Figure 1: Enter Caption

Relatório tabela Hash

Igor Terplak Gutierrez e Patrick Froes

Outubro de 2024

Contents

1 Introdução

Neste projeto, foram implementadas três tabelas hash com diferentes funções de hashing e métodos de tratamento de colisões. A implementação foi realizada em Java, e cada tabela foi projetada para armazenar um grande número de registros de maneira eficiente, minimizando o tempo de busca e inserção. A análise incluiu a comparação do desempenho de cada abordagem de hashing e tratamento de colisões.

2 Estrutura e Implementação

As tabelas hash foram construídas com diferentes tamanhos para observar o impacto do dimensionamento na eficiência da distribuição de elementos e na ocorrência de colisões.

Tabela	**Tamanho**
Tabela 1	1.000
Tabela 2	10.000
Tabela 3	100.000

2.1 Funções de Hashing

As funções de hashing utilizadas nas tabelas hash são as seguintes:

- **Hash por Divisão**: `hashIndex = chave % tamanhoTabela`
- **Hash por Multiplicação**: utiliza a constante $A = 0.6180339887$, conhecida como razão áurea, para dispersar melhor os índices.
- **Hash com Mistura de Bits**: utiliza operações de XOR e deslocamentos de bits para embaralhar os bits da chave, seguido por uma operação de módulo.

2.2 Função de buscar

A função buscar é responsável por localizar um elemento específico em uma tabela hash de acordo com seu identificador (id). Ela percorre a tabela hash, contando as comparações realizadas durante o processo e informando se a chave foi encontrada ou não.

2.3 Função de inserir

A função inserir insere um número especificado de chaves aleatórias na tabela hash, distribuindo-as com base na função de hash e no método de tratamento de colisões selecionados.

2.4 Função H

A função h calcula o índice da chave com base no tipo de função de hash escolhida pelo usuário. Ela retorna um valor inteiro entre 0 e tamanhoTabela - 1, indicando a posição na tabela hash onde a chave será inserida. Cada método de hash tem uma abordagem distinta para manipular a chave, ajudando a distribuir os elementos.

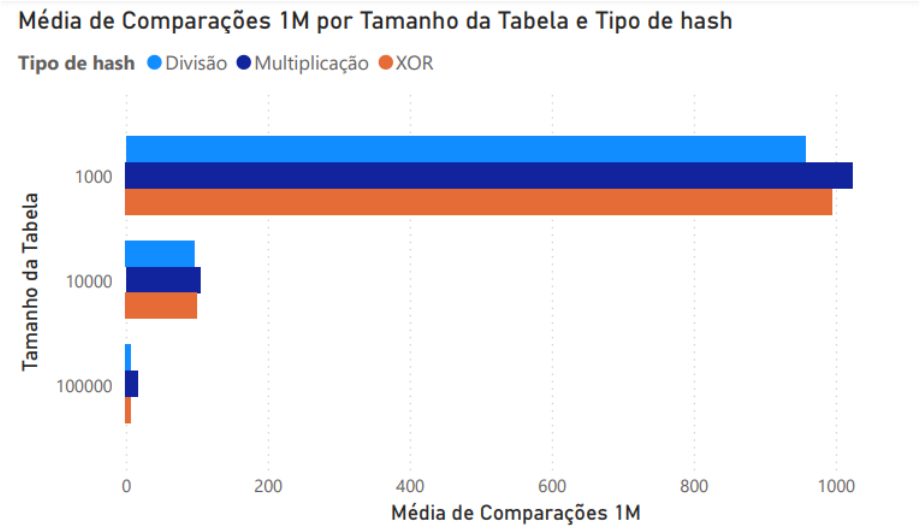
2.5 Métodos de Tratamento de Colisões

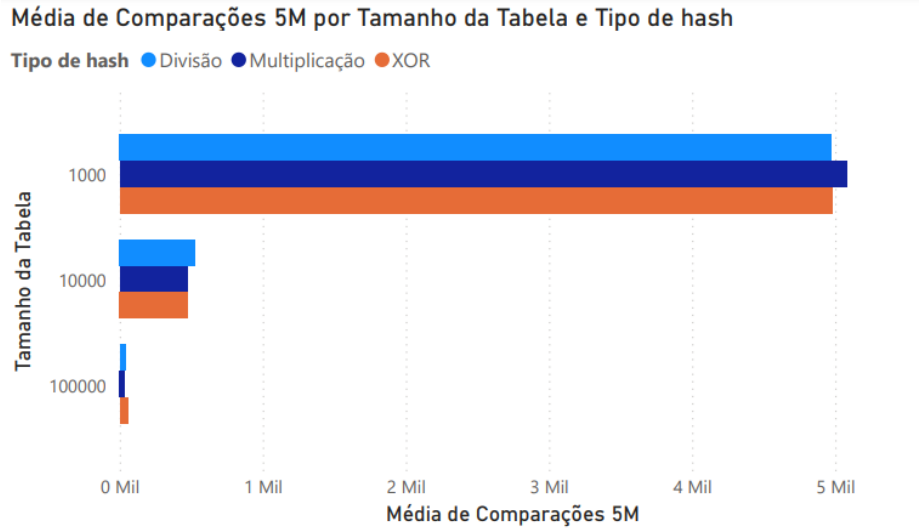
Foram implementados dois métodos de tratamento de colisões:

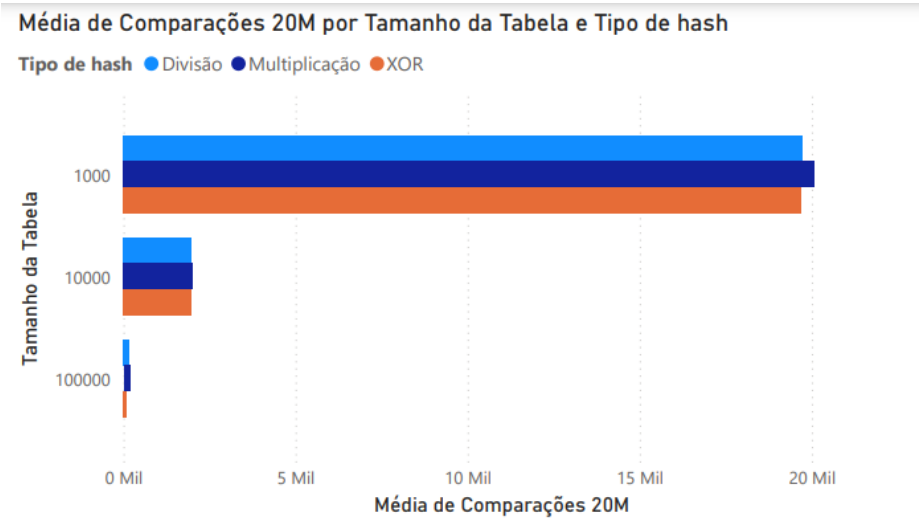
- **Encadeamento**: os elementos que colidem para o mesmo índice da tabela são armazenados em uma lista encadeada.
- **Rehashing Linear**: no caso de colisão, uma nova posição é calculada, incrementando linearmente o índice original até que uma posição vazia seja encontrada.

3 Resultados dos Testes

3.1 Média de Comparações

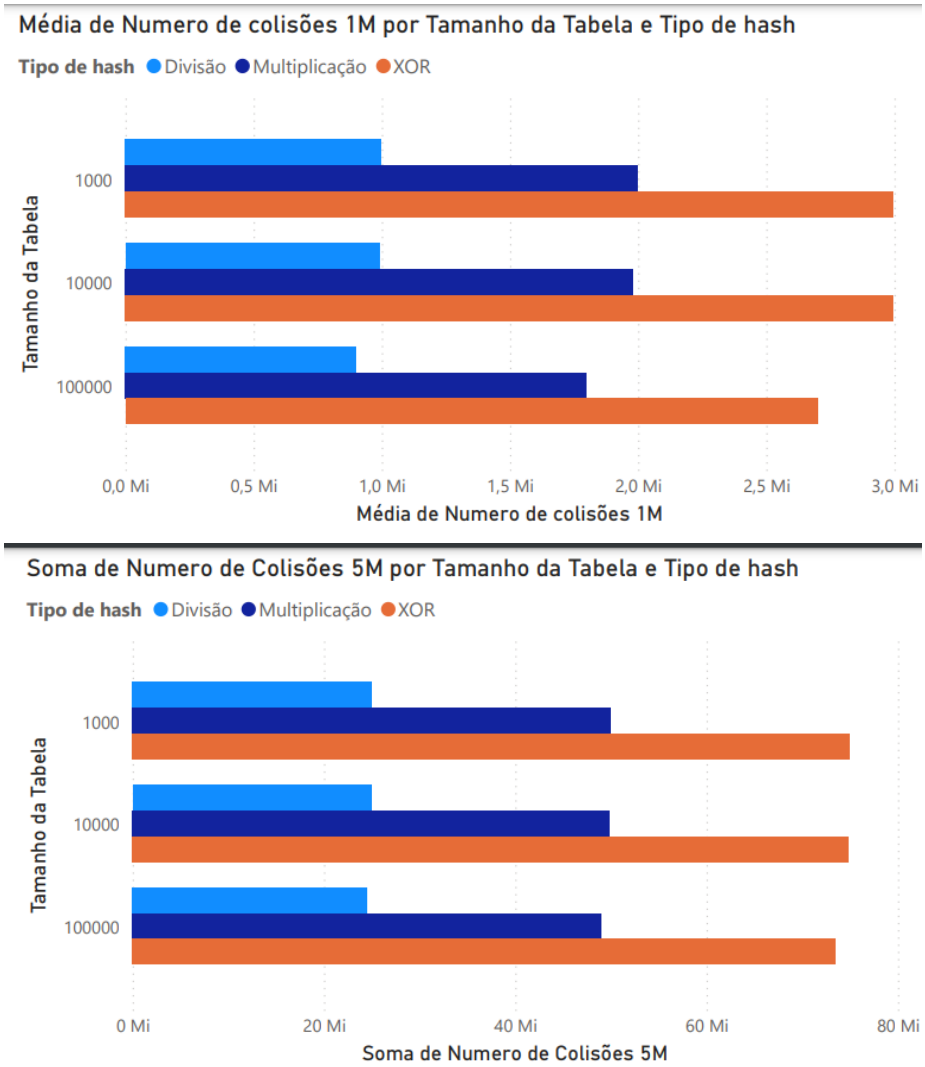


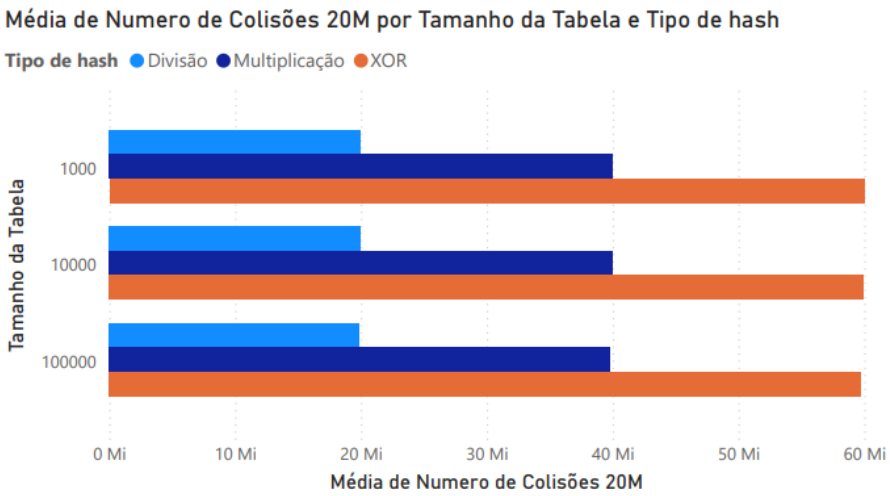




Tipo de hash	Tamanho da Tabela	Média de Comparações 1M	Média de Comparações 5M	Média de Comparações 20M
Divisão	1000	957,00	4977,00	19724,00
Divisão	10000	97,00	529,00	1990,00
Divisão	100000	7,00	40,00	174,00
Multiplicação	1000	1024,00	5080,00	20079,00
Multiplicação	10000	105,00	475,00	2032,00
Multiplicação	100000	17,00	34,00	192,00
XOR	1000	995,00	4978,00	19703,00
XOR	10000	100,00	479,00	1963,00
XOR	100000	7,00	57,00	85,00

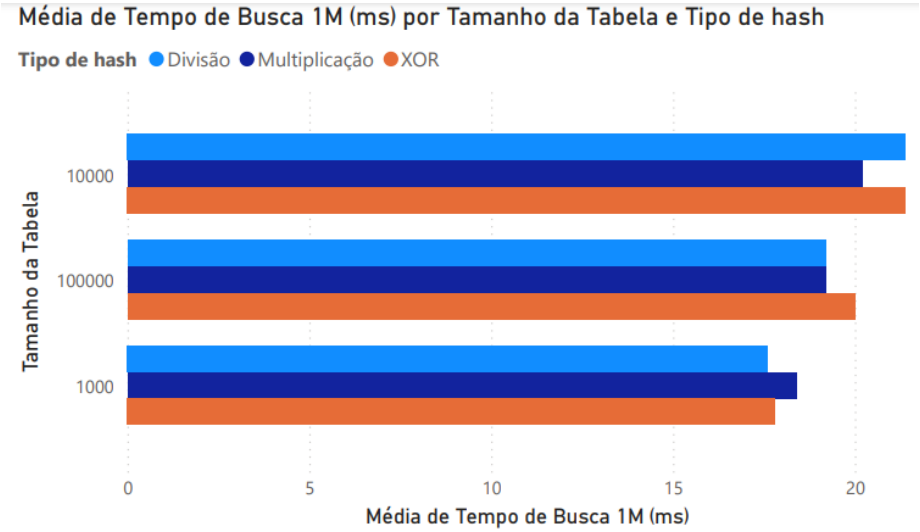
3.2 Média de Numero de colisões

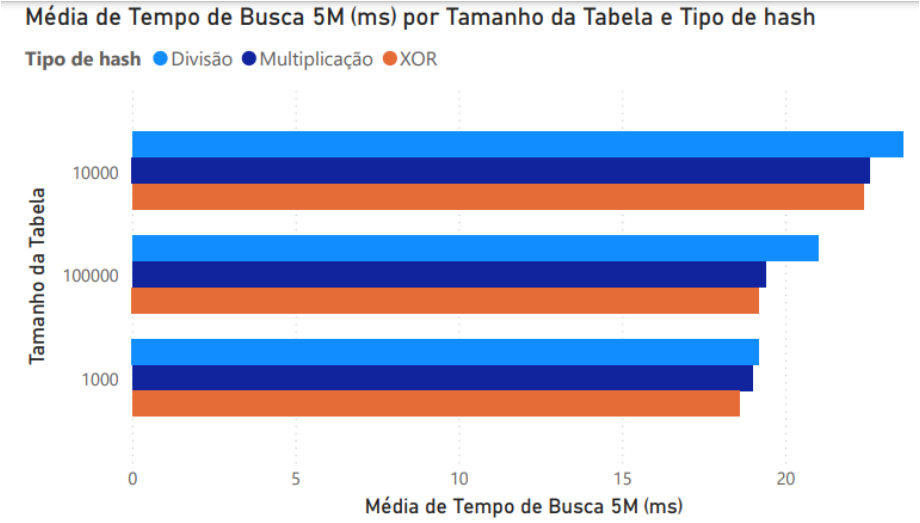


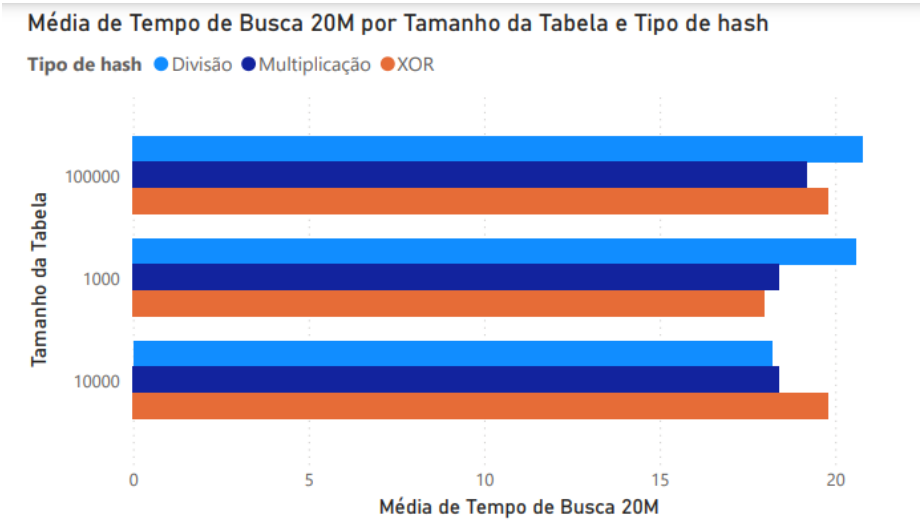


Tipo de hash	Tamanho da Tabela	Média de Numero de colisões 1M	Média de Numero de Colisões 5M	Média de Numero de Colisões 20M
Divisão	1000	999000,00	4999000,00	19999000,00
Divisão	10000	990000,00	4990000,00	19990000,00
Divisão	100000	900006,00	4900000,00	19900000,00
Multiplicação	1000	1998000,00	9998000,00	39998000,00
Multiplicação	10000	1980000,00	9980000,00	39980000,00
Multiplicação	100000	1800007,00	9800000,00	39800000,00
XOR	1000	2997000,00	14997000,00	59997000,00
XOR	10000	2997000,00	14970000,00	59970000,00
XOR	100000	2700016,00	14700000,00	59700000,00

3.3 Média de Tempo de Busca

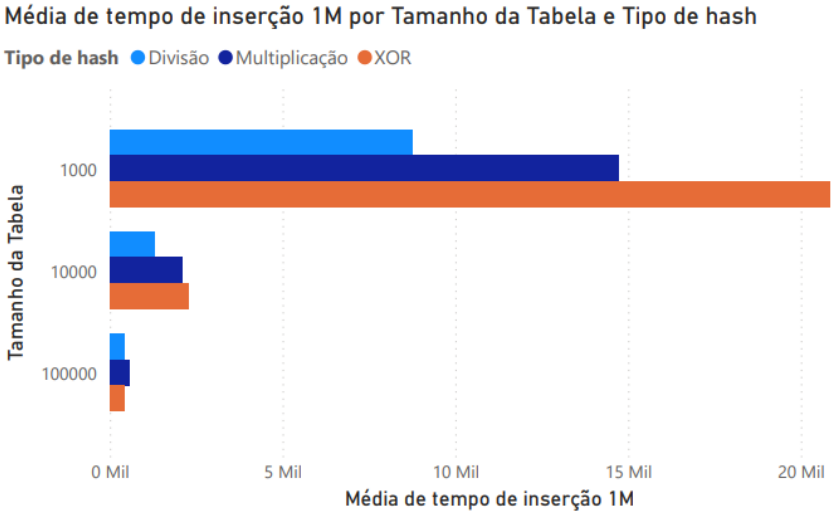


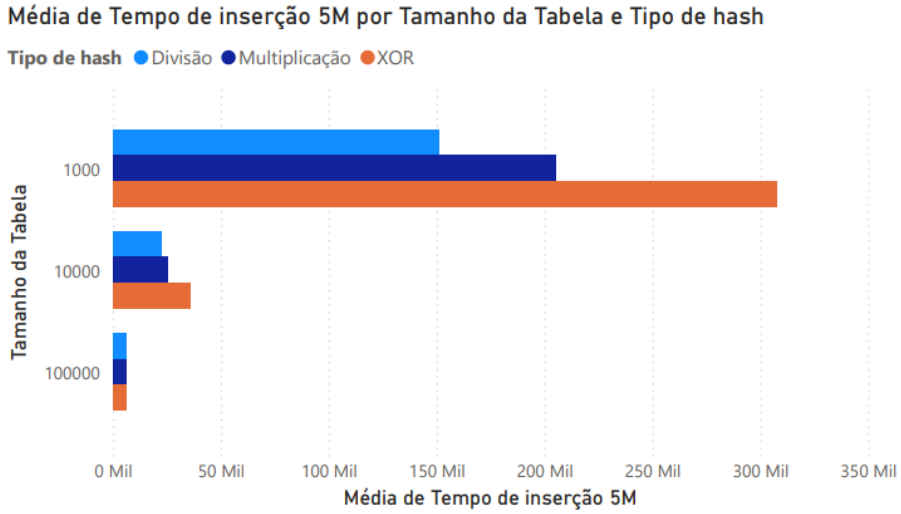


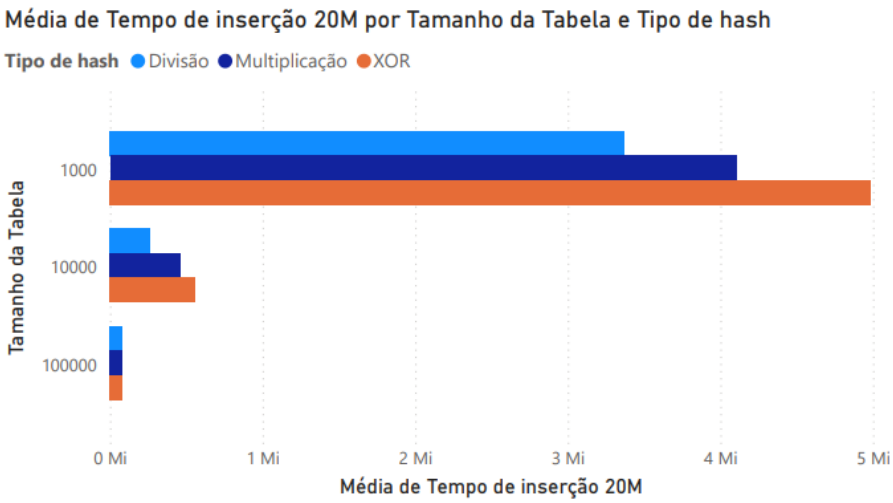


Tipo de hash	Tamanho da Tabela	Média de Tempo de Busca 1M	Média de Tempo de Busca 5M	Média de Tempo de Busca 20M
Divisão	1000	17,60	19,20	20,60
Divisão	10000	21,40	23,60	18,20
Divisão	100000	19,20	21,00	20,80
Multiplicação	1000	18,40	19,00	18,40
Multiplicação	10000	20,20	22,60	18,40
Multiplicação	100000	19,20	19,40	19,20
XOR	1000	17,80	18,60	18,00
XOR	10000	21,40	22,40	19,80
XOR	100000	20,00	19,20	19,80

3.4 Média de tempo de inserção

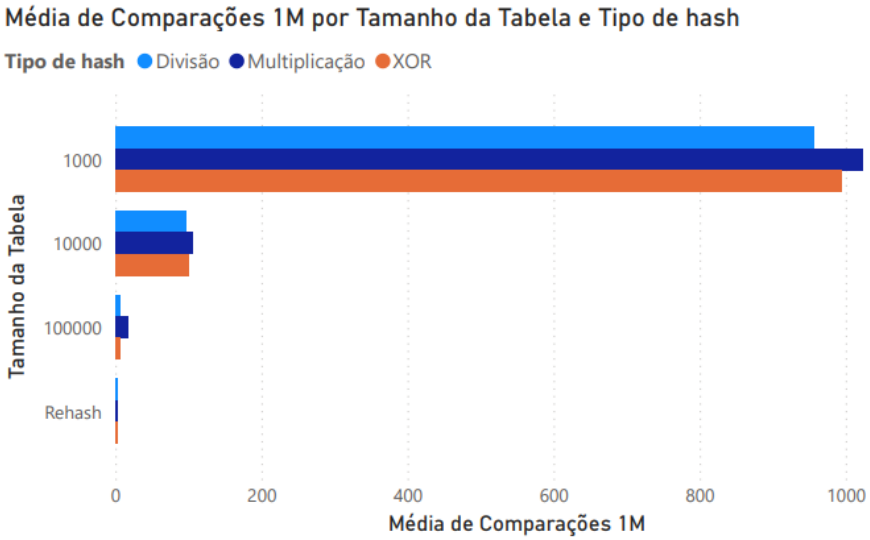


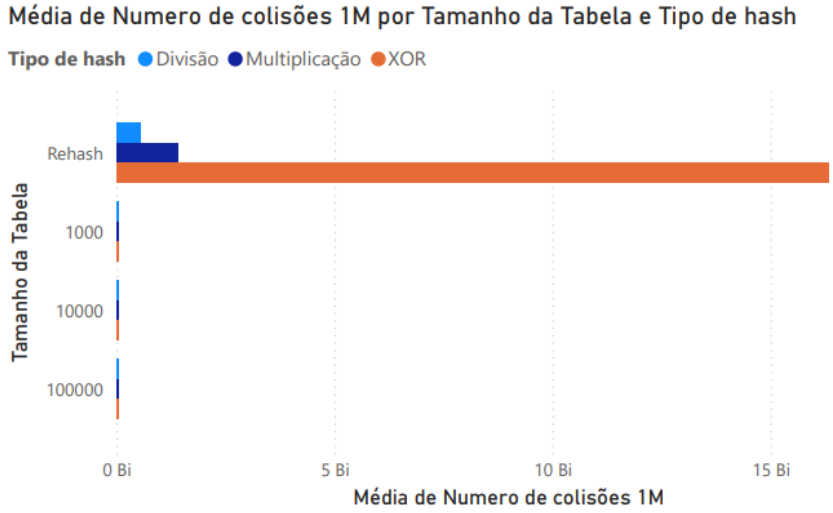


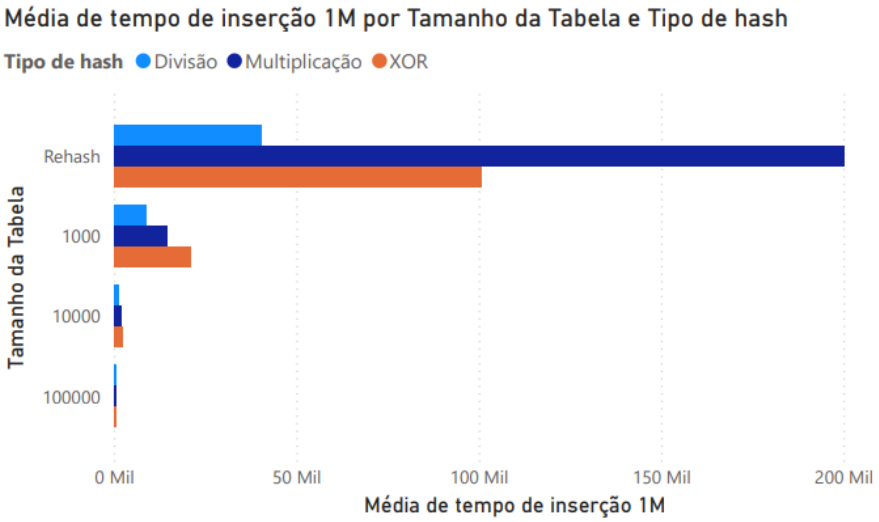


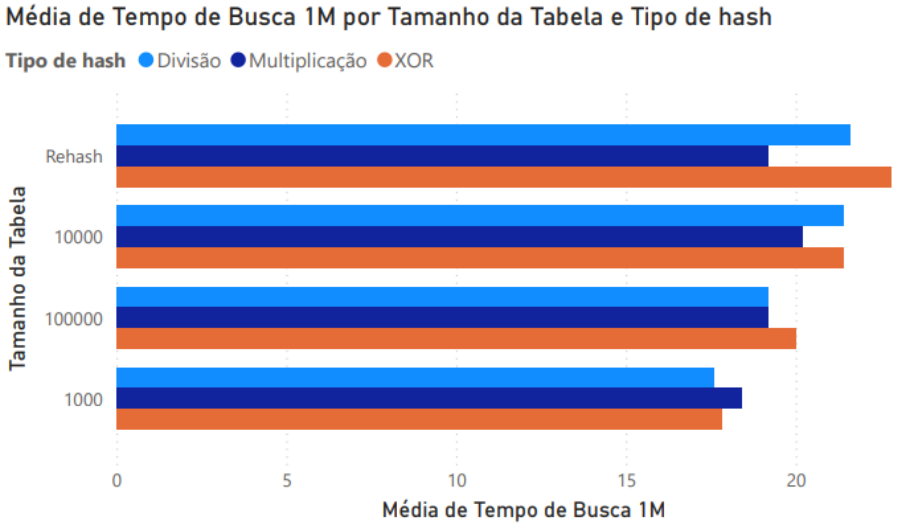
Tipo de hash	Tamanho da Tabela	Média de tempo de inserção 1M	Média de Tempo de inserção 5M	Média de Tempo de inserção 20M
Divisão	1000	8751,40	151076,60	3370597,60
Divisão	10000	1280,20	22475,00	261915,80
Divisão	100000	410,20	5982,40	80393,00
Multiplicação	1000	14708,80	205048,80	4105016,60
Multiplicação	10000	2076,40	25539,40	462707,40
Multiplicação	100000	547,60	6074,80	83983,60
XOR	1000	20847,00	307726,00	4984099,40
XOR	10000	2266,80	35558,20	556270,00
XOR	100000	401,60	6088,80	81585,20

3.5 comparação de rehash









4 Análise de Desempenho

Nesta seção, analisaremos o desempenho de diferentes métodos de hash: Hash por Divisão, Hash por Multiplicação e Hash por XOR. Avaliaremos a eficiência desses métodos em termos de colisões e distribuição de chaves.

4.1 Comparação entre Métodos de Hash

- **Hash por Divisão:** é simples, mas pode ter mais colisões

- **Hash por Multiplicação:** é mais eficaz para distribuir chaves, mas um pouco mais complicado.
- **Hash por XOR:** oferece a melhor dispersão ao embaralhar as chaves, mas requer uma compreensão mais técnica.

4.2 Comparação entre Métodos de Tratamento de Colisões

Rehashing Simples é mais suscetível a problemas de desempenho em tabelas hash muito carregadas, enquanto Listas Encadeadas oferecem uma maneira mais robusta e flexível de lidar com colisões, embora com um custo de memória maior.

5 Conclusão

Neste projeto, fizemos as implementações de tabelas hash utilizando diferentes funções de hashing e métodos de tratamento de colisões. As análises realizadas mostraram que a escolha da função de hash e do método de tratamento de colisões tem um impacto significativo no desempenho da tabela hash.

Os resultados obtidos, tanto em termos de média de comparações quanto em tempo de busca e inserção, confirmaram a importância de uma implementação cuidadosa e a escolha cautelosa das técnicas de hashing e tratamento de colisões. Esses fatores são importantes para a eficiência de aplicações que requerem operações rápidas em grandes volumes de dados.