**COMPARAÇÃO: HASH LINEAR E ÁRVORE B+**

Marco Aurélio Monteiro Lima¹,

Pâmela Evelyn Carvalho²

**RESUMO**

Esse documento apresenta uma breve apresentação das estruturas de dados Árvore B+ e *Hash* Linear. Ambas essas estruturas são muito utilizadas em bancos de dados, dada a sua boa performance na manipulação de registros. É feita uma breve comparação de desempenho entre as estruturas, utilizando implementações na linguagem de programação C++. Esse trabalho foi solicitado na disciplina de Banco de Dados II, ministrada no segundo semestre de 2017 pelo Prof. Dr. Marcos Ribeiro.

**Palavras-chaves:** Árvore B+. *Hash* linear.Banco de Dados.

**ABSTRACT**

*This document presents a brief presentation of the B Tree plus and Linear Hash data structures. Both of these structures are heavily used in databases, given their good record-keeping performance. A brief comparison of performance across structures is made using implementations in the C ++ programming language. This work was requested in the discipline of Database II, given in the second semester of 2017 by Prof. Dr. Marcos Ribeiro.*

***Keywords****: Tree B +. Linear Hash. Database.*

Graduando em Engenharia de Computação da IFMG. marco.monteirolima@gmail.com

² Graduando em Engenharia de Computação da IFMG. pamelaecarvalho@gmail.com

**1 INTRODUÇÃO**

**1.1 Árvore B+**

A árvore B+ é uma estrutura de dados semelhante a árvore B, entretanto, todos os seus registros completos se localizam nas folhas. Os nós intermediários e a raiz contêm apenas sinalizadores e são neles que se localizam as chaves de pesquisa e os ponteiros indicando em qual lado (direita ou esquerda) estará essa chave. Todos os elementos menores que a chave contida na raiz ficarão à direita, e os elementos maiores e iguais ficarão à esquerda. As folhas, que contém os registros, são ligadas através de uma lista encadeada. Isso facilita a busca por registros de forma sequencial. “[...]os nós de árvores B+ são páginas de disco e nós não folha contém apenas chaves de pesquisa, e ponteiro para nós, enquanto nós folhas podem conter os registros de dados reais. ” (GEHRKE, RAMAKRISHNAN, 2008, p. 302).

As operações de inserção e remoção devem sempre respeitar a ordem da árvore. A inserção é feita no nó-folha, se for o primeiro elemento a ser inserido, esse elemento será a raiz da árvore B+. Quando esse nó atinge o número máximo de elementos, o nó é dividido, e o nó com o maior valor sobe e vira raiz, porém o elemento que virou raiz continua na folha também. A remoção na árvore B+ assim como na inserção, as operações são feitas apenas nas folhas.

A árvore B+ é muito utilizada em banco de dados *SQLServer* e *Oracle* pois a operação de busca é muito eficiente log (n).

* 1. **Hash linear**

O *Hash* linear é um *hashing* do tipo dinâmico que ajusta inserções e exclusões. Nele não há requerimento de diretório, é capaz de manipular longas cadeias de *overflow* e é mais flexível em relação à divisão de *buckets*.

A nova ideia por trás do *Hashing* Linear é que um diretório pode ser evitado por uma escolha inteligente do *bucket* a ser dividido. (GEHRKE, RAMAKRISHNAN, 2008, p. 320).

O índice é usado para suportar consultas de correspondências exatas. Comparando com o índice da Árvore B+ que também suporta correspondências exatas, o *hash* linear tem um custo esperado O (1).

Diferentemente do *Hashing* Extensível, quando uma inserção dispara uma divisão, o *bucket* no qual a entrada de dados é inserida não é necessariamente o *bucket* que é dividido. Uma página de *overflow* é adicionada para armazenar a entrada de dados recém inserida, como no *Hashing* Estático.

(GEHRKE, RAMAKRISHNAN, 2008, p. 316)

A função de *hash* muda dinamicamente e em qualquer instante dado pode haver no máximo duas funções de *hashing* usadas pelo esquema. O *hashing* linear é bastante utilizando em sistemas de banco de dados comerciais. É usado em aplicativos onde a consulta de correspondência exata é a consulta mais importante, como à *Hash join*.

**2 DESENVOLVIMENTO**

Ambas as estruturas de dados Árvore B+ e *Hash* Linear foram implementadas na linguagem C++ na IDE do *CodeBlocks* utilizando a plataforma Linux com auxílio da biblioteca *boost* que é utilizada para manipulação de *strings*. Ambas estruturas utilizaram uma base de dados de um arquivo de texto. Para carregar os dados foi implementada uma função de leitura de arquivo. De modo geral, essa função lê cada linha do arquivo atribui a uma *string*. A função *split* da biblioteca *boost*, explode a *string* lida cada vez que encontra o delimitador informado no caso deste uma virgula, atribui a um vetor de *strings* cada separação feita. Foi implementada uma função para converter cada *string* do vetor em um inteiro para que fosse possível efetuar a inserção ou remoção na estrutura.

**2.1 HASH LINEAR**

Na implementação do *Hash* Linear, primeiramente, foi definida uma variável global que armazena o tamanho da tabela *hash*. Foi criada uma classe *hash*, que define os campos dos registros, esse possui dez inteiros. Foi implementada uma classe Tabela na qual essa terá as funções de operação com a tabela *hash,* tais como inserir, buscar, deletar e outras.

Foi implementada uma função que retorna o resto da divisão do índice pelo tamanho da tabela, função de comportamento do *hash* linear.

O método de inserção utiliza como parâmetro dez inteiros, sendo seu primeiro inteiro o índice do registro e seus demais informações do registro. Ele percorre a tabela e busca a posição aonde o registro será inserindo e então o insere.

O procedimento busca por igualdade recebe como parâmetro o índice do registro, e então percorre toda a tabela até encontrar o registro, se ele existir. Se o indice existe, ele exibe os dados do registro. Já a busca por intervalo recebe dois parâmetros um intervalo mínimo e o intervalo máximo, ele começa buscando o valor mais próximo ou igual ao índice mínimo, quando o acha começa a exibir os registros até que se alcance o limite superior informado.

O procedimento de remoção, recebe como parâmetro o índice que se deseja remover, é feita uma busca na tabela até que se encontre o índice, ou que seja informado que o índice não se encontra na tabela, se o índice estiver presente ele removido junto com seus dados de registro.

Na função que inicializa o programa é solicitado que o usuário informe o tamanho da tabela. Em seguida é exibido um menu de opções na qual é feito os testes com a estrutura.

**2.1 ÁRVORE B+**

De início foi definido um tipo de abstrato de dados, que armazena os dados de um nó ou seja: uma lista de índices, quantidade de índices do nó, se é um nó folha, um ponteiro para o próximo nó, uma lista encadeada para apontar para os remanescentes de um nó e um tipo abstrato de dados que a estrutura armazena. Um outro tipo abstrato de dados foi definido, este especifica o tipo de informação que a estrutura irá armazenar neste caso, dez valores inteiros.

Uma função criaçãofoi implementada para alocar um novo nó em memória sempre que fosse preciso e uma função para limpar os registros da memória também foi implementada, para evitar lixos de memória.

O método de inserção da árvore faz primeiramente uma busca na árvore para localizar aonde deve ser inserido o índice. Se o nó a ser inserido tiver espaços vagos ou seja, não estiver cheio o índice é inserido. Porém se este nó possuir todos os seus espaços cheios é feita uma divisão do nó, criando dois novos nós de forma que cada um tenha o número mínimo de chaves.

Um procedimento de busca foi feito de forma recursiva, para que se possa recuperar os dados inseridos da estrutura sempre que preciso. O método de remoção chama o procedimento de busca para verificar se o índice que se deseja retirar existe, se sim, é feita uma análise do nó, aonde o índice se encontra. Esse nó precisa obedecer a ordem da árvore, ou seja só pode haver uma exclusão nesse nó, se, após a exclusão ele possuir o número mínimo de chaves. Se não, é feito um deslocamento de índices, ou seja, se uns dos “vizinhos” do nó, houver índices acima do número mínimo, é transferido esse índice extra para o nó que será feita a exclusão. Se ambos os vizinhos não tiverem índices extras é feita uma fusão dos nós onde um dos seus vizinhos se une a ele formando um novo nó. Depois dos procedimentos necessários é feita a exclusão.

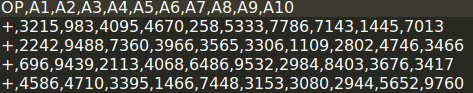
Uma observação importante que se deve lembrar, é que na Árvore B+ os registros estão apenas na folha, logo operações de remoção e inserção ocorrem apenas na folha.

Por fim na função de inicialização do programa, é solicitado ao usuário que informe a ordem da árvore, em seguida é exibido um menu com opções, com as operações que podem ser executadas na árvore.

**3 CONCLUSÕES**

A seguir uma comparação entre o desempenho *Hash* linear e Árvore B+. O teste foi feito utilizando uma base de dados disponibilizada pelo professor que segue essa estrutura:

Figura 1 – Arquivo das comparações



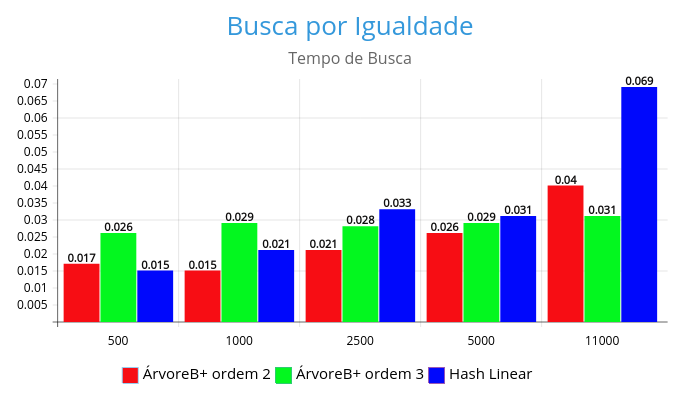
Na primeira linha se encontra o cabeçalho do arquivo e as linhas abaixo são os registros. As informações são separadas por virgula. Os resultados abaixo estão sujeitos a variação de acordo com o hardware do computador utilizado, compilador, etc. Abaixo serão apresentados alguns resultados obtidos pelo teste de caixa preta do compilador da IDE *CodeBlocks*.

**3.1 BUSCA**

Tabela 1 - Busca por igualdade

|  |  |  |  |
| --- | --- | --- | --- |
| **Busca Por igualdade** | | | |
| **Quantidade** | **Arvore B+ ordem 2** | **Arvore B+ ordem 3** | **Hash linear** |
| 500 | 0.017 | 0.01 | 0.015 |
| 1000 | 0.015 | 0.029 | 0.021 |
| 2500 | 0.021 | 0.028 | 0.033 |
| 5000 | 0.026 | 0.029 | 0.031 |
| 11000 | 0.040 | 0.031 | 0.069 |

Gráfico 1- Busca por igualdade. Registros x tempo



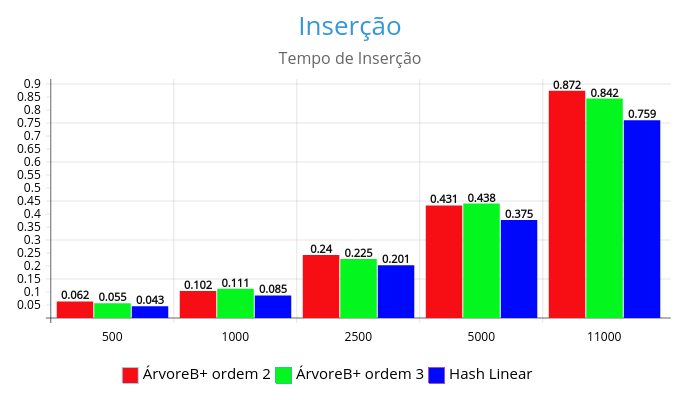
Quando se trata de poucos registros o *Hash* linear e a Árvore B+ possui desempenho semelhantes, porém quando a quantidade de dados aumenta a árvore possui um desempenho melhor que o *hash*, devido ao fato que a árvore possui sinalizadores que facilitam a navegação na mesma indicando o menor caminho.

**3.1 INSERÇÃO**

Tabela 2 - Inserção

|  |  |  |  |
| --- | --- | --- | --- |
| **Inserção** | | | |
| **Quantidade** | **Arvore B+ ordem 2** | **Arvore B+ ordem 3** | **Hash linear** |
| 500 | 0.062 | 0.055 | 0.043 |
| 1000 | 0.102 | 0.111 | 0.085 |
| 2500 | 0.240 | 0.225 | 0.201 |
| 5000 | 0.431 | 0.438 | 0.375 |
| 11000 | 0.872 | 0.842 | 0.759 |

Gráfico 2 - Inserção. Registros x tempo



De acordo com os dados coletados, o *Hash* linear teve melhor desempenho por bem pouco na inserção, porém, o desempenho da árvore tende a melhorar à medida que sua ordem aumenta, pois, os nós tendem a dividir menos vezes à medida que enchem. É preciso considerar também que a inserção utiliza a busca por igualdade, que tem melhor desempenho na árvore.

**4 REFERÊNCIAS**

RAMAKRISHNAN, R; GEHRKE J., Sistemas de banco de dados: 3 ed. São Paulo: McGraw-Hill, 2008.