**Trabalho Tabela Hash**

**Brunno Tatsuo**

Engenharia de Software

Pontifícia Universidade Católica do Paraná (PUCPR) – Curitiba, PR – Brasil

cruz.brunno@pucpr.edu.br

**Abstract.** In this project, the task is to implement and analyze the performance of different hash tables in Java, using chaining as the strategy. The project involves several essential steps, starting with the selection of five distinct sizes for the hash table array, ranging from 10,000 to 200,000 elements. Furthermore, three variations of hash functions, such as modulo, multiplication, and folding, are explored to assess their impact on performance. The generation of five random data sets with varying sizes, ranging from 20,000 to 5 million elements, is conducted. The insertion of elements into each hash table using the different hash functions is followed by the evaluation of insertion time and the number of collisions. Additionally, searching for elements in each hash table is performed, measuring search time and the number of comparisons. These results are then presented in tables and graphs, enabling a comparative analysis of the performance of hash tables and hash functions, contributing to a deeper understanding of how these choices impact the operation of data structures in practical scenarios.

**Resumo.** Neste trabalho, têm a tarefa de implementar e analisar o desempenho de diferentes tabelas hash em Java, usando estratégia de encadeamento. O projeto envolve diversas etapas essenciais, começando pela seleção de cinco tamanhos distintos para o vetor da tabela hash, variando de 10 mil a 200 mil elementos. Além disso, são exploradas três variações de função hash, como o resto da divisão, multiplicação e dobramento, para avaliar seu impacto no desempenho. A geração de cinco conjuntos de dados aleatórios com tamanhos variados, que vão de 20 mil a 5 milhões de elementos. A inserção de elementos em cada tabela hash, utilizando as diferentes funções hash, é seguida pela avaliação do tempo de inserção e do número de colisões. Além disso, a busca dos elementos em cada tabela hash, é realizada, medindo o tempo de busca e o número de comparações. Esses resultados são então apresentados em tabelas e gráficos, permitindo uma análise comparativa do desempenho das tabelas hash e funções hash, contribuindo para uma compreensão mais profunda de como essas escolhas impactam o funcionamento da estrutura de dados em cenários práticos.

1. **Introdução**

Tabelas hash são estruturas de dados fundamentais em ciência da computação, amplamente utilizadas para armazenar e recuperar informações de forma eficiente. Elas desempenham um papel crucial em diversas aplicações, desde bancos de dados até sistemas de busca na web. No entanto, a eficiência e o desempenho de uma tabela hash dependem fortemente da função de hash escolhida. Neste relatório, exploraremos três abordagens diferentes para a construção de tabelas hash: a função de multiplicação, o dobramento e a divisão. Analisaremos as vantagens e desvantagens de cada uma dessas técnicas, além de apresentar exemplos práticos de como elas podem ser implementadas e avaliadas em relação ao seu desempenho. Esta análise permitirá uma compreensão mais profunda das tabelas hash e de como a escolha da função de hash pode influenciar a eficiência do armazenamento e recuperação de dados em aplicações reais.

A função por divisão apresenta vantagens notáveis, como sua notável simplicidade de implementação e compreensão, seu desempenho eficaz quando se utiliza um tamanho de tabela primo e sua eficácia especialmente para chaves que são números inteiros. No entanto, esta abordagem não está isenta de desvantagens, tais como o potencial para colisões frequentes, que pode ocorrer se as chaves não estiverem distribuídas uniformemente ou se o tamanho da tabela não for escolhido adequadamente. Além disso, a função de divisão pode não funcionar bem para tipos de chave que não sejam inteiros, e o tamanho da tabela desempenha um papel crítico em seu desempenho, sendo crucial escolher um tamanho adequado para obter resultados satisfatórios.

A função por dobramento apresenta diversas vantagens, incluindo sua versatilidade, que a torna eficaz para uma variedade de tipos de chaves, como números inteiros e strings, uma distribuição razoável das chaves quando o processo de dobramento é bem projetado e a facilidade de implementação devido à sua simplicidade. Entretanto, essa abordagem não está isenta de desvantagens, como o potencial para colisões, que pode ser mais frequente dependendo da estratégia de dobramento utilizada, a necessidade de cuidado na implementação para garantir sua qualidade, sujeita a erros, e a possível ineficiência quando se lidam com chaves grandes ou mais complexas.

A função por multiplicação oferece várias vantagens, incluindo simplicidade na implementação, uma boa distribuição das chaves quando ajustada corretamente e a adaptabilidade aos diferentes tamanhos de tabelas hash. No entanto, também apresenta desvantagens, como a necessidade de experimentação para escolher os parâmetros ideais, a potencial lentidão devido a operações de ponto flutuante em algumas arquiteturas de hardware e a sensibilidade à distribuição das chaves, o que pode afetar seu desempenho quando as chaves não estão uniformemente distribuídas.

1. **Código**

**Classe No**

public class Node {

private int chave;

private Node proximo;

public Node(int chave){

this.chave = chave;

}

public int getChave() {

return chave;

}

public void setChave(int chave) {

this.chave = chave;

}

public Node getProximo() { return proximo;

}

public void setProximo(Node proximo) {

this.proximo = proximo;

}

}

**Classe Tabela**

import java.util.Random;

public class Tabela {

private Node[] tabela;

private int tamanho;

private int hash;

private int colissao;

public Tabela(int tamanho,int conj,int hash,int seed){

this.tabela = new Node[tamanho];

this.tamanho = tamanho;

this.hash = hash;

this.colissao = 0;

Random rd = new Random(seed);

long start = System.nanoTime();

for(int i= 0 ;i<conj;i++){

inserir(rd.nextInt(999\_999\_999));

}

long end = System.nanoTime();

// imprimir();

System.out.println("Tempo Decorrido: "+(end-start)+" Nanosegundos");

System.out.println("Número de Colisões: "+ getColissao());

}

public int getColissao(){

return colissao;

}

public void inserir(int chave){

inserir(chave,this.hash);

}

private void inserir(int chave,int hash){

int i=-1;

if (hash==1){

i =h(chave);

}else if(hash==2) {

i = h2(chave);

} else if(hash==3) {

i = h3(chave);

}

Node novoNo = new Node(chave);

if (tabela[i]!=null){

Node temp = tabela[i];

colissao++;

while (true){

if(temp.getProximo()==null){

temp.setProximo(novoNo);

break;

}else{

if (chave<temp.getProximo().getChave()){

novoNo.setProximo(temp.getProximo());

temp.setProximo(novoNo);

break;

} else{

temp=temp.getProximo();

colissao++;

}

}

}

} else{

tabela[i]=novoNo;

}

}

public void buscar(int chave){

buscar(chave,this.hash);

}

private void buscar(int chave,int hash){

String msg="";

int comparacao=0;

long start2 = System.nanoTime();

int i=-1;

if(hash==1){

i = h(chave);

} else if(hash==2) {

i = h2(chave);

} else if(hash==3) {

i = h3(chave);

}

Node temp=tabela[i];

while (temp!=null){

if(temp.getChave()==chave){

msg ="Encontrado";

break;

}else{

temp=temp.getProximo();

comparacao++;

}

}

if(temp==null) {

msg ="Não Encontrado";

}

long end2 = System.nanoTime();

System.out.printf("\n%d %s! Tempo decorrido: "+ (end2- start2)+" Nanosegundos, Número de Comparações: %d" ,chave,msg,comparacao);

}

private int h(int chave){

return chave%tamanho;

}

private int h2(int chave){

int a = chave/100\_000;

chave = chave%100\_000;

int c =chave;

return (a+c)%tamanho;

}

private int h3(int chave){

double a = ((0.63274838\*chave)%1)\*tamanho;

return (int) a;

}

public void imprimir(){

System.out.println("Posição | Chave | Proximo");

for(int i = 0;i< tamanho;i++){

if(tabela[i]==null){

System.out.printf(" %6d | %9s\n",i,"Vazio");

}else{

String msg = String.format(" %6d | %09d",i,tabela[i].getChave());

Node temp = tabela[i];

while(temp.getProximo()!=null){

temp=temp.getProximo();

msg += String.format(" -> %09d",temp.getChave());

}

System.out.println(msg);

}

}

System.out.println("Nuúmero de Colissões: "+colissao);

}

}

**Classe Main**

public class Main {

public static void main(String[] args) {

//Parte 1 teste de inserção

int seed = 81; //{24,33,12,50,81}

int tamanho = 200\_000;//{10\_000,25\_000,50\_000;100\_000,200\_000}

int[] conjuntos = {20\_000,100\_000,500\_000,1\_000\_000,5\_000\_000};

for(int conj:conjuntos){

System.out.println("\nTamanho: "+tamanho+" Conjunto: "+conj+" seed: "+seed);

Tabela tabela1 = new Tabela(tamanho,conj,1,seed);

Tabela tabela2 = new Tabela(tamanho,conj,2,seed);

Tabela tabela3 = new Tabela(tamanho,conj,3,seed);

}

//Parte 2 para realizar o teste de busca.

Tabela tabela1 = new Tabela(tamanho,5\_000\_000,1,seed);

Tabela tabela2 = new Tabela(tamanho,5\_000\_000,2,seed);

Tabela tabela3 = new Tabela(tamanho,5\_000\_000,3,seed);

tabela1.buscar(345396281);

tabela1.buscar(987765423);

tabela1.buscar(117321717);

tabela1.buscar(618692254);

tabela1.buscar(10899880);

System.out.println();

tabela2.buscar(345396281);

tabela2.buscar(987765423);

tabela2.buscar(117321717);

tabela2.buscar(618692254);

tabela2.buscar(10899880);

System.out.println();

tabela3.buscar(345396281);

tabela3.buscar(987765423);

tabela3.buscar(117321717);

tabela3.buscar(618692254);

tabela3.buscar(10899880);

}

}

1. **Resultados**

**Tabela com 10.000 vetores:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **20 mil** | **100 mil** | **500 mil** | **1 milhão** | **5 milhões** |
| **Divisão** | 1.833.300 | 6.318.800 | 114.446.400 | 415.951.400 | 13.484.877.600 |
| 1.772.900 | 6.737.600 | 113.216.900 | 428.809.500 | 14.198.211.000 |
| 1.791.900 | 6.626.300 | 102.658.000 | 372.307.200 | 13.456.763.400 |
| 1.745.700 | 6.518.100 | 118.170.400 | 479.917.900 | 13.980.719.700 |
| 2.248.500 | 10.338.200 | 268.819.900 | 874.683.700 | 17.216.114.700 |
| **Dobramento** | 2.007.500 | 5.854.700 | 168.441.100 | 390.300.100 | 17.170.869.500 |
| 1.597.900 | 6.327.600 | 175.355.400 | 384.467.100 | 15.787.082.600 |
| 1.619.400 | 6.843.100 | 178.052.300 | 420.987.700 | 14.850.190.600 |
| 1.553.400 | 5.920.700 | 170.739.700 | 433.357.100 | 15.305.760.600 |
| 1.701.300 | 7.526.800 | 125.510.100 | 356.206.800 | 18.367.952.400 |
| **Multiplicação** | 3.879.900 | 17.070.900 | 271.912.400 | 698.064.600 | 15.495.408.600 |
| 4.026.800 | 17.525.400 | 250.924.100 | 539.611.500 | 16.248.864.600 |
| 4.082.600 | 17.265.200 | 249.498.800 | 621.160.600 | 17.111.344.900 |
| 3.705.200 | 18.119.600 | 213.506.500 | 544.931.000 | 16.765.170.600 |
| 4.058.800 | 23.766.600 | 189.001.100 | 591.777.900 | 20.431.492.100 |

**Tabela 01. Tempo de inserção em nanosegundos na tabela hash com 10.000 vetores.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **20 mil** | **100 mil** | **500 mil** | **1 milhão** | **5 milhões** |
| **Divisão** | 15.734 | 296.044 | 6.494.539 | 25.492.313 | 627.553.240 |
| 15.646 | 294.352 | 6.503.110 | 25.529.539 | 627.730.178 |
| 15.746 | 294.825 | 6.491.068 | 25.470.886 | 627.362.323 |
| 15.601 | 294.573 | 6.497.014 | 25.482.341 | 627.580.147 |
| 15.861 | 294.701 | 6.500.243 | 25.489.110 | 627.371.057 |
| **Dobramento** | 15.627 | 296.001 | 6.496.059 | 25.491.412 | 627.499.363 |
| 15.608 | 294.378 | 6.502.231 | 25.522.534 | 627.618.632 |
| 15.885 | 295.330 | 6.492.373 | 25.471.913 | 627.317.105 |
| 15.888 | 294.514 | 6.498.219 | 25.481.226 | 627.533.102 |
| 15.731 | 293.781 | 6.498.218 | 25.479.651 | 627.414.772 |
| **Multiplicação** | 15.819 | 296.628 | 6.496.780 | 25.496.736 | 627.565.375 |
| 15.757 | 295.350 | 6.504.182 | 25.526.779 | 627.697.628 |
| 15.557 | 295.435 | 6.493.200 | 25.478.395 | 627.333.339 |
| 15.565 | 294.955 | 6.502.700 | 25.484.467 | 627.610.618 |
| 15.521 | 294.567 | 6.496.619 | 25.470.902 | 627.437.956 |

**Tabela 02. Número de colisões na tabela hash com 10.000 vetores.**

**Tabela com 25.000 vetores:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **20 mil** | **100 mil** | **500 mil** | **1 milhão** | **5 milhões** |
| **Divisão** | 1.760.300 | 5.137.500 | 54.169.600 | 346.210.500 | 7.482.934.500 |
| 1.641.500 | 4.941.200 | 57.545.300 | 345.952.800 | 8.542.793.800 |
| 1.998.600 | 5.512.300 | 59.139.400 | 339.410.400 | 8.235.391.800 |
| 1.759.799 | 5.045.999 | 56.497.899 | 291.490.500 | 8.378.732.201 |
| 1.724.199 | 5.012.500 | 63.284.000 | 332.920.200 | 8.608.060.000 |
| **Dobramento** | 1.462.400 | 4.214.700 | 72.558.800 | 258.497.700 | 8.072.539.200 |
| 1.414.700 | 4.423.300 | 70.140.700 | 235.875.800 | 8.249.439.600 |
| 1.451.000 | 4.257.600 | 76.187.700 | 201.776.700 | 7.402.900.000 |
| 1.401.601 | 4.171.800 | 69.924.900 | 239.930.100 | 7.464.106.599 |
| 1.494.000 | 4.279.301 | 89.214.901 | 191.692.701 | 8.310.159.400 |
| **Multiplicação** | 3.889.400 | 15.961.700 | 162.304.700 | 413.086.500 | 8.908.835.000 |
| 3.594.600 | 15.495.100 | 188.074.900 | 350.044.500 | 9.678.316.100 |
| 3.823.700 | 15.633.900 | 176.955.600 | 370.764.900 | 8.097.373.100 |
| 3.911.201 | 15.777.600 | 173.977.400 | 435.201.699 | 9.663.582.201 |
| 3.754.699 | 17.107.700 | 166.886.199 | 362.142.400 | 9.832.617.900 |

**Tabela 03. Tempo de inserção em nanosegundos na tabela hash com 25.000 vetores.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **20 mil** | **100 mil** | **500 mil** | **1 milhão** | **5 milhões** |
| **Divisão** | 7.101 | 138.084 | 2.734.423 | 10.483.958 | 252.528.732 |
| 7.184 | 138.134 | 2.742.701 | 10.504.769 | 252.561.294 |
| 7.209 | 137.409 | 2.733.943 | 10.474.794 | 252.407.069 |
| 7.078 | 137.807 | 2.739.225 | 10.483.602 | 252.518.815 |
| 7.117 | 137.575 | 2.740.725 | 10.485.750 | 252.440.363 |
| **Dobramento** | 7.171 | 138.513 | 2.736.718 | 10.483.477 | 252.481.579 |
| 7.125 | 137.629 | 2.741.007 | 10.499.606 | 252.540.459 |
| 7.120 | 137.584 | 2.736.252 | 10.481.740 | 252.420.344 |
| 7.157 | 137.679 | 2.739.415 | 10.484.998 | 252.541.795 |
| 7.202 | 138.094 | 2.742.036 | 10.488.592 | 252.454.422 |
| **Multiplicação** | 7.196 | 138.177 | 2.739.442 | 10.487.394 | 252.502.032 |
| 7.146 | 137.830 | 2.740.107 | 10.496.285 | 252.564.542 |
| 7.102 | 137.904 | 2.734.979 | 10.475.592 | 252.398.250 |
| 7.181 | 137.427 | 2.740.619 | 10.482.308 | 252.526.922 |
| 7.015 | 137.458 | 2.738.740 | 10.479.898 | 252.468.469 |

**Tabela 04. Número de colisões na tabela hash com 25.000 vetores.**

**Tabela com 50.000 vetores:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **20 mil** | **100 mil** | **500 mil** | **1 milhão** | **5 milhões** |
| **Divisão** | 1.738.001 | 4.589.800 | 43.847.800 | 213.550.800 | 5.762.178.700 |
| 1.676.400 | 4.460.801 | 421.22.501 | 185.304.001 | 5.149.747.200 |
| 1.703.201 | 4.225.699 | 48.827.199 | 178.570.501 | 5.311.424.400 |
| 1.674.000 | 4.868.500 | 41.814.699 | 181.879.000 | 5.503.968.000 |
| 1.638.401 | 4.705.300 | 41.580.000 | 184.868.201 | 5.130.937.300 |
| **Dobramento** | 1.431.300 | 4.247.601 | 44.374.399 | 183.499.100 | 5.591.848.499 |
| 1.144.200 | 3.584.900 | 45.087.701 | 235.457.900 | 4.600.174.500 |
| 1.435.700 | 3.763.600 | 41.256.800 | 223.571.499 | 4.469.548.000 |
| 1.421.100 | 3.600.300 | 42.159.900 | 186.936.200 | 4.309.534.200 |
| 1.561.800 | 3.845.400 | 46.816.400 | 172.868.201 | 4.692.780.800 |
| **Multiplicação** | 3.444.000 | 16.005.699 | 180.473.700 | 339.734.599 | 6.372.426.601 |
| 3.553.900 | 15.206.600 | 143.836.001 | 299.866.800 | 6.117.288.200 |
| 3.662.500 | 16.351.000 | 153.627.099 | 322.520.199 | 6.085.087.201 |
| 3.742.100 | 15.950.500 | 216.799.200 | 318.462.700 | 6.147.900.999 |
| 3.755.899 | 15.177.700 | 196.880.301 | 318.799.500 | 5.388.587.600 |

**Tabela 05. Tempo de inserção em nanosegundos na tabela hash com 50.000 vetores.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **20 mil** | **100 mil** | **500 mil** | **1 milhão** | **5 milhões** |
| **Divisão** | 3.719 | 78.529 | 1.472.417 | 5.473.569 | 127.503.493 |
| 3.856 | 78.653 | 1.478.100 | 5.485.574 | 127.528.836 |
| 3.854 | 78.402 | 1.472.920 | 5.468.313 | 127.442.263 |
| 3.760 | 78.522 | 1.475.867 | 5.475.381 | 127.501.553 |
| 3.749 | 78.426 | 1.476.966 | 5.474.729 | 127.467.814 |
| **Dobramento** | 3.770 | 78.446 | 1.474.665 | 5.472.164 | 127.481.036 |
| 3.761 | 78.476 | 1.477.018 | 5.482.110 | 127.500.994 |
| 3.795 | 78.225 | 1.473.216 | 5.471.208 | 127.445.409 |
| 3.778 | 78.239 | 1.476.526 | 5.474.442 | 127.499.622 |
| 3.794 | 78.401 | 1.474.944 | 5.471.012 | 127.458.338 |
| **Multiplicação** | 3.810 | 78.468 | 1.476.116 | 5.478.888 | 127.496.652 |
| 3.791 | 78.454 | 1.474.717 | 5.478.077 | 127.505.268 |
| 3.663 | 78.035 | 1.475.105 | 5.470.068 | 127.433.802 |
| 3.785 | 78.466 | 1.476.713 | 5.473.996 | 127.508.866 |
| 3.711 | 78.056 | 1.475.322 | 5.468.942 | 127.455.948 |

**Tabela 06. Número de colisões na tabela hash com 50.000 vetores.**

**Tabela com 100.000 vetores**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **20 mil** | **100 mil** | **500 mil** | **1 milhão** | **5 milhões** |
| **Divisão** | 1.652.100 | 4.044.000 | 37.101.200 | 157.220.001 | 3.386.304.899 |
| 1.767.200 | 3.957.101 | 36.706.099 | 141.604.100 | 3.270.228.699 |
| 2.456.301 | 3.990.000 | 35.341.700 | 137.775.900 | 3.232.197.800 |
| 1.853.500 | 3.980.300 | 36.283.300 | 153.350.099 | 3.355.982.300 |
| 1.636.200 | 4257599 | 34.415.600 | 189.542.200 | 3.258.063.000 |
| **Dobramento** | 1.388.501 | 3.540.900 | 28.676.801 | 186.233.800 | 2.872.203.399 |
| 1.772.900 | 3.400.000 | 31.108.201 | 194.857.400 | 2.937.536.599 |
| 1.322.000 | 3.059.800 | 32.182.400 | 181.845.201 | 2.900.539.600 |
| 1.655.200 | 3.183.500 | 30.758.301 | 229.653.999 | 2.845.989.100 |
| 1.474.500 | 3.132.800 | 28.628.800 | 157.898.700 | 2.939.590.700 |
| **Multiplicação** | 3.793.899 | 14.525.200 | 119.959.899 | 259.473.600 | 4.024.111.399 |
| 4.219.100 | 14.431.000 | 127.371.100 | 323.658.600 | 4.027.228.900 |
| 4.012.201 | 14.219.900 | 128.381.001 | 373.060.299 | 3.732.574.700 |
| 3.666.101 | 14.788.200 | 123.679.700 | 258.205.700 | 3.781.598.300 |
| 3.598.200 | 14.527.001 | 149.593.701 | 303.855.299 | 3.645.678.499 |

**Tabela 07. Tempo de inserção em nanosegundos na tabela hash com 100.000 vetores.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **20 mil** | **100 mil** | **500 mil** | **1 milhão** | **5 milhões** |
| **Divisão** | 1.915 | 43.330 | 825.810 | 2.948.485 | 64.956.308 |
| 1.898 | 43.380 | 823.885 | 2.949.390 | 64.969.808 |
| 1.944 | 43.476 | 825.833 | 2.951.786 | 64.950.727 |
| 1.985 | 43.420 | 824.248 | 2.946.974 | 64.929.827 |
| 1.985 | 43.632 | 826.490 | 2.950.197 | 64.976.401 |
| **Dobramento** | 1.961 | 43.301 | 825.462 | 2.948.970 | 64.939.004 |
| 1.916 | 43.307 | 825.242 | 2.948.472 | 64.951.166 |
| 1.937 | 43.477 | 825.628 | 2.951.810 | 64.967.593 |
| 1.960 | 43.486 | 824.947 | 2.948.520 | 64.937.618 |
| 1.956 | 43.351 | 825.465 | 2.948.623 | 64.966.862 |
| **Multiplicação** | 1.911 | 43.038 | 824.767 | 2.945.072 | 64.941.788 |
| 1.960 | 43.251 | 825.645 | 2.952.708 | 64.970.446 |
| 1.953 | 43.394 | 825.480 | 2.951.868 | 64.960.940 |
| 1.896 | 43.302 | 826.220 | 2.948.457 | 64.941.195 |
| 1.973 | 43.378 | 825.574 | 2.949.205 | 64.969.677 |

**Tabela 08. Número de colisões na tabela hash com 100.000 vetores.**

**Tabela com 200.000 vetores**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **20 mil** | **100 mil** | **500 mil** | **1 milhão** | **5 milhões** |
| **Divisão** | 2.020.800 | 3.546.900 | 31.663.100 | 113.263.700 | 2.034.079.899 |
| 1.921.700 | 3.551.600 | 30.414.800 | 104.253.200 | 1.753.958.299 |
| 2.180.399 | 3.717.800 | 31.410.101 | 102.439.900 | 1.965.357.401 |
| 1.961.701 | 3.447.000 | 28.307.500 | 123.743.799 | 1.963.908.099 |
| 2.027.199 | 3.891.701 | 30.939.599 | 115.217.101 | 2.087.638.100 |
| **Dobramento** | 1.426.900 | 3.251.901 | 28.927.801 | 183.665.301 | 2.740.058.200 |
| 1.702.901 | 3.122.700 | 27.055.701 | 194.754.000 | 2.784.983.901 |
| 1.632.400 | 3.199.900 | 27.850.200 | 201.295.699 | 2.846.237.600 |
| 1.788.200 | 3.037.200 | 27.397.400 | 166.623.500 | 2.757.253.600 |
| 1.374.601 | 3.101.100 | 28.379.900 | 179.912.699 | 2.849.815.200 |
| **Multiplicação** | 3.478.200 | 15.352.300 | 123.923.701 | 272.165.300 | 2.721.942.999 |
| 3.620.901 | 14.850.601 | 125.420.500 | 267.457.200 | 2.850.120.399 |
| 3.787.900 | 14.088.300 | 101.911.399 | 286.476.001 | 2.796.732.100 |
| 3.819.500 | 14.935.701 | 97.176.901 | 312.004.601 | 3.059.472.101 |
| 3.604.300 | 14.716.000 | 127.115.400 | 293.803.001 | 2.817.887.199 |

**Tabela 09. Tempo de inserção em nanosegundos na tabela hash com 200.000 vetores.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **20 mil** | **100 mil** | **500 mil** | **1 milhão** | **5 milhões** |
| **Divisão** | 980 | 23.196 | 470.150 | 1.650.496 | 33.661.004 |
| 989 | 23.209 | 470.713 | 1.650.460 | 33.658.449 |
| 997 | 23.109 | 470.468 | 1.647.937 | 33.644.897 |
| 1.026 | 23.217 | 470.793 | 1.649.064 | 33.662.163 |
| 968 | 23.137 | 471.033 | 1.650.232 | 33.651.811 |
| **Dobramento** | 1.864 | 41.946 | 801.637 | 2.861.653 | 62.866.854 |
| 1.875 | 42.072 | 801.800 | 2.864.014 | 62.892.362 |
| 1.908 | 42.206 | 801.367 | 2.862.247 | 62.850.082 |
| 1.888 | 41.951 | 801.559 | 2.860.310 | 62.871.609 |
| 1.906 | 41.988 | 801.817 | 2.861.902 | 62.841.567 |
| **Multiplicação** | 1.001 | 22.977 | 470.362 | 1.650.725 | 33.663.221 |
| 993 | 23.148 | 470.377 | 1.651.615 | 33.656.947 |
| 933 | 22.984 | 470.752 | 1.648.963 | 33.643.035 |
| 1.013 | 23.078 | 470.938 | 1.649.551 | 33.661.607 |
| 992 | 22.994 | 469.814 | 1.648.310 | 33.647.414 |

**Tabela 10. Número de colisões na tabela hash com 200.000 vetores.**

**Gráfico, Gráfico de linhas

Descrição gerada automaticamente**

**Figura 01. Tempo médio por tamanho de tabela inserção de 20 mil dados.**

**Gráfico, Gráfico de linhas

Descrição gerada automaticamente**

**Figura 02. Tempo médio por tamanho de tabela inserção de 100 mil dados.**

**Gráfico, Gráfico de linhas

Descrição gerada automaticamente**

**Figura 03. Tempo médio por tamanho de tabela inserção de 500 mil dados.**

**Gráfico, Gráfico de linhas

Descrição gerada automaticamente**

**Figura 04. Tempo médio por tamanho de tabela inserção de 1 milhão dados.**

**Gráfico, Gráfico de linhas

Descrição gerada automaticamente**

**Figura 05. Tempo médio por tamanho de tabela inserção de 5 milhões dados.**

**Gráfico, Gráfico de linhas

Descrição gerada automaticamente**

**Figura 06. Número de colisões por tamanho de tabela inserção de 20 mil dados.**

**Gráfico, Gráfico de linhas

Descrição gerada automaticamente**

**Figura 07. Número de colisões por tamanho de tabela inserção de 100 mil dados.**

**Gráfico, Gráfico de linhas

Descrição gerada automaticamente**

**Figura 08. Número de colisões por tamanho de tabela inserção de 500 mil dados.**

**Gráfico, Gráfico de linhas

Descrição gerada automaticamente**

**Figura 09. Número de colisões por tamanho de tabela inserção de 1 milhão dados.**

**Gráfico, Gráfico de linhas

Descrição gerada automaticamente**

**Figura 10. Número de colisões por tamanho de tabela inserção de 5 milhões dados.**

**Busca**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Chave** | | **Divisão** | **Dobramento** | **Multiplicação** |
| **345396281** | **Tempo** | **2600** | **3400** | **2700** |
| **Comparações** | **4** | **21** | **12** |
| **987765423\*** | **Tempo** | **4500** | **12100** | **2500** |
| **Comparações** | **28** | **55** | **20** |
| **117321717** | **Tempo** | **1400** | **1100** | **800** |
| **Comparações** | **0** | **0** | **0** |
| **618692254** | **Tempo** | **4100** | **3100** | **2600** |
| **Comparações** | **14** | **21** | **17** |
| **10899880** | **Tempo** | **1100** | **900** | **1000** |
| **Comparações** | **1** | **2** | **1** |

**Tabela 11. Tabela de tempo em nanosegundos e comparações para valores na tabela hash com 200.000 vetores.**

**\*Valor não encontrado na tabela.**

1. **Conclusão**

Ao inserirmos na tabela hash a função hash teve muita influência, como podemos ver quando usado a função de multiplicação ela chega a ser 3 vezes mais devagar que as demais, contudo o tamanho da tabela tem ainda maior influência, já que ao passarmos de 100000 vetores a velocidade de inserção é similar.

Quanto ao número de colisões, independente da função hash escolhida, tem-se um valor parecido na hora da inserção, o que podemos dizer é que o número de colisões é inversamente proporcional ao número de vetores, quando dobramos o número de vetores diminuem-se pela metade o número de colisões.

Quanto ao processo de busca o tempo foi similar nas situações onde o valor foi encontrado, já quando não foi encontrado o dobramento teve uma performance menor, o que já era esperado pois há uma concentração dos valores quando feito o dobramento.