PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ
ESCOLA POLITÉCNICA

RESOLUÇÃO DE PROBLEMAS ESTRUTURADOS NA COMPUTAÇÃO
PROFESSOR: ANDREY CABRAL MEIRA

ALUNO: RODRIGO AUGUSTO FIGUEIRA MOREIRA

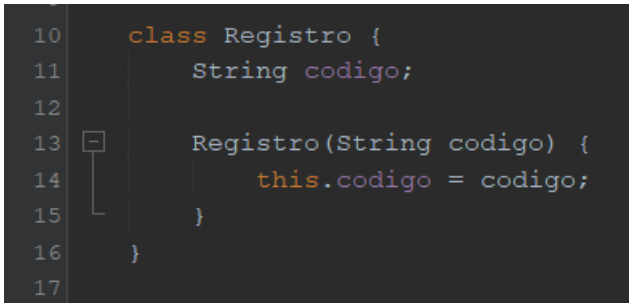
ATIVIDADE AVALIAÇÃO / RA3
RELATÓRIO DO PROJETO TABELA HASH

CURITIBA
2024

INTRODUÇÃO

Foram escolhidas as funções hash de resto de divisão a qual calcula o hash de uma chave numérica ao dividir seu valor por um número primo, utilizando o resto como índice, soma de dígitos, esta, soma todos os dígitos de uma chave numérica para obter o índice hash e multiplicação por caractere este ao calcular o hash multiplicando cada caractere da chave por uma constante, acumulando o valor para gerar um índice único. Para gerar os gráficos foi utilizada a extensão XChart, a seed sendo a 19.

EXPLICAÇÃO DO FUNCIONAMENTO DO CÓDIGO E JUSTIFICATIVA



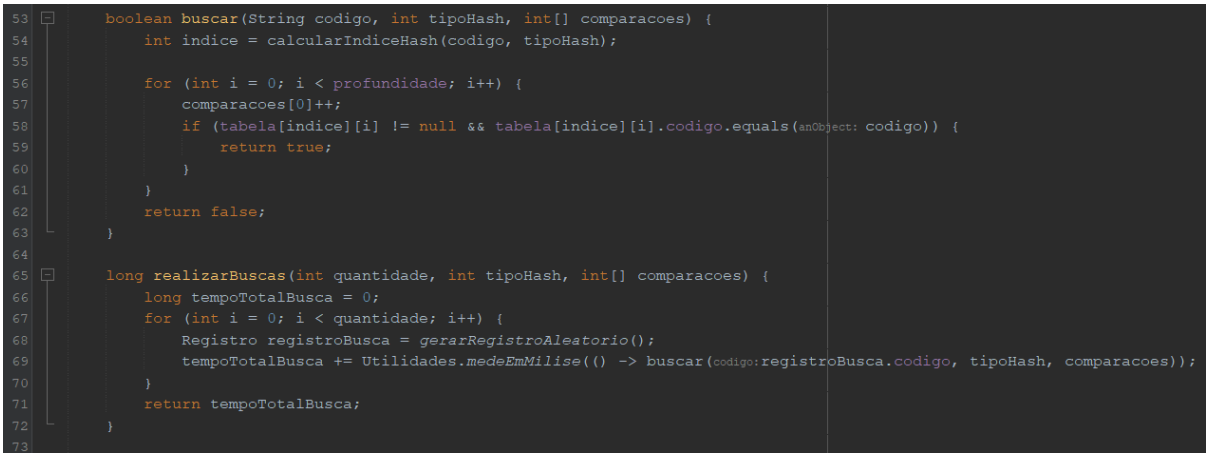
Classe Registro: Define um item na tabela com um código, ele é utilizado como chave para operações de hashing.

```java
class TabelaHashEncadeamento {

    Registro[][] tabela;
    int tamanho;
    int profundidade;

    TabelaHashEncadeamento(int tamanho, int profundidade) {
        this.tamanho = tamanho;
        this.profundidade = profundidade;
        tabela = new Registro[tamanho][profundidade];
    }

    void inserir(Registro registro, int tipoHash, int[] colisoes) {
        int indice = calcularIndiceHash(codigo:registro.codigo, tipoHash);

        boolean inserido = false;
        for (int i = 0; i < profundidade; i++) {
            if (tabela[indice][i] == null) {
                tabela[indice][i] = registro;
                inserido = true;
                break;
            }
        }
        if (!inserido) {
            colisoes[0]++;
        }
    }

    void realizarInsercoes(int quantidade, int tipoHash, int[] colisoes) {
        for (int i = 0; i < quantidade; i++) {
            Registro registro = gerarRegistroAleatorio();
            inserir(registro, tipoHash, colisoes);
        }
    }
```

Classe TabelaHashEncadeamento: Implementa uma tabela hash usando o tipo de encadeamento para colisões. Ela funciona por uma matriz de Registro chamada tabela, onde cada linha representa um índice calculado por hash e cada coluna representa um nível de profundidade no caso de colisões. Justificativa: Escolhi esta abordagem de matriz devido a melhor organização de dados assim como tornar o funcionamento do código mais eficiente.

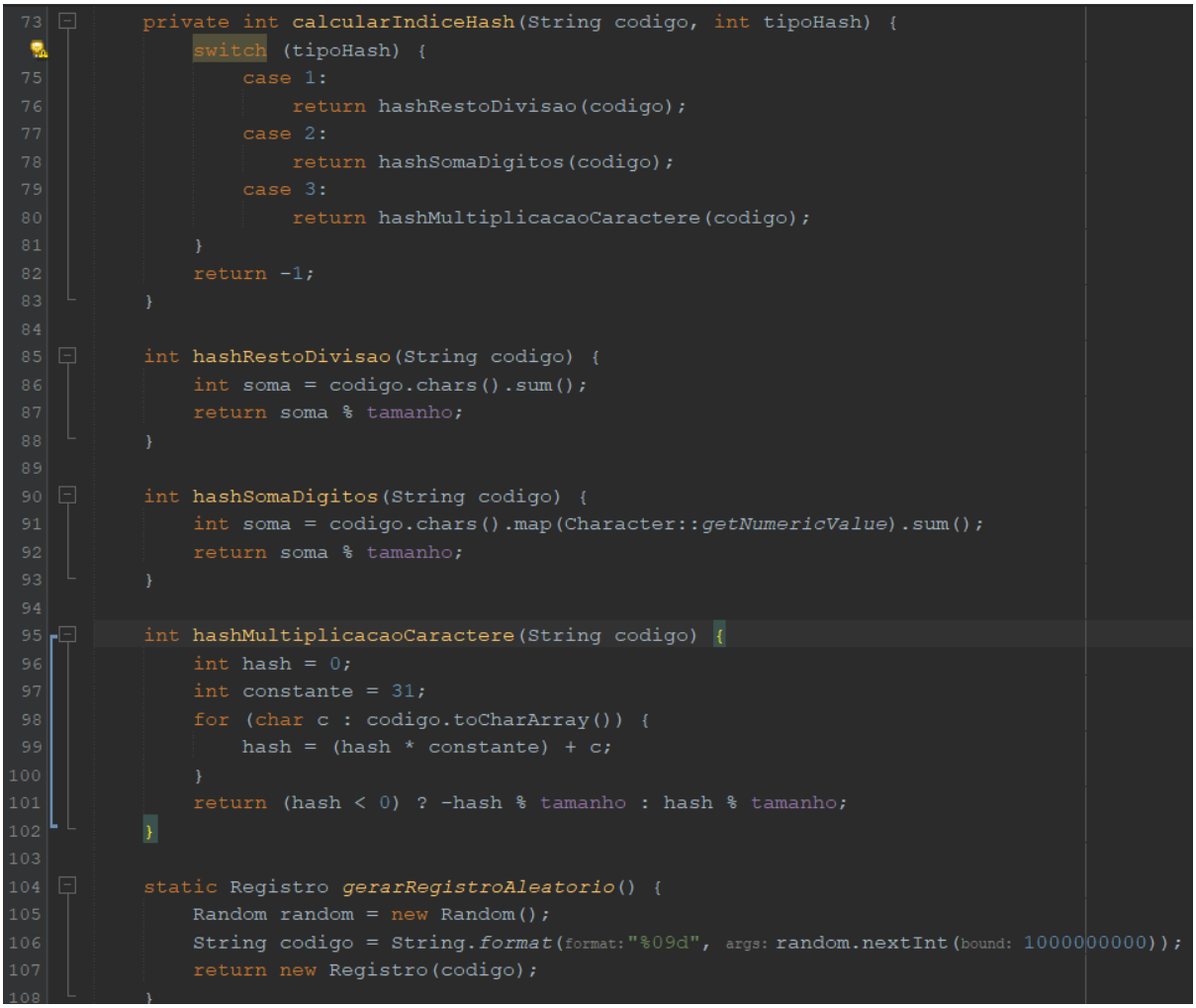
Construtor TabelaHashEncadeamento: Inicializa a tabela hash com um tamanho específico e profundidade de encadeamento.

- **Método inserir:** Método para inserir um registro na tabela, utilizando um dos três tipos de hash.
- **Método realizarIncersoes:** Gera múltiplos registros aleatórios e usa inserir para armazená-los na tabela, ele também conta colisões.



```
53      boolean buscar(String codigo, int tipoHash, int[] comparacoes) {
54          int indice = calcularIndiceHash(codigo, tipoHash);
55
56          for (int i = 0; i < profundidade; i++) {
57              comparacoes[0]++;
58              if (tabela[indice][i] != null && tabela[indice][i].codigo.equals(anObject: codigo)) {
59                  return true;
60              }
61          }
62          return false;
63      }
64
65      long realizarBuscas(int quantidade, int tipoHash, int[] comparacoes) {
66          long tempoTotalBusca = 0;
67          for (int i = 0; i < quantidade; i++) {
68              Registro registroBusca = gerarRegistroAleatorio();
69              tempoTotalBusca += Utilidades.medeEmMilise(() -> buscar(codigo:registroBusca.codigo, tipoHash, comparacoes));
70          }
71          return tempoTotalBusca;
72      }
73
```

- **Método buscar**: Ele calcula o índice usando a função hash especificada, e o contador de comparações é incrementado a cada comparação de registro.
- **Método realizarBuscas**: Mede o tempo total de várias operações de busca na tabela, criando registros aleatórios para simular as buscas e utilizando o método medeEmMilise da classe Utilidades para medir o tempo em miliesegundos de cada execução da função buscar; ele acumula esses tempos de execução e retorna o tempo total necessário para o conjunto de buscas.



```
73      private int calcularIndiceHash(String codigo, int tipoHash) {
74          switch (tipoHash) {
75              case 1:
76                  return hashRestoDivisao(codigo);
77              case 2:
78                  return hashSomaDigitos(codigo);
79              case 3:
80                  return hashMultiplicacaoCaractere(codigo);
81          }
82          return -1;
83      }
84
85      int hashRestoDivisao(String codigo) {
86          int soma = codigo.chars().sum();
87          return soma % tamanho;
88      }
89
90      int hashSomaDigitos(String codigo) {
91          int soma = codigo.chars().map(Character::getNumericValue).sum();
92          return soma % tamanho;
93      }
94
95      int hashMultiplicacaoCaractere(String codigo) {
96          int hash = 0;
97          int constante = 31;
98          for (char c : codigo.toCharArray()) {
99              hash = (hash * constante) + c;
100         }
101         return (hash < 0) ? -hash % tamanho : hash % tamanho;
102     }
103
104     static Registro gerarRegistroAleatorio() {
105         Random random = new Random();
106         String codigo = String.format(format:"%09d", args: random.nextInt(bound: 1000000000));
107         return new Registro(codigo);
108     }
```

- **Funções de Hashing**:
  - hashRestoDivisao: Calcula o índice somando o valor dos caracteres e usando o resto da divisão pelo tamanho da tabela.
  - hashSomaDigitos: Gera o índice somando os valores numéricos de cada caractere e tirando o resto da divisão pelo tamanho.
  - hashMultiplicacaoCaractere: Utiliza uma constante multiplicativa para calcular o hash, gerando um valor que é reduzido ao índice pelo tamanho da tabela.
- **Método gerarRegistroAleatorio**: Gera um Registro com um código aleatório de nove dígitos para testes de inserção e busca na tabela hash, utiliza random.

```java
    public static void main(String[] args) {
        int[] tamanhosTabela = {100, 1000, 10000};
        int profundidadeEncadeamento = 10;
        int[] tamanhosConjunto = {1000000, 5000000, 20000000};

        for (int tipoHash = 1; tipoHash <= 3; tipoHash++) {
            long[] temposInsercao = new long[Utilidades.comprimento(array: tamanhosTabela)];
            long[] temposBusca = new long[Utilidades.comprimento(array: tamanhosTabela)];
            int[] colisoes = new int[Utilidades.comprimento(array: tamanhosTabela)];
            int[] comparacoes = new int[Utilidades.comprimento(array: tamanhosTabela)];

            for (int i = 0; i < Utilidades.comprimento(array: tamanhosTabela); i++) {
                int tamanhoTabela = tamanhosTabela[i];
                TabelaHashEncadeamento tabela = new TabelaHashEncadeamento(tamanho: tamanhoTabela, profundidade: profundidadeEncadeamento);
                int[] colisoesAtual = {0};
                long tempoInicioInsercao = System.currentTimeMillis();

                tabela.realizarInsercoes(tamanhosConjunto[i], tipoHash, colisoes: colisoesAtual);
                long tempoFimInsercao = System.currentTimeMillis();
                temposInsercao[i] = tempoFimInsercao - tempoInicioInsercao;
                colisoes[i] = colisoesAtual[0];

                int[] comparacoesAtual = {0};
                long tempoTotalBusca = tabela.realizarBuscas(quantidade: 5, tipoHash, comparacoes: comparacoesAtual);
                temposBusca[i] = tempoTotalBusca / 5;
                comparacoes[i] = comparacoesAtual[0];
            }

            String[] seriesNames = {
                "Tempo de Insercao (ms) - Hash " + tipoHash,
                "Tempo de Busca (ms) - Hash " + tipoHash,
                "Colisoes (Normalizadas) - Hash " + tipoHash,
                "Comparacoes na Busca (Normalizadas) - Hash " + tipoHash
            };
            //Grafico
```

Classe RA03: A classe RA03 executa a lógica principal, testando a tabela hash em diferentes tamanhos e profundidades. Justificativa: O main lida com a construção do código em si chamando métodos para auxiliá-lo.

```java
            //Grafico
            double[] tamanhosTabelaDouble = new double[Utilidades.comprimento(array: tamanhosTabela)];
            double[] temposInsercaoDouble = new double[Utilidades.comprimentoLong(array: temposInsercao)];
            double[] temposBuscaDouble = new double[Utilidades.comprimentoLong(array: temposBusca)];
            double[] colisoesNormalizadas = new double[Utilidades.comprimento(array: colisoes)];
            double[] comparacoesNormalizadas = new double[Utilidades.comprimento(array: comparacoes)];
            double fatorNormalizacao = 10000.0;

            for (int i = 0; i <Utilidades.comprimento(array: tamanhosTabela); i++) {
                tamanhosTabelaDouble[i] = tamanhosTabela[i];
                temposInsercaoDouble[i] = temposInsercao[i];
                temposBuscaDouble[i] = temposBusca[i];
                colisoesNormalizadas[i] = colisoes[i] / fatorNormalizacao;
                comparacoesNormalizadas[i] = comparacoes[i] / fatorNormalizacao;
            }

            double[][] yDataSeries = {temposInsercaoDouble, temposBuscaDouble, colisoesNormalizadas, comparacoesNormalizadas};
            Utilidades.gerarGrafico(
                "Desempenho da Tabela Hash - Tipo de Hash " + tipoHash,
                xAxisTitle: "Tamanho da Tabela",
                yAxisTitle: "Tempo (ms) / Quantidade Normalizada",
                xDado: tamanhosTabelaDouble,
                serieNome:seriesNames,
                yDado: yDataSeries
            );
        }
            System.out.println(x: "Hash 1. . . RestoDivisao");
            System.out.println(x: "Hash 2. . . SomaDigitos");
            System.out.println(x: "Hash 3. . . MultiplicacaoCaractere");
    } }
```

- **Gráficos**: E responsável por organizar os dados de desempenho da tabela hash para que sejam representados graficamente. Ele primeiro converte os arrays de tamanhos da tabela; tempos de inserção; tempos de busca; colisões e comparações para o tipo double, preparando-os para o gráfico. A normalização é feita para as colisões e comparações, dividindo seus valores pelo fatorNormalizacao (definido 10000.0) para que possam ser comparados em uma escala mais ajustada aos tempos de inserção e busca. Em seguida ele cria um array seriesNames para nomear cada série de dados no gráfico, e um array bidimensional yDataSeries que agrupa todas as séries a serem exibidas.

Por fim, o método chama Utilidades.gerarGrafico, passando o título do gráfico, os títulos dos eixos, os dados do eixo x (tamanhos da tabela) e os dados do eixo y (séries de desempenho) para gerar e exibir o gráfico com uma interface gráfica Justificativa: Optei por utilizar XChart.

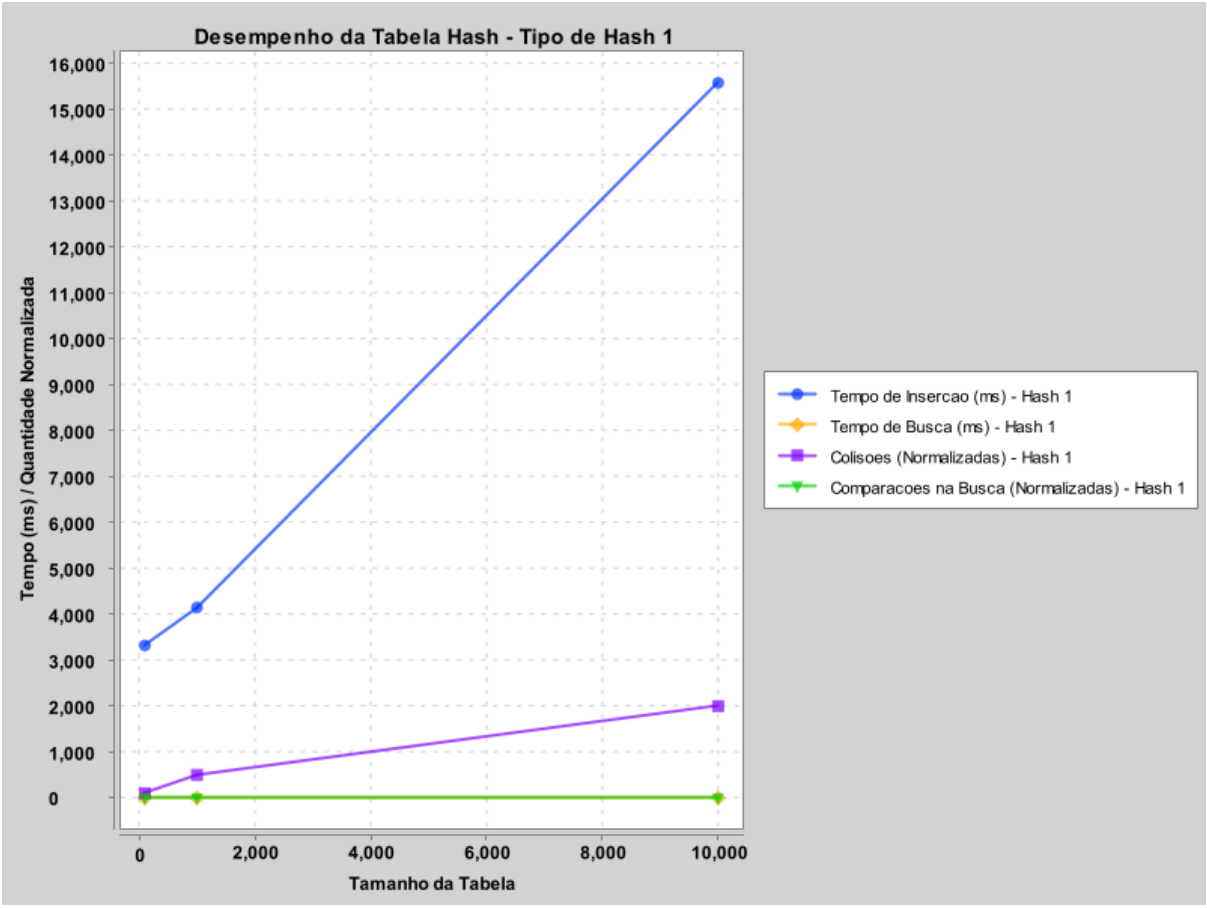
```java
public class Utilidades {
    public static int[] geraRandomArray(int tam, long seed) {
        Random rand = new Random(seed);
        int[] array = new int[tam];
        for (int i = 0; i < tam; i++) {
            array[i] = rand.nextInt();
        }
        return array;
    }
    public static double medeEmMilise(Runnable task) {
        long inicio = System.nanoTime();
        task.run();
        long fim = System.nanoTime();
        return (fim - inicio) / 1e6;
    }
    public static void gerarGrafico(String titulo, String xAxisTitle, String yAxisTitle,
                                    double[] xDado, String[] serieNome, double[][] yDado) {

        XYChart chart = new XYChartBuilder().width(width: 800).height(height:600)
                .title(title: titulo)
                .xAxisTitle(xAxisTitle)
                .yAxisTitle(yAxisTitle)
                .build();

        for (int i = 0; i < comprimentoString(array: serieNome); i++) {
            chart.addSeries(serieNome[i], xData: xDado, yDado[i]);
        }

        new SwingWrapper<>(chart).displayChart();
    }
```

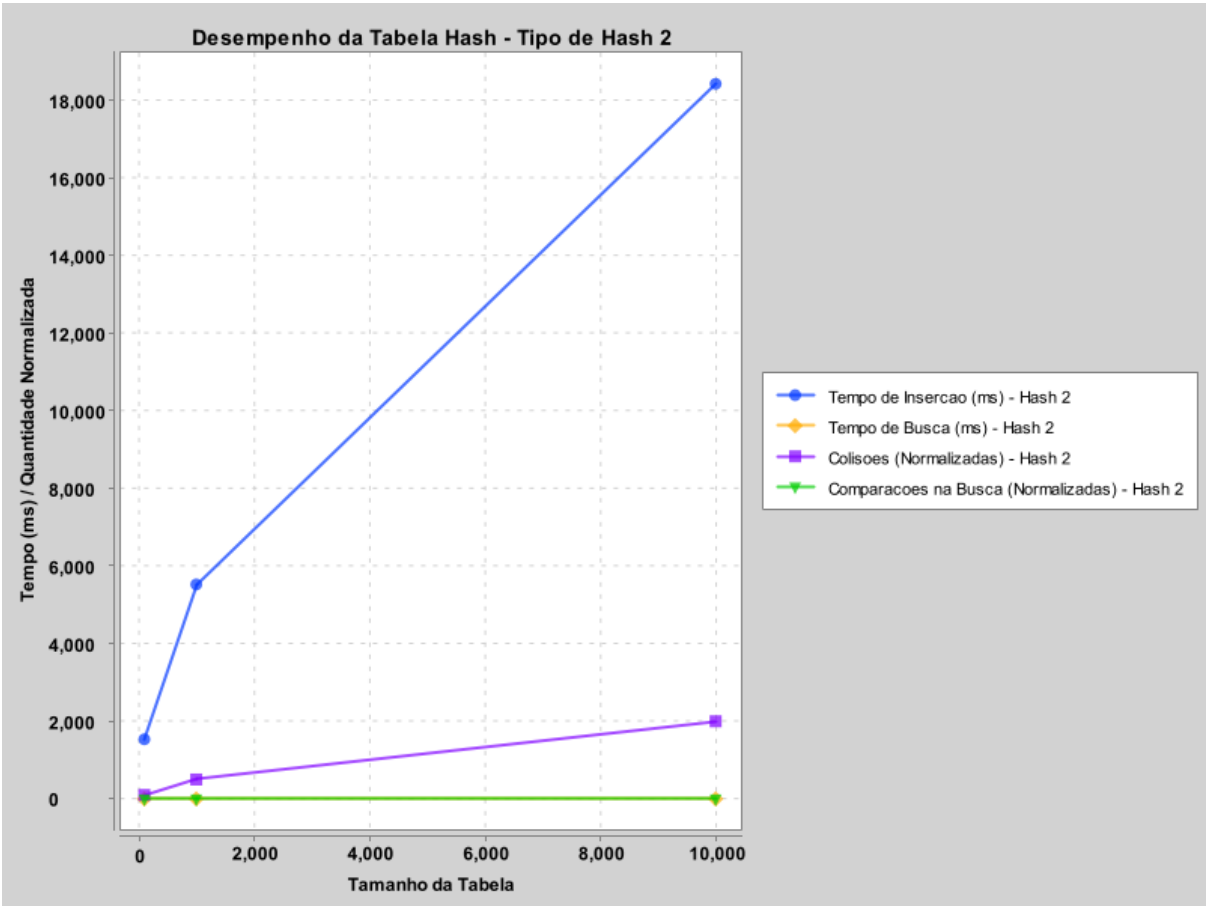
Classe Utilidades: Uma classe de suporte que é utilizada no RA03 e RA04. Justificativa: Escolhi fazer desta maneira pois quis evitar redundância de componentes que são utilizados em ambos os RAs e facilitar legibilidade nos mesmos.

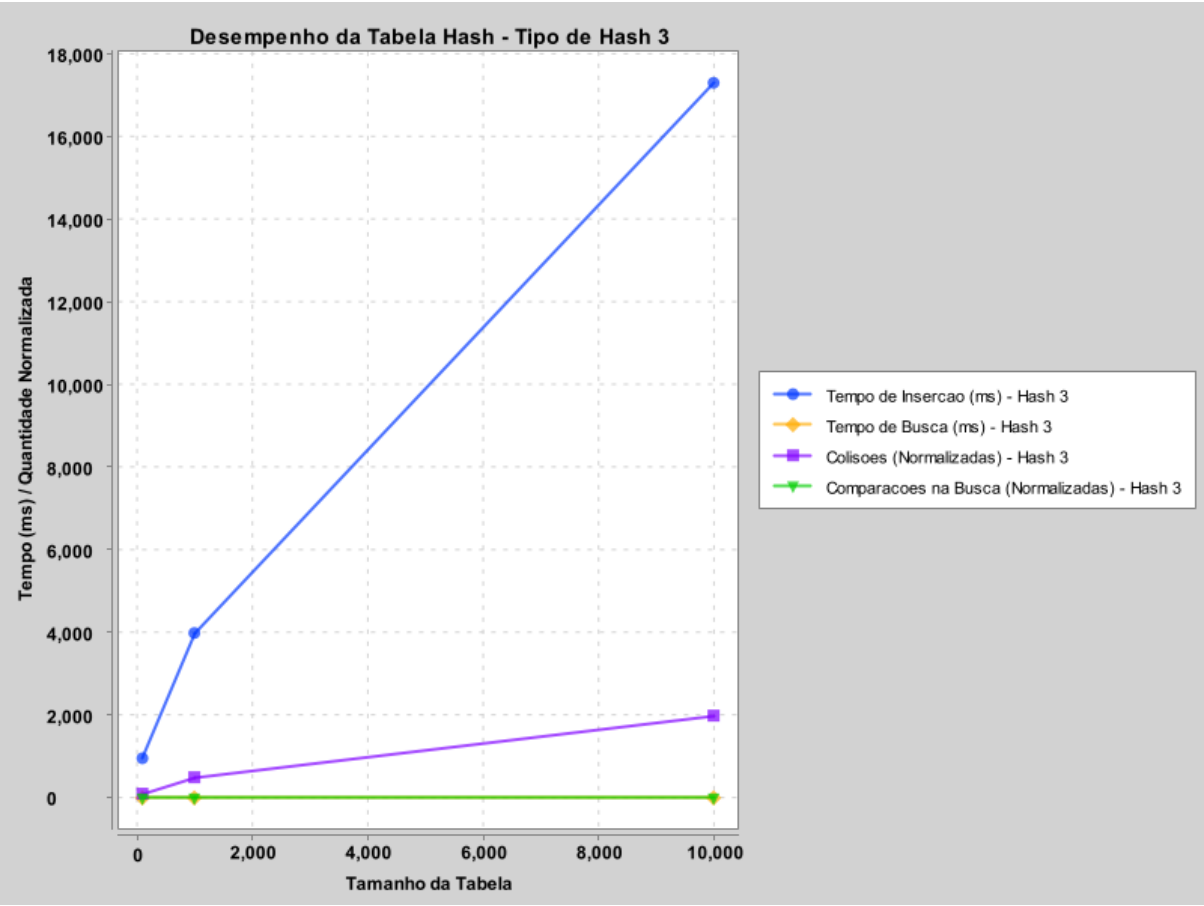
ANÁLIZE DO RESULTADO

Considerando os fatores de tempo de inserção, busca, colisões e comparações na busca o único fator o qual se destaca é o de tempo de incursões o qual deixa claro que o primeiro hash (resto divisão) é o mais eficiente no cenário experimentado pois este tem uma escalada do tempo de inserção mais retardatária em comparação aos outros objetos de amostra, como hash 2 (soma digitos) e hash 3 (multiplicação caractere). Nota-se também que o tempo de colisão assim como o de inserção aumentam conforme maior volume de dados.

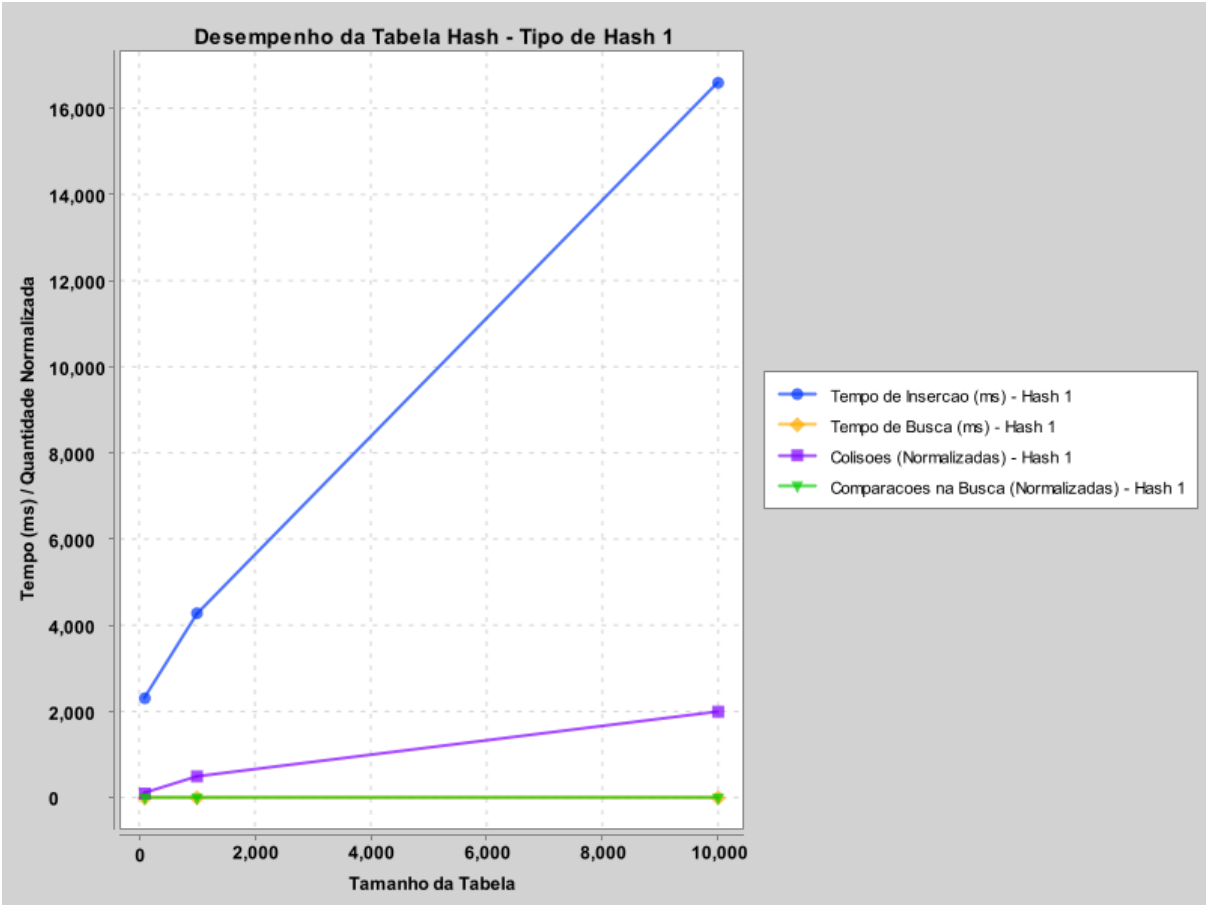
Primeira execução

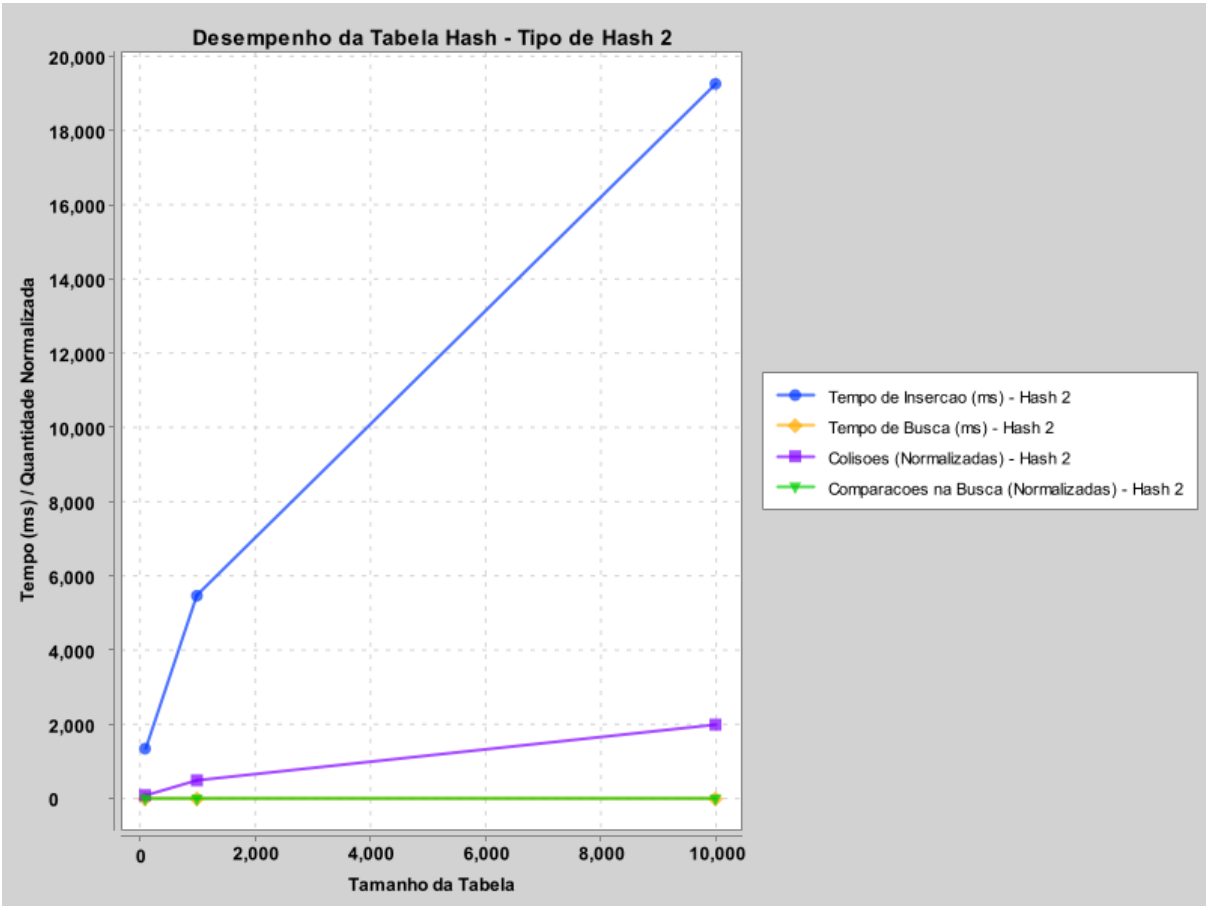

Desempenho da Tabela Hash - Tipo de Hash 1
Tempo (ms) / Quantidade Normalizada
16,000
15,000
14,000
13,000
12,000
11,000
10,000
9,000
8,000
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
0
2,000
4,000
6,000
8,000
10,000
Tamanho da Tabela
Tempo de Insercao (ms) - Hash 1
Tempo de Busca (ms) - Hash 1
Colisoes (Normalizadas) - Hash 1
Comparacoes na Busca (Normalizadas) - Hash 1

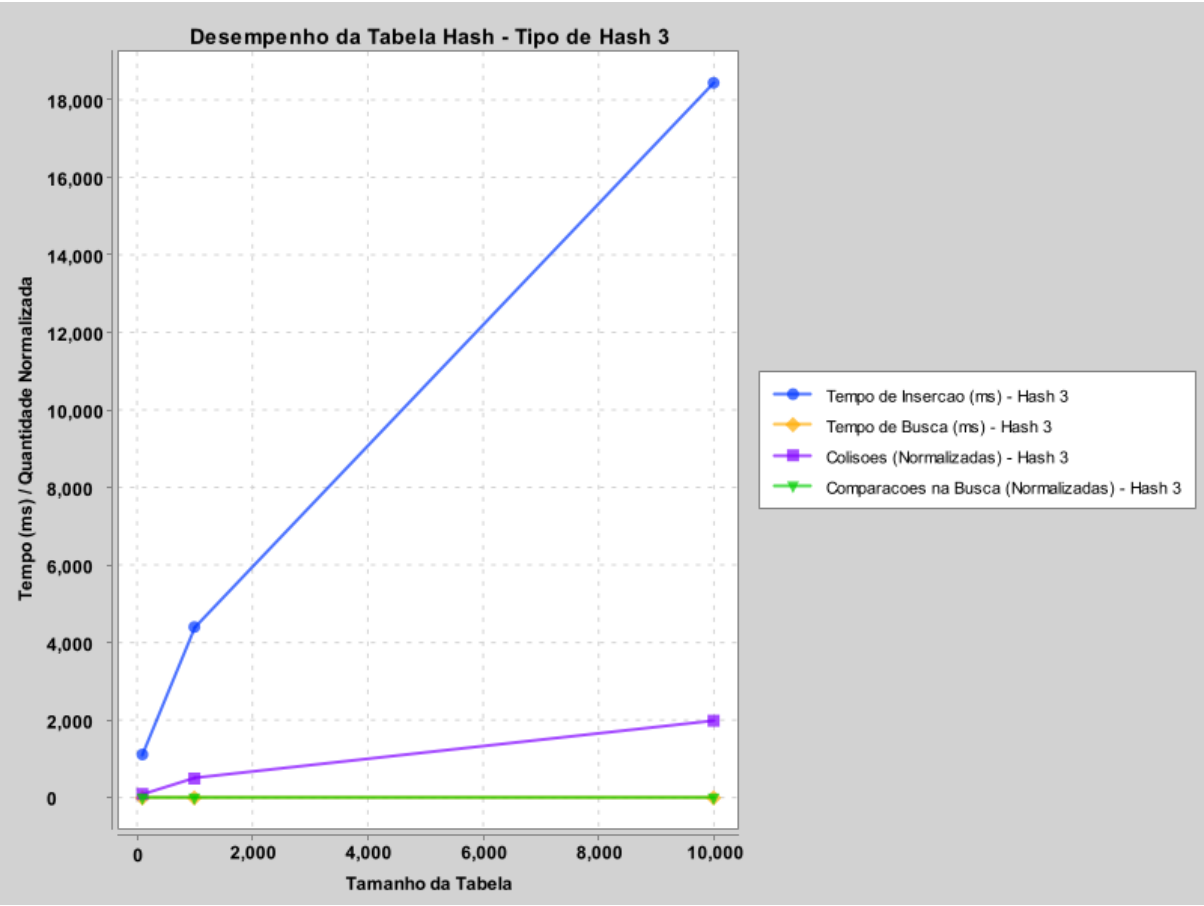

Desempenho da Tabela Hash - Tipo de Hash 2
Tempo (ms) / Quantidade Normalizada
18,000
16,000
14,000
12,000
10,000
8,000
6,000
4,000
2,000
0
0
2,000
4,000
6,000
8,000
10,000
Tamanho da Tabela
Tempo de Insercao (ms) - Hash 2
Tempo de Busca (ms) - Hash 2
Colisoes (Normalizadas) - Hash 2
Comparacoes na Busca (Normalizadas) - Hash 2



Segunda execução


Desempenho da Tabela Hash - Tipo de Hash 1
Tempo (ms) / Quantidade Normalizada
16,000
14,000
12,000
10,000
8,000
6,000
4,000
2,000
0
0
2,000
4,000
6,000
8,000
10,000
Tamanho da Tabela
Tempo de Insercao (ms) - Hash 1
Tempo de Busca (ms) - Hash 1
Colisoes (Normalizadas) - Hash 1
Comparacoes na Busca (Normalizadas) - Hash 1


Desempenho da Tabela Hash - Tipo de Hash 2
Tempo (ms) / Quantidade Normalizada
20,000
18,000
16,000
14,000
12,000
10,000
8,000
6,000
4,000
2,000
0
0
2,000
4,000
6,000
8,000
10,000
Tamanho da Tabela
Tempo de Insercao (ms) - Hash 2
Tempo de Busca (ms) - Hash 2
Colisoes (Normalizadas) - Hash 2
Comparacoes na Busca (Normalizadas) - Hash 2


Desempenho da Tabela Hash - Tipo de Hash 3
Tempo (ms) / Quantidade Normalizada
18,000
16,000
14,000
12,000
10,000
8,000
6,000
4,000
2,000
0
0
2,000
4,000
6,000
8,000
10,000
Tamanho da Tabela
Tempo de Insercao (ms) - Hash 3
Tempo de Busca (ms) - Hash 3
Colisoes (Normalizadas) - Hash 3
Comparacoes na Busca (Normalizadas) - Hash 3