Relatório Hash

Arthur Przygocki[1], **Carlos Eduardo Rodrigues Mello**[2], **Henrique Tetilha Golias**[1]

[1]Pontifícia Universidade Católica do Paraná - PUCPR

1. Hash

A tabela hash é uma estrutura de dados utilizada para armazenar e recuperar informações de forma eficiente. Ela é projetada para realizar operações de busca, inserção e remoção em tempo quase constante, independentemente do tamanho dos dados.

A ideia principal por trás de uma tabela hash é o uso de uma função de hash, que mapeia os elementos que você deseja armazenar para posições em uma estrutura de array. Nas tabelas estamos usando respectivamente os tamanhos de conjuntos de dados: 20000, 100000 e 1000000

Github: https://github.com/Henrick1/AvHash—RPEC

2. Hash por Divisão

Ele calcula o índice na tabela dividindo a chave pela capacidade da tabela e usando o resto da divisão como índice. É uma abordagem simples, mas pode resultar em colisões se não for escolhido o tamanho da tabela apropriado.

Divisão - 100			
Número de Colisões	Tempo Inserção (mili)	Tempo Busca (nano)	Comparações
19900	7	444	5
99900	271	72	5
499900	15886	76	5

Divisão - 10000			
Número de Colisões	Tempo Inserção (mili)	Tempo Busca (nano)	Comparações
11335	0	34	5
99900	3	0	5
990000	990	140	5

Divisão - 100000			
Número de Colisões	Tempo Inserção (mili)	Tempo Busca (nano)	Comparações
1867	0	29	5
36709	1	36	5
900004	112	88	5

3. Hash por Multiplicação

A multiplicação é uma técnica que envolve multiplicar o valor do elemento a ser armazenado por uma constante, extrair a parte fracionária do resultado e em seguida, multiplicar esse valor pela quantidade total de espaços na tabela hash para obter o índice para ajudar a distribuir os elementos de forma uniforme na tabela.

Envolve a multiplicação da chave por uma constante real (0 ¡ A ¡ 1) e a aplicação de um operador de módulo para calcular o índice na tabela de hash. Isso ajuda a distribuir os elementos de forma uniforme na tabela.

Multiplicação - 100			
Número de Colisões	Tempo Inserção (mili)	Tempo Busca (nano)	Comparações
20000	31.2	5186	822.8
100000	1178.6	20460	5169
1000000	390325.8	787960	51947.2

Multiplicação - 10000			
Número de Colisões	Tempo Inserção (mili)	Tempo Busca (nano)	Comparações
18867.8	0.8	84	14.6
99999.4	13.2	364	58
1000000	3863.2	6108	535.4

Multiplicação - 100000			
Número de Colisões	Tempo Inserção (mili)	Tempo Busca (nano)	Comparações
5192.8	0.4	76	5.6
76755.8	3	88	9.8
999995.4	648	1084	59.2

4. Hash por Dobramento

Consiste em dividir a chave em partes iguais, somar essas partes e, em seguida, aplicar um operador de módulo para calcular o índice na tabela. Essa técnica é frequentemente usada em sistemas que processam valores maiores do que o número de slots disponíveis na tabela hash, pois permite distribuir os valores de entrada uniformemente pelos slots.

Dobramento - 100			
Número de Colisões	Tempo Inserção (mili)	Tempo Busca (nano)	Comparações
20000	43.2	10052	2791.4
100000	2737.2	35352	14395.6
1000000	661197.4	805552	143756.4

Dobramento - 10000			
Número de Colisões	Tempo Inserção (mili)	Tempo Busca (nano)	Comparações
19956	1.2	44	19
100000	44.6	296	117
1000000	8861.6	796	802

Dobramento - 100000			
Número de Colisões	Tempo Inserção (mili)	Tempo Busca (nano)	Comparações
18405.4	0.8	24	2
99613.8	19	48	9
1000000	7136.2	384	99

5. Análise

De todos os métodos de hash, o mais rápido na maioria dos casos foi por divisão. O hash por multiplicação foi o método que mais houve colisões e o hash por dobramento teve o maior número de comparações. O hash por divisão tende a ser mais eficaz. O hash por multiplicação e o hash por dobramento podem ser vantajosos em cenários com dados mais variados, mas podem exigir mais recursos computacionais.