Relatório do projeto da disciplina EDPA de 2017-2

Caio César[1], Michelle Oliveira[1], Yuri M. Dias[1], Welington G. Rodrigues[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)

{caiyo.projeto,mchrisso,yurimathe.yp,galvao.eu}@gmail.com

Resumo. Tabelas Hash são funções conhecidas e eficientes para guardar elementos com uma grande dispersão de valores. Neste trabalho, uma implementação de tabela hash com endereçamento aberto e sondagem linear e quadrática foi feita, com o intuito de demonstrar na prática a eficiência e os problemas das tabelas, comparando as duas estruturas.

1. Introdução

Hashing, também chamada de tabela hash ou tabela de dispersão, é uma forma eficiente de implementar dicionários, que pode ser utilizada na maioria dos casos quando a performance deve ser superior a outras estruturas como arrays, listas ligadas e árvores binárias balanceadas. Essa estrutura oferece a possibilidade de que as operações tenham tempo de busca amortizado de $\mathcal{O}(1)$, já no pior caso as operações podem chegar a $\mathcal{O}(n)$ [Sedgewick and Wayne 2011].

Quando o número de chaves armazenadas é baixo com relação ao número total de chaves possíveis, essa estrutura se torna uma boa escolha. O índice do arranjo é computado para encontrar sua respectiva posição, portanto não temos um acesso direto ao elemento, porém é utilizada uma função para atribuir o índice ao elemento. Uma boa fórmula ou função hash para calcular o índice é aquela na qual cada chave inserida tem a mesma probabilidade de passar por quaisquer das x posições, independentemente de qualquer outra chave que ocupa a tabela [Cormen 2012].

Neste trabalho, foram realizados alguns testes sobre uma implementação de uma tabela hash com endereçamento aberto, com o objetivo de verificar duas abordagens de sondagem: a sondagem linear, que é a busca sequencial após uma colisão; e a sondagem quadrática, que utiliza de uma função crescente quadrática, para descobrir a próxima posição que deve ser verificada após o conflito.

2. Metodologia

Para realizar a implementação do Hashing, foi utilizada a linguagem C++ no desenvolvimento do projeto, sendo que para cada valor de n, as seguintes métricas foram coletadas:

- Fator de carga;
- Tamanho real da tabela;
- Tempo total de execução das inserções, em millisegundos;
- Total de colisões na estrutura;
- Média de colisões por cada elemento;
- Número de elementos gerados repetidos;
- Número de comparações total na estrutura;

- Média de comparações por elemento na estrutura;
- Número de falhas de inserções;

O tamanho da tabela foi calculado com base no fator de carga, que seria o fator de carga almejado e no número de n. Além disso, foi executado o mesmo teste utilizando o próximo número primo depois do tamanho calulado.

Isso iria gerar uma diferença no fator de carga real no final das inserções, que também foi registrada pelo programa, porém essa diferença era geralmente muito pequena para um n grande, não influenciando muito no fator de carga final da tabela.

2.1. Geração dos elementos de entrada

Para gerar os n elementos de entrada, foi utilizada a biblioteca padrão do C++, sendo que os números gerados pela entrada foram de 0 até *INT64_MAX*, que teria o máximo de 9223372036854775807, sendo de 64 bits. A versão de 64 bits foi escolhida sobre a de 32 por gerar uma maior dispersão dos elementos, pois com números menores (por exemplo, 16 bits), a porcentagem de números repetidos é muito alta, o que não é um bom caso para a tabela hash.

2.2. Função de Dispersão

Na implementação, foi utilizada a função de hash mais simples o possível, apenas fazendo uma operação de módulo no valor que será inserido, onde k é o elemento a ser inserido, e M é o tamanho da tabela.

$$h(k) = k \bmod M \tag{1}$$

Nos testes realizados, o tamanho da tabela foi alterado em relação ao fator de carga escolhido. Além disso, os testes foram repetidos escolhendo o próximo primo como o tamanho da tabela, com o objetivo de evitar agrupamentos primários, que podem acontecer mais facilmente quanto o tamanho da tabela é comumente divisível, como por exemplo, por um número par.

2.3. Funções de tratamento de colisão

2.3.1. Sondagem Linear

Para a sondagem linear a fórmula abaixo foi utilizada, onde k é o elemento a ser inserido, i é o número de colisões encontradas, e M é o tamanho da tabela, sendo $i > 1$:

$$h_2(k) = (h(k) + i) \bmod M \tag{2}$$

2.3.2. Sondagem Quadrática

Para a sondagem quadrática a fórmula abaixo foi utilizada, onde k é o elemento a ser inserido, i é o número de colisões encontradas, e M é o tamanho da tabela, sendo $i > 1$:

$$h_3(k) = (h(k) + i^2) \bmod M \tag{3}$$

2.4. Computador utilizado

O computador utilizado para os testes foi um modelo Dell Inspiron 15R SE 7520. O processador é um Intel Core i7-3632QM de 3.2GHz, com 8GB de memória RAM, sendo o sistema operacional Linux Mint 18.1 Serena.

2.5. Compilador utilizado

O compilador utilizado foi o GNU/gcc 5.4.0, compilado em 20160609, versão Ubuntu 5.4.0-6ubuntu1 16.04.5. As flags de C++ utilizadas, além do padrão do CMake, foram: *-std=c++11 -Wall*.

3. Resultados

Os diferentes n escolhidos foram testados com diversos fatores de carga. Devido a pouca diferença entre os resultados para um n pequeno ($n = 10.000$, por exemplo), aqui foram colocados os dados do $n = 10.000.000$. O seguintes tamanho de entrada foram testados:

- $n = 10.000$;
- $n = 100.000$;
- $n = 1.000.000$;
- $n = 10.000.000$;

Nos resultados, foram contabilizados os fatores de carga reais para as inserções que utilizam o próximo número primo como tamanho da tabela.

3.1. Execução x Fator de Carga

Os fatores de carga escolhidos foram de 0.1 até 1. O fator de carga 1 foi escolhido para demonstrar a diferença entre os diversos fatores de carga, pois apesar de ser o pior fator de carga o possível para uma tabela hash de encadeamento aberto, ele é útil para demonstrar a eficácia dos diferentes fatores de carga para uma tabela qualquer. Porém, o fator de carga 1 foi omitido dos gráficos para não gerar uma distorção muito grande que atrapalhasse a visualização dos outros dados.

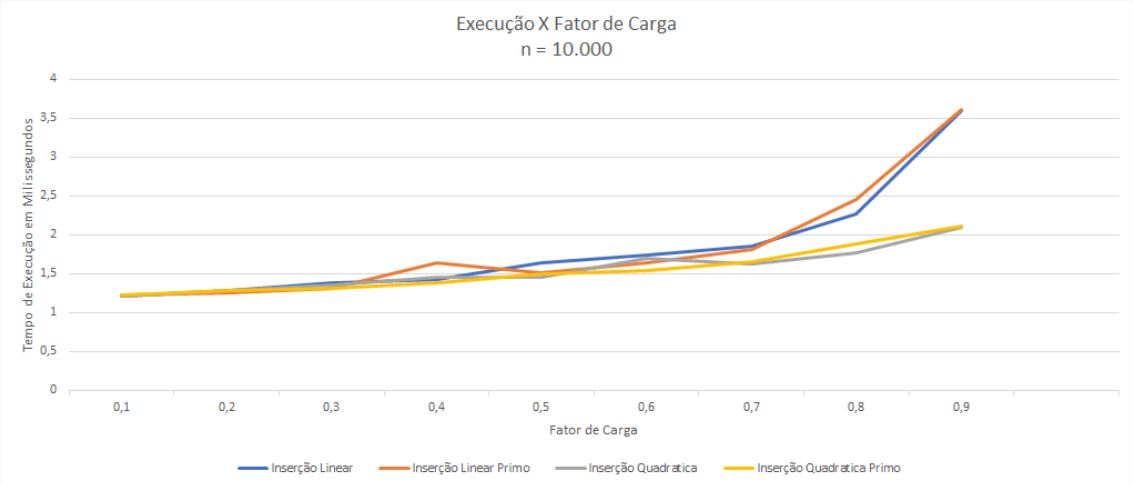

Figura 1. Execução x Fator de carga $n = 10.000$

Para a maior parte dos testes realizados, quanto mais próximo de 1 o fator de carga se aproximava mais colisões foram detectadas e, consequentemente, maior o tempo de execução para a inserção dos dados na tabela. Em alguns casos particulares como representado na figura 2, o fator de carga 0.5 não influenciou o desempenho de forma esperada, porém isso pode ter acontecido devido a algum pico de processamento no computador utilizado para testes, não necessariamente tendo algum significado estatístico.

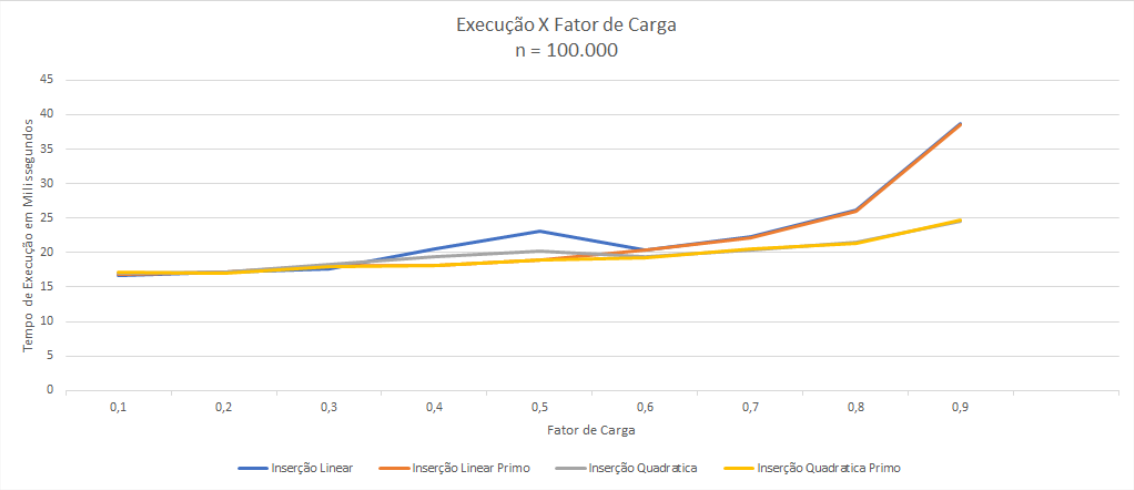

Figura 2. Execução x Fator de carga $n = 100.000$

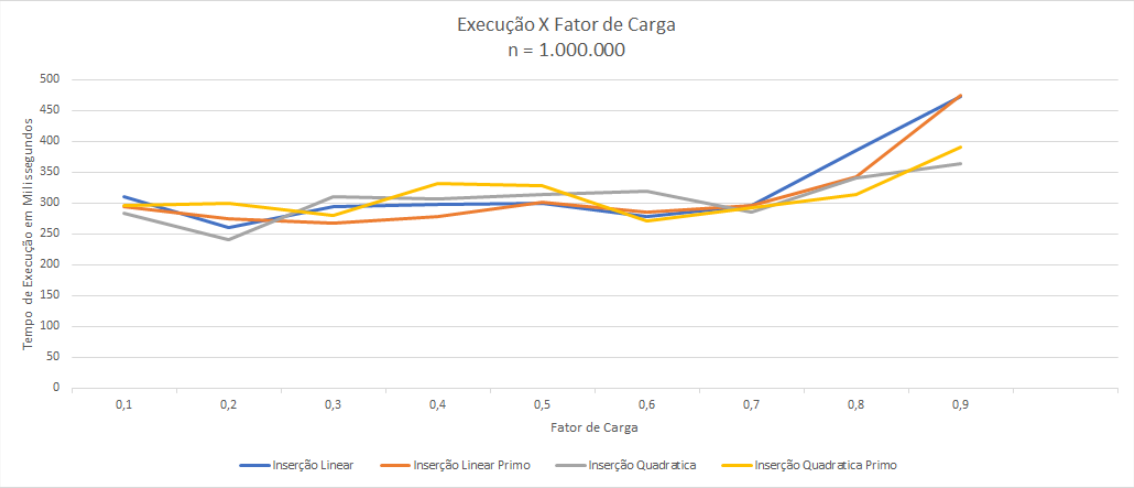

Figura 3. Execução x Fator de carga $n = 1.000.000$

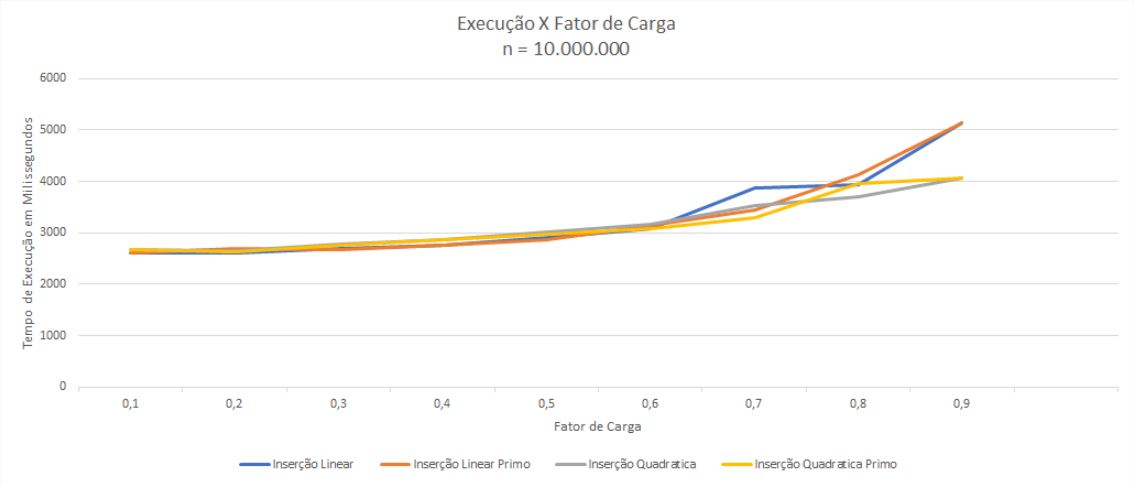

Figura 4. Execução x Fator de carga $n = 10.000.000$

3.2. Comparações x Fator de Carga

Uma comparação entre o número total de comparações e o fator de carga da tabela é mais justa em relação ao tempo de execução, pois o número de comparações não dependeria de outros fatores externos além do tamanho da tabela, da função hash e dos elementos escolhidos. Para isso, foram comparados os resultados para todos os n escolhidos nas tabelas a seguir:

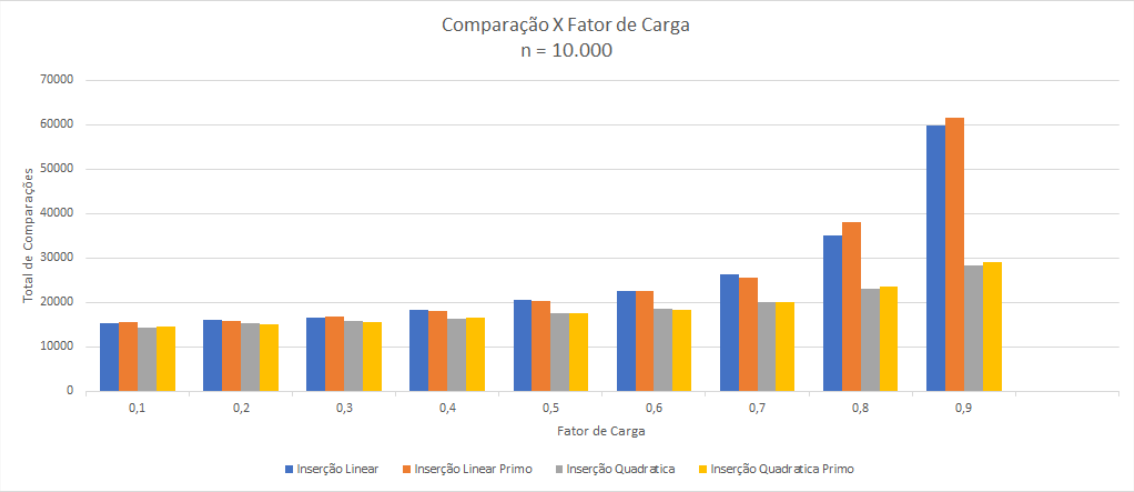

Figura 5. Comparações x Fator de carga $n = 10.000$

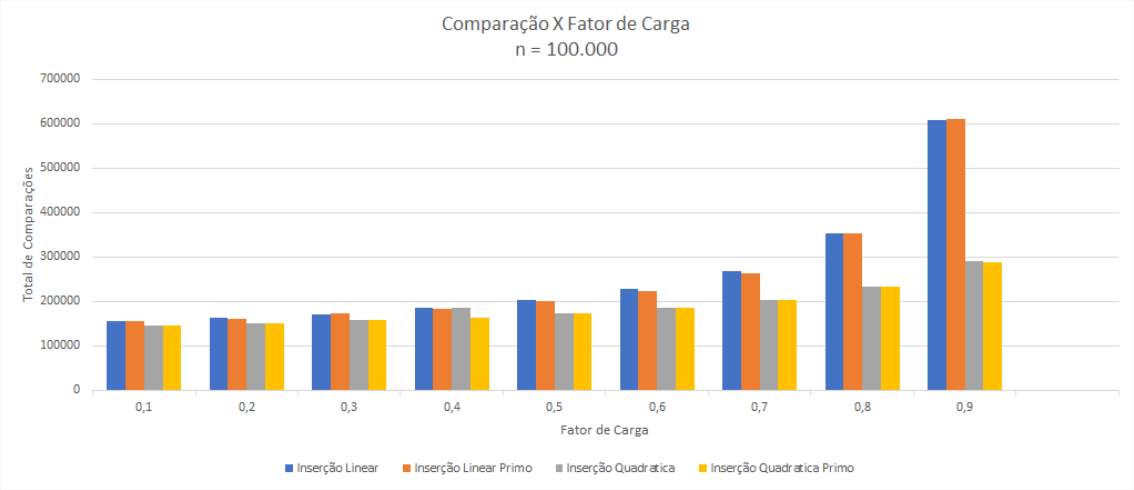

Figura 6. Comparações x Fator de carga $n = 100.000$

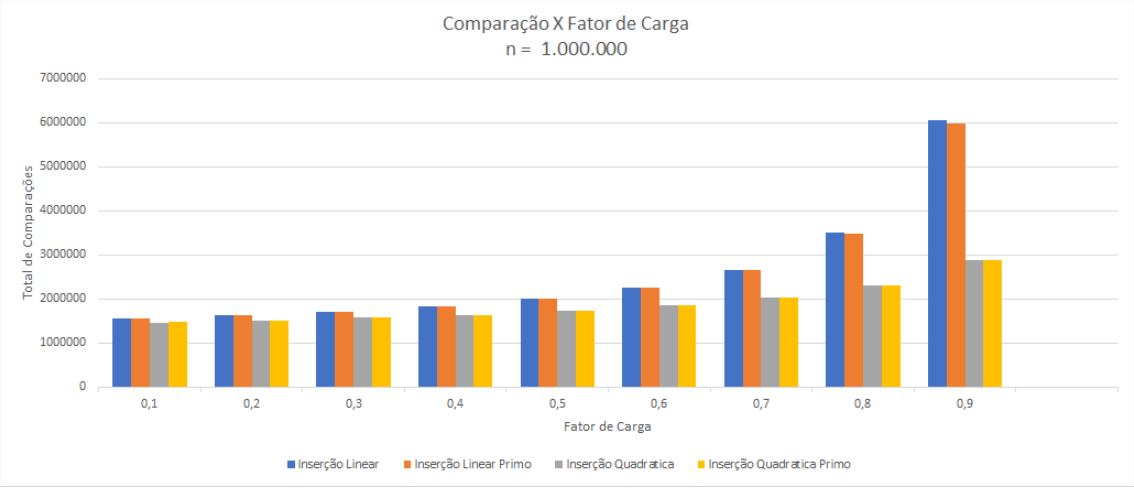

Figura 7. Comparações x Fator de carga $n = 1.000.000$

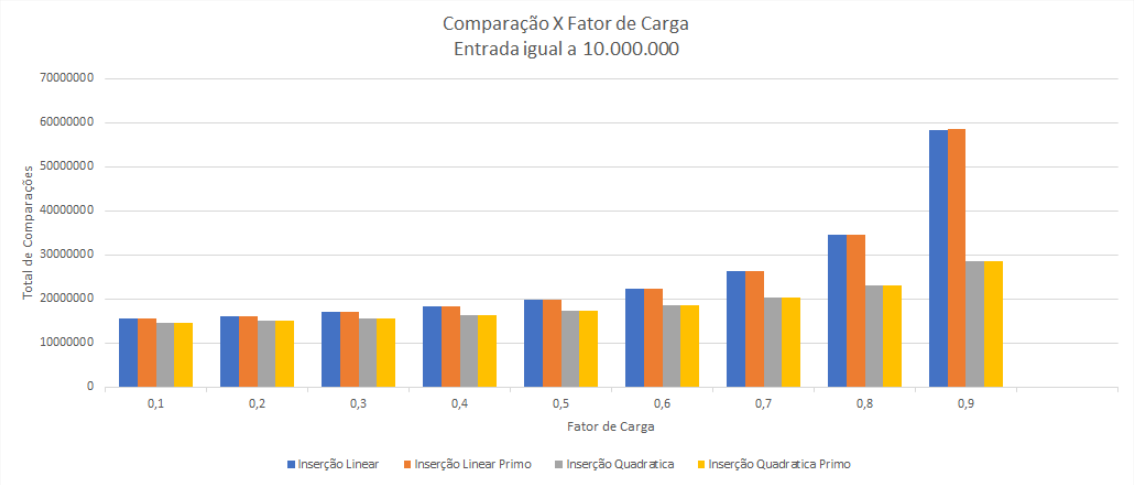

Figura 8. Comparações x Fator de carga $n = 10.000.000$

3.3. Tabelas

As tabelas com os dados a seguir consideram apenas o $n = 10.000.000$, pois não teríamos espaço para demonstrar todos os dados neste trabalho. Para cada fator de carga foram analisadas outras métricas das sondagens linear e quadrática além das dispostas nos gráficos, tais como, total de colisões, total de comparações, média de colisões e média de comparações.

Nos testes realizados, para um $n = 10.000.000$ em uma inserção linear, estas métricas podem ser observadas conforme a tabela 1:

Tabela 1. Inserção linear

Fator Carga	Tamanho Real	Tempo (ms)	Total Colisões	Média Colisões	Total Comparações	Média Comparações
1	10000000	146710,57	2682053266	26820,53	2692053266	26920,53
0,9	11000000	5147,26	48336253	439,42	58336253	530,32
0,8	12000000	3947,06	24568749	204,73	34568749	288,07
0,7	13000000	3862,78	16382258	126,01	26382258	202,94
0,6	14000000	3074,87	12323360	88,02	22323360	159,45
0,5	15000000	2905,02	9839585	65,59	19839585	132,26
0,4	16000000	2752,87	8240996	51,50	18240996	114,00
0,3	17000001	2688,87	7039264	41,40	17039264	100,23
0,2	18000001	2611,02	6156908	34,20	16156908	89,76
0,1	19000001	2605,24	5467874	29,28	15467874	81,40

A sondagem quadrática utilizou da fórmula 3 para tratar as colisões, e para um $n = 10.000.000$ as métricas resultantes podem ser observadas conforme a tabela 2:

Tabela 2. Inserção Quadrática

Fator Carga	Tamanho Real	Tempo (ms)	Total Colisões	Média Colisões	Total Comparações	Média Comparações
1	10000000	7940,21	59156035	591,56	69156035	691,56
0,9	11000000	4071,40	18510989	168,28	28510989	259,19
0,8	12000000	3705,77	13023745	108,53	23023745	191,86
0,7	13000000	3532,51	10209700	78,53	20209700	155,45
0,6	14000000	3176,09	8468592	60,48	18468592	131,91
0,5	15000000	3021,79	7245510	48,30	17245510	114,97
0,4	16000000	2865,86	6353876	39,71	16353876	102,21
0,3	17000001	2778,48	5641017	33,18	15641017	92,00
0,2	18000001	2649,87	6156908	34,20	16156908	89,76
0,1	19000001	2605,24	5467874	28,77	15467874	81,40

A sondagem linear utilizando número primo consistiu em definir o tamanho da tabela hash mediante o próximo número primo dado o tamanho da entrada. A adoção desta técnica permitiu diminuir as colisões e melhorou o tempo de processamento. Os resultados dessa aplicação para $n = 10.000.000$ podem ser observadas na tabela 3:

Tabela 3. Inserção Linear Utilizando Número Primo

Fator Carga	Tamanho Real	Tempo (ms)	Total Colisões	Média Colisões	Total Comparações	Média Comparações
1	10000019	142063,19	2600008203	26000,03	2610008203	26100,03
0,9	11000027	3454,71	16403279	126,17	26403279	203,10
0,8	12000000	4134,88	24558813	204,65	34558813	287,98
0,7	13000027	3454,71	16403279	126,17	26403279	203,10
0,6	14000029	3148,11	12306738	87,90	22306738	159,33
0,5	15000017	2877,02	9851510	65,67	19851510	132,34
0,4	16000057	2760,40	8216832	51,35	18216832	113,85
0,3	17000023	2677,65	7040458	41,41	17040458	100,23
0,2	18000041	2691,19	6156246	34,20	16156246	89,75
0,1	19000013	2610,38	5474871	28,81	15474871	81,44

A sondagem quadrática utilizando número primo utilizou da fórmula 3 para tratar as colisões, e a tabela 4 apresenta as métricas para $n = 10.000.000$.

Tabela 4. Inserção Quadrática Utilizando Número Primo

Fator Carga	Tamanho Real	Tempo (ms)	Total Colisões	Média Colisões	Total Comparações	Média Comparações
1	10000019	8002,64	59158881	591,58	69158881	691,58
0,9	11000027	4055,66	18525264	168,41	28525264	259,31
0,8	12000017	3955,13	13012910	108,44	23012910	191,77
0,7	13000027	3295,53	10224260	78,64	20224260	155,57
0,6	14000029	3078,26	8476318	60,54	18476318	131,97
0,5	15000017	2974,22	7256662	48,37	17256662	115,04
0,4	16000057	2872,39	6337946	39,61	16337946	102,11
0,3	17000023	2749,83	5642616	33,19	15642616	92,01
0,2	18000041	2631,42	5086081	28,25	15086081	83,81
0,1	19000013	2674,64	4625238	24,34	14625238	76,97

4. Conclusão

As sondagens linear e quadrática são bastante utlizadas pela sua fácil implementação para tratamentos de colisões em tabelas hash com endereçamento aberto.

Os resultados obtidos para diferentes valores de n permitiram observar o comportamento das duas abordagens conforme o valor de n variava de 10.000 até $10.000.000$. Os resultados mostraram que conforme o valor do fator de carga aumenta, maiores são as colisões e, consequentemente, o tempo de processamento aumenta para ambas abordagens.

A ideia de utilizar o próximo número primo em relação a n com o intuito de diminuir as colisões não mostrou-se tão eficaz nos cenários utlizandos quando o fator de carga é próximo de $0,1$, no entanto quando o fator de carga aproximava-se de 1 percebeu-se a diminuição de colisões e tempo de processamento em relação as sondagens tradicionais com a função hash mais simples.

Referências

Cormen, T. H. (2012). *Algoritmos (Em Portuguese do Brasil)*. Elsevier.

Sedgewick, R. and Wayne, K. (2011). *Algorithms (4th Edition)*. Addison-Wesley Professional.