Implementação de Tabela de Hash em Java

Joel Sepulveda Martins[1]**,Victor Portelinha, Lucas Geremias**[1]

1

Abstract. *https://github.com/JoelSepulvedaMartins/-Struct-Data-/tree/Version2/HashTable*
Este artigo descreve a implementação de uma tabela de hash em Java com foco em várias técnicas de hash como por exemplo a de dobramento. A tabela de hash é uma estrutura de dados eficiente para armazenar e recuperar dados com base em chaves. Discutiremos a criação da tabela, a função de hash por dobramento e as operações de inserção e remoção de elementos na tabela.

1. Introdução

Este artigo explora várias funções de hash implementadas em uma estrutura de dados de tabela de hash. As funções de hash desempenham um papel fundamental na distribuição eficiente de elementos na tabela de hash. Vamos analisar diferentes funções de hash e seus impactos nas colisões.

2. Funções de Hash

A classe `HashTable` implementa várias funções de hash para calcular o índice de uma chave na tabela de hash. A escolha da função de hash afeta diretamente o desempenho da tabela. Vamos discutir cada função implementada.

2.1. Hash por Módulo

A função `hashModTam` calcula o índice da tabela de hash através do operador de módulo. Ela calcula `key % tamanho` e lida com valores negativos, garantindo que o índice seja sempre não negativo.

[language=Java] private int hashModTam(int key) int hash = key if (hash ¡ 0) hash += this.tamanho; return hash;

2.2. Hash por Multiplicação do Fator

A função `hashFactor` utiliza a multiplicação da chave pelo fator `FactorHash` escolhido. Ela ajusta a parte fracionária do resultado multiplicando pelo tamanho da tabela.

[language=Java] private int hashFactor(int key, double Factor) this.FactorHash = Factor; double hashValue = key * FactorHash; int intPart = (int) hashValue; hashValue -= intPart; hashValue *= this.tamanho; if (hashValue ¡ 0) hashValue = -hashValue; tempHash = (int) hashValue; return tempHash

2.3. Hash por Dobramento de Dígitos

A função `digitHashFolding` realiza o dobramento dos dígitos da chave, somando os dígitos sequencialmente. Isso é feito dividindo a chave em partes menores e somando-as.

[language=Java] private int digitHashFolding(int key) int hashValue = 0; int keyCopy = key; while (keyCopy ¿ 0) int digit = keyCopy hashValue += digit; keyCopy /= 10; return hashValue

2.4. Hash por Remoção dos 3 Últimos Dígitos

A função `hashFoldingLast3` remove os 3 últimos dígitos da chave, somando-os para calcular o índice da tabela de hash.

[language=Java] private int hashFoldingLast3(int key) int hashValue = 0; while (key ¿ 0) int digit = key hashValue += digit; key /= 1000; return hashValue

2.5. Hash por Bits Significativos

A função `significantBitHash` utiliza os bits mais significativos da chave para calcular o índice da tabela de hash. Isso ajuda a reduzir colisões ao considerar apenas os bits importantes.

[language=Java] private int significantBitHash(int key) int hashValue = 0; int mask = 0x7FFFFFFF; while (key ¿ 0) int digitGroup = key mask; hashValue $^{=}digitGroup; key >>= 7; if(hashValue < 0)hashValue = -hashValue; returnhashValue$

3. Conclusão

As funções de hash desempenham um papel fundamental na eficiência de uma tabela de hash. A escolha da função de hash certa pode minimizar colisões e melhorar o desempenho. Este artigo explorou várias funções de hash implementadas na classe `HashTable` e suas características. A compreensão dessas funções é essencial ao trabalhar com tabelas de hash em aplicações do mundo real.

4. Destarte

Uma tabela de hash é uma estrutura de dados que permite armazenar e recuperar informações com base em chaves. A técnica de hash por dobramento é uma abordagem para calcular o índice de uma chave na tabela. Neste artigo, abordaremos a implementação de uma tabela de hash em Java, com foco na técnica de hash por dobramento.

5. Implementação da Tabela de Hash

Nossa implementação da tabela de hash inclui as seguintes características:

- Tamanho da tabela configurável.
- Tratamento de colisões usando listas encadeadas.
- Função de hash por dobramento.
- Operações de inserção e remoção de elementos.

6. Função de Hash por Dobramento

A função de hash por dobramento consiste em somar os dígitos de uma chave após dividi-la em partes menores. A implementação em Java é a seguinte:

```
// Código Java da função de hash por dobramento
```

Nesta função, a chave é dividida em dígitos, somados e usados como índice na tabela de hash.

7. Operações da Tabela de Hash

Nossa implementação inclui as seguintes operações:

- ‘addElement(int data)‘: Adiciona um elemento à tabela de hash.
- ‘removeElement(int data)‘: Remove um elemento da tabela de hash.

Por exemplo, a operação ‘addElement‘ é implementada da seguinte forma:

```
// Código Java da operação addElement
```

Esta operação calcula o índice com base na chave usando a função de hash e trata colisões usando listas encadeadas.

8. Conclusão

A implementação de tabelas de hash em Java com hash por dobramento é uma técnica eficaz para armazenar e recuperar dados com base em chaves. Neste artigo, abordamos a criação da tabela, a função de hash e as operações de inserção e remoção de elementos. Essa estrutura de dados é amplamente utilizada em aplicativos que requerem rápida recuperação de informações com base em chaves.

Rehashing em Tabelas de Hash November 4, 2023

9. Introdução

Este artigo aborda o conceito de rehashing em tabelas de hash e como ele é implementado em uma classe de tabela de hash. O rehashing é uma técnica importante para garantir o desempenho de uma tabela de hash à medida que ela se enche. Exploraremos como a classe `HashTable` implementa o rehashing em seu código.

10. Rehashing na Classe `HashTable`

A classe `HashTable` implementa o rehashing como uma estratégia para lidar com colisões e manter um fator de carga aceitável. Quando o fator de carga da tabela de hash ultrapassa um limite definido, o rehashing é acionado.

10.1. Fator de Carga e Limite

O fator de carga de uma tabela de hash é calculado como a razão entre o número de elementos armazenados na tabela e o tamanho total da tabela. O limite de fator de carga (`loadFactorThreshold`) é um valor predefinido que determina quando o rehashing deve ocorrer. Se o fator de carga exceder esse limite, o rehashing será acionado.

10.2. Implementação do Rehashing

Quando o rehashing é acionado, a classe `HashTable` realiza as seguintes etapas:

1. Calcula um novo índice para o elemento usando uma função de hash alternativa. No exemplo dado, a função `significantBitHash` é usada como função de rehashing.
   [language=Java] posicao = significantBitHash(data);

2. Marca o elemento como ”rehashed” para evitar que ele seja movido novamente durante o mesmo processo de rehashing.
   [language=Java] this.tabela[posicao].setRehashed(true);
3. Incrementa o contador de rehashings (`numRehashes`) para rastrear o número de rehashings adicionais realizados.
   [language=Java] this.numRehashes++;
4. Chama a função `addElement` para inserir o elemento na nova posição após o rehashing.
   [language=Java] addElement(data);
5. Zera o contador de rehashings (`numRehashes`) para evitar rehashings em cascata.
   [language=Java] this.numRehashes = 0;

Essa estratégia de rehashing permite que a tabela de hash cresça dinamicamente e evite colisões excessivas à medida que mais elementos são inseridos.

11. Conclusão

O rehashing é uma técnica fundamental para garantir a eficiência de tabelas de hash à medida que elas são preenchidas com elementos. A classe `HashTable` implementa o rehashing como uma estratégia para controlar o fator de carga e evitar colisões. Compreender o funcionamento do rehashing é crucial ao trabalhar com tabelas de hash em aplicações do mundo real.

Implementação

November 4, 2023

12. Introdução

Este artigo aborda o conceito de rehashing em tabelas de hash e como ele é implementado em uma classe de tabela de hash. O rehashing é uma técnica importante para garantir o desempenho de uma tabela de hash à medida que ela se enche. Exploraremos como a classe `HashTable` implementa o rehashing em seu código.

13. Rehashing na Classe `HashTable`

A classe `HashTable` implementa o rehashing como uma estratégia para lidar com colisões e manter um fator de carga aceitável. Quando o fator de carga da tabela de hash ultrapassa um limite definido, o rehashing é acionado.

13.1. Fator de Carga e Limite

O fator de carga de uma tabela de hash é calculado como a razão entre o número de elementos armazenados na tabela e o tamanho total da tabela. O limite de fator de carga (`loadFactorThreshold`) é um valor predefinido que determina quando o rehashing deve ocorrer. Se o fator de carga exceder esse limite, o rehashing será acionado.

13.2. Implementação do Rehashing

Quando o rehashing é acionado, a classe `HashTable` realiza as seguintes etapas:

1. Calcula um novo índice para o elemento usando uma função de hash alternativa. No exemplo dado, a função `significantBitHash` é usada como função de rehashing.
   [language=Java] posicao = significantBitHash(data);
2. Marca o elemento como ”rehashed” para evitar que ele seja movido novamente durante o mesmo processo de rehashing.
   [language=Java] this.tabela[posicao].setRehashed(true);
3. Incrementa o contador de rehashings (`numRehashes`) para rastrear o número de rehashings adicionais realizados.
   [language=Java] this.numRehashes++;
4. Chama a função `addElement` para inserir o elemento na nova posição após o rehashing.
   [language=Java] addElement(data);
5. Zera o contador de rehashings (`numRehashes`) para evitar rehashings em cascata.
   [language=Java] this.numRehashes = 0;

Essa estratégia de rehashing permite que a tabela de hash cresça dinamicamente e evite colisões excessivas à medida que mais elementos são inseridos.

14. Conclusão

O rehashing é uma técnica fundamental para garantir a eficiência de tabelas de hash à medida que elas são preenchidas com elementos. A classe `HashTable` implementa o rehashing como uma estratégia para controlar o fator de carga e evitar colisões. Compreender o funcionamento do rehashing é crucial ao trabalhar com tabelas de hash em aplicações do mundo real.

15. PoweBY Graphics

References

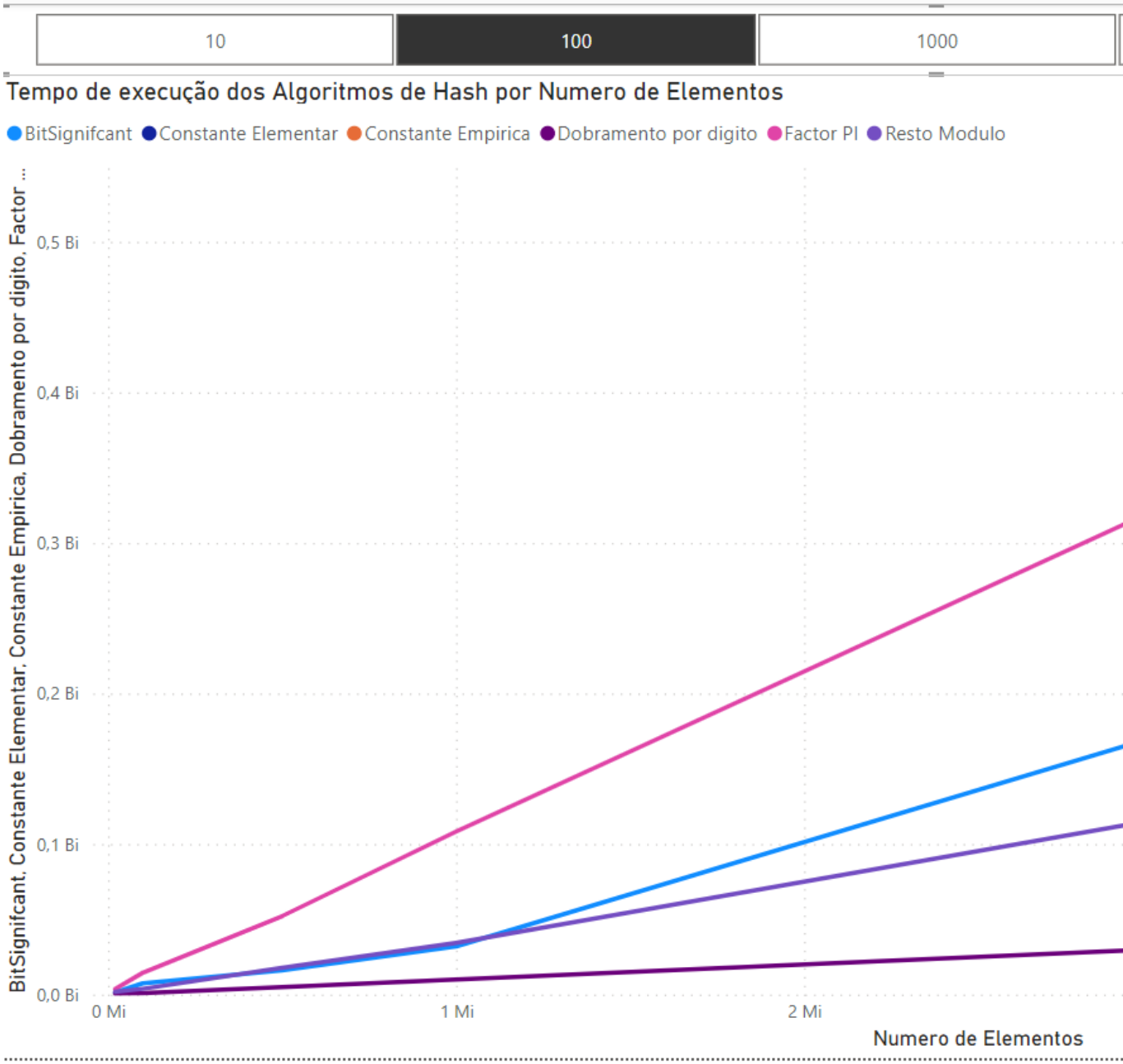

Figure 1. Relatorio

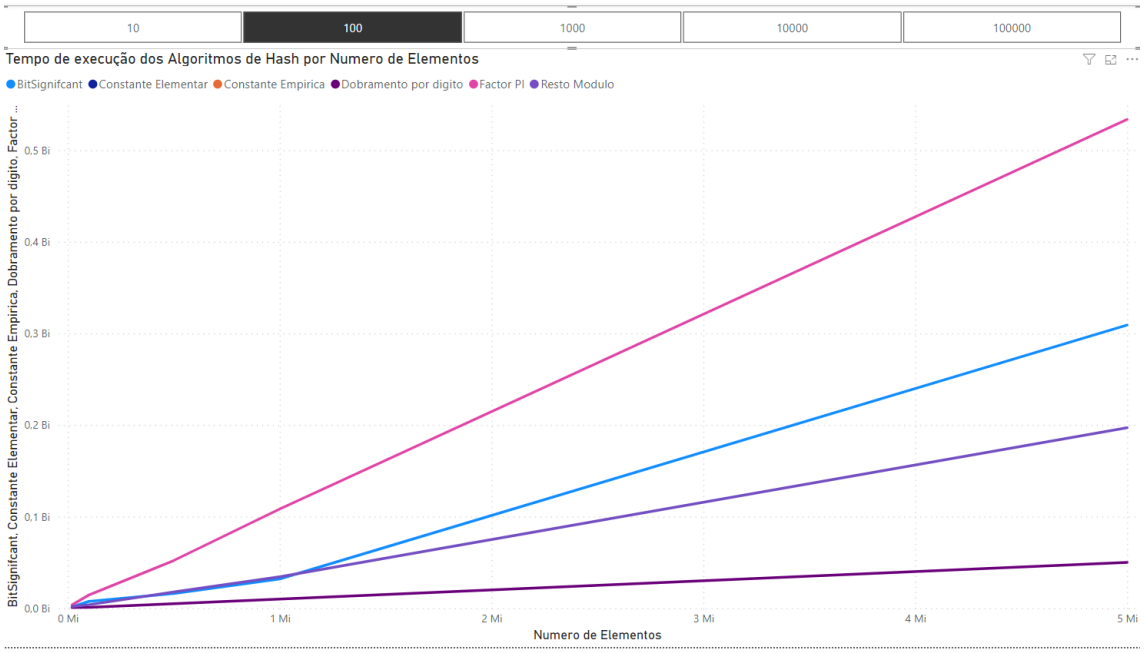

Figure 2. Enter Relatorio

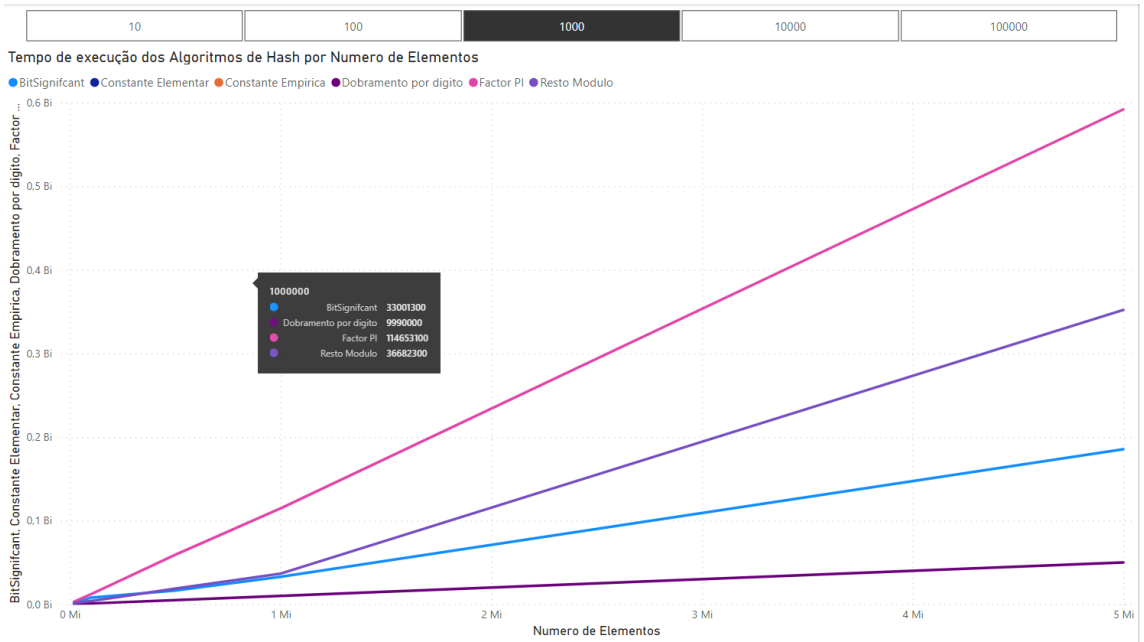

Figure 3. Enter Relatorio

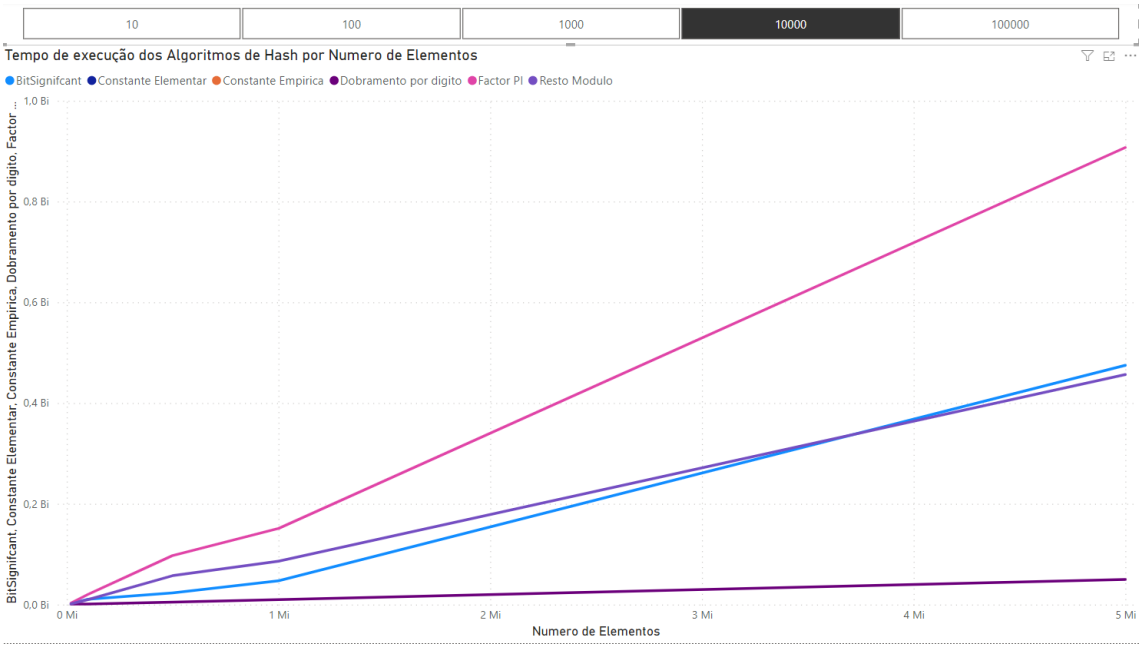

Figure 4. Enter Relatorio

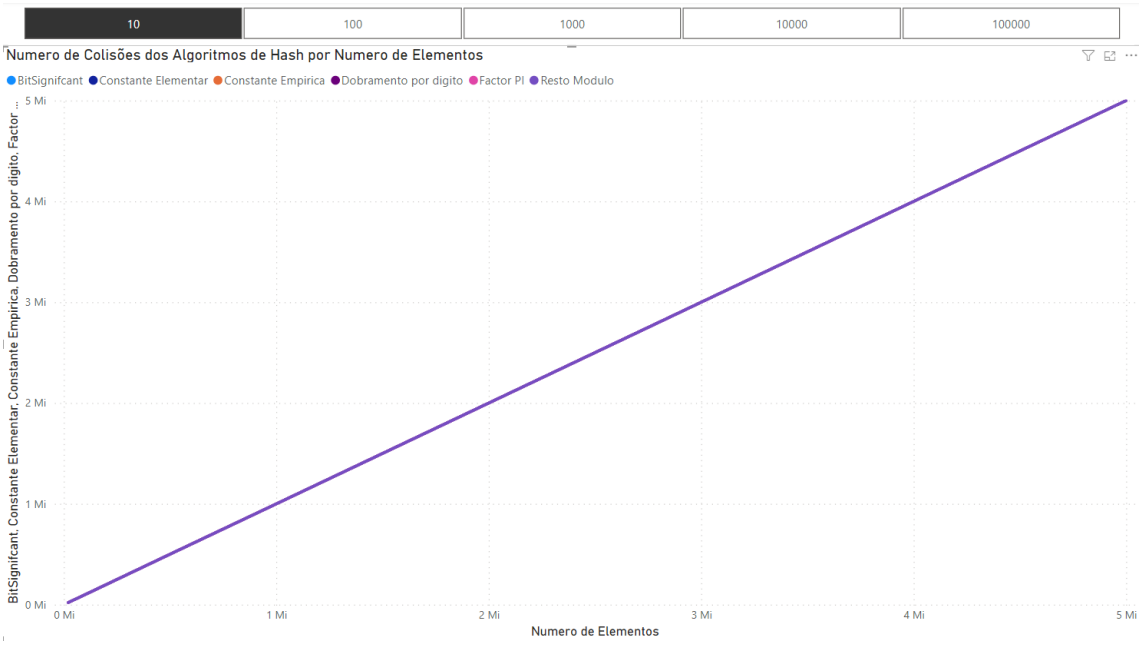

Figure 5. Enter Relatorio

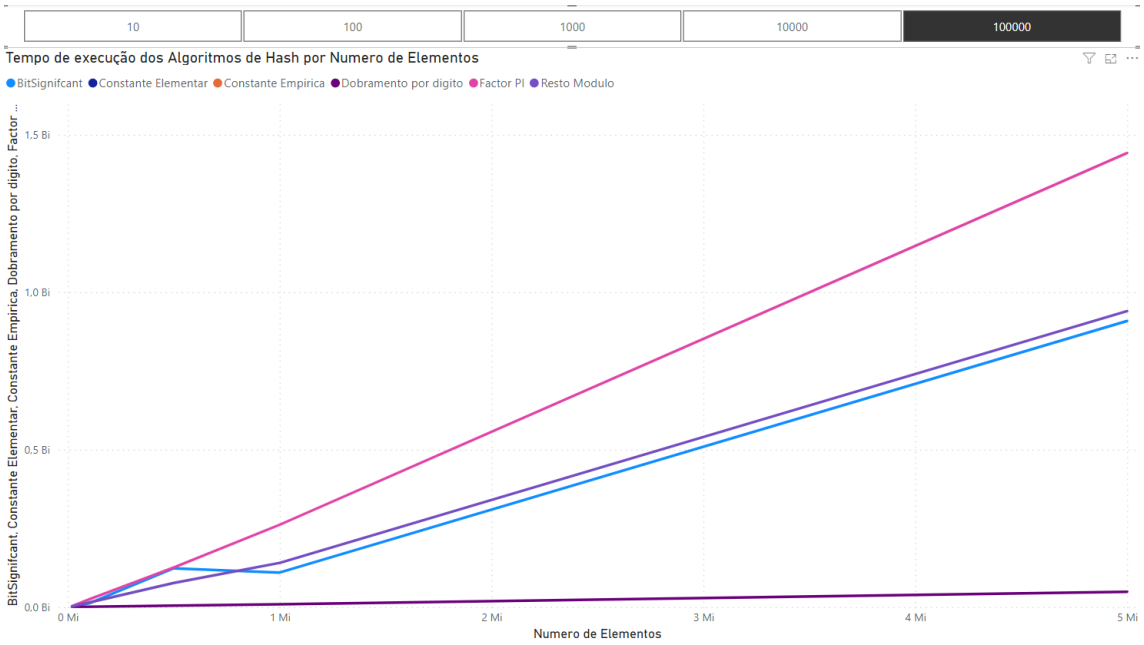

Figure 6. Enter Relatorio

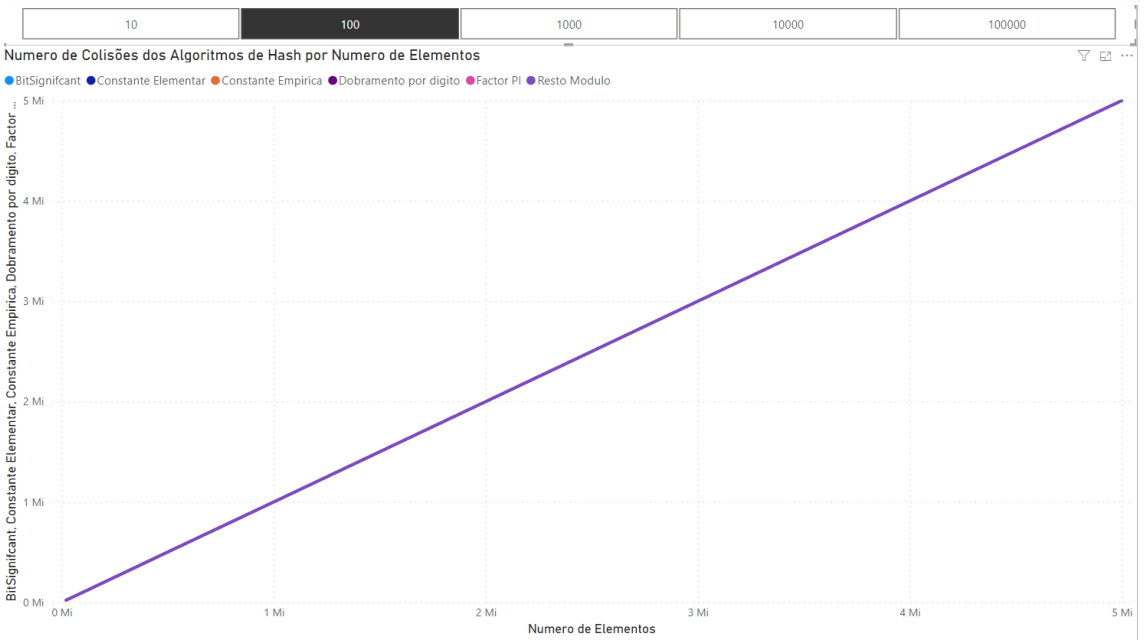

Figure 7. Enter Relatorio

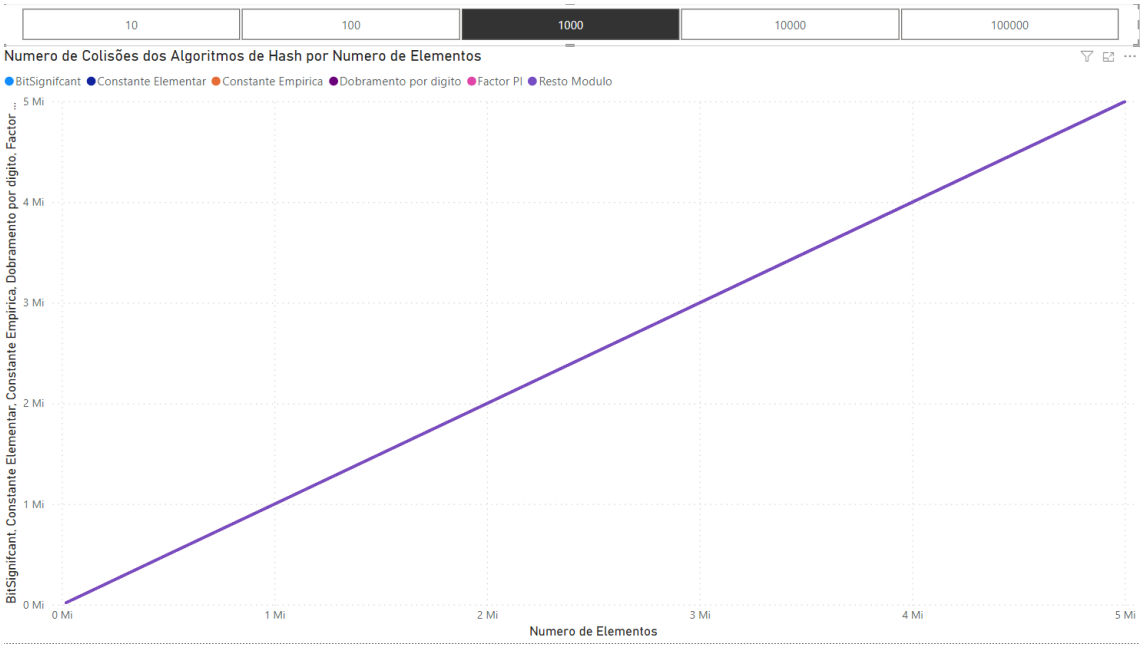

Figure 8. Enter Relatorio

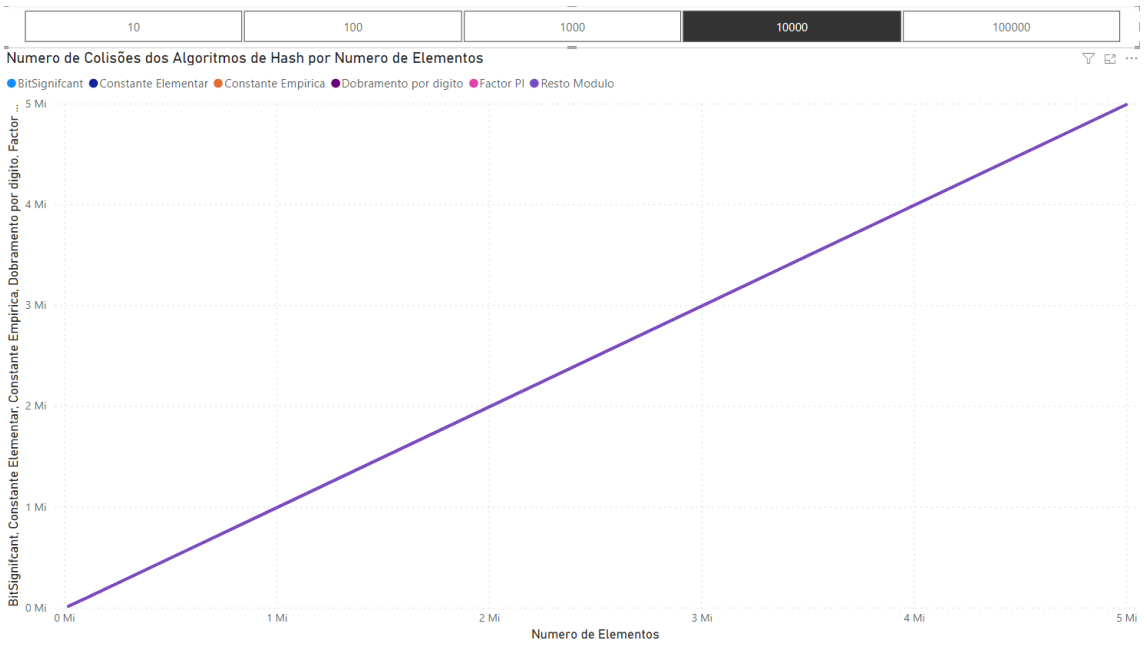

Figure 9. Enter Relatorio

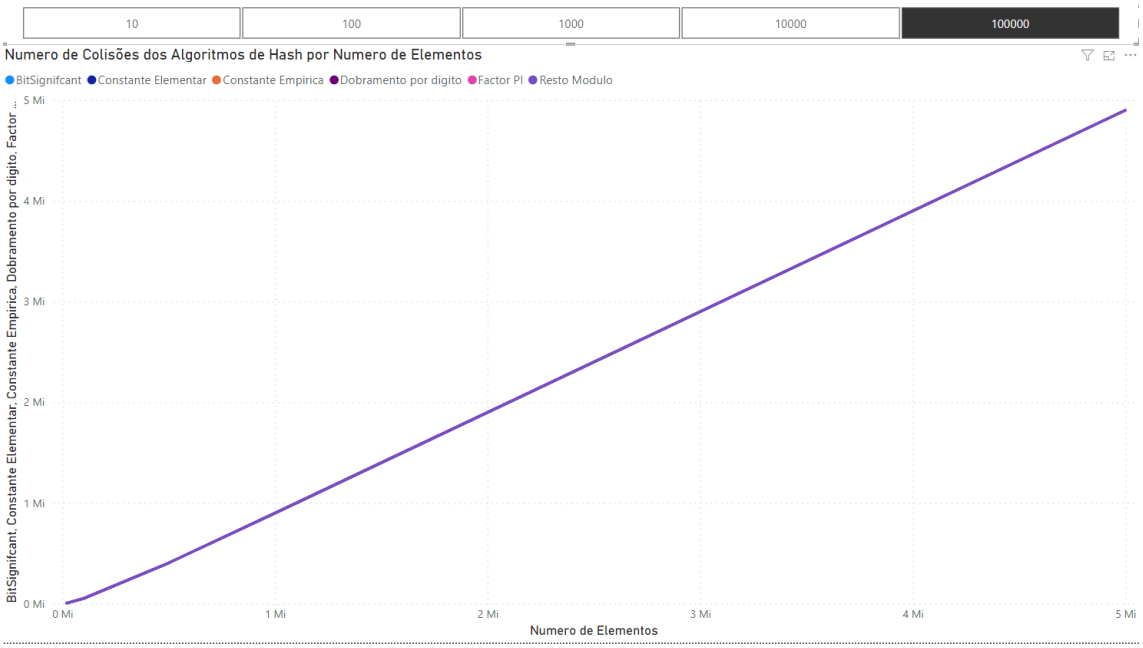

Figure 10. Enter Relatorio