FACULDADE DE EDUCAÇÃO TECNOLÓGICA DO ESTADO DO RIO DE JANEIRO

ADRIEL CAVALCANTE, DOUGLAS GARCIA, KAREN KNUP, RAFAELA ALEXANDRE.

ESTUDO DE CASO: Árvores B+ e B*

Trabalho acadêmico submetido ao curso de graduação em Análise e Desenvolvimento de Sistemas da Faculdade de Educação Tecnológica do Estado do Rio de Janeiro como nota parcial da segunda avaliação da disciplina de Estrutura e dados.

Orientadora: Claudia Ferlin

Rio de Janeiro

2023

Sumário

Introdução

As estruturas de dados desempenham um papel fundamental na organização e manipulação eficiente de informações em sistemas de armazenamento e bancos de dados. Duas estruturas de dados amplamente utilizadas nesse contexto são as árvores B+ e B*.

As árvores B+ e B* são variantes das árvores B, projetadas para otimizar o desempenho de consultas, busca e armazenamento de dados em sistemas que lidam com grandes volumes de informações. Elas são especialmente adequadas para cenários em que a eficiência de leitura e a capacidade de busca por intervalo são essenciais.

Neste trabalho, exploraremos em detalhes as características, funcionamento e aplicações das árvores B+ e B*. Discutiremos como essas estruturas de dados foram projetadas para superar as limitações das árvores B convencionais, visando melhorar o desempenho e a utilização do espaço de armazenamento em sistemas de bancos de dados e armazenamento em disco.

Inicialmente, apresentaremos os conceitos básicos das árvores B+, fornecendo uma visão geral de sua estrutura hierárquica, ordenação e balanceamento. Discutiremos como as árvores B+ lidam com a inserção e remoção de chaves, além de como a estrutura de lista encadeada nas folhas facilita a recuperação sequencial e a busca por intervalo.

Em seguida, abordaremos as árvores B*, destacando as diferenças fundamentais em relação às árvores B+ e como essas diferenças contribuem para uma utilização mais eficiente do espaço de armazenamento em disco. Exploraremos os conceitos de fatores de preenchimento e compartilhamento de nós folhas, discutindo como essas técnicas otimizam o armazenamento de dados e minimizam o número de acessos a disco necessários.

Além disso, apresentaremos exemplos práticos de inserção, remoção e busca em árvores B+ e B*, ilustrando como essas estruturas de dados podem ser aplicadas em cenários reais. Discutiremos também as vantagens e desvantagens dessas estruturas em relação a outras estruturas de árvore, permitindo uma compreensão mais abrangente de seu potencial e limitações.

Por fim, destacaremos as principais aplicações das árvores B+ e B* em sistemas de gerenciamento de bancos de dados, sistemas de arquivos e outros contextos onde a eficiência de leitura, busca por intervalo e armazenamento de dados são cruciais.

Com este trabalho, buscamos fornecer uma visão aprofundada sobre as árvores B+ e B*, explorando suas características, funcionamento e aplicações. Esperamos que este estudo contribua para uma compreensão sólida dessas estruturas de dados e inspire o uso efetivo das mesmas em sistemas de armazenamento e bancos de dados de grande escala.

1. Estruturas

1.1. Árvore B

A B-Tree (Árvore B) é uma estrutura de dado usada para armazenar com eficiência e recuperar grandes quantidades de dados em um dispositivo de armazenamento secundário, como por exemplo discos rígidos.

Essa árvore foi feita para superar as limitações das ABBs (Árvores de Busca Binária) em diversos sentidos:

- Balanceamento — A ABB pode se tornar desbalanceada após várias inserções ou remoções de elemento. Nesse sentido, a B-Tree mantém um balanceamento adequado para garantir a eficácia das operações, pois na B-Tree todas as folhas estão no mesmo nível e a diferença de altura entre os filhos de um nó não excede um certo valor máximo.
- Eficiência em disco — A ABB não é otimizada para acesso a disco. Ela é uma estrutura de dados que reside completamente em memória principal. Portanto, quando a quantidade de dados excede a capacidade da memória principal, é necessário armazenar a árvore em disco, o que pode resultar em um desempenho significativamente reduzido devido aos frequentes acessos a disco.

  O mesmo não ocorre na B-Tree, pois ela geralmente segue uma política de escrita em disco que minimiza os acessos de escrita desnecessários. As alterações nos dados são agrupadas e gravadas em disco em lotes, reduzindo o número total de operações de gravação e melhorando a eficiência em disco.
- Escalabilidade — A ABB pode não ser adequada para cenários em que a quantidade de dados é muito grande porque à medida que o número de elementos na árvore aumenta, a altura da árvore também aumenta linearmente, o que pode levar a um desempenho inferior em operações de busca, pois o número de comparações necessárias para localizar um elemento aumenta proporcionalmente à altura da árvore.

  Ao contrário da ABB, a B-Tree é altamente escalável e é projetada para lidar com conjuntos de dados grandes e crescentes de maneira eficiente. Sua estrutura hierárquica balanceada permite que ela mantenha um desempenho consistente mesmo quando a quantidade de dados aumenta.
- Inserção e Remoção eficientes — A inserção e remoção de elementos em uma ABB podem ser ineficientes em comparação com a B-Tree. Em uma ABB, a inserção e remoção de elementos podem exigir a reestruturação completa da árvore para

manter sua propriedade de ordenação. Isso pode ser custoso em termos de tempo e recursos, especialmente quando a árvore é grande.

Nesse sentido, a B-Tree é mais vantajosa, pois ao inserir um novo elemento em uma B-Tree, a estrutura da árvore pode ser ajustada localmente sem precisar reestruturar a árvore inteira. Isso é possível porque a B-Tree permite que os nós tenham um número mínimo e máximo de chaves. Se a inserção exceder o número máximo de chaves permitido em um nó, ele é dividido em dois nós, e a chave mediana é promovida para o nó pai. Esse processo de divisão ocorre recursivamente, se necessário, até acomodar a nova chave.

Da mesma forma, a remoção em uma B-Tree também é eficiente. A remoção de um elemento não requer uma reestruturação completa da árvore. Se a remoção de uma chave causar uma escassez de chaves em um nó, a redistribuição das chaves entre os nós irmãos é realizada para manter a propriedade de balanceamento da B-Tree.

- Busca por intervalo eficiente — A busca por intervalo em uma ABB pode ser ineficiente em comparação com a B-Tree. Na ABB, não há uma estrutura específica para suportar eficientemente a busca por um intervalo de chaves. É necessário percorrer a árvore e verificar cada elemento individualmente, o que pode resultar em um desempenho inferior em comparação com a B-Tree, que é otimizada para suportar operações de busca por um intervalo de valores.
- Vantagens e Desvantagens da B-Tree:

Vantagens	Desvantagens
Ela minimiza o número de acessos ao disco, reduzindo a quantidade de E/S necessária para ler ou gravar dados.	A implementação correta da B-Tree pode ser complexa e exigir um conhecimento avançado de estruturas de dados e algoritmos para manter a propriedade de balanceamento da árvore.
A B-Tree mantém um equilíbrio entre a altura da árvore e o número de elementos armazenados em cada nó, sendo assim mantém o equilíbrio e o balanceamento da árvore.	A B-Tree requer um ***overhead*** de memória relativamente maior em comparação com outras estruturas de dados, devido aos ponteiros adicionais necessários para manter as referências entre os nós. Isso pode ser uma preocupação em sistemas com restrições de memória.
A B-Tree é altamente escalável e pode lidar com grandes conjuntos de dados. À medida	Em ambientes concorrentes, a implementação de operações de inserção, remoção e busca em uma

que o número de elementos aumenta, a altura da árvore permanece relativamente baixa, o que mantém o desempenho consistente mesmo para grandes volumes de dados.	B-Tree pode se tornar mais complexa devido à necessidade de garantir a consistência e evitar conflitos entre as transações concorrentes.
A B-Tree é projetada para maximizar a utilização eficiente do espaço de armazenamento, sendo assim, ela usa o espaço com eficiência.	_____

Observação: Overhead — é o custo adicional em termos de recursos, como processamento, memória, rede ou armazenamento, que são necessários para realizar tarefas secundárias ou processos auxiliares em uma determinada operação

1.2. Árvore B+

Embora as árvores B e B+ sejam variações da mesma estrutura de dados, existem algumas limitações na B-Tree em comparação com a B+:

- Eficiência em busca por intervalo — A B-Tree não é otimizada para busca por intervalo, pois requer percorrer a árvore da raiz até as folhas para encontrar os elementos desejados. Isso pode resultar em mais acessos a disco e um desempenho inferior em comparação com a B+ que utiliza a lista ligada de nós folha para realizar a busca por intervalo de forma mais eficiente.
- Utilização de espaço — A B-Tree tem uma utilização de espaço menos eficiente em comparação com a B+. Na B-Tree, os nós não folha também armazenam chaves, enquanto na B+ apenas os nós folha armazenam as chaves. Isso significa que a B-Tree usa mais espaço para armazenar as mesmas chaves, o que pode se tornar uma limitação em sistemas com restrições de armazenamento.
- Complexidade de implementação — A B-Tree possui uma complexidade de implementação maior em comparação com a B+. As operações de inserção, remoção e busca na B-Tree envolvem a reestruturação completa da árvore para manter seu balanceamento. Isso requer um cuidado adicional e pode ser mais complexo de implementar e depurar do que na B+.

- Desempenho em leitura sequencial — A B-Tree pode ter um desempenho inferior em leitura sequencial em comparação com a B+. Como a B-Tree possui nós não folha que armazenam chaves, pode haver mais níveis a serem percorridos para encontrar os elementos desejados em uma leitura sequencial, resultando em um desempenho mais lento em relação à B+.
- Benefícios da lista ligada de nós folha — A B-Tree não aproveita os benefícios da lista ligada de nós folha da B+. A lista ligada facilita a navegação eficiente pelos nós folha e permite uma busca por intervalo mais rápida. Essa característica é especialmente útil em aplicações que exigem operações de intervalo frequentes.
- Vantagens e Desvantagens da B+:

Vantagens	Desvantagens
A B+ é especialmente eficiente em operações de busca por intervalo. Devido à estrutura de nós folha ligados em uma lista, é possível percorrer os nós folha de forma sequencial, facilitando a recuperação de dados dentro de um determinado intervalo.	A B+ requer mais espaço para armazenar as chaves, uma vez que elas estão presentes tanto nos nós internos quanto nos nós folha. Isso pode ser uma desvantagem em sistemas com restrições de armazenamento, especialmente quando as chaves são grandes.
A B+ tem uma maior chance de aproveitar o cache de memória devido ao acesso sequencial aos nós folha. Isso resulta em menos leituras de disco, o que pode melhorar significativamente o desempenho em sistemas com alta latência de disco.	A implementação da B+ pode ser mais complexa do que outras estruturas de dados, pois requer o gerenciamento adequado dos ponteiros e a manipulação dos nós internos e folha. O algoritmo de inserção e remoção pode exigir um tratamento cuidadoso para manter a integridade da árvore.
A B+ é otimizada para leituras sequenciais, uma vez que os dados estão organizados nos nós folha em uma ordem definida. Isso é particularmente útil em cenários onde há uma leitura frequente de grandes volumes de dados em ordem.	Embora a B+ seja eficiente em consultas e leituras sequenciais, o desempenho de escrita pode ser relativamente mais lento. Isso ocorre porque a inserção e remoção de elementos exigem a atualização dos nós internos e a reestruturação da árvore.
Devido à eficiência em busca por intervalo e leitura sequencial, a B+ é frequentemente utilizada em bancos de dados e sistemas de	_____

gerenciamento de informações, onde consultas por intervalo são comuns. Ela permite consultas eficientes e rápidas em grandes conjuntos de dados.	

1.3. Árvore B*

A Árvore B*, assim como a Árvore B+, é uma variação da B-Tree (Árvore B). No entanto, a B-Tree possui diferenças quando colocada lado a lado com a B*:

- Utilização de espaço — A B-Tree pode ter uma utilização de espaço menos eficiente em comparação com a B*. Na B-Tree, os nós não folha também armazenam chaves, enquanto na B* é permitido armazenar um número maior de chaves nos nós não folha. Isso significa que a B* pode acomodar mais chaves na mesma quantidade de espaço, o que pode ser benéfico em sistemas com restrições de armazenamento.
- Eficiência em busca por intervalo — A B-Tree não é otimizada para busca por intervalo, pois requer percorrer a árvore da raiz até as folhas para encontrar os elementos desejados. Isso pode resultar em um desempenho inferior em comparação com a B*, que permite uma busca por intervalo mais eficiente usando os nós não folha como ponto de partida.
- Balanceamento — A B-Tree requer reequilíbrio completo da árvore após a inserção ou remoção de elementos para manter o balanceamento. Esse reequilíbrio completo pode ser mais custoso em termos de tempo e recursos em comparação com a B*, que possui um algoritmo de reequilíbrio parcial mais eficiente.
- Complexidade de implementação — O algoritmo de reequilíbrio completo da B-Tree envolve várias etapas e pode ser mais desafiador de implementar e manter em comparação com o algoritmo de reequilíbrio parcial da B*.
- Desempenho em leitura sequencial — A B-Tree pode ter um desempenho inferior em leitura sequencial em comparação com a B*. Devido ao armazenamento de chaves nos nós não folha da B-Tree, pode haver mais níveis a serem percorridos para encontrar os elementos desejados em uma leitura sequencial, resultando em um desempenho mais lento em relação à B*.
- Vantagens e Desvantagens da B*:

Vantagens	Desvantagens
A B* permite uma utilização mais eficiente de espaço em comparação com outras	A B* pode ter uma complexidade de implementação maior do que outras

estruturas de dados, como a B-Tree. Isso ocorre porque a B* permite armazenar mais chaves nos nós não folha, reduzindo a altura da árvore e, consequentemente, o número de acessos a disco.	estruturas de dados, como a B-Tree. O algoritmo de reequilíbrio parcial pode ser mais desafiador de implementar e depurar, exigindo um cuidado adicional para manter a integridade da árvore.
A B* oferece uma eficiência aprimorada em operações de busca. Com mais chaves nos nós não folha, a B* reduz o número de níveis que precisam ser percorridos para encontrar uma chave desejada. Isso resulta em um desempenho mais rápido e menor tempo de acesso a disco em comparação com outras estruturas de dados.	A B* pode ter um desempenho inferior em operações de busca por intervalo em comparação com a B+. Embora a B* seja otimizada para buscas individuais, a busca por intervalo pode exigir percorrer vários nós folha, o que pode afetar o desempenho em comparação com a B+.
A B* apresenta um desempenho de escrita melhorado em comparação com a B-Tree. Isso ocorre porque a B* utiliza um algoritmo de reequilíbrio parcial, que requer menos operações de reestruturação da árvore durante a inserção e remoção de elementos. Isso resulta em um tempo de escrita mais rápido e menor sobrecarga de processamento.	_____
A B* é otimizada para leituras sequenciais. Com a estrutura de nós não folha armazenando mais chaves, a B* reduz o número de níveis que precisam ser percorridos em uma leitura sequencial, melhorando o desempenho geral da operação.	_____

1.4. Diferença entre as árvores B, B+ e B*

B	B+	B*
Os nós da árvore B contêm tanto as chaves quanto os ponteiros para os filhos. Eles podem ter um número variável de chaves e filhos.	Os nós da árvore B+ também contêm chaves e ponteiros para filhos, mas apenas as chaves são armazenadas nos nós folha. Os nós folha estão ligados em uma lista para permitir uma busca eficiente por intervalo.	A árvore B* é uma extensão da árvore B+ e possui a mesma organização dos nós, mas permite que mais chaves sejam armazenadas nos nós não folha, melhorando a eficiência de busca.
A árvore B pode ter um fator de preenchimento mais baixo, o que significa que ela pode ter mais nós vazios em comparação com a B+ e a B*.	Ambas as árvores têm um fator de preenchimento mais alto, o que resulta em uma utilização mais eficiente de espaço, uma vez que mais chaves são armazenadas nos nós.	_____
A busca por intervalo na árvore B requer percorrer a árvore da raiz até as folhas, verificando cada chave. Não há uma estrutura de nós folha específica para busca por intervalo.	As árvores B+ e B* possuem nós folha que estão ligados em uma lista. Isso permite que a busca por intervalo seja realizada de forma eficiente, percorrendo apenas os nós folha em vez de percorrer a árvore inteira.	_____
A árvore B é projetada para operações eficientes em disco, mas a necessidade de percorrer a árvore da raiz às folhas pode resultar em mais acessos a disco em comparação com a B+ e a B*.	As árvores B+ e B* são otimizadas para operações em disco, pois a busca por intervalo pode ser realizada percorrendo apenas os nós folha, minimizando o número de acessos a disco.	_____

2. Áreas de aplicação

As Árvores B tem como vantagem, seu auto-balanceamento e múltiplos índices, assim podendo manter uma altura razoável para ter uma busca eficiente, além de inserções e remoções constantes em suas páginas. Por isso, esse tipo de árvore é usada em sistemas com foco em **armazenamento**, como os destacados:

- Sistemas de arquivos: Árvores B+, são a base para a gestão de dados na memória secundária em HDD ou SSD, como o NTFS do Windows ou o XFS do Linux. Seu uso, deve-se pelo fato de sua ótima eficiência em busca e excelente organização hierárquica dos índices, facilitando o acesso aos arquivos. E já que o acesso é rápido, torna a inserção e remoção eficiente, principalmente por conta das folhas serem encadeadas entre si, podendo realizar alterações em massa num espaço predeterminado.

- Banco de dados: Árvores B+ são muito usadas para banco de dados SQL, alguns bancos NoSQL também a usam, mas geralmente possuem outros focos. Exemplos de sistemas que a usam: MySQL, MongoDB, Oracle. O uso de B+ se dá pela necessidade de buscas constantes no banco de dados, em colaboração com o ACID(Atomicity, Consistency, Isolation, Durability), permitindo a consistência dos dados. As buscas em intervalos são mais simples por conta de sua estrutura. Por exemplo: Numa query SQL que deseja todos dados entre Y e Z, bastando apenas encontrar o primeiro Y e seguir sequencialmente pelas folhas até o último de Z.

- GIS(Geographic Information System): Árvores B tendem a ter presença em sistemas GIS, costumam estar em combinação com a R-Tree(Árvore Retangular), mas não é foco. A árvore B* é mais frequente pela sua alta eficiência em indexação, sem desperdício de espaço, por conta de sua indexação constante em espaços adjacentes(vizinhos), para sistemas do tipo, que possuem a necessidade de armazenar dados espaciais, como Polígonos e Pontos constantemente e os mantendo próximos e reajustando o espaço para uma melhor consulta. Nesse caso, a Árvore B é mais auxiliar, para armazenar os dados em si.

3. Inclusão

3.1. Árvore B

A árvore do tipo B realiza a inclusão em algumas etapas, sendo a primeira a busca pela folha adequada (a folha adequada é a que segue a ordem numérica crescente da esquerda para direita), e essa busca parte da raiz e percorre a árvore inteira até encontrar o local de destino. Após chegar na folha adequada, é feita uma verificação para garantir que o número de chaves seja menor do que o número de nós filhos permitidos pela estrutura (folhas = chaves + 1). Caso esse número não ultrapasse, a inserção termina aqui, caso contrário, a folha a qual ocorreria a inserção é dividida em duas, e a chave do meio sobe para o nó pai, e caso aconteça novamente a mesma situação neste, é realizada outra divisão e este processo ocorre recursivamente, em ultimo caso até a raiz, onde esta também seria dividida e seria criada uma nova raiz.

3.2. Árvore B+

A inclusão na árvore B herda algumas características mas com detalhes que a diferencia bastante da B-Tree comum: a busca pela localização da chave a ser inserida ainda parte da raiz, porém agora as chaves só podem estar nas folhas, enquanto os nós do meio executam papel apenas na navegação. Caso uma chave seja inserida em uma folha e seja necessária uma divisão, uma cópia da chave mais à esquerda da folha mais à direita é promovida para executar esse papel de navegação. Também é bom notar que quando as folhas são divididas é gerado ou aumentado, caso já exista, um encadeamento da esquerda para a direita entre as folhas.

3.3. Árvore B*

Já a árvore B* também herda características da B-Tree na inserção, mas com com várias outras peculiaridades únicas, como por exemplo o fato de só serem divididas e criados novos nós para as folhas, quando as folhas atuais já estiverem com todos os campos de chave preenchidos (a B* utiliza esse método para evitar consumo desnecessário de memória e facilitar buscas). Um exemplo disso é a capacidade de emprestar chaves para folhas irmãs; mais especificamente, quando várias (nesse caso duas) folhas filhas de um mesmo nó, uma preenchida e outra com vagas para chaves, vão participar de uma inserção onde uma chave deve ser inserida na folha cheia, esta passará uma das chaves (no caso se for a folha da esquerda, a chave mais a direita e vice-versa) para o pai, e a chave do pai que separava as duas folhas vai para a outra folha, a com vaga. Outra situação ocorre quando todos os espaços de chaves já foram preenchidos e entre duas folhas referenciadas

por um nó pai, nessa situação o conteúdo das duas folhas somado com o elemento a ser inserido são divididos entre três novas folhas, reiniciando o processo de preenchimento.

4. Remoção

4.1. Árvore B

A exclusão de um elemento de uma árvore B envolve algumas etapas específicas para garantir que a estrutura e as propriedades da árvore sejam preservadas.De uma forma genérica, o algoritmo de exclusão funciona da seguinte forma:

1. Localização do nó contendo o elemento a ser excluído: Percorra a árvore B começando pelo nó raiz e siga os ramos apropriados para encontrar o nó que contém o elemento a ser excluído. Isso envolve a comparação do elemento com as chaves dos nós internos e a seleção do caminho correto até chegar ao nó folha que contém o elemento.
2. Remoção do elemento do nó folha: Ao encontrar o nó folha que contém o elemento a ser excluído, remova-o do nó. Dependendo da implementação da árvore B, pode haver algumas regras específicas para o tratamento de elementos duplicados ou a redistribuição de valores de chaves.
3. Reequilíbrio da árvore, se necessário: A remoção de um elemento pode violar as propriedades da árvore B, como o número mínimo de chaves em cada nó ou a ordem das chaves. Nesse caso, algumas ações podem ser necessárias para reequilibrar a árvore e manter suas propriedades. Isso pode incluir a redistribuição de chaves entre os nós ou a fusão de nós vizinhos.
4. Atualização dos nós pai: Após a reequilibração da árvore, é necessário atualizar os nós pai dos nós que foram afetados pela exclusão. Isso pode envolver a modificação das chaves ou a remoção de um nó quando ocorre a fusão.
5. Verificação se a raiz foi modificada: Se a exclusão resultar em uma modificação na raiz da árvore B, é necessário atualizar a referência para a nova raiz, caso contrário, a estrutura da árvore será afetada.
6. Fim da exclusão: Após todas as modificações necessárias terem sido realizadas, a exclusão do elemento da árvore B é concluída. O elemento foi removido com sucesso e a estrutura da árvore foi mantida conforme as propriedades da árvore B.

A exclusão em uma árvore B apresenta vantagens em relação às árvores AVL e árvores binárias de busca. A redução do número de rotações necessárias, a eficiência em

termos de armazenamento, a redução da fragmentação e o melhor desempenho em cenários com acesso aleatório são benefícios significativos da exclusão em árvores B.

4.2. Árvore B+

A árvore B e a árvore B+ são estruturas de dados amplamente utilizadas em computação. Na Árvore B, a exclusão do nó interno é muito complexa, exigindo que a árvore passe por várias transformações. Por outro lado, na Árvore B+, a exclusão de qualquer nó é fácil, pois todos os nós são encontrados na folha. Cada uma dessas árvores possui suas características e vantagens específicas, tornando-as adequadas para diferentes cenários de aplicação.

Quando um elemento é removido de uma árvore B+, pode ocorrer uma situação em que o número de chaves (elementos) na folha em que a remoção foi realizada fica abaixo de um limite mínimo pré-determinado. Nesse caso, duas ações que devem ser executadas para manter a integridade da árvore:

- Redistribuir chaves de folhas adjacentes: Isso significa que algumas chaves das folhas adjacentes, que têm o mesmo pai, serão transferidas para a folha que possui um número insuficiente de chaves. Essa redistribuição tem como objetivo garantir que todas as folhas tenham um número mínimo de chaves, evitando assim que alguma folha fique muito vazia.
- Concatenar folhas adjacentes: Se a redistribuição não for suficiente para alcançar o número mínimo de chaves na folha, a próxima ação é combinar (concatenar) essa folha com uma ou mais folhas adjacentes que também possuam o mesmo pai. Isso resultará em uma única folha contendo todas as chaves das folhas envolvidas na concatenação. Essa operação é realizada para evitar folhas excessivamente vazias e melhorar a eficiência da árvore B+.

Essas ações são importantes para manter a estrutura e o equilíbrio da árvore B+ após uma remoção, garantindo que todas as folhas tenham um número mínimo de chaves e otimizando o desempenho da árvore em operações futuras.

As operações de inserção e remoção são otimizadas na árvore B+, pois as alterações ocorrem principalmente nos nós folha, sem afetar a estrutura da árvore.

4.3. Árvore B*

A vantagem da remoção na árvore B* em relação à árvore B tradicional está relacionada ao desempenho e à minimização da fragmentação da estrutura da árvore.

Na árvore B tradicional, quando uma chave é removida de um nó interno, pode ser necessário reorganizar a estrutura da árvore, realizando fusões ou redistribuições de chaves entre os nós adjacentes para manter o balanceamento. Essa reestruturação pode ser um processo complexo e pode exigir várias operações para garantir que a árvore permaneça balanceada.

Por outro lado, na árvore B*, a remoção de uma chave em um nó interno não requer imediatamente a reestruturação da árvore. As chaves de intervalo existentes no nó interno são mantidas mesmo que o número de chaves regulares caia abaixo de um limite mínimo,. Isso evita a necessidade de reequilibrar a árvore imediatamente após a remoção de uma chave.

Quando uma chave é removida de uma folha em uma árvore B*, ocorre uma verificação para determinar se a folha fica abaixo do limite mínimo de ocupação, que é definido como 1/3 da capacidade máxima da folha.

Se a remoção de uma chave resultar em uma folha com um número menor de chaves do que o limite mínimo de ocupação, a técnica 2/3 pode ser aplicada. Nesse caso, ocorre uma reestruturação da árvore.

A reestruturação na remoção envolve a redistribuição das chaves da folha atual com outra folha adjacente. As chaves das duas folhas são combinadas em uma única folha, que terá aproximadamente 2/3 das chaves originais.

Essa redistribuição de chaves entre as folhas adjacentes visa manter um número suficiente de chaves em cada folha, evitando que alguma folha fique excessivamente vazia. Dessa forma, busca manter o balanceamento da árvore mesmo após a remoção de chaves.

É importante mesmo com a redistribuição na remoção, ainda pode ser necessário ajustes adicionais na estrutura da árvore, como fusões ou redistribuições em níveis superiores da árvore, caso a redistribuição das chaves nas folhas inferiores afete o balanceamento em níveis superiores.

Essa reestruturação adiada na árvore B* evita a fragmentação desnecessária da estrutura da árvore e reduz a quantidade de reorganização necessária. Como resultado, o desempenho das operações de remoção pode ser melhorado, uma vez que não é necessário realizar imediatamente um "split" ou uma fusão após cada remoção.

No entanto, é importante observar que, em algum momento, a reestruturação da árvore ainda pode ser necessária para manter o balanceamento. Portanto, embora a árvore B* adie o "split" e a reorganização após a remoção, eventualmente essas operações podem ser realizadas para garantir que a árvore permaneça balanceada a longo prazo.

5. Busca

Nas árvores binárias, a busca é feita usando o algoritmo de busca binária, no qual se baseia numa lista de dados classificados, onde é pego a metade da lista como base da busca, indo para a esquerda caso o valor buscado for menor e para direita caso o valor buscado for maior. Tendo no pior dos casos, O(log n) possibilidades, algo que seria O(n) numa busca convencional. As árvores binárias são estruturadas justamente pensando na busca.

5.1. Árvore B

Na árvore B, a busca é usando o algoritmo de busca binária, porém cada nó possui 1 apontamento tanto para esquerda quanto para direita, começando por sua raiz e navegando para esquerda ou direita com base na busca e o nó que está sendo testado. Ao achar a página(o conjunto de Nós, como pode ver na imagem abaixo, um exemplo é o 14 e o 16). A busca a percorre até achar o nó buscado.

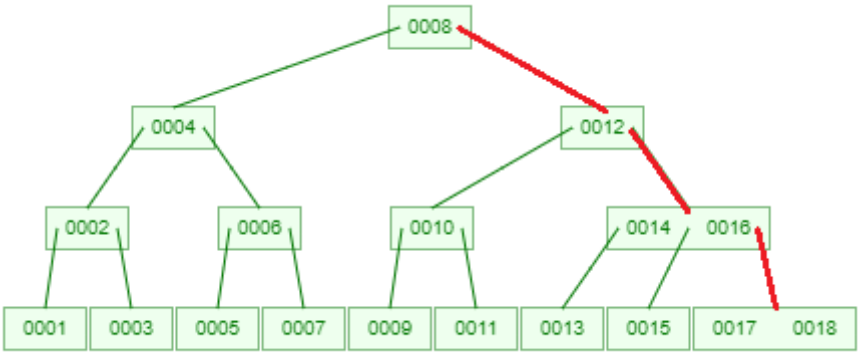

figura X: Busca em Árvore B

5.2. Árvore B+

Na árvore B+, a busca é da mesma forma que na Árvore B, porém, como podemos consultar apenas as folhas, não teremos casos em que a busca se encerra num galho. A vantagem da busca da Árvore B+, é mais por conta do encadeamento das folhas, como no exemplo abaixo, estou buscando o valor 10. Caso eu queira saber o valor posterior ao buscado, eu não precisarei realizar outra busca, já que as folhas estão encadeadas.

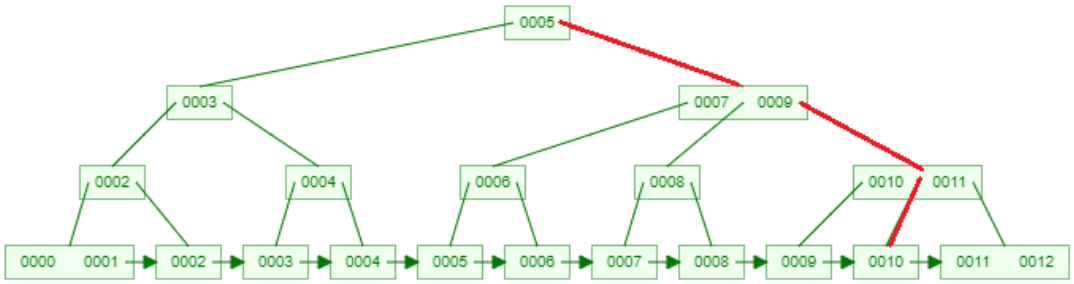

figura X: Busca em Árvore B+

5.3. Árvore B*

Na árvore B*, a busca é literalmente da mesma forma que na Árvore B, tendo em vista que as folhas não são encadeadas, sua mudança é significativa em relação à inclusão e exclusão.

Conclusão

Com base no que foi dito, percebe-se que as árvores B+ e B* são estruturas de dados poderosas e eficientes projetadas para lidar com grandes volumes de informações em sistemas de armazenamento e bancos de dados. Elas superam as limitações das árvores B convencionais, oferecendo melhor desempenho e utilização do espaço de armazenamento em disco.

As árvores B+ destacam-se pela sua estrutura hierárquica, ordenação e balanceamento, permitindo uma recuperação sequencial eficiente e busca por intervalo. A presença da estrutura de lista encadeada nas folhas facilita a manipulação de dados e a busca por intervalo, tornando-as adequadas para cenários em que a eficiência de leitura é crucial.

Por outro lado, as árvores B* diferenciam-se das árvores B+ pela aplicação de técnicas como fatores de preenchimento e compartilhamento de nós folhas. Essas técnicas otimizam o armazenamento de dados e reduzem o número de acessos a disco necessários, proporcionando uma utilização mais eficiente do espaço de armazenamento em disco.

Ambas as estruturas de dados têm vantagens e desvantagens, e a escolha entre elas depende do contexto e dos requisitos específicos do sistema. No entanto, tanto as árvores B+ quanto as árvores B* são amplamente utilizadas em sistemas de gerenciamento de

bancos de dados, sistemas de arquivos e outros contextos onde a eficiência de leitura, busca por intervalo e armazenamento de dados são cruciais.

Com este estudo, esperamos ter fornecido uma visão aprofundada sobre as árvores B+ e B*, explorando suas características, funcionamento e aplicações. Ao compreender suas vantagens e desvantagens, os profissionais e pesquisadores podem tomar decisões informadas ao projetar e implementar sistemas de armazenamento e bancos de dados de grande escala. O uso efetivo das árvores B+ e B* pode resultar em melhor desempenho, eficiência e utilização de recursos, contribuindo para o avanço da área de sistemas de informação.

Referências

Adam Drozdek, Cengage Learning, Árvores de multivias(B,B+,B*) 2013. Disponível em <https://moodle.ufsc.br/pluginfile.php/2562394/mod_resource/content/1/MultiwayTree_AdamDrozdek.pdf> Acesso em 26 de junho de 2023 às 12:34

B+ TREE, Java T Point, estrutura de dados 2011-2021. Disponível em <https://www.javatpoint.com/b-plus-tree> Acesso em 26 de junho de 2023 às 13:06

B-TREEs, usfca.edu, B-tree algorithm visualization. Disponível em <https://www.cs.usfca.edu/~galles/visualization/BTree.html> Acesso em 26 de junho de 2023 às 15:52

Árvore B, Wikipedia <Árvore B – Wikipédia, a enciclopédia livre (wikipedia.org)> Acesso em 28 de junho de 2023 às 18:23

B*-Trees, GeeksforGeeks <B*-Trees implementation in C++ - GeeksforGeeks> Acesso em 27 de junho de 2023 às 17:50

Combining R-Tree and B-Tree to Enhance Spatial Queries Processing, GeeksforGeeks <https://aast.edu/pheed/staffadminview/pdf_retreive.php?url=44355_153_1_PID2979945(1).pdf&staffttype=staffpdf> Acesso em 28 de junho de 2023 às 20:40

B+ Tree, Wikipedia <https://en.wikipedia.org/wiki/B%2B_tree> Acesso em 28 de junho de 2023 às 19:37

ArvoreB+ e B*, Gpec<http://www.gpec.ucdb.br/pistori/disciplinas/ed/aulas_II/bp.htm> Acesso em 25 de junho de 2023 às 15:00
ÁrvoreB*,Dra.CristinaDutradeAguiar
Ciferri<http://wiki.icmc.usp.br/images/f/f0/SCC0215012015arvoreBvariacoesBestrela.pdf> Acesso em 22 de junho de 2023 às 17:50

Árvore B+ - Remoção, Chintia Caliari <https://www.youtube.com/watch?v=_kYOntAPyCE> Acesso em 20 de junho de 2023 às 14:50

ÁrvoreB+,Unicamp<https://www.ic.unicamp.br/~thelma/gradu/MC326/2010/Slides/Aula11b-ArvoreB+_pre_fixada.pdf> Acesso em 26 de junho de 2023 às 19:00

Árvore B+, Programiz <https://www.programiz.com/dsa/b-plus-tree> Acesso em 24 de junho de 2023 às 20:30

B-Tree & B*-Tree Explained - Algorithms & Data Structures #23 <(140) B-Tree & B*-Tree Explained - Algorithms & Data Structures #23 - YouTube> Acesso em 24 de junho de 2023 às 22:20