

Tabela Hash

Motivação

- ✎ Os métodos de pesquisa vistos até agora buscam informações armazenadas com base na **comparação de suas chaves**
- ✎ Para obtermos **algoritmos eficientes**, armazenamos os elementos **ordenados** e tiramos proveito dessa ordenação
- ✎ **hashing** (tabela de dispersão) ou método de cálculo de endereço
- No caso médio é possível encontrar a chave em **tempo constante**

Conceitos Básicos

- ✎ Índices em vetores ou listas sequenciais são utilizados para acessar informações
- ✎ No entanto, se quisermos acessar uma informação de um determinado conteúdo (e não posição)?
 - **temos que procurá-lo**

Família	1	2	3	4	5	6
	José Maria	Leila	Artur	Jolinda	Gisela	Alciene


`Família[1] = "José Maria"`

`Família[3] = "Artur"`

`Família[2] = "Leila"`

Em qual posição está "Alciene" ?

Conceitos Básicos

 **Ideal:** Parte da informação poderia ser utilizada na recuperação

Definição de Hash (1/3)

- ✎ Hash é uma generalização da noção mais simples de um arranjo comum, sendo uma estrutura de dados do tipo **dicionário**
- ✎ **Dicionários** são estruturas especializadas em prover as operações de **inserir**, **pesquisar** e **remover**.
- ✎ A idéia central do Hash é utilizar uma **função**, aplicada sobre **parte** da informação (**chave**), para retornar o **índice** onde a informação deve ou deveria estar armazenada.

Definição de Hash (2/3)

- ✎ Esta função que mapeia a chave para um índice de um arranjo é chamada de **Função de Hashing**
- ✎ A estrutura de dados Hash é comumente chamada de **Tabela Hash**.

Definição de Hash (3/3)

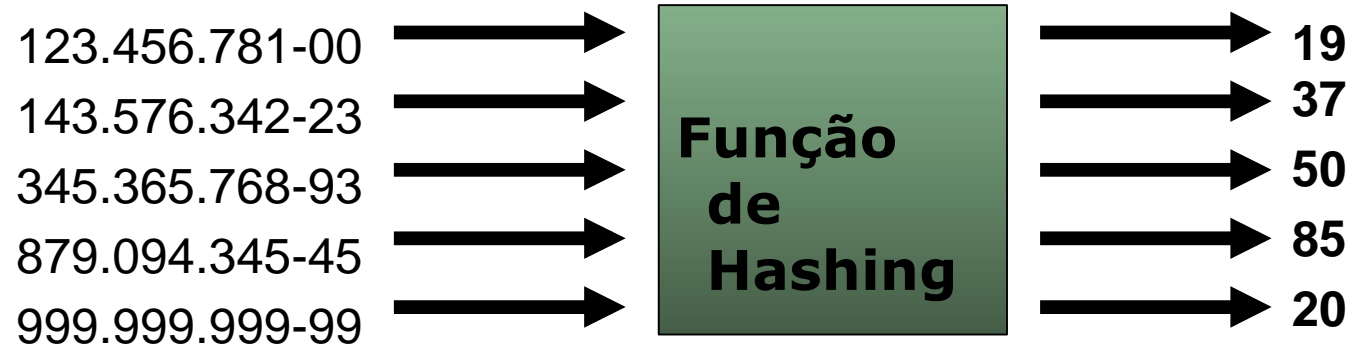


Tabela Hash

19	123.456.781-00; Fausto Silva; Av. Canal. Nº 45.
20	
...	
37	143.576.342-23; Carla Perez; Rua Celso Oliva. Nº 27.
...	
50	345.365.768-93; Gugu Liberato; Av. Atlântica. S/N.
...	
85	879.094.345-45 ; Hebe Camargo; Rua B. Nº 100.
...	

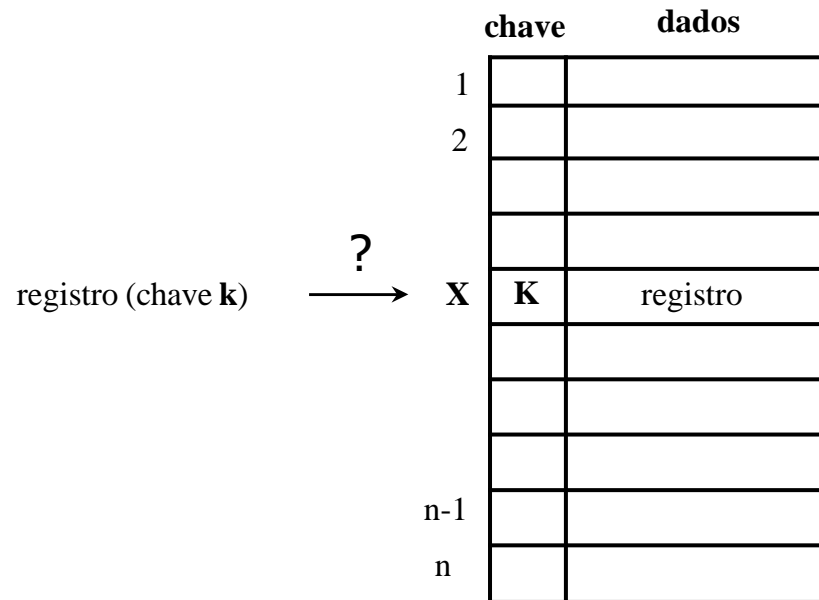
Tabela Hash

Tabela Hash

- é uma estrutura de dados especial
- armazena as informações desejadas associando **chaves**

 **Objetivo:** a partir de uma **chave**, fazer uma busca rápida e obter o valor desejado.

Ilustração de uma Tabela Hash



Como o registro (com chave **K**) foi armazenado na posição **X** na Tabela Hash ao lado?

Resp: Através de uma **Função de Hashing**.

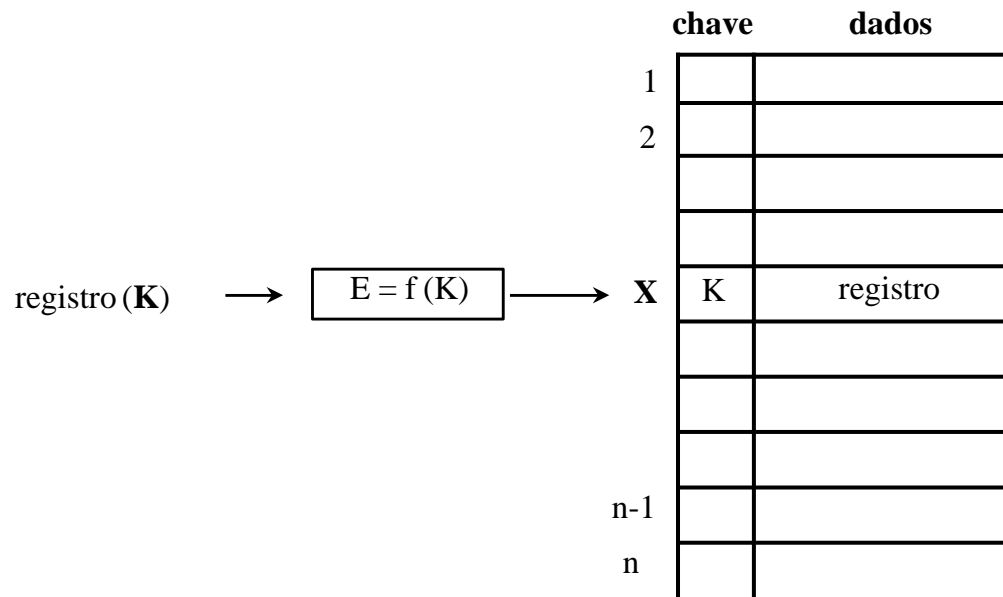
Como representar Tabelas Hash?

- ✎ **Vetor**: cada posição do vetor guarda uma informação. Se a **função de hashing** aplicada a um conjunto de elementos determinar as informações I_1, I_2, \dots, I_n , então o vetor $V[1 \dots N]$ é usado para representar a tabela hash
- ✎ **Vetor + Lista Encadeada**: o vetor contém ponteiros para as listas que representam as informações.

Função de Hashing

- ✂ A **Função de Hashing** é a responsável por gerar um **índice** a partir de uma determinada **chave**.
- ✂ O **ideal** é que a função forneça **índices únicos** para o conjunto das chaves de entrada possíveis.
- ✂ A função de Hashing é **extremamente importante**, pois ela é responsável por distribuir as informações pela Tabela Hash.

Ilustração da Função de Hashing



✂ Os valores da chave podem ser **numéricos**, **alfabéticos** ou **alfa-numéricos**.

✂ Executam a transformação do valor de uma chave em um endereço, pela aplicação de operações aritméticas e/ou lógicas

f: C → E

f: função de cálculo de endereço

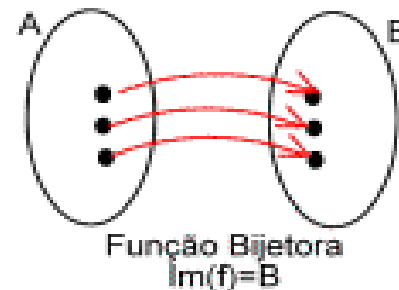
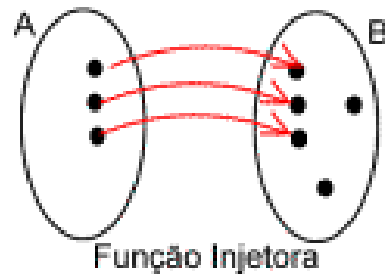
C: espaço de valores da chave (domínio de f)

E: espaço de endereçamento (contradomínio de f)

Hashing Perfeito

✎ Característica:

- Para quaisquer chaves x e y diferentes e pertencentes a A , a função utilizada fornece **saídas diferentes**;



Exemplo de Hashing Perfeito

- ✎ Armazenamento de alunos de uma **determinada turma** de um **curso específico**
- ✎ Cada aluno é identificado unicamente pela sua matrícula.

```
struct aluno {  
    int mat;           // 4 bytes    = 4 bytes  
    char nome[81];     // 1 byte * 81 = 81 bytes  
    char email[41];    // 1 byte * 41 = 41 bytes  
};                    Total = 126 bytes  
typedef struct aluno Aluno;
```

Exemplo de Hashing Perfeito

- ✎ O número de dígitos **efetivos** na matrícula são 7
- ✎ Para permitir um acesso a qualquer aluno em **ordem constante**, podemos usar o número de matrícula do aluno como **índice** de um vetor
- ✎ Um problema é que pagamos um preço alto para ter esse acesso rápido. **Porque?**

Exemplo de Hashing Perfeito

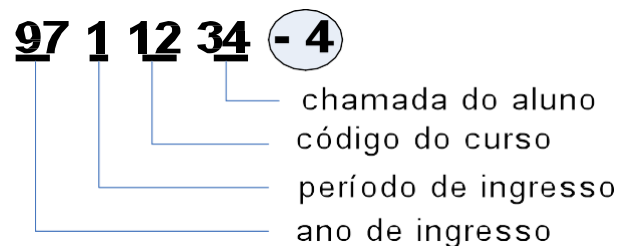
- ✍ O número de dígitos **efetivos** na matrícula são **7**
- ✍ Para permitir um acesso a qualquer aluno em **ordem constante**, podemos usar o número de matrícula do aluno como **índice** de um vetor
- ✍ Um problema é que pagamos um preço alto para ter esse acesso rápido. **Porque?**
 - **Resp:** Visto que a matrícula é composta de 7 dígitos, então podemos esperar uma matrícula variando de 0000000 a 9999999. Portanto, precisaríamos dimensionar um vetor com **DEZ Milhões de elementos**

Exemplo de Hashing Perfeito

✎ Para economizar mais ainda: **Hashing**.

Como construir Tabela Hash usando hashing perfeito?

Resp: Identificando as **partes significativas** da chave.



Exemplo de Hashing Perfeito

- ✎ Esta parte mostra a **dimensão** que a **Tabela Hash** deverá ter.
- ✎ Por exemplo, dimensionando com apenas 100 elementos, ou seja, `Aluno* tabAlunos[100]`.

Função que aplicada sobre matrículas de alunos retorna os índices da tabela

Exemplo de Hashing Perfeito (6/6)

- ✎ Supondo que a **turma** seja do 2º semestre de 2005 (código **052**) e do **curso** de Sistemas de Informação (código **35**).

Qual seria a função de hashing **perfeito** !?

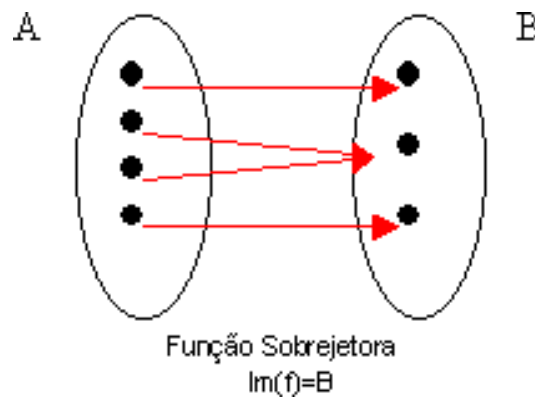
```
int funcao_hash(int matricula) {  
    int valor = matricula - 0523500;  
    if((valor >= 0) & (valor <=99)) then  
        return valor;  
    else  
        return -1;  
}
```

Acesso: dada **mat** ✂ tabAlunos[funcao_hash(mat)]

Hashing Imperfeito

✎ Características:

- Existe chaves x e y diferentes e pertencentes a A , onde a função Hash utilizada fornece **saídas iguais**;



Exemplo de Hashing Imperfeito

- ✎ Suponha que queiramos armazenar as seguintes chaves: **C**, **H**, **A**, **V**, **E** e **S** em um vetor de $P = 7$ posições (0..6) conforme a seguinte

$$\text{função } f(k) = k(\text{código ASCII}) \% P.$$

- ✎ Tabela ASCII: C (67); H (72); A (65); V (86); E (69) e S (83).

Exemplo de Hashing Imperfeito

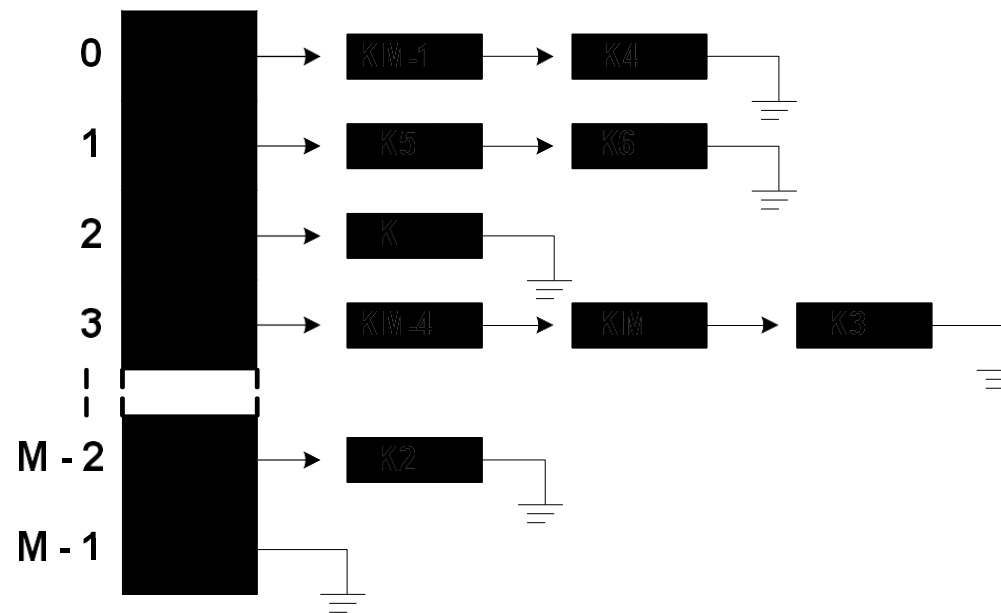
chave	$K = \text{ord}(\text{chave})$	$i_1 = h(K)$
C	67	4
H	72	2
A	65	2
V	86	2
E	69	6
S	83	6

Colisões

- ✎ Quando duas ou mais chaves geram o mesmo endereço da Tabela Hash, dizemos que houve uma **colisão**.
- ✎ É comum ocorrer colisões.
- ✎ Principais causas:
 - em geral o número N de chaves possíveis é **muito maior** que o número de **entradas** disponíveis na tabela.
 - **não** se pode garantir que as funções de hashing possuam um **bom potencial de distribuição (espalhamento)**.

Encadeamento

 **Característica Principal:** A informação é armazenada em estruturas encadeadas



Encadeamento

✎ P = 7 posições (0..6) e a **função** $f(k) = k(\text{código ASCII}) \% P$.

