IMPLEMENTAÇÃO DE ÍNDICE (ÁRVORE B+ E HASH)

Carlos Eduardo de Sousa, Denilson Aparecido de Morais

Instituto Federal de Minas Gerais (IFMG) – Campus Bambuí

carloscadu1811999@gmail.com, denilsonapmorais@gmail.com

RESUMO

Este artigo acompanha a entrega dos arquivos para o Trabalho 1 da disciplina Bancos de Dados 2, do curso Engenharia de Computação no IFMG - Campus Bambuí e apresenta os resultados, observações e comparações feitas sobre o tempo de execução de algoritmos simulando as estruturas de índice em árvore B+ e *hash*, apresentados conceitualmente durante o período letivo.

Palavras-chave: Árvore B+. *Hash*. Banco de Dados. Índices

1 INTRODUÇÃO

Grande parte das consultas a um banco de dados se referem a somente uma pequena parcela dos registros armazenados nas tabelas. É altamente ineficiente para o SGBD ter de ler cada registro na tabela para verificar algum dado. Assim sendo, o sistema deveria conseguir localizar esses registros diretamente. Para isso, existem estruturas especiais nas tabelas, chamadas de índices. Os índices são uma das ferramentas de otimização mais conhecidas e utilizadas pelos desenvolvedores de bancos de dados. O emprego de indexação em tabelas pode aumentar significativamente a performance em consultas ao banco de dados.

Existem vários tipos de índices que podem ser criados em tabelas, de acordo com técnicas variadas. Neste trabalho trataremos especificamente dos índices em árvore B+ e Índice de *Hashing*.

A árvore B+ é uma extensão da árvore B que permite operações eficientes de inserção, exclusão e pesquisa. A árvore B+ é mais usada por SGBDs comerciais por oferecer consultas por intervalos, é usada para armazenar grande quantidade de dados que não podem ser armazenados na memória principal, devido ao fato de que a mesma é sempre limitada. Os nós internos (chaves para acessar registros) da árvore B+ são armazenados na memória principal enquanto os nós folha são armazenados na memória secundária. Os nós internos da árvore B+ são frequentemente chamados de nós de índice.

Uma alternativa à Árvore B+, que foi desenvolvida por Fagin, Nievergelt, Pippenger e Strong em 1978-9, que estende algoritmos de busca e os aplica em busca externa é chamada de *Hash Extensível*. Esta estrutura é uma implementação de busca que normalmente requer apenas um ou dois acessos a disco, inclusive para a inserção (na maioria dos casos). O *Hash Extensível* combina características de *hashing*, *multiway-tries* (Frakes, W. et al., 1992; Sedgewick et al., 1998) e métodos de acesso sequencial.

A ideia básica do *Hash Extensível* é usar uma função de *hashing*, que mapeia valores em um campo de pesquisa com um determinado número de *buckets* para encontrar a página na qual uma entrada de dados desejada se encontra. Usamos um esquema simples chamado *Hashing* Estático para introduzir a ideia. Este esquema, como o ISAM, sofre do problema de longas cadeias de *overflow*, o que pode afetar o desempenho. Uma solução para este problema, é o esquema de *Hashing* Extensível que usa um diretório para suportar inserções e exclusões.

Visto isto, o objetivo do presente trabalho foi implementar duas estruturas de indexação: a primeira baseada em *hash* e segunda baseada na árvore B+, e comparar o tempo de execução de suas operações básicas (inserção, remoção, busca por igualdade e busca por intervalo (exceto o *hash*)).

2 METODOLOGIA OU MATERIAL E MÉTODO

Para realizar a implementação das estruturas de indexação, utilizou-se a linguagem de programação *python* pelo fato da mesma possuir uma sintaxe bastante simples, e por ser muito popular possibilita a busca de diversos materiais de aprendizagem na internet. Como ambiente de desenvolvimento, utilizou-se o *Visual Studio Code* para a implementação dos algoritmos. As bases de dados sintéticos utilizadas nos experimentos foram produzidas por meio do gerador fornecido pelo professor da disciplina, Marcos R. Ribeiro, disponível em https://ribeiromarcos.github.io/siogen/.

A produção do código para a árvore B + tomou tempo consideravelmente maior do que a da produção do código de *hash* extensível, dada a grande diferença de sofisticação entre os dois. A proposta inicial do trabalho era a implementação das operações de inserção, deleção, busca por igualdade e busca por intervalo, porém devido à complexidade e o curto período de tempo, não conseguimos efetuar com maestria a implementação da deleção e nem das busca por intervalo na árvore B+.

A estrutura da Árvore B + consiste em:

- Uma estrutura para ler os dados do registro
- Classe nó básica - Cada nó carrega um par chave-valor, um status que indica se o nó é folha ou não e a sua ordem, que é o número máximo de valores que cada nó pode armazenar.
- Classe Arvore B+ - contem nós que serão divididos uma vez que estejam cheios. Nas divisões a chave é inserida no nó-pai, agindo como referência.

1 - Busca na árvore B+

Todos os nós internos na árvore representam o intervalo de nós folha. Começamos da raiz e comparamos com os valores e a chave do nó interno que queremos encontrar e seguimos o ponteiro filho em cada nó e chegamos ao nó folha.

2 – Inserção árvore B+

Na inserção, primeiramente procura-se o nó, depois se insere a nova chave. Deve-se encontrar a folha a qual a nova chave pertence. Se houver espaço na folha, a nova chave é inserida. Senão, a

folha deve ser dividida *splitting* em duas e suas chaves distribuídas entre elas. A divisão de um nó em um determinado nível da árvore implica em uma nova entrada (chave) no nível superior àquele nó.

Já a estrutura do hash extensível é bem mais simples que a estrutura da Árvore B+. Ela consiste em:

- Uma estrutura para ler os dados do registro.
- Uma lista de diretórios com referências para os *buckets*.
- *Buckets*, que são os repositórios dos dados.
- Uma classe *hash extensível* onde estão todas as operações básicas.

1.1 – Busca *hash extensível*

Através da função de *hashing* utilizada pela implementação, a chave do item é transformada em um código *hash*, que é um inteiro. A partir do seu resultados é localizado o *bucket* onde estão todos os itens com aquele mesmo código *hash*, é realizada uma busca sequencial para localizar o item desejado.

2.1- Inserção *hash extensível*

Aqui primeiro é obtido onde a chave será armazenada, caso caiba e a chave não exista no hash, ela é adicionada, senão verificamos se as profundidades são iguais, se sim, iremos duplicar o diretório.

3 - Nos laços de repetição *FOR*

- Atualiza as profundidades tanto do *bucket*, quanto do diretório;
- Particiona o *bucket*, realoca os endereços do *bucket* no diretório;
- Redistribui os valores e tenta inserir a chave.

Se as profundidades não são iguais, iremos atualizar as profundidades do *bucket*, particionar o *bucket*, realocar os endereços do *bucket* no diretório e reinserir os valores e tentar inserir a chave.

4 – Deleção *hash extensível*

Para remover passamos para o método a chave que desejamos e aplicamos a função *hash* para descobrir em qual *bucket* está, após isso removemos o item da posição desejada.

3 RESULTADOS E DISCUSSÃO

Os experimentos foram realizados em dois notebooks diferentes e com as seguintes especificações:

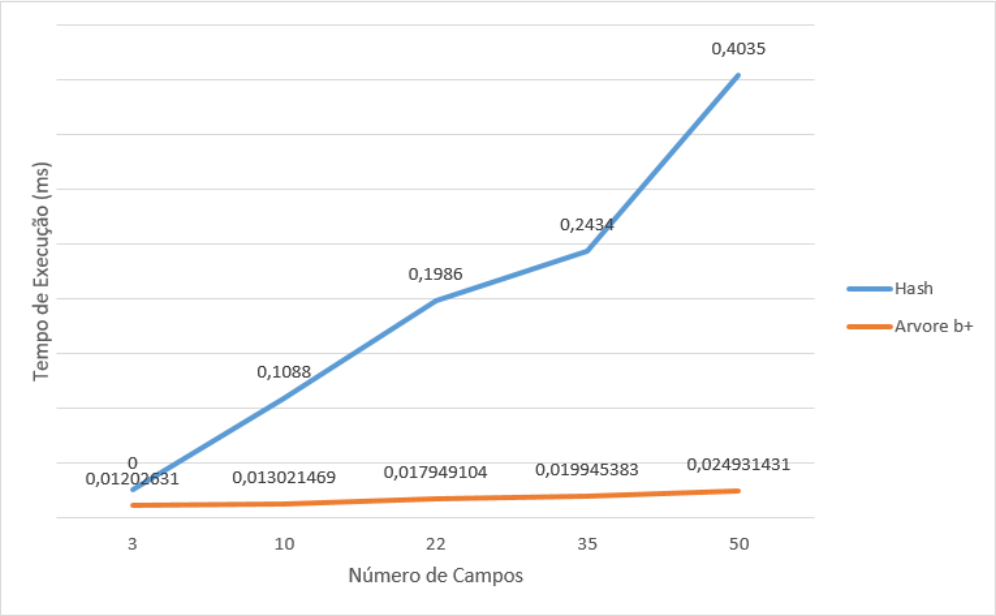
- Processador - Intel(R) Core(TM) i7-7500U CPU @2.70 GHz
  Memória RAM - 12,00 GB
  Windows 10
- Processador – Core i3 7020u
  Memória RAM – 4,0 GB
  Ssd240

Windows 10

O algoritmo da Árvore B+ foi executado do primeiro notebook citado, já o algoritmo *hash* no segundo.

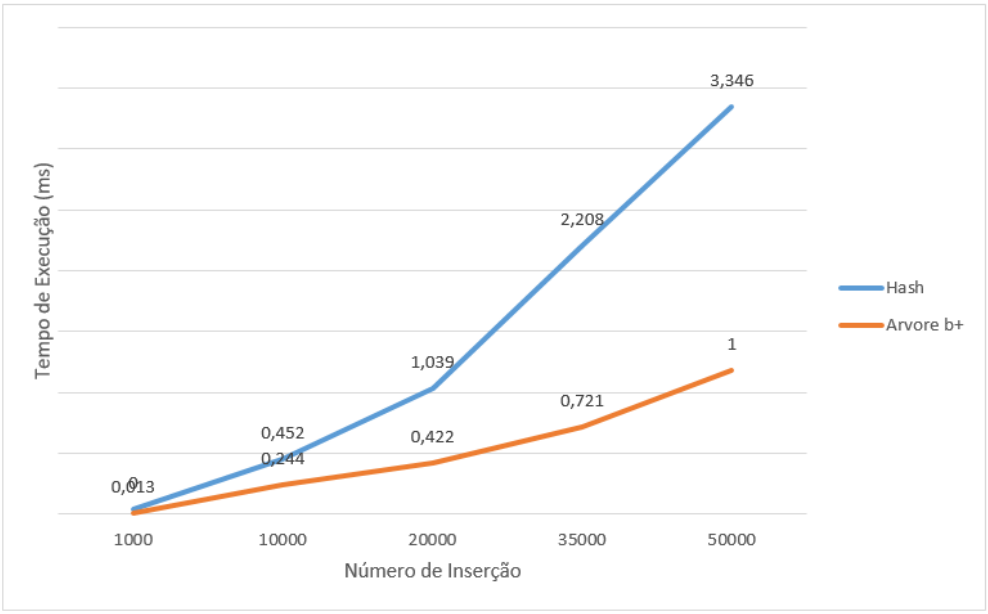
Os resultados do experimento foram gráficos dos tempos de execução coletados. Logo abaixo, são mostrados os gráficos comparativos dos experimentos com o número de campos, número de inserções e número de deleções:

Figura 1 - Experimento sobre número de campos



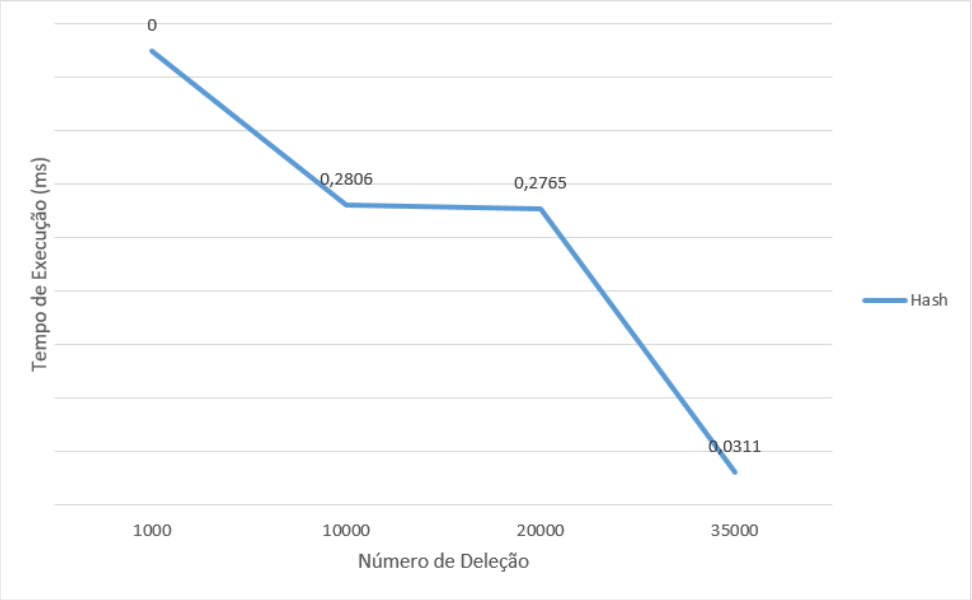
Fonte: Autores, 2022

Figura 2 - Experimento sobre número de inserções



Fonte: Autores, 2022

Figura 3 - Experimento sobre número de deleções



Fonte: Autores, 2022

Como podemos observar, à medida que aumentamos o número de inserções aumenta também o tempo de execução de ambos os algoritmos. Já na deleção do *hash* é possível ver que o tempo diminui à medida que aumentamos as deleções

Dos experimentos propostos, não foi bem sucedida a execução do processo de deleção na estrutura de árvore B + e nem o experimento sobre o tamanho da página, portanto, deve-se considerar este fator implícito sobre os gráficos gerados. Apesar disso a discrepância foi de uma ordem de magnitude que permite uma comparação clara entre o desempenho dos dois algoritmos.

4 CONCLUSÃO

Os experimentos desenvolvidos neste trabalho deixam claro a superioridade do índice de árvore B+ sobre o índice *hash*. Pode-se inferir, com esses resultados, o impacto positivo da sofisticação dos algoritmos e estruturas de dados nas operações e desempenho dos SGBDs, assim como a relevância dessa sofisticação levando em conta a tendência crescente de produção, transferência e armazenamento de dados no mundo contemporâneos.

Em suma, após todas as implementações, testes e análises conseguimos ver o quão importante são as estruturas de índice baseadas em *hash* extensível e da árvore B+, já que estas estruturas estão no nosso dia a dia, seja em um programa de computador ou em um aplicativo para smartphone. Por fim, conseguimos perceber o porquê das buscas por igualdade serem sempre preferíveis a utilizar estruturas baseadas em *hash* (por serem mais rápidas que árvore B+) e buscas por intervalo com árvores B+ (por suportarem e também serem bem rápidas).

REFERÊNCIAS BIBLIOGRÁFICAS

DOS REIS, Fabio. **O que são Índices em Bancos de Dados – Indexação em Tabelas**. [S. l.], 15 out. 2019. Disponível em: http://www.bosontreinamentos.com.br/bancos-de-dados/o-que-sao-indices-em-bancos-de-dados-indexacao-em-tabelas/. Acesso em: 12 mar. 2022.

NETO, M. C. T. **Estrutura de dados para Indexação de Grandes Volumes de Dados**. 2001.

Frakes,W. B. e Baeza-Yates,R. **Information Retrieval: Data Structure & Algorithms**. Prentice Hall, 1992.

Sedgewick, R., Wyk, J. van, **Algorithms in C++, Part 1-4**. p. 560-564; 646-662. Third Edition. Addison-Wesley, 1998.