***Árvore B+***

**O que é Árvore B?**

Na ciência da computação, uma **Árvore B** é uma estrutura de dados auto balanceada projetada para armazenar dados classificados e facilitar operações como busca, inserção, remoção e acesso sequencial. Trata-se de uma generalização das árvores binárias de busca, onde cada nó pode conter mais de duas chaves, tendo um número variável de filhos que depende da ordem da árvore (mm).

A Árvore B foi projetada com foco em sistemas de armazenamento secundário (como discos rígidos ou SSDs), otimizando operações de entrada e saída (E/S). Graças à sua estrutura, permite que grandes volumes de dados sejam organizados eficientemente, minimizando o número de acessos ao disco para buscar ou manipular informações.

Sua principal aplicação ocorre em **bancos de dados** e **sistemas de arquivos**, onde é crucial lidar com grandes volumes de registros. A Árvore B compartilha algumas similaridades com a árvore Rubro-Negra, mas é considerada mais eficiente para casos que envolvem armazenamento em memória secundária, devido à redução no número de operações de E/S necessárias.

**Características e vantagens:**

* Os nós da Árvore B possuem entre chaves, permitindo um balanceamento eficiente.
* Todas as folhas da Árvore B estão no mesmo nível, garantindo que o tempo de acesso seja uniforme.
* Sua complexidade para inserção, remoção e busca é onde nn é o número de chaves armazenadas.
* Excelente desempenho para operações em larga escala, como em bancos de dados ou sistemas que gerenciam milhões de registros.

A Árvore B foi introduzida em 1970 por Rudolf Bayer e Edward M. McCreight. O significado da letra “B” ainda é debatido, mas acredita-se que esteja relacionado ao sobrenome do criador (Bayer) ou à palavra "Balanced" (Balanceada).

**O que é Árvore B+?**

A **Árvore B+** é uma variação da Árvore B que aprimora certas características para aplicações específicas. Em uma Árvore B+, apenas as folhas armazenam os dados (ou registros associados às chaves), enquanto os nós internos contêm apenas chaves usadas como índices para direcionar a busca.

Diferente da Árvore B, onde os dados podem ser encontrados tanto nas folhas quanto nos nós internos, a Árvore B+ concentra os valores nas folhas, garantindo uma navegação mais eficiente para acesso sequencial.

**Características e vantagens da Árvore B+:**

1. **Separação de índices e dados:** Os nós internos servem apenas como índice, enquanto os dados reais são armazenados exclusivamente nas folhas. Isso reduz a quantidade de informações que precisam ser manipuladas em cada nó interno.
2. **Conexão entre folhas:** As folhas da Árvore B+ são conectadas por meio de ponteiros, formando uma **lista duplamente encadeada**. Essa estrutura permite que buscas sequenciais sejam realizadas de forma eficiente, além de suportar varreduras completas dos dados.
3. **Busca otimizada:** Embora a complexidade seja ), assim como na Árvore B, a separação clara entre índices e dados facilita buscas em blocos específicos, especialmente em sistemas que dependem de discos.
4. **Ideal para sistemas de banco de dados:** A Árvore B+ é amplamente usada em sistemas de gerenciamento de bancos de dados, onde operações como consultas sequenciais e acesso por intervalo (range queries) são comuns.

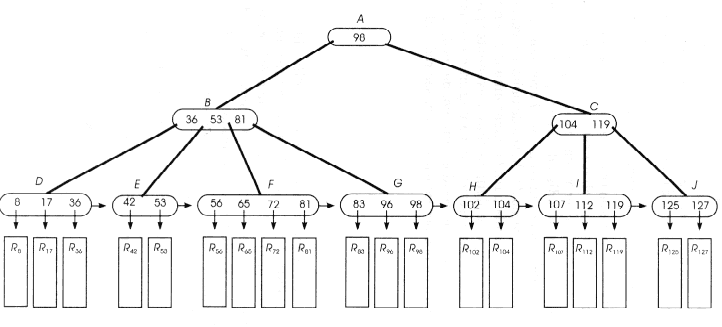
|  |  |  |
| --- | --- | --- |
| **Aspecto** | **Árvore B** | Árvore B+ |
| Armazenamento de dados | Nós internos | Somente folhas |
| Acesso sequencial | Não diretamente suportado | Suporta por meio de ponteiros |
| Busca | Mais complexidade em consultas por intervalo | Mais eficientes em consultas por intervalos |
| Eficiência de leitura | Menos eficiente em grandes volumes | Mais eficiente devido à estrutura linear |

**Exemplo de aplicação em banco de dados:**

* **Árvore B:** Um banco de dados usa uma Árvore B para armazenar índices e dados juntos. Cada nó carrega informações completas, e uma busca pode encontrar o registro em um nó interno, economizando tempo em certas operações.
* **Árvore B+:** Um sistema que frequentemente realiza buscas sequenciais ou por intervalo (como consultar todas as vendas de um cliente em um mês) utiliza uma Árvore B+, onde as folhas são conectadas. Isso permite acessar os registros diretamente, sem precisar navegar entre nós internos.

Com essa estrutura, as Árvore B e B+ formam a base de muitos sistemas de arquivos modernos, como NTFS (Windows) e EXT4 (Linux), além de bancos de dados relacionais, como MySQL e PostgreSQL.

Imagem de exemplo de como é o formato de uma árvore B: <https://www.inf.ufsc.br/~aldo.vw/estruturas/Estruturas.GerArq-2.html>



**Definições da Árvore B**

* Uma árvore B é uma árvore de busca onde cada nó é possível ter **múltiplos filhos** e armazenas **múltiplas chaves**
* Ela é projetada para ser eficiente em sistemas de discos, pois minimiza o tempo e a quantidade de acesso devido a sua estrutura de N-filhos
* Como ela é uma generalização da árvore de busca, todo nó pode ainda conter no máximo 2 filhos

**Propriedade da Árvore B**

1. **Nível de Balanceamento:**

* A árvore B é sempre balanceada, ou seja, todos os caminhos da raiz até as folhas possuem o mesmo comprimento. Isso garante que a árvore não fique desbalanceada, mantendo a complexidade de busca, inserção e remoção logarítmica (O(log N)).

1. **Chave e filhos:**

* Cada nó (exceto a raiz) contém entre , onde m é o grau da árvore (o número máximo de filhos que um nó pode ter).
* Cada nó contém entre
* A **raiz** pode ter entre 2 ou m filhos, mas só pode ter uma chave caso seja uma folha

1. **Ordem de Árvore:**

* O **grau** de uma árvore B (também chamado de **ordem**) é o número máximo de filhos que um nó pode ter. A ordem define quantas chaves e filhos os nós podem armazenar.

1. **Chaves Ordenadas:**

* As chaves dentro de cada nó são **ordenadas** em ordem crescente, o que permite que as buscas binárias ocorram dentro do nó.
* As chaves no nó interno atuam como **limites** para decidir em qual subárvore a chave buscada deve estar.

1. **Subárvores e Regras de Relacionamento:**

* Se um nó interno contém a chave k, então todas as chaves na subárvore à esquerda de k são **menores** que k, e todas as chaves na subárvore à direita de k são **maiores** que k.
* A árvore é dividida de maneira que, ao navegar de um nó interno para um nó filho, as chaves sejam **ordem crescente** entre os nós.

**Regras da Árvore B e B+**

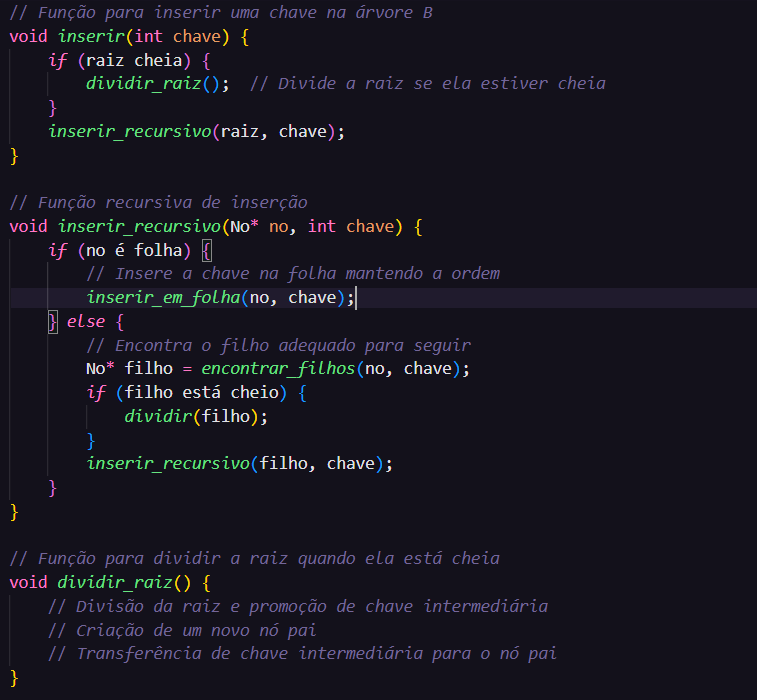
Lembrando que até mesmo a árvore B+ segue estás regras na hora de sua aplicação

1. **Inserção na árvore**

A inserção ela é feita inicialmente com a árvore vazia, assim garantimos que o nó possa ser criado. Com a nó (Raiz) criado, podemos executar uma mini busca para ver se a árvore está vazia ou não, caso não localize nenhum valor naquele local, sua inserção é feita

**A busca é realizada de forma descendente**:

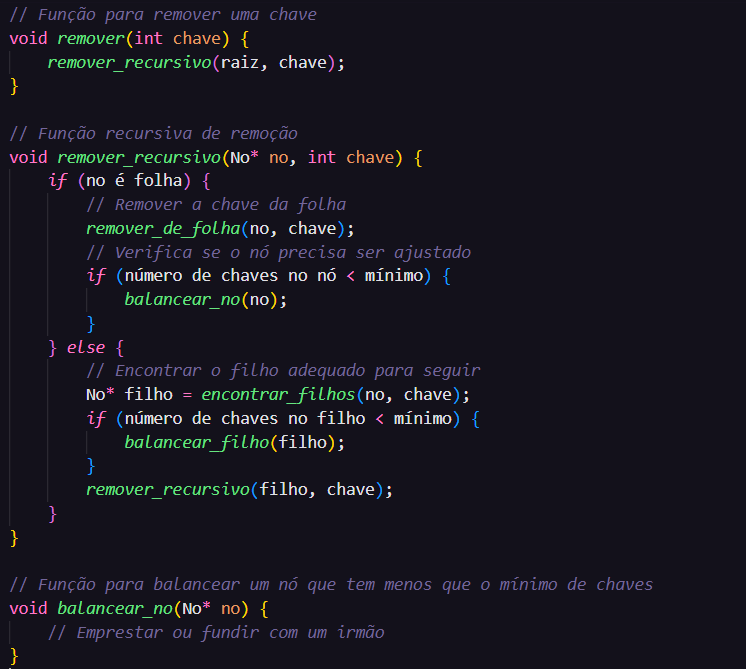
1. Começa na raiz, localizando o intervalo em que a chave pode estar.
2. Segue para o filho correspondente.
3. Continua até alcançar uma folha ou encontrar a chave.



1. **Remoção na chave de busca**

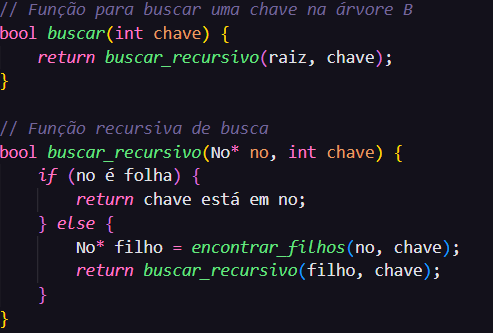
A remoção é obviamente análoga à inserção, com apenas o fato que na hora da exclusão precisa lidar com as propriedades da árvore em questão sejam mantidas. A remoção funciona com base na fundição de nós ou empréstimos de locais para o nó. Ao remover uma chave, três cenários podem ocorrer:

1. **A chave está em uma folha:**
   * Basta removê-la, verificando se o nó não fica abaixo do limite mínimo de chaves
   * Caso fique, redistribuímos ou fundimos nós.
2. **A chave está em um nó interno:**
   * Substituímos pela **maior chave da subárvore esquerda** ou **menor chave da subárvore direita**.
   * Em seguida, removemos essa chave da folha onde ela está.
3. **Fusão:**
   * Se um nó e seu irmão têm menos de m/2m/2m/2 chaves, eles são fundidos, e a chave pai é integrada ao novo nó.

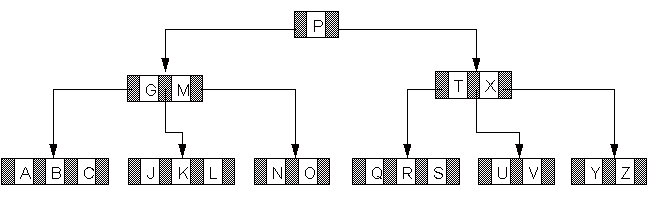


1. **Busca na Árvore B/B+**

A busca é extremamente parecida com a árvore de busca binária com um único, porém, diferente da busca binária onde você tem uma decisão de caminhos com 2 vias de rotas para prosseguir, na busca da árvore, você terá uma sessão de vários blocos de listas para se averiguar para onde cada valor irá se encaixar



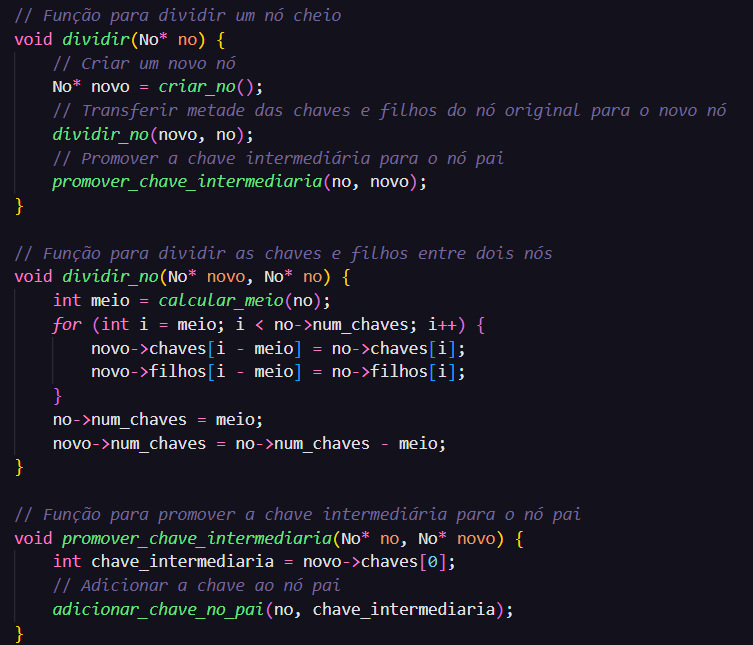
Por exemplo, temos está árvore, onde é separada por letras cada filho/raiz



Vamos supor que queremos colocar outro “Z” dentro da árvore, a primeira coisa que ele irá perguntar é se o Z é maior o não que P, como ele é maior que P, então vai para a direita. Agora, verificamos novamente, se ele é **menor ou maior que T**, caso ele fosse maior que T, porém que X, ele não iria para a direita por completo, ele cairia para a folha onde contém o “U e V”

1. **Divisão de Nó ou Split**

O slit ocorre quando um nó está cheio, funcionando como uma espécie de promotor de chave intermediária. A função do slit é dividir e “subir” o valor central do nó para cima, ou caso o nó que sofreu o split seja uma raiz, ele cria uma raiz com um nó novo, e aquela antiga raiz se torna uma folha



**Comparando com outros tipos**

Segue uma comparação detalhada entre **Árvores B**, **Árvores B+**, **Árvores AVL**, e **Árvores Red-Black**, destacando suas diferenças em desempenho, aplicação prática e complexidade teórica:

**Comparação com Outras Estruturas**

**Árvores AVL**

* **Definição:**  
  As Árvores AVL (Adelson-Velsky e Landis) são árvores binárias de busca auto-balanceadas. Elas mantêm uma diferença de altura (balance factor\text{balance factor}) entre os subárvores esquerda e direita de no máximo 1.
* **Desempenho:**  
  As AVL são altamente balanceadas, o que garante tempos de busca menores em comparação com outras estruturas. No entanto, esse balanceamento rigoroso pode levar a uma maior sobrecarga em operações de inserção e remoção, devido à necessidade de rotações frequentes.
* **Aplicação Prática:**  
  São utilizadas quando o foco é a **eficiência em buscas** e o volume de inserções/remoções não é muito grande, como em tabelas de roteamento ou pequenas bases de dados na memória.

**Árvores Red-Black**

* **Definição:**  
  Uma Árvore Red-Black é uma árvore binária de busca auto balanceada que utiliza um conjunto de regras baseadas em cores (nós somos vermelhos ou pretos) para manter o balanceamento sem ser tão rigorosa quanto as AVL.
* **Desempenho:**  
  As Red-Black Trees oferecem tempos de inserção e remoção melhores do que as AVL, pois seu balanceamento é mais flexível. Porém, a busca pode ser ligeiramente mais lenta devido ao balanceamento menos rigoroso.
* **Aplicação Prática:**  
  São amplamente usadas em **sistemas operacionais** (e.g., Linux usa Red-Black Trees em estruturas como o Schedule) e em **bibliotecas padrão**, como a implementação de mapas ordenados

**Árvores B e B+**

* **Definição:**  
  As Árvores B e B+ são árvores de ordem variável, projetadas para trabalhar eficientemente em armazenamento secundário. Diferentemente das AVL e Red-Black, elas não são binárias; os nós podem conter várias chaves e ter múltiplos filhos.
* **Desempenho:**
  + **Busca:** Mais eficiente em sistemas de armazenamento secundário, pois minimizam o número de operações de entrada e saída (E/S).
  + **Inserção/Remoção:** O custo é mas as operações podem ser mais complexas devido à redistribuição ou divisão de nós.
  + **Varreduras sequenciais:** A Árvore B+ é superior devido aos ponteiros entre folhas.
* **Aplicação Prática:**  
  Utilizadas em **bancos de dados** e **sistemas de arquivos**, onde é necessário lidar com grandes volumes de dados e minimizar o acesso ao disco. Exemplos incluem MySQL (B+) e NTFS (B).

***Tabela Comparativa de Complexidade***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operação** | **Árvore AVL** | **Árvore Red-Black** | **Árvore B** | **Árvore B+** |
| inserção |  |  |  |  |
| remoção |  |  |  |  |
| busca |  |  |  |  |
| Varredura sequencial | Ineficiente | Ineficiente | Moderado | Muito eficiente |

**Análise Teórica e Prática**

**1. Busca:**

* **AVL e Red-Black:** Ambas oferecem busca mas as AVL são ligeiramente mais rápidas devido ao balanceamento mais rígido.
* **Árvore B e B+:** Embora a complexidade teórica seja a mesma, elas são otimizadas para leitura de blocos em disco, tornando-as mais eficientes para sistemas de armazenamento secundário.

**2. Inserção e Remoção:**

* **AVL:** Operações podem ser custosas devido à necessidade de rotações frequentes para manter o balanceamento.
* **Red-Black:** Mais eficiente que AVL, pois permite maior flexibilidade no balanceamento.
* **B/B+:** Inserção e remoção podem ser complexas devido à redistribuição ou divisão de nós, mas o impacto é mitigado em cenários onde as operações são agrupadas.

**3. Armazenamento e Aplicação:**

* **AVL e Red-Black:** Usadas em memória principal, ideais para conjuntos de dados menores e dinâmicos.
* **B e B+:** Focadas em armazenamento secundário, manipulam grandes volumes de dados com eficiência, sendo predominantes em sistemas de banco de dados e arquivos.

**Quando usar cada estrutura?**

* **AVL:** Quando a busca precisa ser extremamente rápida e o conjunto de dados é relativamente pequeno, como em tabelas de roteamento.
* **Red-Black:** Quando operações de inserção e remoção são mais frequentes, e a busca não precisa ser tão rigorosamente otimizada.
* **B e B+:** Para sistemas que manipulam grandes quantidades de dados em memória secundária, como bancos de dados e sistemas de arquivos.
  + **B+:** Preferível para aplicações que exigem consultas sequenciais ou acesso por intervalo.

Com essas comparações, é possível escolher a estrutura mais adequada para diferentes cenários com base nas características de desempenho e requisitos do sistema.