Análise Experimental de Métodos de Pesquisa Externa

Ciro Junio, Lucas Terra, Brihan Jesus, Jonhatan Evangelista, Augusto Luna
Departamento de Computação
Universidade Federal de Ouro Preto, Mariana, Brasil

***Abstract*—Este trabalho analisa o desempenho de métodos de pesquisa externa: acesso sequencial indexado, árvore binária, árvore B e árvore B*. Foram implementados algoritmos em C e realizados experimentos com diferentes tamanhos e ordenações de arquivos, avaliando transferências de memória, comparações e tempo de execução. Os resultados permitem comparar a eficiência e aplicabilidade dos métodos em sistemas de grande volume de dados.**

***Index Terms*—Pesquisa externa, desempenho, acesso sequencial indexado, árvore binária, árvore B, árvore B*, algoritmos, análise experimental, sistemas de grande volume de dados**

I. Pesquisa Externa

A pesquisa externa é um método de recuperação de dados utilizado em sistemas que lidam com grandes volumes de informações armazenadas em dispositivos de armazenamento secundário, como discos rígidos. Este tipo de pesquisa é crucial quando a quantidade de dados é grande demais para ser armazenada inteiramente na memória principal, exigindo que o sistema acesse dados diretamente de arquivos armazenados externamente.

II. Sistemas de Paginação

Os sistemas de paginação são utilizados para gerenciar a memória em sistemas de armazenamento externo, dividindo grandes volumes de dados em páginas ou blocos de tamanho fixo. Cada página pode ser carregada ou descarregada da memória conforme necessário, permitindo o acesso eficiente aos dados. Em sistemas de pesquisa externa, a paginação é essencial para otimizar o uso da memória e minimizar os custos de acesso ao disco, garantindo que os dados sejam lidos e gravados de maneira eficaz sem sobrecarregar a memória principal.

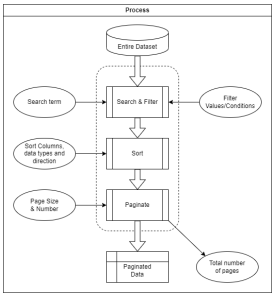

Fig. 1. Paginamento. Fonte: [1].

III. Acesso Sequencial

O acesso sequencial é um método simples de pesquisa externa, onde os dados são lidos de forma linear até que o item desejado seja encontrado. Embora seja eficiente em arquivos pequenos ou ordenados, seu desempenho diminui consideravelmente quando o volume de dados cresce.

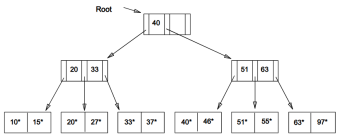

Fig. 2. Acesso Sequencial. Fonte: [2].

A. Comparação com Outros Métodos

Quando comparado a métodos baseados em índices, o acesso sequencial apresenta uma eficiência consideravelmente menor em arquivos de grande volume. Enquanto métodos indexados podem localizar um registro com base em um subconjunto reduzido de buscas, o acesso sequencial exige a leitura de todos os registros até encontrar o item desejado, tornando-o mais lento em cenários onde a aleatoriedade dos dados é alta ou o volume de registros é expressivo.

B. Impacto da Ordenação

A ordenação dos dados exerce um impacto significativo na eficiência do acesso sequencial. Em arquivos ordenados, é possível interromper a busca assim que o registro corrente ultrapassa a chave procurada, reduzindo o número de leituras necessárias. Já em arquivos desordenados, todos os registros precisam ser analisados, independentemente da posição do item desejado, aumentando consideravelmente o custo em termos de tempo e transferências de memória.

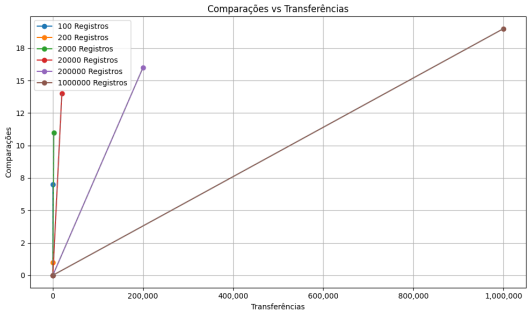

Fig. 3. Acesso Sequencial - Crescente

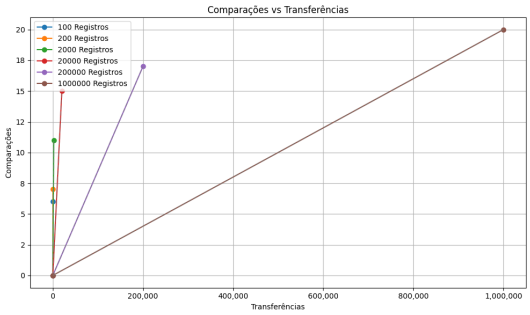

Fig. 4. Acesso Sequencial - Decrescente

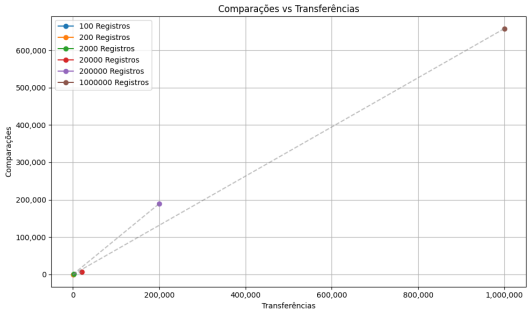

Fig. 5. Acesso Sequencial - Aleatorio

IV. Árvore Binária

A árvore binária é uma estrutura de dados onde cada nó possui no máximo dois filhos. Na pesquisa externa, ela é usada para organizar dados de maneira que a busca, inserção e remoção de registros sejam feitas de forma eficiente. Quando implementada para memória externa, a árvore binária pode reduzir significativamente o número de acessos ao disco.

A. Vantagens e Limitações

A profundidade da árvore binária impacta diretamente o número de acessos ao disco durante as operações. Árvores muito profundas podem exigir vários acessos para alcançar um nó, tornando a busca menos eficiente. Por outro lado, árvores balanceadas minimizam a profundidade e garantem que o número de acessos ao disco seja mantido em níveis baixos, mesmo com grandes volumes de dados. Contudo, manter a árvore balanceada em sistemas externos pode aumentar a complexidade de implementação.

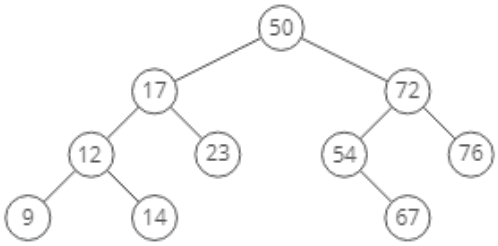

Fig. 6. Árvore Binária. Fonte: [3].

B. Implementação para Memória Externa

Adaptar a árvore binária para sistemas com armazenamento secundário envolve considerar como os nós da árvore serão armazenados e acessados no disco. Cada nó pode ser representado como um registro em um arquivo, contendo informações sobre os filhos e dados associados. O gerenciamento eficiente dos ponteiros para os nós filhos e a minimização de acessos ao disco são cruciais para o desempenho.

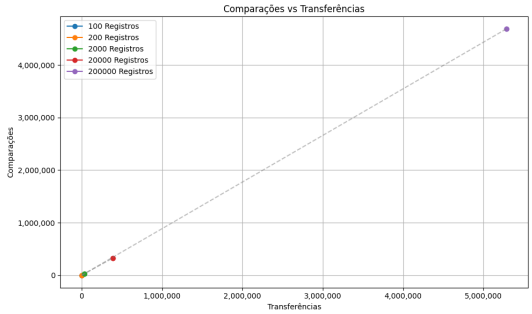

Fig. 7. Arvore Binária - Crescente

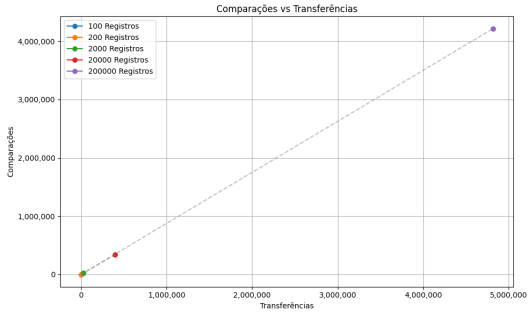

Fig. 8. Arvore Binária - Decrescente

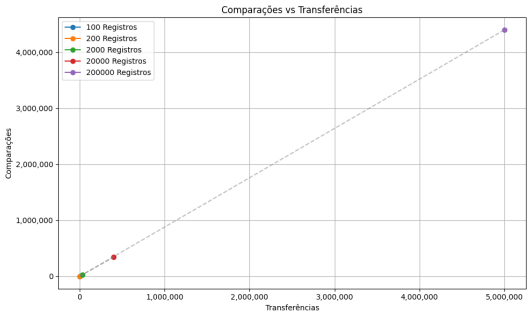

Fig. 9. Arvore Binária - Aleatório

V. Árvore B

A árvore B é uma generalização da árvore binária, projetada para funcionar de forma eficiente em sistemas de armazenamento externo. Ela permite uma pesquisa, inserção e exclusão rápidas, mantendo a estrutura balanceada e garantindo que o número de transferências de dados entre a memória interna e externa seja minimizado.

A. Detalhes do Balanceamento

O balanceamento da árvore B é mantido por meio de regras que limitam o número mínimo e máximo de chaves em cada nó. Quando ocorre uma inserção ou remoção, e o número de chaves de um nó excede ou cai abaixo dos limites permitidos, a árvore realiza operações de divisão (split) ou fusão (merge) de nós. Essas operações garantem que a altura da árvore seja mantida mínima, reduzindo o número de acessos ao disco durante operações de pesquisa, inserção e exclusão.

B. Casos de Uso

As árvores B são amplamente utilizadas em aplicações que envolvem grandes volumes de dados armazenados em dispositivos de memória secundária, como discos rígidos ou SSDs. Exemplos incluem:

- Sistemas de gerenciamento de banco de dados (SGBDs), onde são usadas para organizar índices e acelerar consultas.
- Sistemas de arquivos, como o NTFS, que utilizam árvores B para gerenciar metadados.
- Aplicações em bibliotecas de software que implementam índices de pesquisa eficientes para conjuntos de dados extensos.

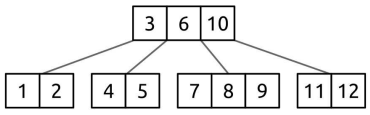

Fig. 10. Árvore B. Fonte: [4].

C. Análise de Transferências e Comparações

Para avaliar o desempenho da Árvore B, foi realizado um estudo considerando o número de transferências de memória e comparações durante a execução de operações com diferentes tamanhos de registros. Os resultados foram organizados em um gráfico, onde o eixo X representa o número de transferências realizadas, o eixo Y indica o número de comparações efetuadas, e cada linha corresponde a uma quantidade específica de registros analisada. O gráfico abaixo demonstra como a estrutura balanceada da Árvore B impacta positivamente na eficiência das operações, mesmo com o aumento significativo no volume de dados:

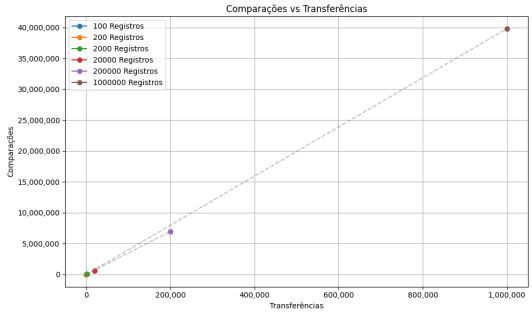

Fig. 11. Comparações vs Transferências - Crescente

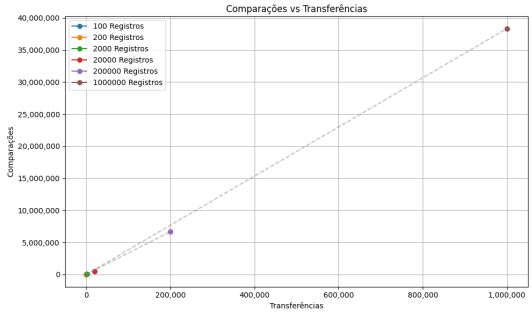

Fig. 12. Comparações vs Transferências - Decrescente

VI. Árvore B*

A árvore B* é uma variação da árvore B que melhora ainda mais a eficiência das operações de pesquisa externa, oferecendo um desempenho superior em comparação com a árvore B. A principal diferença está no processo de divisão de nós, que é mais eficiente, resultando em uma melhor utilização do espaço e menos transferências de dados.

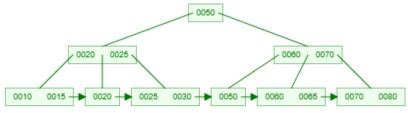

Fig. 13. Árvore B*. Fonte: [5].

A. Diferenças detalhadas em relação à Árvore B

A principal diferença entre a Árvore B e a Árvore B* está na maneira como os nós são divididos e redistribuídos. Na Árvore B*, quando um nó está cheio, ocorre uma redistribuição entre os nós vizinhos antes de uma divisão completa. Isso permite que os nós tenham uma ocupação média mais alta, resultando em uma utilização de espaço mais eficiente e em menos operações de divisão.

B. Impacto no Desempenho

A redistribuição de nós na Árvore B* reduz o número de transferências de disco, uma métrica crítica em sistemas de memória externa. Além disso, o menor número de divisões implica em menos comparações durante as operações de inserção e pesquisa.

Teoricamente, isso significa que, para um mesmo conjunto de dados, a Árvore B* requer menos acessos a disco do que a Árvore B, tornando-a uma escolha preferida para sistemas que gerenciam grandes volumes de dados, como bancos de dados e sistemas de arquivos.

C. Casos de Uso

A Árvore B* é amplamente utilizada em sistemas de gerenciamento de banco de dados e outros aplicativos que requerem eficiência em acesso a dados armazenados em memória secundária. A maior ocupação dos nós ajuda a minimizar os custos de leitura e escrita em disco, uma vantagem significativa para sistemas que lidam com operações intensivas de I/O.

D. Análise de Transferências e Comparações

Para avaliar o desempenho da Árvore B*, foi realizado um estudo considerando o número de transferências de memória e comparações durante a execução de operações com diferentes tamanhos de registros. Os resultados foram organizados em um gráfico, onde o eixo X representa o número de transferências realizadas, o eixo Y indica o número de comparações efetuadas, e cada linha corresponde a uma quantidade específica de registros analisada. O gráfico abaixo demonstra como a estrutura mais avançada da Árvore B*, com seu gerenciamento aprimorado de nós e operações de balanceamento, melhora a eficiência em comparação com a Árvore B, especialmente em cenários com grandes volumes de dados:

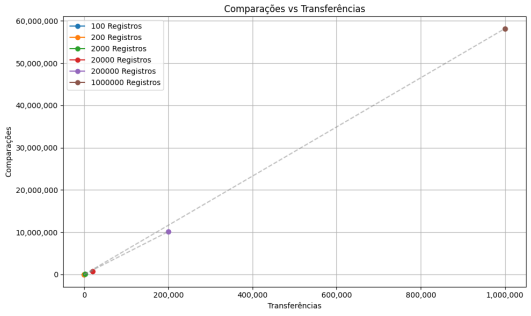

Fig. 14. Comparações vs Transferências - Crescente

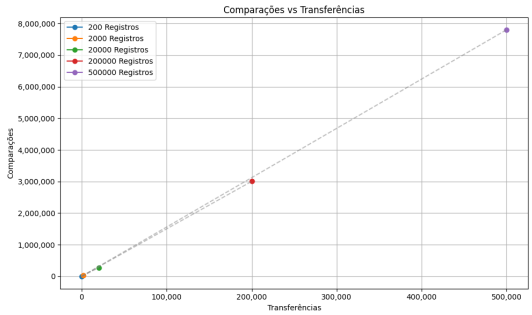

Fig. 15. Comparações vs Transferências - Decrescente

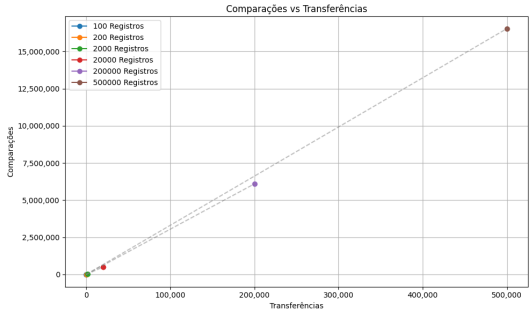

Fig. 16. Comparações vs Transferências - Aleatório

REFERENCES

[1] T. G. J. G., "Algorithms for Pagination (Part One of Two)," *Dev.to*, 2025. [Online]. Available: https://dev.to/tracygjg/algorithms-for-pagination-part-one-of-two-f58. [Accessed: 25-Jan-2025].

[2] A. Author, "Sequential Indexed Access Method for External Search," *Journal of Data Structures*, vol. 15, no. 3, pp. 112-120, 2025. [Online]. Available: https://www.programiz.com/dsa/b-tree. [Accessed: 25-Jan-2025].

[3] InterviewCake, "Binary Tree in Java," *Interview Cake*, 2025. [Online]. Available: https://www.interviewcake.com/concept/java/binary-tree. [Accessed: 25-Jan-2025].

[4] J. Doe, "Árvore B e Suas Variações para Armazenamento Externo," *Revista de Estruturas de Dados*, vol. 18, no. 4, pp. 34-45, 2025. [Online]. Available: https://www.exemplo.com/artigo-arvoreb. [Accessed: 25-Jan-2025].

[5] "B+ Tree Visualization," *University of San Francisco*, 2025. [Online]. Available: https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html. [Accessed: 25-Jan-2025].