3. Tabelas de Hash

As tabelas de hash são um tipo de estruturação para o armazenamento de informação, de uma forma extremamente simples, fácil de se implementar e intuitiva de se organizar grandes quantidades de dados.

Possui como idéia central a divisão de um universo de dados a ser organizado em subconjuntos mais gerenciáveis.

A estruturação da informação em tabelas de hash, visa principalmente permitir armazenar e **procurar** rapidamente grande quantidade de dados.

As tabelas de hash são constituídas por 2 conceitos fundamentais:
- **Tabela de Hash**: Estrutura que permite o acesso aos subconjuntos.
- **Função de Hashing**: Função que realiza um mapeamento entre valores de chaves e entradas na tabela.

Quando se pretende armazenar um grande conjunto de dados, classificáveis segundo um critério, é necessário:
- Criar um critério simples para dividir este universo em subconjuntos com base em alguma qualidade do domínio das chaves.
    - Possuir um índice que me permita encontrar o início do subconjunto certo, depois de calcular o hashing. Isto é a **tabela de hash**.
- Saber em qual subconjunto procurar e colocar uma chave.
    - Saber quantos subconjuntos eu quero e criar uma regra de cálculo que me diga, dada uma chave, em qual subconjunto devo procurar pelos dados com esta chave ou colocar este dado, caso seja um novo elemento. Isto é chamado de **função de hashing**.
- Gerenciar estes subconjuntos bem menores por algum método simples.
    - Possuir uma ou um conjunto de estruturas de dados para os subconjuntos. Existem duas filosofias: **hashing fechado** (ou de endereçamento aberto) ou o **hashing aberto** (ou encadeado).

Alguns dos problemas que se colocam quando usamos tabelas de hash são:
- Determinar uma função de hashing que minimize o número de colisões;
- Obter os mecanismos eficientes para tratar as colisões.

Considere as seguintes definições:

N – Número de elementos a armazenar

n – Tamanho do array de apontadores para listas

3.1. Hashing Aberto ou de Encadeamento Separado (Separate Chaining Hashing)

A estruturação de dados segundo o Hashing Aberto, é talvez a forma mais intuitiva de se implementar o conceito de Hashing, na qual se utiliza a ideia de se ter um array de apontadores, com dimensão **n**, contendo cada elemento do array uma ligação (por exemplo para listas ligadas, árvores, etc.) para uma lista representando o conjunto dos elementos a armazenar.

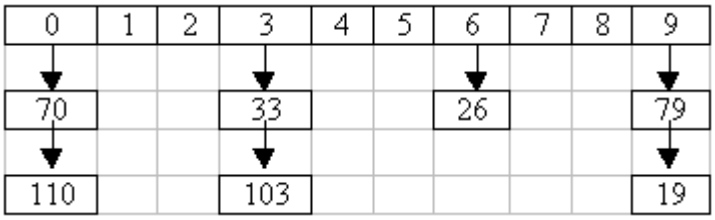
A procura de um elemento no hashing aberto efectua-se da seguinte forma:

- Calcular a partir de uma chave qual entrada do array é a cabeça da lista que se pretende.
- Utiliza-se uma técnica qualquer para pesquisa dentro da lista de elementos armazenados, geralmente utiliza-se a técnica de pesquisa sequencial em lista ligada.

O exemplo I, apresentado de seguida, é um exemplo de uma estruturação de dados segundo uma tabela de hash aberto.

Exemplo I: Pretende-se armazenar números numa tabela de hash por forma a melhorar o processo de procura desses elementos armazenados. Os elementos a guardar são os seguintes: 19, 26, 33, 70, 79, 103, 110. Defina a função de hashing e construa a tabela de hash.

Tabela de Hash



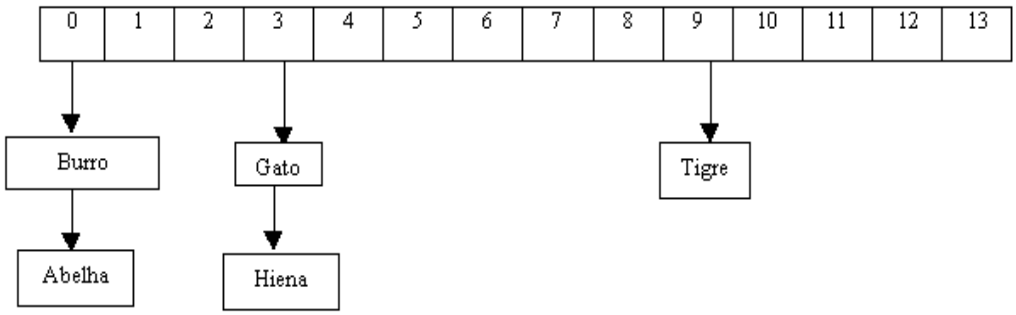
Exemplo II: Cada elemento do array é um apontador para uma lista de estruturas com o mesmo valor da função de hash.

Supondo que o array contém um tamanho **Tam**=14 e que a função de hash $f_h(X)$ calcular os seguintes valores para as seguintes chaves.

Função de Hash:

Chaves	A,B	C,D	E,F	G,H	I,J	K,L	M,N	O,P	Q,R	S,T	U,V	X,Y	W,Y	Z
Hash	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Array de Listas Ligadas:



Propriedades do Separate Chaining:

- Separate Chaining reduz o número de comparações, em média, de **Tam** se comparado com a procura sequencial.
- Necessidade de espaço em memória para armazenar o array de listas ligadas.

3.2. Hashing Fechado ou de Endereçamento Aberto (Open Addressing Hashing)

A estruturação de dados segundo o Hashing Aberto, não sendo a forma mais intuitiva de se implementar o conceito de Hashing, era a forma mais utilizada antigamente, na qual se utiliza somente uma estrutura de dados de tamanho fixo para implementar o hashing. Utiliza-se uma função se hashing que dado um chave devolve a primeira posição onde procurar na tabela de hash.

Esta estruturação utiliza, na sua implementação, um array onde são armazenados os valores a guardar, não se utilizando quaisquer apontadores.

A inserção e procura de um elemento no hashing fechado efectua-se da seguinte forma:

- Calcular a partir de uma chave qual entrada do array é a cabeça da lista que se pretende.
- A função de hashing define qual a primeira posição no array onde poderá eventualmente ser colocado/procurado o elemento a inserir/procurar, se esta posição do array estiver ocupada, procura-se no array, de forma sequencial, uma posição livre no array onde colocar/procurar o elemento.
- O Processo de procura de uma posição vazia numa array (lista circular) só termina quando o programa chega de novo ao ponto de partida. Só neste momento é que se pode considerar que uma chave não existe ou que não existe uma posição vazia.

Quando a função de hashing indica uma posição no array, e essa posição encontra-se ocupada, diz-se que ocorreu uma **colisão**.

Exemplo II: Pretende-se armazenar números numa tabela de hash por forma a melhorar o processo de procura desses elementos armazenados. Os elementos a guardar são os seguintes: 19, 26, 33, 70, 79, 103, 110, 203, 210. Defina a função de hashing e construa a tabela de hash fechada.

Tabela de Hash fechada

1	210	6	26
2		7	
3	33	8	
	103		
4	203	9	19
			79
5		0	70
			110

Função de Hash:
Indexar número pelo último digito

3.3. Complexidade

A ordem de complexidade de tempo de uma implementação de hashing é linear **O(n)** e **T(n)**, e compõe-se de três elementos:

- O tempo para calcular a função de hashing
- O tempo para encontrar o início da lista, indicado pela função de hashing, através da tabela de hash
- O tempo para encontrar o elemento a procurada dentro da lista.

Estas complexidades são diferentes para as duas filosofias de implementação de hashing: aberto ou fechado, na qual:

- **T(n)** vai tender a ser algo em torno de **N/n**, no caso ideal.
- Nos casos menos ideais, **T(n)** varia muito.

3.4. Vantagens

- **Simplicidade**, na medida em que é muito fácil de definir um algoritmo para implementar hashing.
- **Escalabilidade**, dado que podemos adequar o tamanho da tabela de hashing ao número de elementos a armazenar **N** esperado para a aplicação.
- **Eficiência para grande número de elementos**, para trabalharmos com problemas envolvendo **N** = 100.000 de dados, podemos imaginar uma tabela de hash com **n**=1.000 entradas, onde temos uma divisão do espaço de procura da ordem de **N**/1.000 de imediato.
- **Aplicação imediata a arquivos**, os métodos de hashing, tanto de endereçamento aberto como fechado, podem ser utilizados praticamente sem nenhuma alteração em um ambiente de dados persistentes utilizando arquivos em disco.

3.5. Desvantagens

- **Dependência da escolha de função de hashing**, para que o tempo de acesso médio seja mantido, é necessário que a função de hashing divida o universo dos dados de entrada em conjuntos de tamanho aproximadamente igual.
- **Ineficiência para os últimos elementos das listas ligadas**, para o qual o acesso a estes elementos pode ser mais demorado do que em implementações em árvore.

3.6. Funções de Hashing

As funções de hashing, têm o objectivo de transformar o valor de chave de um elemento de dados em uma posição para este elemento em um dos subconjuntos definidos.
O requisito mais importante de uma função de hashing é o de distribuir uniformemente as chaves pelos vários índices de forma a minimizar o número de colisões.

Uma função de hashing é uma transformação de um conjunto de chaves num conjunto de destinos. Esta transformação deve dividir o conjunto de chaves o mais uniforme possivel pelo conjunto de destinos.

A probabilidade de uma chave transformada pela função de hashing, pertencer a uma lista qualquer, deve ser uniforme. Se a função de Hashing não dividir uniformemente as chaves, estão a tabela de hashing pode degenerar.
O pior caso de degeneração é aquele onde todas as chaves caem no mesmo destino.

Utilizar a primeira letra de uma palavra para função de hashing, não é uma boa escolha na medida que a distribuição das palavras que começam por uma determinada letra, não é uma distribuição uniforme (pense quantos palavras começam com "X" ?).

Principais Funções de Hashing:

- Divisão
- Meio do Quadrado
- Folding ou Desdobramento
- Análise de Dígitos

As funções de hashing necessitam sempre de uma chave. Vamos de seguida assumir que as chaves são números inteiros. Se tal não acontecer, é normalmente fácil descobrir uma função que transforme a chave, seja esta um único dado, seja uma string ou mesmo mais que uma chave (ex: Nome e Nº BI, o qual implica uma funções mais complexa) num índice inteiro.

3.6.1. Função de Hashing: Divisão

Suponhamos que o array tem dimensão **n**.
Uma função de hashing simples e eficiente será:

$$h(x) = (x \bmod \mathbf{n}) + 1,$$

quando **n** é primo. O tamanho do array não é normalmente um número primo e por isso usa-se antes:

$$h(x) = ((x \bmod \mathbf{p}) \bmod \mathbf{n}) + 1,$$

onde **p** é um número primo maior do que **n**.
Outra função que tem um bom comportamento é:

$$h(x)= ((\mathbf{a}*x + \mathbf{b}) \bmod \mathbf{p}) \bmod \mathbf{n}) + 1,$$

em que **a** e **b** são menores do que **p** e $\mathbf{a} \neq 0$.

Por exemplo, para chaves do tipo string, a forma mais imediata é a de somar os códigos de todos os caracteres da string. Esta é no entanto uma forma "muito pouco injectiva" de fazer a conversão. Poderemos minorar este problema se entrarmos em consideração com a posição dos caracteres.

Exemplo III: Definir uma função de hashing que divida os seguinte valores de forma mais ou menos uniforme: 119, 226, 343, 710, 749, 1003, 1100, 2003, 2100

Tabela de Hashing: 100 elementos
Função de hashing: F(x) = (x mod 100)
Donde se obtém a seguinte distribuição:

CHAVE "x"	F(x)	
119	119	
226	226	
343	343	
710	710	
749	749	
1003	3	*
1100	100	*
2003	3	*
2100	100	*

Nota: Com esta função de hashing obteve-se colisões nas seguintes chaves:

- 1003 e 2003
- 1100 e 2100

3.6.2. Função de Hashing: Meio do Quadrado

Quando se pretende distribuir uma grande quantidade de dados com uma distribuição muito pouco uniforme, pode-se efectuar uma transformação sobre estes valores de modo a tentar obter uma distribuição mais uniforme.

O método do Meio Quadrado elevado um número ao quadrado e retira deste resultado parte dos bits. Estes bits dependem do número inicial e deste modo pode-se obter uma distribuição diferente das chaves.

Esta função de hashing, efectua os cálculos em dois passos:

1. Eleva-se a chave ao quadrado
2. Utiliza-se um determinado número de dígitos ou bits do meio do resultado.

3.6.3. Função de Hashing: Folding ou Desdobramento

3.6.4. Função de Hashing: Análise de Dígitos