesmo para árvores com muitos nós. Isso faz da árvore AVL uma escolha popular para a implementação de estruturas de dados de árvore, especialmente quando a eficiência de busca e inserção é importante.

Árvore - B

A Árvore B é uma estrutura de dados de árvore de busca binária que é usada para armazenar dados de forma organizada para permitir operações de busca, inserção e exclusão eficientes. Ela mantém uma propriedade chamada balanceamento, garantindo que as folhas da árvore estejam equilibradas e que cada nó tenha no máximo um número fixo de filhos, o que garante uma complexidade de tempo de busca logarítmica.

Métodos:

Busca: o algoritmo de busca começa comparando o valor procurado com o valor do nó raiz. Se o valor for encontrado, a busca termina. Se o valor procurado for menor que o valor do nó atual, a busca é continuada na subárvore à esquerda. Se o valor procurado for maior, a busca é continuada na subárvore à direita. O processo é repetido até encontrar o valor ou chegar a uma folha vazia.

Inserção: o algoritmo de inserção começa procurando o local correto para inserir o novo valor, seguindo o mesmo processo descrito acima na busca. Quando o local correto é encontrado, o novo valor é adicionado e o nó é verificado para ver se precisa ser dividido para manter o balanceamento da árvore.

Exclusão: o algoritmo de exclusão começa procurando o valor a ser excluído, seguindo o mesmo processo descrito acima na busca. Quando o valor é encontrado, há três casos possíveis:

1. O nó a ser excluído é uma folha, então ele pode ser removido diretamente;

2. O nó a ser excluído tem um filho, então ele pode ser substituído pelo seu filho;

3. O nó a ser excluído tem dois filhos, então ele pode ser substituído pelo seu sucessor inorder (o nó com o valor mais próximo, mas menor que o nó a ser excluído).

Em seguida, o nó é verificado para ver se precisa ser unido com outro nó para manter o balanceamento da árvore.

Split:
O split em uma árvore B é o processo de dividir um nó que atingiu o seu limite máximo de chaves em dois nós separados.

O processo de split pode ser descrito da seguinte forma:

1. **Escolha da chave média:** O primeiro passo é escolher a chave média do nó cheio e movê-la para o nó pai.

2. **Criação de novos nós:** Em seguida, os elementos a esquerda da chave média são colocados em um novo nó, enquanto os elementos a direita são colocados em outro novo nó.

3. **Atualização do nó pai:** O nó pai é atualizado para incluir a chave média e apontar para os dois novos nós criados.

4. **Verificação de overflow no nó pai:** Se o nó pai também estiver cheio, o processo de split é repetido recursivamente.

5. **Propagação da chave:** A chave média é propagada para cima na árvore, se necessário, até chegar ao nó raiz.

6. **Atualização da árvore:** A árvore é atualizada para incluir as novas chaves e os novos nós criados.

Complexidade:

A Árvore B garante que a altura da árvore seja logarítmica em relação ao número de nós, o que significa que o tempo de busca, inserção e exclusão é de ordem logarítmica, ou seja, O(log n).

Isso significa que, em média, o número de passos necessários para realizar uma operação na Árvore B é proporcional ao logaritmo do número de nós na árvore. Isto é considerado muito eficiente, especialmente quando comparado com estruturas de dados lineares, como listas encadeadas, onde as operações de busca, inserção e exclusão são de ordem linear, ou seja, O(n).