Tabela Hash

Victor Silva Camargo, Vinícius Silva Camargo

Pontifícia Universidade Católica do Paraná´
(PUCPR) Curitiba – PR – Brazil

Abstract. *A hash function is an algorithm that transforms data into a fixed-length sequence of characters, known as a ”hash” or ”fingerprint.” These func- tions are used to optimize data retrieval, protect data integrity, and accelerate algorithms. They play a crucial role in data structures like hash tables and in cryptography. The choice or design of hash functions is critical to avoid colli- sions and ensure security and efficiency.*

Resumo. *Uma função de hash e´ um algoritmo que transforma dados em uma seque^ncia de caracteres de tamanho fixo, conhecida como ”hash” ou ”impressão digital”. Essas funções são usadas para otimizar a pesquisa de dados, proteger a integridade de informações e acelerar algoritmos. Elas desempenham um papel essencial em estruturas de dados como tabelas de dispersão (hash tables) e em criptografia. A escolha ou design de funções de hash é crucial para evitar colisões e garantir a segurança e eficiência.*

1. Hash com Lista encadeada

Para o nosso trabalho escolhemos o encadeamento porque tentamos usar o hiraishin mas estava dando problemas de mais, sobre a lista encadeada, a lista encadeada com hash e uma estrutura de dados que combina características de uma lista encadeada é uma tabela de dispersão. Ela é usada para armazenar e recuperar informações eficientemente, onde cada elemento é associado a uma chave única, calcula-se um valor de hash a partir dessa chave e usa-o para determinar a posição do elemento na estrutura. As colisões, quando várias chaves te^m o mesmo valor de hash, são resolvidas através de listas encadeadas associadas a cada slot da tabela de dispersão. A eficie^ncia depende da função de hash e do dimensionamento adequado da tabela. Essa estrutura é comum em dicionários, bancos de dados e sistemas de indexação.

2. Estrutura do Código

A princípio a estrutura do código foi dividida por conjunto de dados tendo ao todo tendo 5 códigos executáveis, onde cada um deles usa 3 func¸o˜es hash diferentes para inserir o conjunto de dados nas diferentes tabelas hash, o projeto possui também 3 códigos que implementam as 3 funções hashs diferentes, todas as 3 usam o no´ e lista encadeada.

3. Resultados

Executamos o código com diferentes tamanhos do vetor sendo 10, 100, 1000, 10000 e 100000 com cinco conjuntos de dados aleatórios com 20 mil, 100 mil, 500 mil, 1 milhão e 5 milhões de elementos cada, na tabela tem as letras representam os métodos sendo I para inserção B para Busca, e todos os valores de tempo estão em milisegundos.

I	100000	1000000	5000000	7000000	8000000
Divisão Tempo	4	6	10	9	9
Divisão Colisões	1880	194	48	27	25
Multiplicação Tempo	5	7	9	7	2
Multiplicação Colisões	1930	194	44	25	20
Dobramento Tempo	2	1	1	1	1
Dobramento Colisões	19939	19939	19939	19939	19939
B					
Divisão Tempo	0	0	0	0	0
Divisão Comparações	15	15	15	15	15
Multiplicação Tempo	0	0	0	0	0
Multiplicação Comparações	15	15	15	15	15
Dobramento Tempo	0	0	0	0	0
Dobramento Comparações	4018	4018	4018	4018	4018

Figura 1. Tabela hash com 20 mil elementos

I	100000	1000000	5000000	7000000	800000
Divisão Tempo	20	7	24	38	79
Divisão Colisões	36742	4818	983	705	684
Multiplicação Tempo	17	31	67	74	68
Multiplicação Colisões	36565	4767	942	691	575
Dobramento Tempo	8	2	6	7	3
Dobramento Colisões	99934	99934	99934	99934	99934
B					
Divisão Tempo	1	0	0	0	0
Divisão Comparações	26	17	15	15	15
Multiplicação Tempo	0	0	0	0	0
Multiplicação Comparaçõ	30	18	15	15	18
Dobramento Tempo	0	0	1	1	0
Dobramento Comparaçõ	20550	20550	20550	20550	20550

Figura 2. Tabela hash com 100 mil elementos

I	100000	1000000	5000000	7000000	8000000
Divisão Tempo	41	38	58	205	173
Divisão Colisões	400643	106050	23955	17259	15345
Multiplicação Tempo	24	46	164	357	343
Multiplicação Colisões	400656	106409	24724	18016	15685
Dobramento Tempo	18	9	12	9	9
Dobramento Colisões	499930	499930	499930	499930	499930
B					
Divisão Tempo	0	0	0	0	0
Divisão Comparações	37	17	17	15	15
Multiplicação Tempo	0	0	0	0	0
Multiplicação Comparações	36	15	15	15	15
Dobramento Tempo	2	2	3	2	2
Dobramento Comparações	85131	85131	85131	85131	85131

Figure 3. Tabela hash com 500 mil elementos

I	100000	1000000	5000000	6000007	8000000
Divisão Tempo	69	142	106	64	164
Divisão Colisões	900005	567901	368176	213409	60315
Multiplicação Tempo	23	46	63	196	91
Multiplicação Colisões	900004	567757	367734	213081	60450
Dobramento Tempo	27	124	42	31	22
Dobramento Colisões	999928	999928	999928	999928	999928
B					
Divisão Tempo	0	0	0	0	0
Divisão Comparações	94	25	22	18	15
Multiplicação Tempo	0	0	0	0	0
Multiplicação Comparações	92	29	26	24	20
Dobramento Tempo	1	4	5	4	4
Dobramento Comparações	166023	166023	166023	166023	166023

Figura 4. Tabela hash com 1 milhão de elementos

I	100000	1000000	5000000	7000000	8000000
Divisão Tempo	201	378	605	478	669
Divisão Colisões	4900000	4500028	4006899	3163880	1283724
Multiplicação Tempo	289	451	508	497	897
Multiplicação Colisões	4900000	4500021	3164717	3164717	1292015
Dobramento Tempo	125	147	504	272	195
Dobramento Colisões	4999925	4999925	4999925	4999925	4999925
B					
Divisão Tempo	0	1	2	0	0
Divisão Comparações	455	111	67	43	32
Multiplicação Tempo	0	0	47	0	0
Multiplicação Comparaçõ	545	103	57	32	15
Dobramento Tempo	4	1	2	2	31
Dobramento Comparaçõ	1760000	1760000	1760000	1760000	1760000

Figura 5. Tabela hash com 5 milhão de elementos

4. Conclusão

Notou-se que quanto maior a tabela hash, melhor e o desempenho de todas as funções hash devido às colisões serem menos frequentes que as tabelas menores, isso favorece também a busca, fazendo com que as funções entrem menos vezes nas listas encadeadas.

Entre as funções hash a de dobramento teve o maior número de colisões e consequentemente maior número de comparações na busca, enquanto as outras duas equilibraram bastante.

Esses resultados podem variar dependendo do tamanho da tabela hash e da máquina que está executando..