Árvores de busca

Propriedades
As propriedades de uma árvore de busca são as seguintes:

- Classificação: As chaves em cada nó da árvore são classificadas em ordem crescente.

- Não-duplicidade: Cada chave na árvore é única e não há dois nós com a mesma chave.

- Estrutura de árvore: A árvore tem uma raiz, folhas e nós intermediários, formando uma estrutura hierárquica.

- Filhos direcionados: Cada nó tem dois ou mais filhos, cada um apontando para uma subárvore.

- Ordem de filhos: As subárvores à esquerda de um nó contem chaves menores que a chave do nó, e as subárvores à direita contem chaves maiores.

- Balanceamento: A altura da árvore é mantida relativamente curta para garantir que as operações na árvore sejam eficientes.

Estas propriedades garantem que as operações na árvore de busca, como inserção, remoção e busca, possam ser realizadas de forma eficiente e que a árvore mantenha uma estrutura organizada.

O que é?
Uma árvore de busca é uma estrutura de dados de pesquisa que segue uma ordem específica para organizar os nós e permitir uma busca mais eficiente. Cada nó da árvore contém um valor e as regras de comparação determinam a posição de cada nó filho. As árvores de busca mais comuns são a árvore binária de busca e a AVL (árvore balanceada).

Como é sua estrutura?
A estrutura de uma árvore de busca é composta por nós que contêm valores e referenciam outros nós (filhos) em uma estrutura hierárquica. Cada nó tem, no máximo, dois filhos, chamados de filho esquerdo e filho direito.

Como adicionar um elemento?
- Verificar se a árvore está vazia. Se estiver, criar um nó raiz com o valor desejado.
- Caso contrário, começar a partir da raiz e compare o valor do nó atual com o valor a ser adicionado.
- Se o valor a ser adicionado for menor que o valor do nó atual, siga para o filho esquerdo. Se o valor a ser adicionado for maior, siga para o filho direito.
- Repita o processo até encontrar uma folha (nó sem filhos) ou um nó nulo.
- Adicione o novo nó com o valor desejado como filho do nó encontrado.

Observe que a árvore de busca deve manter a propriedade de ordenação, ou seja, todos os nós da sub-árvore esquerda devem ter valores menores que o nó atual, e todos os nós da sub-árvore direita devem ter valores maiores.

Como remover um elemento?

- Comece a partir da raiz e compare o valor do nó atual com o valor a ser removido.
- Se o valor a ser removido for igual ao valor do nó atual, você encontrou o nó a ser removido.
- Se o valor a ser removido for menor que o valor do nó atual, siga para o filho esquerdo. Se o valor a ser removido for maior, siga para o filho direito.
- Repita o processo até encontrar o nó com o valor a ser removido ou um nó nulo.
- Caso não seja encontrado o nó, a remoção não pode ser realizada.
- Caso o nó a ser removido tenha zero ou um filho, basta substituí-lo pelo seu filho.
- Caso o nó a ser removido tenha dois filhos, é necessário encontrar o sucessor inorder (o nó mais à esquerda na sub-árvore direita) ou o antecessor inorder (o nó mais à direita na sub-árvore esquerda) e substituir o nó removido pelo sucessor ou antecessor. Em seguida, remover o sucessor ou antecessor, seguindo os passos 5 ou 6.

Observe que a árvore de busca deve manter a propriedade de ordenação após a remoção do elemento.

Como procurar um elemento?

- Comece a partir da raiz e compare o valor do nó atual com o valor procurado.
- Se o valor procurado for igual ao valor do nó atual, você encontrou o nó procurado.
- Se o valor procurado for menor que o valor do nó atual, siga para o filho esquerdo. Se o valor procurado for maior, siga para o filho direito.
- Repita o processo até encontrar o nó com o valor procurado ou um nó nulo.
- Caso o nó nulo seja encontrado, o elemento não está na árvore.

Observe que, durante o processo, a propriedade de ordenação da árvore é utilizada para decidir por qual caminho seguir, o que torna a busca muito mais eficiente do que, por exemplo, uma busca linear em uma lista.

Árvores binárias de busca

Propriedades

As propriedades de uma árvore binária de busca são as seguintes:

- Classificação: As chaves em cada nó da árvore são classificadas em ordem crescente.

- Não-duplicidade: Cada chave na árvore é única e não há dois nós com a mesma chave.

- Estrutura de árvore: A árvore tem uma raiz, folhas e nós intermediários, formando uma estrutura hierárquica.

- Filhos direcionados: Cada nó tem, no máximo, dois filhos, cada um apontando para uma subárvore.

- Ordem de filhos: As subárvores à esquerda de um nó contem chaves menores que a chave do nó, e as subárvores à direita contem chaves maiores.

- Balanceamento: A altura da árvore é mantida relativamente curta para garantir que as operações na árvore sejam eficientes.

Estas propriedades garantem que as operações na árvore binária de busca, como inserção, remoção e busca, possam ser realizadas de forma eficiente e que a árvore mantenha uma estrutura organizada.

O que é?

Árvore Binária de Busca (Binary Search Tree, BST) é uma estrutura de dados de árvore em que cada nó tem, no máximo, dois filhos. Além disso, em uma árvore binária de busca, para todo nó, o valor de todos os nós na sub-árvore esquerda é menor que o valor do nó atual, e o valor de todos os nós na sub-árvore direita é maior que o valor do nó atual.

Isso permite que operações como inserção, remoção e busca de elementos sejam realizadas de maneira eficiente, com tempo de execução O(log n) em média.

A árvore binária de busca é amplamente utilizada em aplicações que requerem busca rápida de elementos, como em bancos de dados, sistemas de arquivos, entre outros.

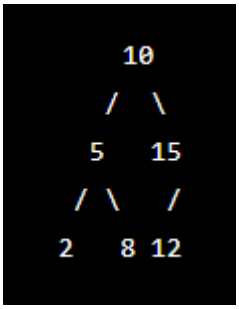
Como é sua estrutura?

A estrutura de uma árvore binária de busca consiste em nós que representam elementos e ligações que representam relações entre os elementos. Cada nó pode ter, no máximo, dois filhos: um filho à esquerda e um filho à direita.

Cada nó na árvore possui um valor associado, e a estrutura da árvore é determinada pela propriedade de ordenação: para todo nó, o valor de todos os nós na sub-árvore esquerda é menor que o valor do nó atual, e o valor de todos os nós na sub-árvore direita é maior que o valor do nó atual.

Isso significa que, para procurar um elemento na árvore, é possível seguir o caminho mais curto até o elemento, comparando o valor do nó atual com o valor procurado. Se o valor procurado for menor que o valor do nó atual, siga para a sub-árvore esquerda; se for maior, siga para a sub-árvore direita.

Abaixo está um exemplo de uma árvore binária de busca com os valores 10, 5, 15, 2, 8 e 12

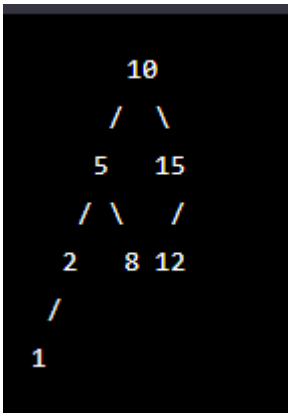


Observe que a estrutura da árvore mantém a propriedade de ordenação, o que permite buscas eficientes.

Como adicionar um elemento?

- Comece na raiz da árvore.
- Compare o valor do nó atual com o valor do elemento que você deseja adicionar.
- Se o valor do elemento for menor que o valor do nó atual, siga para o filho esquerdo. Se o valor do elemento for maior, siga para o filho direito.
- Se chegar a um nó nulo, crie um novo nó com o valor do elemento e o adicione como filho do nó nulo.

Observe que, ao adicionar um elemento, a propriedade de ordenação da árvore é mantida, o que permite buscas eficientes. Aqui está um exemplo de adição de um elemento (1) na árvore mencionada na pergunta anterior



Observe como o elemento 1 foi adicionado como filho esquerdo do nó 2.

Como remover um elemento?

- Encontre o nó a ser removido.
- Se o nó a ser removido não tiver filhos, basta apagar a referência a ele.
- Se o nó a ser removido tiver apenas um filho, substitua a referência a ele pelo filho.
- Se o nó a ser removido tiver dois filhos, encontre o menor nó na subárvore direita do nó a ser removido. Substitua o nó a ser removido pelo menor nó encontrado e remova o menor nó.

Obs: É importante que você mantenha as propriedades de uma árvore binária de busca, como o fato de que o valor da chave do nó esquerdo deve ser menor que a do nó pai e o valor da chave do nó direito deve ser maior que a do nó pai.

Como procurar um elemento?

- Comece pelo nó raiz.
- Compare o valor da chave do nó atual com o valor da chave do elemento que você está procurando.
- Se o valor da chave do elemento que você está procurando é menor que o valor da chave do nó atual, vá para a subárvore esquerda.
- Se o valor da chave do elemento que você está procurando é maior que o valor da chave do nó atual, vá para a subárvore direita.
- Repita os passos 3 e 4 até encontrar o nó com a chave desejada ou chegar em uma folha (nó sem filhos).

Se o elemento for encontrado, você terá o nó correspondente. Caso contrário, você terá chegado a uma folha e poderá concluir que o elemento não está na árvore.

Árvore AVL

Propriedades

As propriedades de uma árvore AVL são as seguintes:

- Classificação: As chaves em cada nó da árvore são classificadas em ordem crescente.
- Não-duplicidade: Cada chave na árvore é única e não há dois nós com a mesma chave.
- Estrutura de árvore: A árvore tem uma raiz, folhas e nós intermediários, formando uma estrutura hierárquica.
- Filhos direcionados: Cada nó tem, no máximo, dois filhos, cada um apontando para uma subárvore.
- Ordem de filhos: As subárvores à esquerda de um nó contem chaves menores que a chave do nó, e as subárvores à direita contem chaves maiores.
- Balanceamento: A altura das subárvores de cada nó da árvore deve diferir em no máximo 1, garantindo assim o balanceamento da árvore.
- Autobalanceamento: A árvore AVL se autobalanceia após cada inserção ou remoção, ajustando a estrutura da árvore de modo a manter o balanceamento.

Estas propriedades garantem que as operações na árvore AVL, como inserção, remoção e busca, sejam realizadas de forma eficiente e que a árvore mantenha um alto nível de balanceamento, o que é fundamental para garantir a eficiência das operações.

O que é?

Uma Árvore AVL é uma variação de uma árvore binária de busca equilibrada. É uma estrutura de dados que mantém a propriedade de uma árvore binária de busca, ou seja, o valor da chave do nó esquerdo é menor que a do nó pai, e o valor da chave do nó direito é maior que a do nó pai. Além disso, uma árvore AVL é equilibrada, o que significa que a altura das subárvores esquerda e direita de cada nó não deve diferir em mais de 1.

A propriedade de equilíbrio é mantida através de rotações. Quando um nó é adicionado ou removido, a altura das subárvores pode ser desequilibrada, então as rotações são realizadas para corrigir isso. Esta propriedade de equilíbrio garante que as operações de busca, inserção e remoção tenham tempo de execução O(log n), onde n é o número de nós na árvore.

Como é sua estrutura?
Uma árvore AVL é uma árvore binária de busca equilibrada, onde cada nó possui as seguintes informações:

- Valor da chave: é o valor que serve como identificador para o nó. É usado para determinar a ordem dos nós na árvore e para realizar buscas.
-
- Filhos esquerdo e direito: cada nó pode ter até dois filhos, um à esquerda e outro à direita.
-
- Altura: é a distância do nó até a folha mais distante. É usada para mantê-lo equilibrado.

O equilíbrio é mantido de forma a garantir que a diferença de altura entre as subárvores esquerda e direita de um nó não seja maior que 1. Quando uma inserção ou remoção desequilibra a árvore, rotações são realizadas para corrigir isso.

A estrutura de uma árvore AVL é importante porque garante que as operações de busca, inserção e remoção tenham tempo de execução O(log n), o que é muito eficiente em comparação a outras estruturas de dados.

Como adicionar um elemento?

- Encontre o local correto para inserção: comece pelo nó raiz e compare o valor da chave do elemento que você está adicionando com o valor da chave dos nós que você está visitando. Se o valor da chave for menor, vá para a subárvore esquerda. Se o valor da chave for maior, vá para a subárvore direita. Repita o processo até encontrar uma folha.
- Adicione o novo nó: crie um novo nó com o valor da chave e os ponteiros para os filhos esquerdo e direito nulos e coloque-o na posição correta na árvore.
- Atualize a altura: atualize a altura da árvore a partir da folha até a raiz.
- Verifique o equilíbrio: verifique se a árvore está equilibrada depois da inserção. Se a diferença de altura entre as subárvores esquerda e direita de um nó for maior que 1, então é necessário realizar uma rotação para corrigir isso.

- Repita o processo de verificação do equilíbrio até que a raiz seja alcançada. Se for necessário, repita as rotações até que a árvore seja equilibrada novamente.

Esses passos garantem que a árvore mantenha a propriedade de equilíbrio e que a adição do novo elemento seja realizada de forma eficiente, com tempo de execução O(log n).

Como remover um elemento?

- Encontre o nó a ser removido: comece pelo nó raiz e compare o valor da chave do elemento que você está removendo com o valor da chave dos nós que você está visitando. Se o valor da chave for menor, vá para a subárvore esquerda. Se o valor da chave for maior, vá para a subárvore direita. Repita o processo até encontrar o nó a ser removido.

- Verifique se o nó tem filhos: se o nó não tiver filhos, basta removê-lo e atualizar as referências dos pais. Se o nó tiver um filho, basta substituí-lo pelo filho. Se o nó tiver dois filhos, é necessário encontrar o sucessor inorder para substituí-lo.

- Atualize a altura: atualize a altura da árvore a partir do nó removido até a raiz.

- Verifique o equilíbrio: verifique se a árvore está equilibrada depois da remoção. Se a diferença de altura entre as subárvores esquerda e direita de um nó for maior que 1, então é necessário realizar uma rotação para corrigir isso.

- Repita o processo de verificação do equilíbrio até que a raiz seja alcançada. Se for necessário, repita as rotações até que a árvore seja equilibrada novamente.

Esses passos garantem que a árvore mantenha a propriedade de equilíbrio e que a remoção do elemento seja realizada de forma eficiente, com tempo de execução O(log n).

Como procurar um elemento?

- Comece pelo nó raiz.

- Compare o valor da chave do elemento que você está procurando com o valor da chave do nó atual.

- Se o valor da chave for menor, vá para a subárvore esquerda. Se o valor da chave for maior, vá para a subárvore direita.

- Repita o processo até encontrar o elemento ou chegar a uma folha nula.

- Se o elemento for encontrado, retorne o nó correspondente. Se o elemento não for encontrado, retorne null.

Esse processo é realizado de forma eficiente, com tempo de execução O(log n), graças à propriedade de equilíbrio da árvore AVL.
Nó +-2 com dois filhos Delta = 0 significa que a árvore não era AVL antes!

Árvore rubro negra

Propriedades
As propriedades de uma árvore Rubro-Negra são as seguintes:

- Classificação: As chaves em cada nó da árvore são classificadas em ordem crescente.

- Não-duplicidade: Cada chave na árvore é única e não há dois nós com a mesma chave.

- Estrutura de árvore: A árvore tem uma raiz, folhas e nós intermediários, formando uma estrutura hierárquica.

- Filhos direcionados: Cada nó tem, no máximo, dois filhos, cada um apontando para uma subárvore.

- Ordem de filhos: As subárvores à esquerda de um nó contem chaves menores que a chave do nó, e as subárvores à direita contem chaves maiores.

- Cor: Cada nó é classificado como vermelho ou preto.

- Propriedades de cor:
  a) A raiz é sempre preta.
  b) Todos os nós folhas são pretos.
  c) Se um nó é vermelho, então seus filhos são pretos.
  d) Para cada nó, o número de nós pretos na sua subárvore esquerda e direita é o mesmo.

Estas propriedades garantem que a árvore Rubro-Negra mantenha um bom equilíbrio e que as operações na árvore, como inserção, remoção e busca, sejam realizadas de forma eficiente. Além disso, a combinação de regras de cor garante que a árvore não tenha altura excessivamente grande, o que ajuda a manter a eficiência das operações.

O que é?
Uma Árvore Rubro-Negra é uma estrutura de dados de árvore binária de busca que garante um tempo de execução logarítmico para as operações de inserção, remoção e busca, além de ser uma forma de balanceamento de árvore.

A diferença entre as árvores Rubro-Negra e outras árvores de balanceamento é que as árvores Rubro-Negra usam uma propriedade adicional, a cor dos nós, para manter o equilíbrio da árvore. Cada nó da árvore é marcado como "rubro" ou "negro". As regras que governam a cor dos nós garantem que a árvore Rubro-Negra esteja balanceada.

A árvore Rubro-Negra é muito utilizada na implementação de estruturas de dados, como Mapas e Conjuntos, em diversas linguagens de programação, pois ela possui um alto desempenho para operações de inserção, remoção e busca.

Como é sua estrutura?
A árvore rubro-negra é uma estrutura de dados de árvore binária que mantém uma propriedade de balanceamento para garantir complexidade de tempo O(log n) para as operações de inserção, deleção e busca. Cada nó na árvore tem uma cor (vermelho ou preto) e segue as seguintes regras:

- Todos os nós folhas (nós null) são considerados pretos.
- Todo nó vermelho tem dois filhos pretos.
- Todo caminho de um nó folha até a raiz contém o mesmo número de nós pretos.

Estas regras garantem que a altura da árvore seja sempre logarítmica em relação ao número de nós, o que mantém sua eficiência.

Como adicionar um elemento?
A inserção em uma árvore rubro-negra envolve os seguintes passos:

- Adicionar o elemento como se fosse em uma árvore binária normal, colocando-o como um nó filho do nó que o precede na ordem correta.
- Colorir o novo nó como vermelho.
- Verificar se as propriedades da árvore rubro-negra foram violadas e, se sim, realizar rotações e recolorações necessárias para corrigi-las.

As rotações e recolorações são necessárias para manter as propriedades da árvore rubro-negra e garantir que a altura da árvore seja sempre logarítmica em relação ao número de nós. Estas operações corrigem as seguintes violações de propriedade:

- Se o pai do novo nó é vermelho, então o avô do novo nó também é vermelho: neste caso, a cor do avô é alterada para preto e as cores dos pais do novo nó e do avô são alteradas para vermelho.
- Se o pai do novo nó é preto, mas o tio do novo nó (irmão do pai) é vermelho: neste caso, a cor do tio é alterada para preto e a cor do pai do novo nó é alterada para vermelho.
- Se o pai do novo nó é preto e o tio do novo nó é preto: neste caso, é necessária uma rotação para balancear a árvore.

Estes passos são executados de forma recursiva, subindo na árvore a partir do nó adicionado, até a raiz, corrigindo quaisquer violações de propriedade.

Como remover um elemento?
A remoção em uma árvore rubro-negra envolve os seguintes passos:

- Localizar o nó a ser removido, como em uma árvore binária normal.

- Se o nó a ser removido tem dois filhos, então o nó de substituição é encontrado (por exemplo, o menor nó na subárvore à direita).
- Substituir o nó a ser removido pelo nó de substituição.
- Verificar se as propriedades da árvore rubro-negra foram violadas e, se sim, realizar rotações e recolorações necessárias para corrigi-las.

As rotações e recolorações são necessárias para manter as propriedades da árvore rubro-negra e garantir que a altura da árvore seja sempre logarítmica em relação ao número de nós. Estas operações corrigem as seguintes violações de propriedade:

- Se o irmão do nó removido é vermelho: neste caso, a cor do irmão é alterada para preto e a cor do pai do nó removido é alterada para vermelho. Em seguida, é realizada uma rotação para balancear a árvore.
- Se o irmão do nó removido é preto e ambos os filhos do irmão são pretos: neste caso, a cor do irmão é alterada para vermelho e o processo é repetido a partir do pai do nó removido.
- Se o irmão do nó removido é preto, mas um dos filhos do irmão é vermelho: neste caso, a cor do filho vermelho é alterada para preto e a cor do irmão é alterada para vermelho. Em seguida, é realizada uma rotação para balancear a árvore.

Estes passos são executados de forma recursiva, corrigindo quaisquer violações de propriedade na subida da árvore a partir do nó substituto até a raiz.

Remoção de um nó negro

Se o substituto for Negro, ele se torna um nó Duplo Negro.

X	Substituto	Resultado
NEGRO	NEGRO	DUPLO NEGRO
NEGRO	NULO NEGRO	NULO DUPLO NEGRO
NEGRO	RUBRO	NEGRO
RUBRO	RUBRO	RUBRO
RUBRO	NULO NEGRO	NULO NEGRO

Como procurar um elemento?

A procura por um elemento em uma árvore rubro-negra é realizada da mesma forma que em uma árvore binária normal:

- Inicie a partir da raiz da árvore.
- Se o elemento atual é igual ao elemento procurado, então a busca é bem-sucedida.
- Se o elemento procurado é menor que o elemento atual, então siga para a subárvore esquerda.

- Se o elemento procurado é maior que o elemento atual, então siga para a subárvore direita.
- Repita os passos 2 a 4 até encontrar o elemento ou chegar a uma folha, o que significa que o elemento não está na árvore.

A estrutura e a propriedade da árvore não afetam diretamente a procura por um elemento. No entanto, a propriedade de balanceamento da árvore rubro-negra garante que a altura da árvore é logarítmica em relação ao número de nós, o que torna a busca por um elemento eficiente.

Pai do novo nó é filho esquerdo e novo nó é filho esquerdo

- Pai fica Negro;
- Avô fica Rubro; e
- Rotaciona o avô para a direita

Pai do novo nó é filho esquerdo e novo nó é filho direito

- Rotaciona o pai para a esquerda;
- Novo nó = filho esquerdo do novo nó;
- Pai fica Negro;
- Avô fica Rubro; e
- Rotaciona o avô para a direita.

Árvore-B

Propriedades

As propriedades de uma Árvore B são:

- Classificação: As chaves em cada nó da árvore são classificadas em ordem crescente.

- Não-duplicidade: Cada chave na árvore é única e não há dois nós com a mesma chave.

- Estrutura de árvore: A árvore tem uma raiz, folhas e nós intermediários, formando uma estrutura hierárquica.

- Ordem de filhos: As subárvores à esquerda de um nó contem chaves menores que a chave do nó, e as subárvores à direita contem chaves maiores.

- Ordem de tamanho: Cada nó pode ter até um número fixo "d" de filhos, com d > 2, onde d é chamado de ordem da árvore.

- Propriedade de balanceamento: Para cada nó, o número de chaves é mantido entre d-1 e 2d-1. Quando um nó atinge sua capacidade máxima de chaves, ele é dividido em dois nós, mantendo a propriedade de balanceamento.

Estas propriedades garantem que a Árvore B mantenha um bom equilíbrio e que as operações na árvore, como inserção, remoção e busca, sejam realizadas de forma eficiente. Além disso, a combinação de regras de tamanho e equilíbrio garante que a altura da árvore não cresça excessivamente, o que ajuda a manter a eficiência das operações.

O que é?
Uma Árvore B é uma estrutura de dados de árvore de busca balanceada. É uma variação de árvores B-tree, que são amplamente utilizadas em sistemas de banco de dados relacionais para armazenar e recuperar grandes quantidades de dados em disco.

As árvores B possuem as seguintes propriedades:

- Cada nó pode ter um número mínimo e máximo de filhos, geralmente definidos como t.
- Cada nó, exceto as folhas, armazena pelo menos t-1 chaves.
- Cada nó, exceto a raiz, tem pelo menos t filhos.
- As chaves armazenadas em um nó são classificadas em ordem crescente.
- As folhas possuem todas as chaves da subárvore abaixo delas.

Estas propriedades garantem que a árvore seja balanceada, o que significa que a altura da árvore é logarítmica em relação ao número de nós. Isso torna a busca, inserção e remoção de elementos muito eficientes. Além disso, as árvores B são projetadas para serem utilizadas em sistemas de armazenamento secundário, como discos, onde a leitura e escrita de grandes quantidades de dados pode ser lenta. A estrutura da árvore permite que várias chaves sejam lidas ou escritas em uma única operação de leitura ou escrita, o que torna a operação muito mais eficiente.

Como é sua estrutura?
A estrutura de uma árvore B é semelhante a uma árvore binária, mas com algumas diferenças importantes:

- Cada nó pode conter várias chaves e vários filhos, em vez de apenas uma chave e dois filhos em uma árvore binária.
- As chaves são classificadas em ordem crescente e armazenadas de forma a maximizar o uso do espaço no nó.
- Cada nó, exceto as folhas, armazena pelo menos t-1 chaves e tem pelo menos t filhos.
- As folhas possuem todas as chaves da subárvore abaixo delas.

A estrutura de uma árvore B permite que sejam feitas buscas, inserções e remoções de elementos de forma muito eficiente, pois garante que a altura da árvore seja logarítmica em relação ao número de nós. Além disso, a estrutura da árvore permite que sejam lidas ou escritas várias chaves em uma única operação, o que torna a operação muito mais eficiente quando se trata de armazenamento secundário, como discos.

Como adicionar um elemento?
A adição de um elemento em uma árvore B envolve algumas etapas importantes para garantir que as propriedades da árvore sejam mantidas:

- Buscar a folha onde o elemento deve ser inserido: A busca começa na raiz da árvore e segue pelos filhos do nó até chegar a uma folha.

- Verificar se a folha tem espaço disponível: Se a folha tiver espaço disponível, o elemento é simplesmente inserido na posição correta de forma a manter a ordem das chaves.

- Caso a folha não tenha espaço disponível: A folha é dividida em dois nós, sendo que a chave central é elevada para o nó pai. Se o nó pai tiver espaço disponível, a chave central é inserida nesse nó. Se não tiver, o nó pai também é dividido e o processo é repetido até chegar a uma raiz que tenha espaço disponível.

Este processo é repetido até que a chave seja inserida na árvore de forma a manter as propriedades da árvore B, garantindo que a altura da árvore seja logarítmica em relação ao número de nós e que as chaves sejam classificadas em ordem crescente.

Como remover um elemento?
A remoção de um elemento em uma árvore B envolve algumas etapas importantes para garantir que as propriedades da árvore sejam mantidas:

- Buscar o elemento: A busca começa na raiz da árvore e segue pelos filhos do nó até encontrar o elemento ou chegar a uma folha vazia.

- Verificar se o elemento está em uma folha: Se o elemento estiver em uma folha, ele é simplesmente removido da folha e as propriedades da árvore são mantidas.

- Verificar se o elemento está em um nó com mais de t-1 chaves: Se o elemento estiver em um nó com mais de t-1 chaves, ele é substituído por outra chave do nó ou por uma chave do filho e as propriedades da árvore são mantidas.

- Verificar se o elemento está em um nó com t-1 chaves: Se o elemento estiver em um nó com t-1 chaves, o nó pode ser combinado com um de seus irmãos para formar um nó com mais de t-1 chaves. Se não for possível combinar os nós, a chave central de um dos irmãos é elevada para o nó pai.

Este processo é repetido até que o elemento seja removido da árvore de forma a manter as propriedades da árvore B, garantindo que a altura da árvore seja logarítmica em relação ao número de nós e que as chaves sejam classificadas em ordem crescente.

Como procurar um elemento?
A procura de um elemento em uma árvore B segue o mesmo processo de busca em uma árvore binária de busca. A busca começa na raiz da árvore e, em cada nó, verifica-se em qual filho o elemento procurado deve ser encontrado. A busca é conduzida até que o elemento seja encontrado ou chegue-se a uma folha vazia.

As propriedades da árvore B afetam a forma como a busca é conduzida. Em particular, como cada nó tem no máximo 2t-1 chaves, a busca precisa verificar apenas os filhos correspondentes às chaves no nó. Além disso, como as chaves em cada nó são classificadas em ordem crescente, a busca não precisa verificar todos os filhos, mas apenas o filho que é provável conter o elemento procurado.

Esta estratégia de busca garante que a árvore B tenha uma altura logarítmica em relação ao número de nós, o que significa que a busca por um elemento na árvore é eficiente, mesmo quando há muitos elementos na árvore.

AVL x Rubro-Negra

- Na teoria, possuem a mesma complexidade computacional (inserção, remoção e busca): “O(log N)”
- Na prática, a árvore AVL é mais rápida na operação de busca, e mais lenta nas operações de inserção e remoção
- A árvore AVL é mais balanceada do que a Rubro-Negra, o que acelera a operação de busca
- Maior custo na operação de inserção e remoção: no pior caso, uma operação de remoção pode exigir “O(log N)” rotações na árvore AVL, mas apenas 3 rotações na árvore Rubro-Negra
- Se sua aplicação realiza de forma intensa a operação de busca, é melhor usar uma árvore AVL
- Se a operação mais usada é a inserção ou remoção, use uma árvore Rubro-Negra
- Árvores Rubro-Negra são de uso mais geral do que as árvores AVL. Isso faz com que elas sejam utilizadas em diversas aplicações e bibliotecas de linguagens de programação