Implementação de Tabela Hash

Guilherme Ferraz

November 8, 2024

1 Introdução

Neste projeto foi implementada uma tabela hash que permite analisar o desempenho de diferentes configurações de tabela hash. O código é composto por 4 classes principais: Main, TabelaHash, No e Registro.

2 Classe Main

A classe Main é responsável pelo processamento e testes da tabela hash. Ela inicia com a definição de algumas constantes importantes:

- Tamanhos das tabelas: 10.000, 100.000 e 1.000.000
- Quantidades de registros: 1.000.000, 3.000.000, 5.000.000
- Número de funções hash: 3
- Número de buscas para cálculo de média: 5
- Seed fixa (42) para garantir a reprodutibilidade dos dados

```
private static final int[] TAMANHOS = {10000, 100000, 1000000};
private static final int[] QUANTIDADES = {1000000, 5000000, 20000000};
private static final int NUM_FUNCOES_HASH = 3;
private static final int NUM_BUSCAS = 5;
private static final long SEED = 42;
```

A execução principal do código ocorre nesta classe, onde são realizados testes com diferentes combinações de tamanho de tabela e quantidade de registros. Para cada combinação, as três funções hash implementadas são testadas. O código realiza inserções e buscas, medindo tempos de execução, contando colisões e comparações.

Os resultados são apresentados no console, mostrando estatísticas para cada configuração testada, incluindo tempo de inserção, número de colisões, tempo médio de busca e número médio de comparações.

A geração de dados é feita através do método gerarDados, que cria conjuntos de dados aleatórios usando a classe Random com a seed fixa, garantindo reprodutibilidade.

```
private static Registro[] gerarDados(int quantidade, Random random) {
    Registro[] dados = new Registro[quantidade];
    for (int i = 0; i < quantidade; i++) {
        dados[i] = new Registro(100000000 + random.nextInt(900000000));
    }
    return dados;
}
```

3 Classe TabelaHash

Esta classe implementa a estrutura da tabela hash e suas operações. Três funções hash diferentes são definidas:

- Hash 1 (Método do resto da divisão): Calcula o resto da divisão da chave pelo tamanho da tabela.
- Hash 2 (Método da multiplicação): Multiplica a chave por uma constante A (0.6180339887), pega a parte fracionária do resultado e multiplica pelo tamanho da tabela.
- Hash 3 (Método do dobramento): Realiza operações de XOR e deslocamento de bits na chave para gerar o hash.

E após teremos um switch que irá retornar cada tipo de hash.

O método inserir na TabelaHash irá inserir novos registros na tabela, calculando o índice usando a função hash selecionada. Caso a posição estiver vazia, vai ser inserido o novo nó nela.

```
public void inserir(Registro registro) {
    int indice = calcularHash(registro.codigo);
    No novoNo = new No(registro);
    if (tabela[indice] == null) {
        tabela[indice] = novoNo;
    } else {
        No atual = tabela[indice];
        while (atual.proximo != null) {
            colisoes++;
            atual = atual.proximo;
        }
        atual.proximo = novoNo;
        colisoes++;
    }
}
```

Logo após teremos o método buscar, ele calcula o índice ao percorrer a lista encadeada, se naquela posição (se existir) e conta o número de comparações realizadas até encontrar o elemento ou chegar ao final da lista.

```
public int buscar(int codigo) {
    int indice = calcularHash(codigo);
    No atual = tabela[indice];
    int comparacoes = 0;
    while (atual != null) {
        comparacoes++;
        if (atual.registro.codigo == codigo) {
            return comparacoes;
        }
        atual = atual.proximo;
    }
    return comparacoes;
}
```

Teremos também ainda dentro da TabelaHash a contagem de colisões, que é realizada durante a inserção, incrementando um contador cada vez que um novo nó é adicionado a uma lista encadeada existente.

```
public long getColisoes() {
    return colisoes;
}
```

4 Classe No

A classe No representa um nó na lista encadeada utilizada para tratar colisões na tabela hash.

5 Classe Registro

A classe Registro é uma estrutura simples que representa um elemento a ser inserido na tabela hash.

6 Gráfico

Segue abaixo os gráficos com os resultados de cada teste

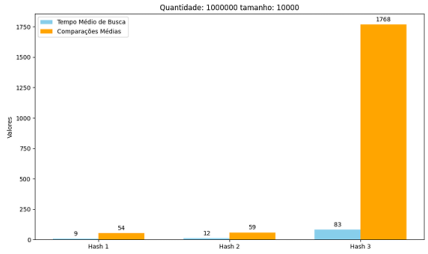

Figure 1: Gráfico 1: Resultados do primeiro teste

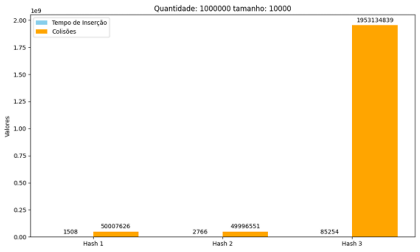

Figure 2: Gráfico 2: Resultados do segundo teste

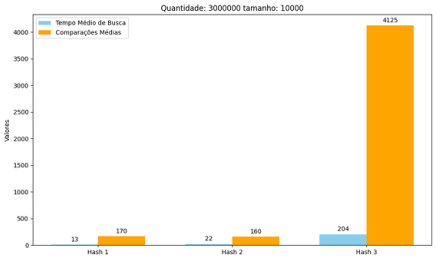

Figure 3: Gráfico 3: Resultados do terceiro teste

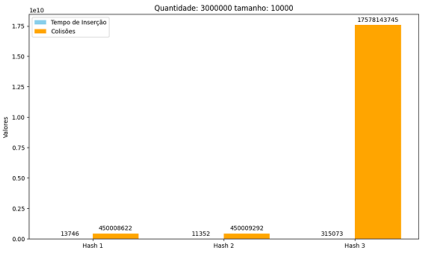

Figure 4: Gráfico 4: Resultados do quarto teste

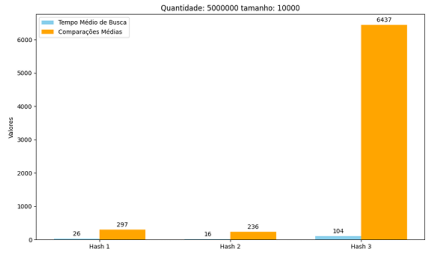

Figure 5: Gráfico 5: Resultados do quinto teste

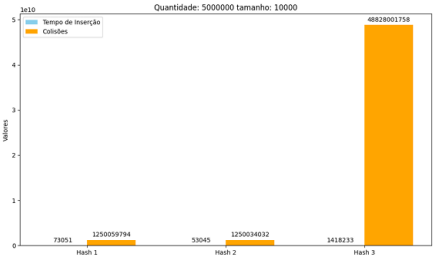

Figure 6: Gráfico 6: Resultados do sexto teste

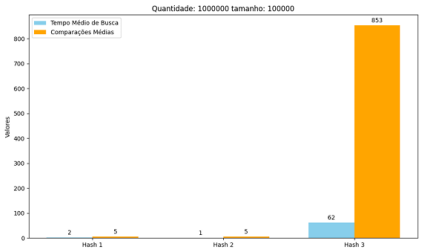

Figure 7: Gráfico 7: Resultados do sétimo teste

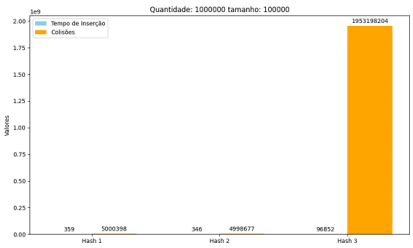

Figure 8: Gráfico 8: Resultados do oitavo teste

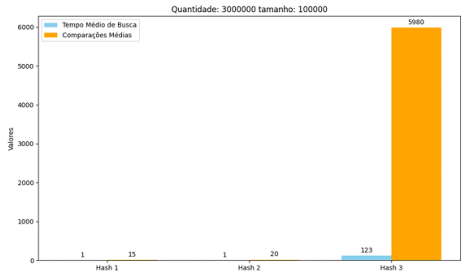

Figure 9: Gráfico 9: Resultados do nono teste

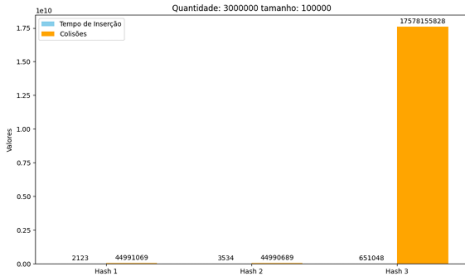

Figure 10: Gráfico 10: Resultados do décimo teste

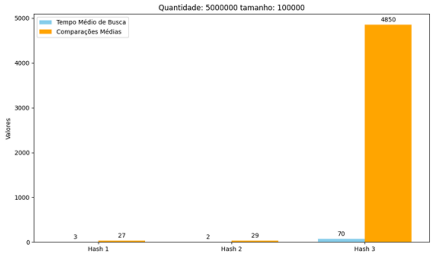

Figure 11: Gráfico 11: Resultados do décimo primeiro teste

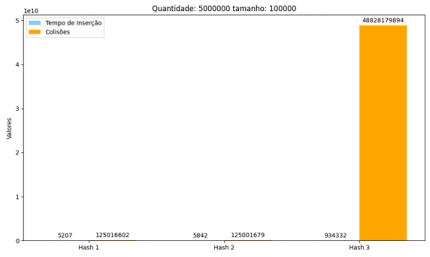

Figure 12: Gráfico 12: Resultados do décimo segundo teste

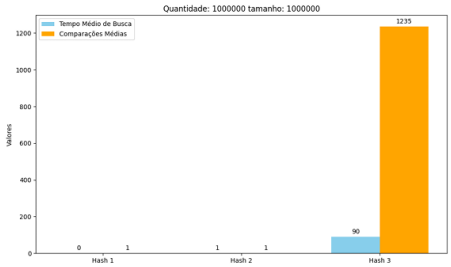

Figure 13: Gráfico 13: Resultados do décimo terceiro teste

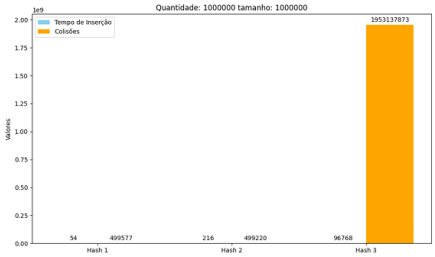

Figure 14: Gráfico 14: Resultados do décimo quarto teste

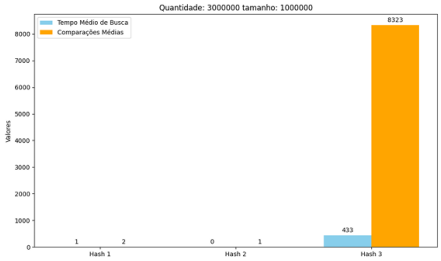

Figure 15: Gráfico 15: Resultados do décimo quinto teste

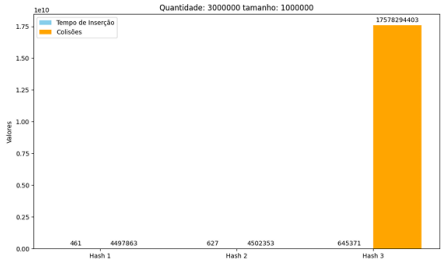

Figure 16: Gráfico 16: Resultados do décimo sexto teste

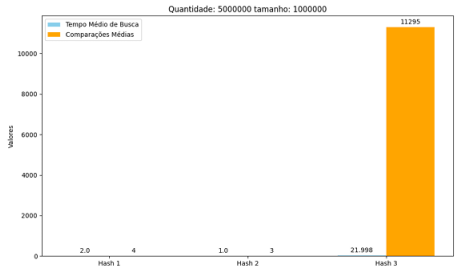

Figure 17: Gráfico 17: Resultados do décimo sétimo teste

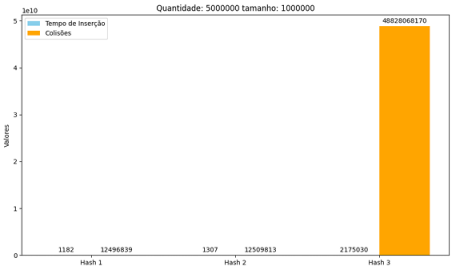

Figure 18: Gráfico 18: Resultados do décimo oitavo teste