Társila Samille Santos da Silveira

Relatório sobre Tabelas de Dispersão

Brasil
2020, v-1.0

Társila Samille Santos da Silveira

Relatório sobre Tabelas de Dispersão

Relatório resumo apresentado à disciplina de Estrutura de Dados Básicas I, como requisito parcial para obtenção de nota referente à unidade III.

Universidade Federal do Rio Grande do Norte - UFRN
Instituto Metrópole Digital - IMD
Bacharelado em Tecnologia da Informação

Brasil
2020, v-1.0

Conteúdo

1 Introdução

Este relatório objetiva fazer um relatório sobre Tabelas de Dispersão, um resumo do conteudo apresentado na terceira unidade.

2 Relatório

Tabela de Dispersão(Hash Table) é uma estrutura de dados que armazena dados de forma associativa. Em uma tabela de Dispersão, os dados são armazenados em um formato de vetor, em que cada valor de dados tem seu próprio índice. O acesso aos dados torna-se muito rápido pois se conhece o índice dos dados desejados.

Por isso é uma estrutura de dados em que as operações de inserção e busca são muito rápidas, independente do tamanho dos dados. A tabela de Dispersão usa um vetor como meio de armazenamento e usa a técnica de Dispersão para gerar um índice onde um elemento deve ser inserido ou localizado.

2.1 Dispersão

Dispersão é uma técnica para converter um intervalo de valores-chave em um intervalo de índices de um vetor. Se usa o operador de módulo para obter uma gama de valores-chave. Considere um exemplo de tabela de Dispersão de tamanho 20 e os seguintes itens devem ser armazenados. Os itens estão no formato (chave, valor).

Tabela 1: Técnica de Dispersão

Numero	**Chave**	**Dispersão**	**Index Vetor**
1	1	1 % 20 = 1	1
2	2	2 % 20 = 2	2
3	*42*	42 % 20 = 2	2
4	4	4 % 20 = 4	4
5	12	12 % 20 = 12	12
6	*14*	14 % 20 = 14	14
7	17	17 % 20 = 17	17
8	13	13 % 20 = 13	13
9	37	37 % 20 = 17	17

2.2 Abordagens

2.2.1 Sondagem Linear

Como podemos ver, pode acontecer que a técnica de Dispersão seja usada para criar um índice já usado do array. Nesse caso, podemos pesquisar o próximo local vazio na matriz olhando para a próxima célula até encontrar uma célula vazia. Essa técnica é chamada de sondagem linear, é uma das abordagens que não utiliza listas ligadas, também há Sondagem Quadratica e Disperção Dupla, mas para fins de simplificação utilizaremos como exemplo a Sondagem Linear.

Tabela 2: Técnica de Dispersão com Sondagem Linear

Numero	**Chave**	**Dispersão**	**Index Array**	**Index Array depois da Sondagem Linear**
1	1	1 % 20 = 1	1	1
2	2	2 % 20 = 2	2	2
3	*42*	42 % 20 = 2	2	3
4	4	4 % 20 = 4	4	4
5	12	12 % 20 = 12	12	12
6	*14*	14 % 20 = 14	14	14
7	17	17 % 20 = 17	17	17
8	13	13 % 20 = 13	13	13
9	37	37 % 20 = 17	17	18

2.2.2 Encadeamento Separado

Outra estratégia, comumente conhecida como encadeamento separado, é manter uma lista de todos os elementos com o mesmo código hash. Pode-se utilizar a implementação da lista da Biblioteca Padrão.

2.3 Item de dados

Defina um item de dados com alguns dados e chave, com base nos quais a pesquisa será conduzida em uma tabela hash.

```
struct DataItem {
        dados internos;
        chave int;
};
```

2.4 Método de Dispersão - Hash

Defina um método de Dispersão para calcular o código hash da chave do item de dados.

```
int hashCode (int key) {
        return key % TAMANHO_TABELA;
}
```

2.5 Operações básicas

A seguir estão as operações primárias básicas de uma tabela de Dispersão.

Pesquisar - Pesquisa um elemento em uma tabela hash.

Inserir - Insere um elemento em uma tabela hash.

Delete - Exclui um elemento de uma tabela hash.

2.5.1 Operação de Pesquisa

Se foi utilizado Sondagem Linear, calcula-se o código hash da chave passada e localiza-se o elemento usando esse código hash como índice na matriz. Se o elemento não for encontrado no código hash calculado, usa-se a análise linear para obter o elemento à frente.

Se foi utilizado Encadeamento Separado, usamos o função de Dispersão(hashCode) para determinar qual lista percorrer, em seguida, pesquisa-se na lista apropriada.

2.5.2 Operação de Inserir

Se foi utilizado Sondagem Linear, calcula-se o código hash da chave passada e localiza-se o índice usando esse código hash como um índice na matriz.

Usa-se sondagem linear para achar uma localização vazia, se um elemento for encontrado no código hash calculado.

Se foi utilizado Encadeamento Separado, usamos o função de Dispersão(hashCode) para determinar qual lista percorrer, verificamos na lista apropriada para ver se o elemento já está no lugar. Se o elemento acaba por ser novo, pode ser inserido no início da lista, desde que seja conveniente e também porque frequentemente acontece que os elementos inseridos recentemente são os mais prováveis para ser acessado em um futuro próximo.

2.5.3 Operação de Excluir

Se foi utilizado Sondagem Linear, calcula-se o código hash da chave passada e localiza-se o índice usando esse código hash como um índice na matriz. Usa-se a análise linear para obter o elemento adiante se um elemento não for encontrado no código hash calculado. Quando encontrado, armazena-se um item fictício para manter o desempenho da tabela hash intacto.

Se foi utilizado Encadeamento Separado, usamos o função de Dispersão(hashCode) para determinar qual lista percorrer, verificamos na lista apropriada para ver se o elemento existe. Se sim, é excluido da lista ligada (LinkedList).

2.6 Comparações

A principal vantagem das tabelas de hash sobre outras estruturas de dados de tabela é a velocidade (Figura 1 e 2). Essa vantagem é mais aparente quando o número de entradas é grande. As tabelas de hash são particularmente eficientes quando o número máximo de entradas pode ser previsto com antecedência, de modo que a matriz de intervalo possa ser alocada uma vez com o tamanho ideal e nunca redimensionada.

Embora as operações em uma tabela hash demorem em média um tempo constante, o custo de uma boa função hash pode ser significativamente mais alto do que o loop interno do algoritmo de pesquisa para uma lista sequencial ou árvore de pesquisa. Portanto, as tabelas de hash não são eficazes quando o número de entradas é muito pequeno (pior caso). (No entanto, em alguns casos, o alto custo de computação da função hash pode ser mitigado salvando o valor do hash junto com a chave.)

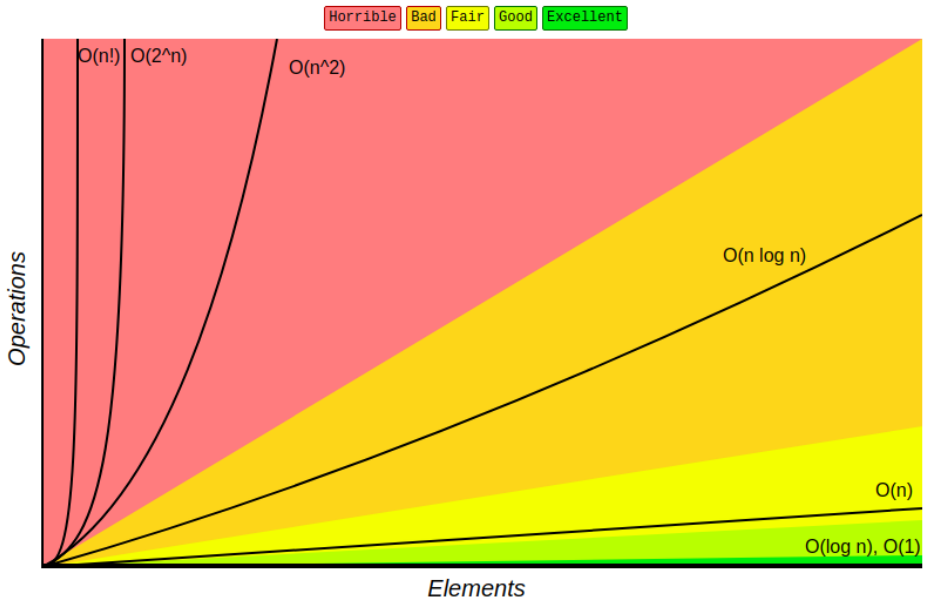

Figura 1: Gráfico de complexidade Big-O

Estrutura de dados	Complexidade de tempo							
	Média				Pior			
	Acesso	Pesquisa	Inserção	Eliminação	Acesso	Pesquisa	Inserção	Eliminação
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)
Pilha	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Fila	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Lista Ligada Individualmente	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Lista duplamente vinculada	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)
Lista de Pular	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)
Tabela Hash	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)
Árvore de pesquisa binária	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)
Árvore cartesiana	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))
Árvore Vermelho-Preto	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))
Árvore Splay	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))
Árvore AVL	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))
Árvore KD	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)

Figura 2: Operações comuns de estrutura de dados

2.7 SHA-1

O termo Secure Hash Algorithm (SHA , Inglês para algoritmo de hash seguro) descreve um grupo de padronizados funções hash criptográficas .Eles são usados para calcular um valor de teste para quaisquer dados digitais (mensagens) e são, entre outras coisas, a base para a criação de uma assinatura digital .

As funções hash criptográficas são projetadas para aceitar strings de comprimento arbitrário como entrada e gerar valores hash de comprimento fixo. A função hash criptográfica deve ser capaz de resistir a todos os ataques de análise criptográfica conhecidos.

No mínimo, ele deve ter os seguintes atributos:

- Resistência à pré-imagem: Dado um valor hash h deve ser difícil encontrar qualquer mensagem m tal que h = hash(m). Este conceito está relacionado ao da função de mão única (ou função unidirecional). Funções que não possuem essa propriedade estão vulneráveis a ataques de pré-imagem.

- Resistência à segunda pré-imagem: Dada uma entrada m1 deve ser difícil encontrar outra entrada m2 tal que hash(m1) = hash(m2). Funções que não possuem essa propriedade estão vulneráveis a ataques de segunda pré-imagem.

- Resistência à colisão: Deve ser difícil encontrar duas mensagens diferentes m1 e m2 tal que hash(m1) = hash(m2). Tal par é chamado de colisão hash criptográfica. Essa propriedade também é conhecida como forte resistência à colisão. Ela requer um valor hash com pelo menos o dobro do comprimento necessário para resistência à pré-imagem; caso contrário, colisões podem ser encontradas através de um ataque do aniversário.

Por serem um tipo específico de função hash, as funções hash criptográficas também são muito adequadas para ser utilizadas para indexar dados em tabelas hash.

No entanto, em comparação com as funções hash padrão, as funções hash criptográficas costumam ser muito mais caras em termos computacionais. Portanto, eles tendem a ser usados onde os usuários precisam se proteger da possibilidade de falsificação por usuários mal-intencionados em potencial.

Bibliografia

Mark Allen Weiss. Data Structure and Algorithms in C++, chapter Chapter 5 – Hashing, pages 193–244. Pearson Education, Inc. as Addison-Wesley, 2014.

https://www.bigocheatsheet.com/