UNIVERSIDADE FEDERAL DE ITAJUBÁ

CIÊNCIA DA COMPUTAÇÃO

DESENVOLVIMENTO DE UMA APLICAÇÃO PARA INDEXAÇÃO DE DADOS UTILIZANDO ÁRVORES B

2022007164 - Gabriel Santos do Amaral
2022004233 - Gustavo Totti Custódio dos Santos
2022008090 - Iago Martins Silva
2024006596 - Isaac do Prado Almeida
2022004897 - Murilo Martinez Zaina

CTCO02 - Algoritmos e Estruturas de Dados II

Prof. Vanessa Cristina

INSTITUTO DE MATEMÁTICA E COMPUTAÇÃO
UNIFEI - Itajubá

2024

UNIVERSIDADE FEDERAL DE ITAJUBÁ
INSTITUTO DE MATEMÁTICA E COMPUTAÇÃO
CIÊNCIA DA COMPUTAÇÃO
CTCO02 - ALGORITMOS E ESTRUTURAS DE DADOS II

Desenvolvimento de uma Aplicação para Indexação de Dados Utilizando Árvores B

1 Introdução

O presente trabalho tem como objetivo desenvolver uma aplicação utilizando a linguagem de programação C, que simule a funcionalidade de um índice em um banco de dados, empregando a estrutura de árvore B. As árvores B são amplamente utilizadas em sistemas de gerenciamento de banco de dados (SGBDs) devido à sua eficiência na indexação e busca de registros, facilitando a recuperação rápida de informações armazenadas.

1.1 Contextualização

As árvores B são um tipo específico de árvore balanceada que permite armazenar e buscar dados de forma eficiente. Introduzidas por Bayer & McCreight (1971), estas estruturas de dados são amplamente utilizadas em Sistemas de Gerenciamento de Banco de Dados (SGBDs) para organizar e manipular grandes volumes de dados. Sua principal vantagem é que elas mantêm os dados ordenados e permitem operações de busca, inserção e remoção em tempo logarítmico $O(\log n)$, onde n é o número de elementos armazenados.

Em um banco de dados normal, um índice cria arquivos classificados que apressariam a localização de um arquivo. Quando é feita uma consulta, um banco de dados utiliza o índice para encontrar rapidamente o registro desejado, evitando a necessidade de percorrer toda a tabela de dados. Isso é especialmente útil em bancos de dados maiores, onde a busca eficiente é crucial para o desempenho do sistema.

1.2 Objetivos

Os objetivos desse projeto incluem:

1. Implementar uma árvore B em memória RAM para simular o índice em um banco de dados.
2. Desenvolver funcionalidades de inserção, remoção e busca utilizando a árvore B.
3. Comparar o desempenho da busca utilizando a árvore B com busca direta no arquivo de dados.
4. Analisar os resultados obtidos para avaliar a eficiência da árvore B na indexação de dados.

1.3 Relevância do Tema

A escolha do tema se justifica pela importância das árvores B na área de ciência da computação, especialmente em sistemas de banco de dados. Compreender e implementar estruturas de dados eficientes é fundamental para o desenvolvimento de aplicações robustas e escaláveis. Além disso, este projeto proporciona uma experiência prática na aplicação de conceitos teóricos em um cenário realista, promovendo o aprendizado ativo (metodologia PBL) e o desenvolvimento de habilidades técnicas e analíticas.

2 Fundamentação Teórica

Nesta seção, abordaremos os conceitos teóricos fundamentais para a compreensão das árvores B e sua aplicação na indexação de bancos de dados. Este embasamento é essencial para a implementação e análise do projeto proposto.

2.1 Árvores B

As árvores B são um tipo de árvore de busca balanceada introduzidas por Bayer & McCreight (1971). Elas são especialmente projetadas para trabalhar com grandes volumes de dados e são amplamente utilizadas em sistemas de gerenciamento de banco de dados (SGBDs) devido à sua eficiência.Comer (1979) destaca a ubiquidade das Árvores B em sistemas de armazenamento de dados devido à sua eficiência e capacidade de manter a estrutura balanceada.

A principal característica das árvores B é a manutenção de dados ordenados e a realização de operações de busca, inserção e remoção em tempo logarítmico $O(logn)$.

2.1.1 Estrutura e Propriedades da Árvore B

As árvores B possuem uma estrutura que garante balanceamento e eficiência. Cada nó da árvore pode conter múltiplas chaves e filhos, e as seguintes propriedades são mantidas:

- **Balanceamento**: Todas as suas folhas estão no mesmo nível, garantindo uma profundidade uniforme e operações eficientes.
- **Capacidade dos Nós**: Cada nó interno pode conter um número variável de chaves, mas esse número está limitado a um intervalo específico definido durante a construção da árvore.
- **Chaves Ordenadas**: As chaves em cada nó são mantidas em ordem crescente, permitindo busca binária eficiente.
- **Divisão de Nós**: Quando um nó atinge a capacidade máxima, ele é dividido, e a chave mediana é promovida ao nó pai, mantendo a árvore balanceada.

As árvores B são uma generalização das árvores binárias de busca, onde cada nó pode armazenar mais de uma chave e mais de um ponteiro para os filhos. Esta característica permite que elas sejam otimizadas para operações de entrada e saída (E/S) em discos, minimizando o número de acessos ao disco necessários para encontrar um registro.

2.1.2 Ordem (m)

Na literatura, a ordem de uma árvore B é a quantidade máxima de filhos que um nó pode ter. A ordem determina a estrutura da árvore e influencia diretamente o seu desempenho. Especificamente:

- **Número máximo de chaves em um nó**: Em uma árvore B de ordem m, um nó pode ter no máximo $m - 1$ chaves.
- **Número mínimo de chaves em um nó**: Um nó (exceto a raiz) deve ter pelo menos $\lceil \frac{m}{2} \rceil - 1$ chaves.
- **Número máximo de filhos de um nó**: Um nó pode ter no máximo m filhos.

Essas propriedades garantem que a árvore permaneça bem organizada e eficiente. Sua altura é proporcional ao logaritmo do número total de nós, o que significa que a árvore cresce em altura muito lentamente à medida que mais nós são adicionados. Escolher uma ordem adequada para a árvore B é crucial para otimizar a eficiência das operações de inserção, remoção e busca.

2.2 Aplicação das Árvores B na Indexação de Bancos de Dados

A eficiência no acesso a dados é crucial em SGBDs, e a indexação é uma técnica chave para melhorar o tempo de resposta das consultas ao banco de dados. Um índice é uma estrutura de dados que permite ao SGBD localizar registros de maneira rápida e eficiente.

2.2.1 Funcionamento da Indexação

A indexação em bancos de dados é realizada criando-se um índice, que é um arquivo ordenado separado do arquivo de dados principal. Este índice contém pares de chave e ponteiro, onde a chave é o valor pelo qual a busca é realizada e o ponteiro aponta para o local do registro correspondente no arquivo de dados. Quando uma consulta é feita utilizando a chave indexada, o SGBD pode usar o índice para localizar rapidamente o registro sem a necessidade de percorrer todo o arquivo de dados. Segundo Comer (1979), as árvores B são particularmente eficientes para esse propósito devido à sua estrutura balanceada e ao fato de que minimizam o número de acessos ao disco.

2.2.2 Vantagens das Árvores B na Indexação

As árvores B são preferidas para a indexação em bancos de dados por várias razões:

- **Eficiência na Busca**: Devido à estrutura balanceada, sua busca pode ser realizada em tempo $O(logn)$, o que é muito mais eficiente do que uma busca linear. Essa eficiência é crucial para manter o desempenho em bancos de dados com grandes volumes de dados. Além disso, as operações logarítmicas são essenciais para o desempenho eficiente em grandes bancos de dados.
- **Inserções e Remoções Rápidas**: A capacidade de inserção e remoção de chaves sem desbalancear a árvore torna as Árvores B ideais para ambientes onde os dados estão em constante mudança. Essa característica é essencial para a operação de SGBDs modernos, que frequentemente enfrentam atualizações dinâmicas.
- **Minimização de Acessos ao Disco**: A estrutura das Árvores B é otimizada para minimizar o número de acessos ao disco, o que é crucial para o desempenho em bancos de dados de grande escala. Essa otimização é uma das principais razões pelas quais esse tipo de árvore são amplamente utilizadas em sistemas de arquivos e bancos de dados modernos.

3 Desenvolvimento

Nesta seção, descreveremos detalhadamente o processo de desenvolvimento da aplicação para indexação de dados utilizando Árvores B, com ênfase nas funções específicas do código implementado. Nosso objetivo é explicar a lógica por trás de cada função e como elas se interconectam para formar uma aplicação eficiente e funcional.

Abordaremos a implementação das principais operações da Árvore B, incluindo busca, inserção, remoção e códigos específicos da árvore B, como emprestar e fundir nós.

3.1 Funções de Busca

As funções de busca são cruciais para sua eficiência, permitindo a localização de registros específicos de forma rápida e eficiente.

3.1.1 Função *buscarNoArquivo*

Essa função é responsável por **buscar um registro específico em um arquivo de dados** baseado na matrícula fornecida. Nesta função, utilizamos as funções **strtok** e **strcspn** para processar e analisar o conteúdo do arquivo de dados de forma eficiente.

A função strtok é usada para dividir uma string em tokens, utilizando delimitadores especificados. No caso da buscarNoArquivo, strtok é utilizada para separar as colunas de cada linha do arquivo de dados, permitindo a extração de informações como matrícula, nome e outros campos. A strtok percorre a string original, substituindo os delimitadores (neste caso, vírgulas) por caracteres nulos e retorna um ponteiro para o início de cada token.

A função strcspn calcula o comprimento do segmento inicial de uma string que não contém nenhum dos caracteres especificados. Na função buscarNoArquivo, strcspn é usada para encontrar a posição do caractere de nova linha no final de cada linha lida do arquivo, permitindo que esse caractere seja substituído por um caractere nulo. Isso facilita a manipulação e comparação das strings.

3.1.2 Função *'buscarElemento'*

A **função *'buscarElemento'* busca uma matrícula específica** na Árvore B e retorna o número da linha correspondente no arquivo de dados. Ela inicia comparando a matrícula buscada com as matrículas nos registros do nó atual, procurando a posição correta na lista ordenada.

Se encontrar a matrícula, retorna o número da linha. Caso contrário, verifica se o nó é uma folha; se for, indica que a matrícula não foi encontrada. Se não for folha, a função continua a busca recursivamente no filho apropriado, explorando a subárvore correspondente.

3.2 Funções de Inserção

As funções de inserção são responsáveis por adicionar novos registros na Árvore, garantindo que a estrutura permaneça balanceada e eficiente.

3.2.1 Função *'inserir'*

A função *'inserir'* adiciona uma nova chave na Árvore B. Se a raiz está cheia, a árvore cresce em altura antes da inserção. Isso envolve a criação de um novo nó que se torna a nova raiz, seguido pela divisão da antiga raiz. Após essa divisão, a função *'inserirNaoCheio'* (3.2.2) é chamada para inserir a nova chave no local apropriado, garantindo que a árvore permaneça balanceada durante a inserção.

3.2.2 Função *'inserirNaoCheio'*

Esta função insere uma nova chave em um nó que não está cheio. Se o nó for folha, a nova chave é inserida diretamente na posição correta. Se não for, a função localiza o filho apropriado para continuar a inserção. Se esse filho estiver cheio, ele é dividido antes da inserção da nova chave.

3.3 Funções de Remoção

As funções de remoção são responsáveis por retirar registros da Árvore, mantendo suas propriedades estruturais.

3.3.1 Função *'remover'*

A função *'remover'* é usada para retirar uma chave da Árvore B. Se a raiz da árvore se torna vazia após a remoção, a árvore é reestruturada para eliminar nós vazios, possivelmente ajustando a raiz. Esta função chama *'removerNo'* (3.3.2) para efetuar a remoção da chave do nó apropriado, garantindo que a árvore permaneça funcional e balanceada.

3.3.2 Função *'removerNo'*

Esta função remove uma chave de um nó específico. Ela localiza a chave a ser removida e, dependendo se o nó é folha ou não, utiliza diferentes estratégias. Se o nó for folha, a chave é removida diretamente. Se o nó não for folha, a função encontra o sucessor ou predecessor da chave e substitui a chave a ser removida por essa nova chave. Após isso, a função continua a remoção no nó apropriado.

3.4 Funções da Árvore B

Para manter a árvore balanceada e eficiente, a Árvore B utiliza algumas funções específicas que serão citadas nesta seção.

3.4.1 Função *'emprestar'*

Assemelha-se com a função de rotação das Árvores binárias. É utilizada quando um **nó não possui chaves suficientes e precisa pegar uma chave emprestada de um nó irmão**. Isso ajuda a manter as propriedades da Árvore, garantindo que todos os nós tenham o número mínimo de chaves necessário.

Esta função ajusta os ponteiros e as chaves dos nós afetados, promovendo uma chave do nó pai para o nó que está emprestando a chave e movendo uma chave do nó irmão para o pai.

3.4.2 Função *'fundir'*

Também conhecida como **Merge**, a função *'fundir'* é utilizada quando um nó não possui chaves suficientes e o empréstimo de uma chave de um nó irmão não é possível. Neste caso, **dois nós irmãos são combinados em um único nó**, movendo a chave mediana do nó pai para o novo nó fundido, e combinando os filhos dos dois nós originais. Isso reduz o número total de nós na árvore, mas mantém suas propriedades balanceadas e eficientes.

3.4.3 Função *'dividirFilho'*

Também conhecida como **Split**, é responsável por dividir um nó filho quando ele está cheio. Quando um nó atinge a capacidade máxima de chaves, a função divide este nó em dois nós menores e promove a chave mediana para o pai. Esta operação garante que a árvore permaneça balanceada e que nenhum nó exceda sua capacidade máxima de chaves.

Ela é chamada durante a inserção de uma nova chave, quando o nó no qual a chave deve ser inserida está cheio. Ao dividir o nó e promover a chave mediana, a função facilita a inserção da nova chave e mantém a árvore balanceada.

3.4.4 Função *preencher*

Esta função é utilizada para garantir que um nó tenha o número mínimo de chaves após uma operação de remoção (3.3). Quando um nó filho possui menos chaves do que o mínimo permitido, a função é chamada.

Ela determina se deve emprestar uma chave de um nó irmão ou fundir o nó filho com um nó irmão, baseado na disponibilidade de chaves presentes nos irmãos. Essa função é essencial para manter a integridade da Árvore B, assegurando que todos os nós, exceto a raiz, mantenham o número mínimo de chaves necessário.

3.5 Tratamento do grau da árvore

Durante o desenvolvimento do projeto, havíamos optado por aceitar apenas graus ímpares para a árvore. Essa escolha simplificaria a implementação das operações principais, como inserção e remoção, e garantiria a manutenção da estrutura balanceada da árvore.

Isso facilitaria a operação de divisão (split), pois sempre há uma chave central clara para promover ao nó pai quando um nó está cheio. Além disso, as operações de redistribuição de chaves entre nós irmãos tornariam-se mais equilibradas, mantendo a árvore consistente e evitando a necessidade de tratar muitos casos especiais.

No entanto, após diversas tentativas e discussões entre os membros de nosso grupo, identificamos que as dificuldades técnicas e o nível de complexidade dessa tarefa seriam muito altos para conseguirmos implementá-la. Assim, nossa árvore B possui um grau padrão, que é utilizada em todos os casos.

4 Resultados

O presente estudo teve como objetivo desenvolver uma aplicação para indexação de dados utilizando Árvores B, analisando detalhadamente o desempenho e a eficiência dessa estrutura de dados em diferentes operações.

Para avaliarmos se a busca utilizando o índice da Árvore B performa melhor do que a busca direta no arquivo, realizamos um experimento comparativo. Ele envolveu a realização de 30 buscas aleatórias, tanto utilizando a Árvore B, quanto buscando diretamente no arquivo.

4.1 Ambiente de Testes

Os testes foram realizados em um computador sob ambiente controlado, cujas especificações são:

- Processador: Intel Core i7-10700 / 2.90GHz, 8 núcleos
- RAM instalada: 8 GB
- Tipo de sistema: Sistema operacional de 64 bits, processador baseado em x64

4.2 Metodologia dos Testes

4.2.1 Geração do Arquivo de Testes

Para os testes realizados, um arquivo contendo mais de 10 mil registros foi gerado aleatoriamente. Cada registro continha informações como matrícula, nome e outros dados relevantes para o teste. O objetivo era criar um conjunto de dados suficientemente grande para avaliar a eficiência das operações de busca.

4.2.2 Configuração da Árvore B

Antes de iniciar os testes de busca, uma Árvore B foi construída utilizando os dados do arquivo gerado. Vale ressaltar que o tempo de construção da Árvore B não foi considerado nos testes, focando exclusivamente no tempo utilizado para busca.

4.2.3 Busca Direta no Arquivo

Para simular a busca direta, implementamos um algoritmo 3.1.1 que percorre o arquivo linha por linha até encontrar o registro desejado. Foram realizadas 30 buscas aleatórias, e o tempo de cada busca foi registrado. Este método serviu como base de comparação para avaliar a eficiência da Árvore B.

4.2.4 Busca utilizando Árvore B

Utilizando a Árvore B previamente construída, implementamos o algoritmo de busca 3.1.2 e realizamos as mesmas 30 buscas aleatórias. O tempo de cada busca foi registrado para comparação com a busca direta no arquivo.

4.2.5 Coleta de Dados

A parir dos testes realizados calculamos a média, o valor máximo e o valor mínimo do tempo gasto em ambas as abordagens.

Os dados coletados foram comparados para determinar qual método de busca foi mais eficiente. A análise incluiu a geração de um gráfico comparativo para visualizar claramente as diferenças de desempenho entre as duas abordagens.

4.3 Resultado dos Testes

O gráfico abaixo mostra a comparação dos tempos de busca direta no arquivo e utilizando a Árvore B:

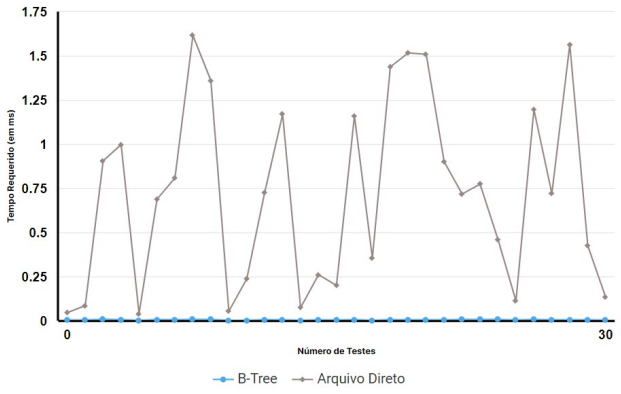

Figura 1: Gráfico do Tempo de Busca: Árvore B x Arquivo Direto

A partir do gráfico 1, podemos observar claramente que a busca utilizando a Árvore B é consistentemente mais rápida do que a busca direta no arquivo, observando que os resultados da busca utilizando a Árvore B no gráfico estão muito próximos à linha do eixo X (quase zero). Para facilitar a visualização dos resultados, uma tabela foi criada mostrando os valores máximo, médio e mínimo do tempo de busca para ambos os métodos.

Tabela 1: Comparação entre o desempenho em Árvore B x Arquivo Direto

	Árvore B (ms)	**Arquivo (ms)**
Tempo máximo	0.008	1.615
Tempo médio	0.005	0.739
Tempo mínimo	0.002	0.039

Os resultados apresentados na Tabela 1 demonstram uma melhora significativa no desempenho da busca utilizando a Árvore B em comparação com a busca direta no arquivo. Especificamente, a busca utilizando a Árvore B apresentou um tempo máximo de 0.008 ms, enquanto a busca no arquivo teve um tempo máximo de 1.615 ms. Isso representa uma diferença

drástica do tempo de busca, indicando que a Árvore B é extremamente eficiente para operações de busca em grandes volumes de dados.

Além disso, o tempo médio de busca com a Árvore B foi de 0.005 ms, comparado a 0.739 ms na busca direta no arquivo. Essa diferença evidencia a superioridade da Árvore B em termos de consistência e rapidez, proporcionando uma experiência de busca muito mais ágil. Além disso, o tempo mínimo de busca também foi menor utilizando a Árvore B (0.002 ms) em comparação ao arquivo (0.039 ms), reforçando a vantagem da estrutura de dados da Árvore B.

Essa análise confirma que a utilização de uma Árvore B para indexação de dados melhora a performance das buscas, tornando-a uma estrutura de dados altamente eficiente para aplicações que necessitam de operações rápidas em grandes volumes de dados.

5 Conclusão

Os resultados deste estudo mostram que a busca utilizando uma Árvore B é mais rápida do que a busca direta em um arquivo. Evidentemente, a eficiência na recuperação de registros foi comprovada pelo menor tempo médio de busca via Árvore B, bem como pelos melhores tempos máximo e mínimo, demonstrando consistência no desempenho.

A análise confirma que a Árvore B é um tipo de estrutura de dados altamente eficaz na indexação de dados e excelente para aplicações que necessitam de operações rápidas em grandes volumes de dados. Se corretamente balanceadas e eficientes, podem garantir respostas rápidas a consultas e operações, como em bancos de dados, sistemas de arquivos e outros ambientes.

Em suma, as conclusões deste estudo fornecem uma base sólida para a implantação das Árvores B em sistemas com alto desempenho e confiabilidade exigindo pesquisa e recuperação informações.

Referências

Bayer, R. & McCreight, E. M. (1971). Organization and maintenance of large ordered indexes. *Acta Informatica*.

Comer, D. (1979). The ubiquitous b-tree. *ACM Computing Surveys*.