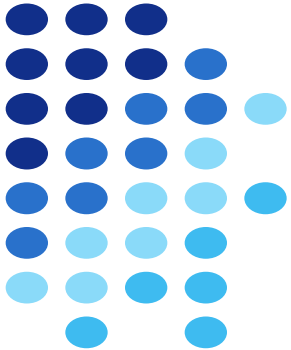
Universidade Federal de Sergipe
Departamento de Sistemas de Informação

SINF0007 – Estrutura de Dados II

Tabelas Hash (hash tables, hashing ou tabela de espalhamento)



5

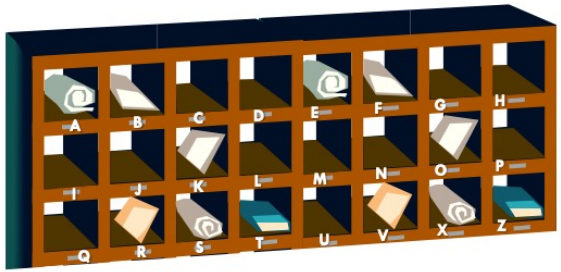
Prof. Dr. Raphael Pereira de Oliveira
raphael.oliveira@academico.ufs.br

Motivação

- Alternativas para acelerar buscas em grandes volumes de dados:
  - Usar um índice (ex. Árvore B, Árvore B+)
  - Usar cálculo de endereço para acessar diretamente o registro procurado em O(1) ➔ **Tabelas Hash**

Exemplo Motivador

- Distribuição de correspondências de funcionários numa empresa
  - Um escaninho para cada inicial de sobrenome
  - Todos os funcionários com a mesma inicial de sobrenome procuram sua correspondência dentro do mesmo escaninho
  - Pode haver mais de uma correspondência dentro do mesmo escaninho



Hashing: Princípio e Funcionamento

- Suponha que existem **n** chaves a serem armazenadas numa tabela de comprimento **m**
  - Em outras palavras, a tabela tem **m** compartimentos (*buckets*)
  - Endereços possíveis: [**0**, **m−1**]
  - Situações possíveis: cada compartimento da tabela pode armazenar **x** registros
  - Para simplificar, assumimos que **x = 1** (cada compartimento armazena apenas **1** registro)

Como Determinar M?

- Uma opção é determinar **m** em função do número de valores possíveis das chaves a serem armazenadas

Hashing: Princípio e Funcionamento

- Se os valores das chaves variam de [**0**, **m−1**], então podemos usar o valor da chave para definir o endereço do compartimento onde o registro será armazenado

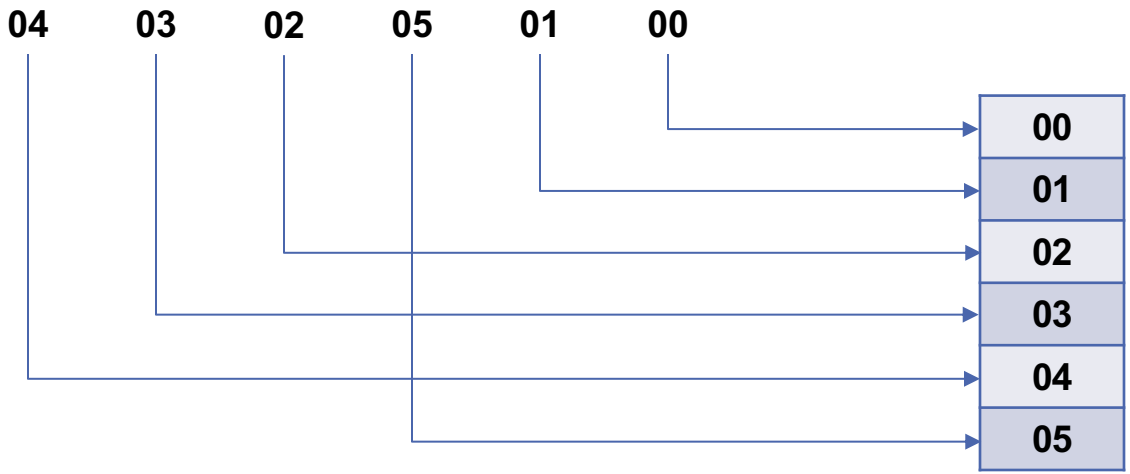
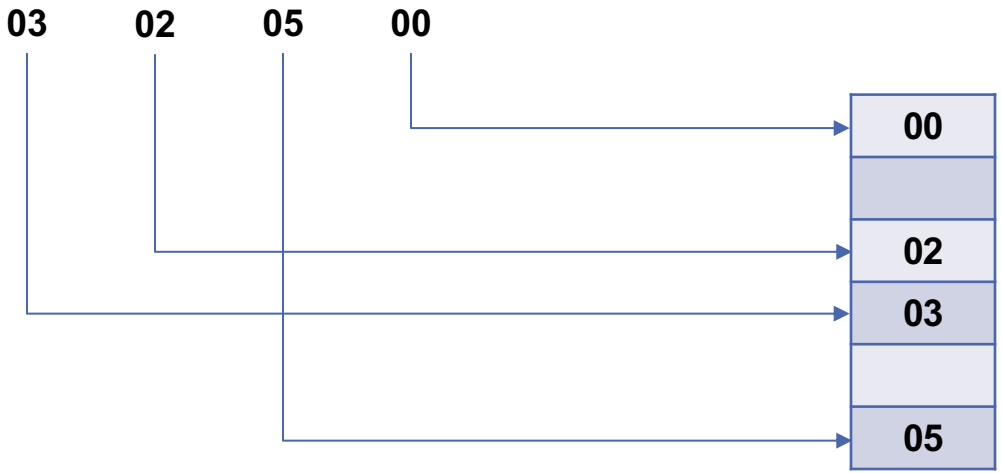


Tabela pode ter espaços vazios

- Se o número **n** de chaves a armazenar é menor que o número de compartimentos **m** da tabela



Mas...

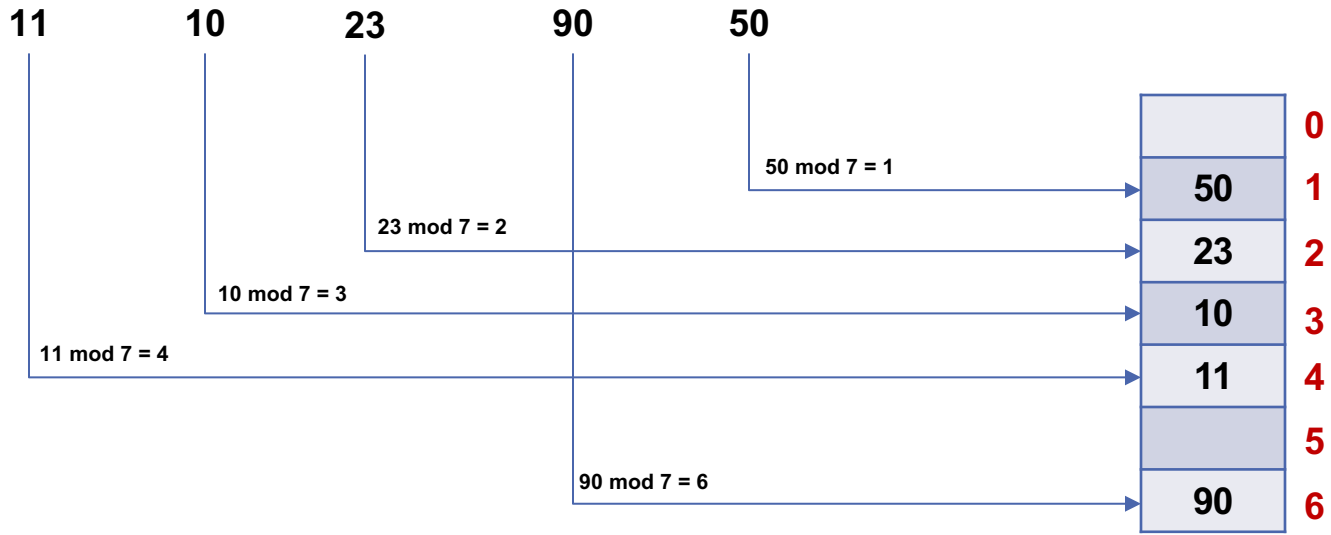
- Se o intervalo de valores de chave é muito grande, **m** é muito grande
- Pode haver um número proibitivo de espaços vazios na tabela se houver poucos registros
- Exemplo: armazenar 2 registros com chaves 0 e 999.999 respectivamente
  - **m** = 1.000.000
  - tabela teria 999.998 compartimentos vazios

Solução

- Definir um valor de **`m`** menor que os valores de chaves possíveis
- Usar uma função **`hash h`** que mapeia um valor de chave **`x`** para um endereço da tabela
- Se o endereço **`h(x)`** estiver livre, o registro é armazenado no compartimento apontado por **`h(x)`**
- Diz-se que **`h(x)`** produz um endereço-base para **`x`**

Exemplo

h(x) = x mod 7



Função Hash H

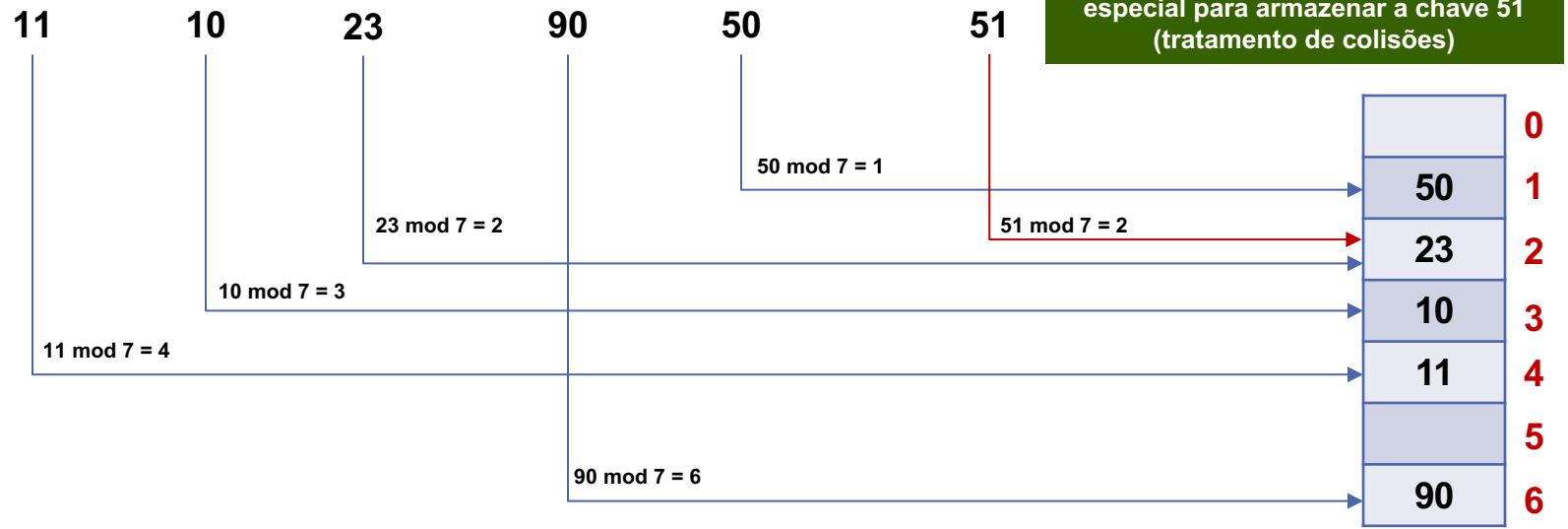
- Infelizmente, a função pode não garantir injetividade, ou seja, é possível que **`x ≠ y`** e **`h(x) = h(y)`**
- Se ao tentar inserir o registro de chave **`x`** o compartimento de endereço **`h(x)`** já estiver ocupado por **`y`**, ocorre uma **`colisão`**
  - Diz-se que **`x`** e **`y`** são sinônimos em relação a **`h`**

Exemplo

$h(x) = x \bmod 7$

A chave 51 colide com a chave 23 e não pode ser inserida no endereço 2!

Solução: uso de um procedimento especial para armazenar a chave 51 (tratamento de colisões)



Características Desejáveis das Funções de Hash

1. Produzir um **número baixo de colisões**
2. Ser **facilmente computável**
3. Ser **uniforme**

Características Desejáveis das Funções de Hash

1. Produzir um **número baixo de colisões**
   - Difícil, pois depende da distribuição dos valores de chave
   - Exemplo: Pedidos que usam o ano e mês do pedido como parte da chave
     - Se a função **h** realçar estes dados, haverá muita concentração de valores nas mesmas faixas

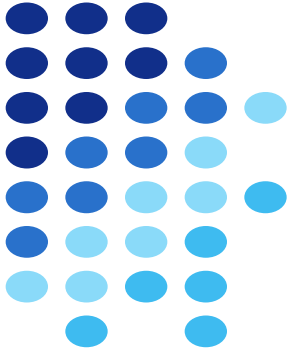
Características Desejáveis das Funções de Hash

2. Ser **facilmente computável**
   - Se a tabela estiver armazenada em disco (nosso caso), isso não é tão crítico
   - Das 3 condições, é a mais fácil de ser garantida
3. Ser **uniforme**
   - Idealmente, a função `h` deve ser tal que todos os compartimentos possuam a mesma probabilidade de serem escolhidos
   - Difícil de testar na prática

Exemplos de Funções de Hash

- Algumas funções de **`hash`** são bastante empregadas na prática por possuírem algumas das características anteriores:
  - Método da Divisão
  - Método da Dobra
  - Método da Multiplicação

Método da Divisão



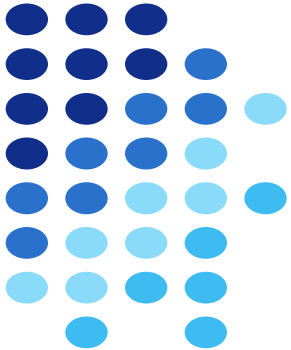
Método da Divisão

- Uso da função **mod**:
  - **h(x) = x mod m**
  - onde **m** é a dimensão da tabela
- Alguns valores de **m** são melhores do que outros
  - Exemplo: se **m** for par, então **h(x)** será par quando **x** for par, e ímpar quando **x** for ímpar => **indesejável**

Método da Divisão

- Estudos apontam bons valores de **m**:
  - Escolher **m** de modo que seja um número primo não próximo a uma potência de 2; ou
  - Escolher **m** tal que não possua divisores primos menores do que 20

Método da Dobra



Método da Dobra

- Suponha a chave como uma sequencia de dígitos escritos em um pedaço de papel
- O método da dobra consiste em “dobrar” este papel, de maneira que os dígitos se superponham
- Os dígitos então devem ser somados, sem levar em consideração o “vai-um”

Exemplo: Método da Dobra

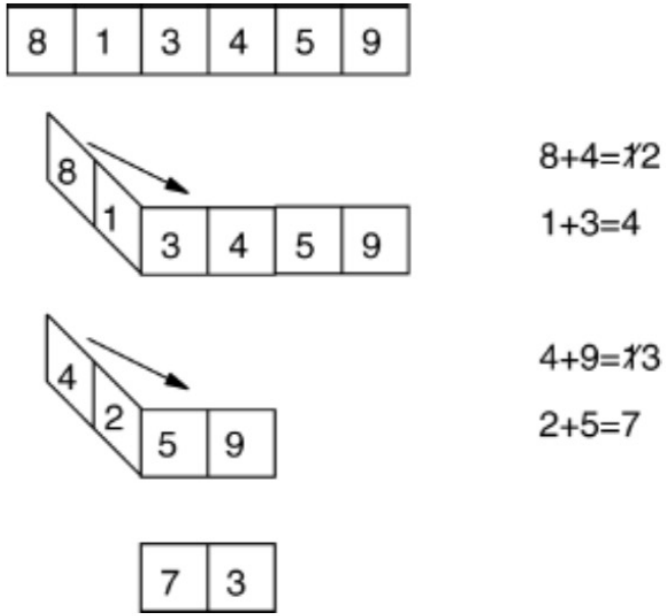


FIGURA 10.4 Método da dobra.

Livro: Jayme Luiz Szwarcfiter e Lilian Markenzon - ***Estruturas de Dados e seus Algoritmos*** *(2010)*

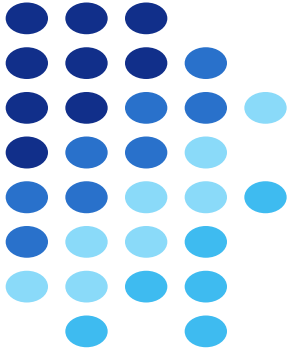
Método da Dobra

- A posição onde a dobra será realizada, e quantas dobras serão realizadas, depende de quantos dígitos são necessários para formar o endereço base
- O tamanho da dobra normalmente é do tamanho do endereço que se deseja obter

Para Treinar...

- Escreva uma função em C que implementa o método da dobra, de forma a obter endereços de 2 dígitos
- Assuma que as chaves possuem 6 dígitos

Método da Multiplicação



Método da Multiplicação

- Multiplicar a chave por ela mesma
- Armazenar o resultado numa palavra de **b bits**
- Descartar os bits das extremidades direita e esquerda, um a um, até que o resultado tenha o tamanho de endereço desejado

Método da Multiplicação

- Exemplo: chave **12**
  - **12** x **12** = **144**
  - **144** representado em binário: **10010000**
  - Armazenar em **10 bits**: **0010010000**
  - Obter endereço de **6 bits** (endereços entre 0 e 63)

0	0	1	0	0	1	0	0	0	0

Método da Multiplicação

- Exemplo: chave **12**
  - **12** x **12** = **144**
  - **144** representado em binário: **10010000**
  - Armazenar em **10 bits**: **0010010000**
  - Obter endereço de **6 bits** (endereços entre 0 e 63)

Método da Multiplicação

- Exemplo: chave **12**
  - **12** x **12** = **144**
  - **144** representado em binário: **10010000**
  - Armazenar em **10 bits**: **0010010000**
  - Obter endereço de **6 bits** (endereços entre 0 e 63)

Método da Multiplicação

- Exemplo: chave **12**
  - **12** x **12** = **144**
  - **144** representado em binário: **10010000**
  - Armazenar em **10 bits**: **0010010000**
  - Obter endereço de **6 bits** (endereços entre 0 e 63)

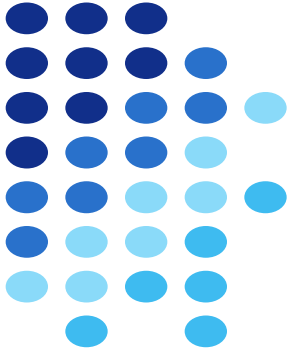
Método da Multiplicação

- Exemplo: chave **12**
  - **12** x **12** = **144**
  - **144** representado em binário: **10010000**
  - Armazenar em **10 bits**: **0010010000**
  - Obter endereço de **6 bits** (endereços entre 0 e 63)

Método da Multiplicação

- Exemplo: chave **12**
  - **12** x **12** = **144**
  - **144** representado em binário: **10010000**
  - Armazenar em **10 bits**: **0010010000**
  - Obter endereço de **6 bits** (endereços entre 0 e 63)

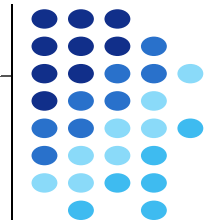
Uso da Função de Hash



Uso da Função de Hash

- A mesma função de hash usada para inserir os registros é usada para buscar os registros

Exemplo: Busca de Registro por Chave



`h(x) = x mod 7`

0	
1	50
2	23
3	10
4	11
5	
6	90

- Encontrar o registro de chave 90
  - **`90 mod 7 = 6`**
- Encontrar o registro de chave 7
  - **`7 mod 7 = 0`**
  - Compartimento 0 está vazio: registro não está armazenado na tabela
- Encontrar o registro de chave 8
  - **`8 mod 7 = 1`**
  - Compartimento 1 tem um registro com chave diferente da chave buscada, e não existem registros adicionais: registro não está armazenado na tabela

E se...

`h(x) = x mod 7`

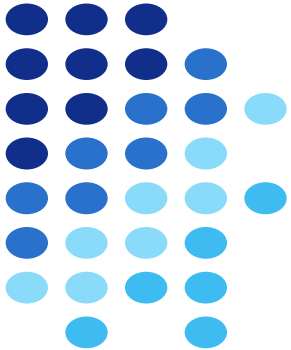
0	
1	50
2	23
3	10
4	11
5	
6	90

- Inserir o registro de chave 30
  - `30 mod 7 = 2`
  - Compartimento 2 está ocupado! O que fazer?

Rodar o código **`1.hash-basico`**

É necessário **tratar colisões**

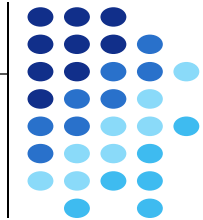
Tratamento de Colisões



Fator de Carga

- O fator de carga de uma tabela hash é $\alpha = n/m$, onde n é o número de registros armazenados na tabela
  - O número de colisões cresce rapidamente quando o fator de carga aumenta
  - Uma forma de diminuir as colisões é diminuir o fator de carga
  - Mas isso não resolve o problema: colisões sempre podem ocorrer

Então, como tratar as colisões?



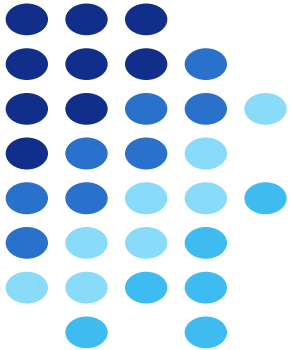
Tratamento de Colisões

- Por Encadeamento (exterior e interior)
- Por Endereçamento Aberto

Tratamento de Colisões

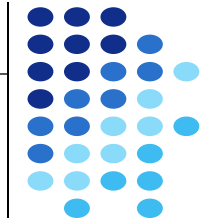
- **Por Encadeamento (exterior e interior)**
- Por Endereçamento Aberto

Tratamento de Colisões por Encadeamento Exterior



Encadeamento Exterior

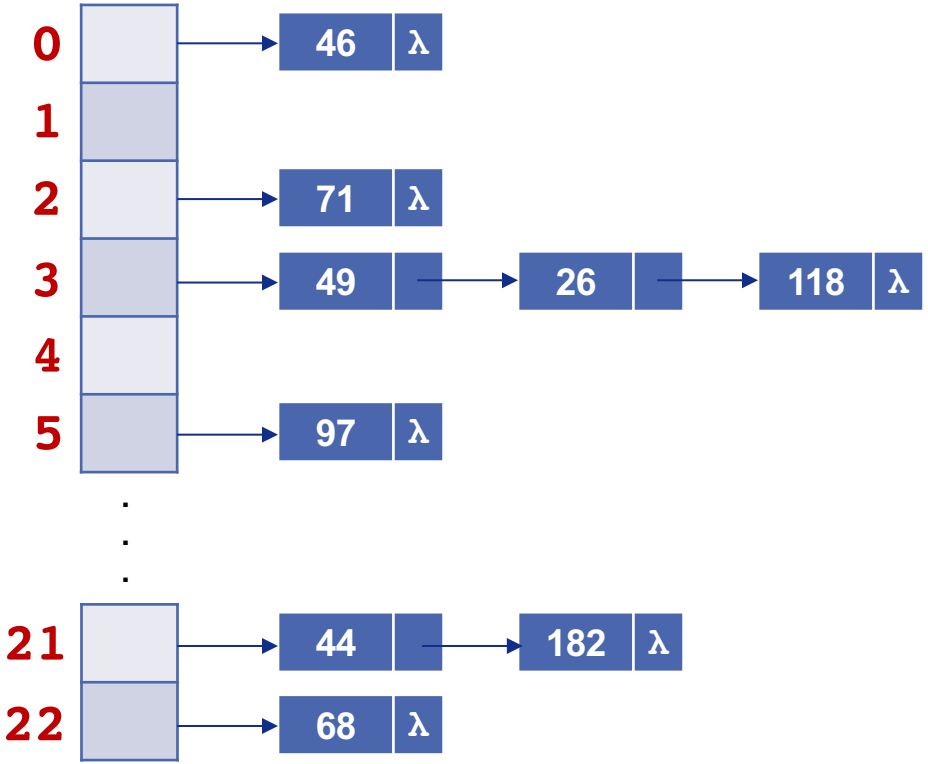
- Manter **m** listas encadeadas, uma para cada possível endereço base
- A tabela base não possui nenhum registro, apenas os ponteiros para as listas encadeadas
- Por isso chamamos de **encadeamento exterior**: a tabela base não armazena nenhum registro



Nós da Lista Encadeada

- Cada nó da lista encadeada contém:
  - um registro
  - um ponteiro para o próximo nó

Exemplo: Encadeamento Exterior



$$h(x) = x \bmod 23$$

Busca em Tabela Hash com Encadeamento Exterior

- Busca por um registro de chave `x`:
  1. Calcular o endereço aplicando a função `h(x)`
  2. Percorrer a lista encadeada associada ao endereço
  3. Comparar a chave de cada nó da lista encadeada com a chave `x`, até encontrar o nó desejado
  4. Se final da lista for atingido, registro não está lá

Inserção em Tabela Hash com Encadeamento Exterior

- Inserção de um registro de chave **`x`**:
  1. Calcular o endereço aplicando a função **`h(x)`**
  2. Buscar registro na lista associada ao endereço **`h(x)`**
  3. Se registro for encontrado, sinalizar erro
  4. Se o registro não for encontrado, inserir no final da lista

Exclusão em Tabela Hash com Encadeamento Exterior

- Exclusão de um registro de chave **`x`**:
  1. Calcular o endereço aplicando a função **`h(x)`**
  2. Buscar registro na lista associada ao endereço **`h(x)`**
  3. Se registro for encontrado, excluir registro
  4. Se o registro não for encontrado, sinalizar erro

Rodar o código
`2.hash-encadeamento-exterior`

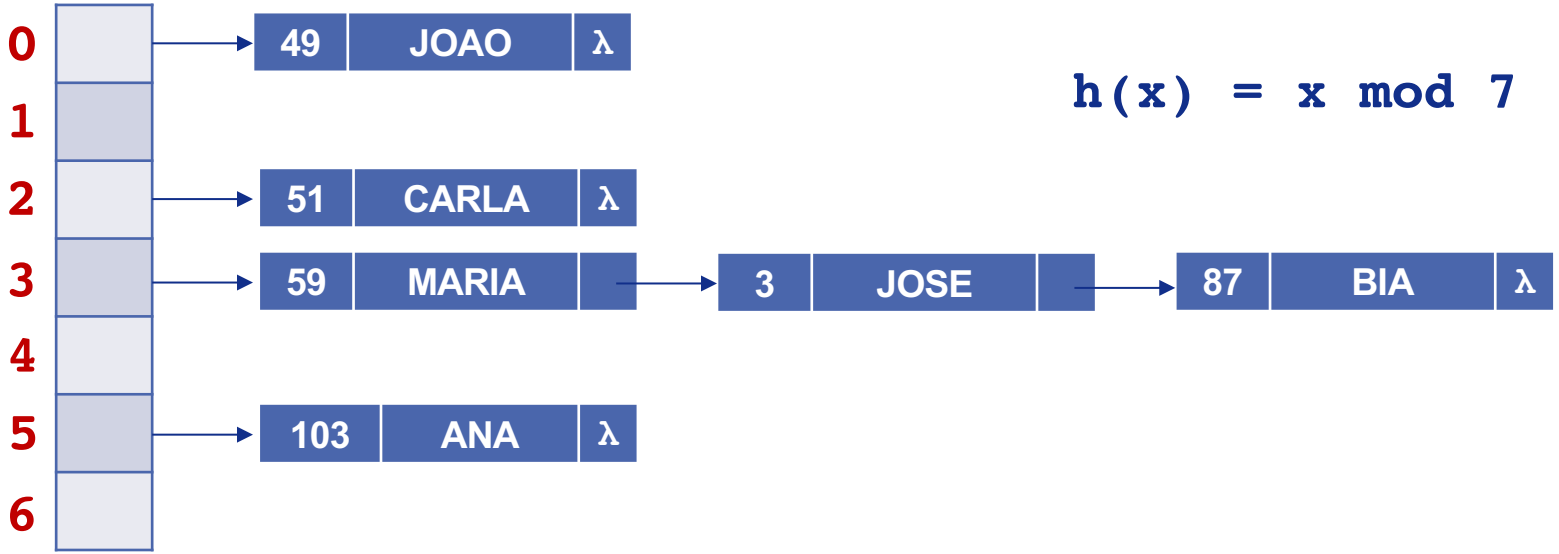
Complexidade no Pior Caso

- É necessário percorrer uma lista encadeada até o final para concluir que a chave não está na tabela
- Comprimento de uma lista encadeada pode ser O(n)
- Complexidade no pior caso: **O(n)**

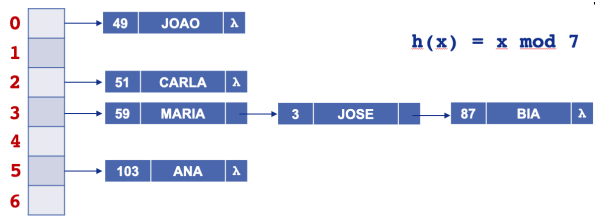
Implementação em Disco

- Normalmente, usa-se um arquivo para armazenar os compartimentos da tabela, e outro para armazenar as listas encadeadas
- Ponteiros para NULL são representados por -1

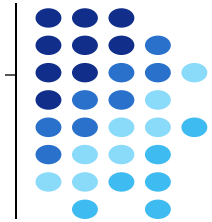
Exemplo


0
1
2
3
4
5
6
49
JOAO
λ
51
CARLA
λ
59
MARIA
3
JOSE
87
BIA
λ
103
ANA
λ
h(x) = x mod 7

Estrutura dos Arquivos



Inserir 49



Arquivo `clientes.dat`
(cliente)

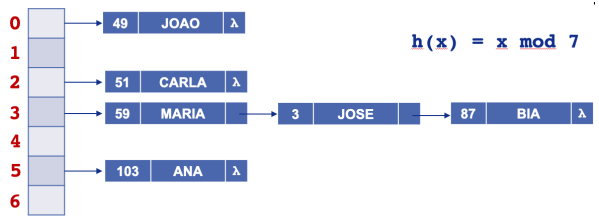
	CodCliente	Nome	Prox	Ocupado
0				
1				
2				
3				
4				
5				
6				
7				
...				

Arquivo `tabHash.dat`
(compartimento_hash)

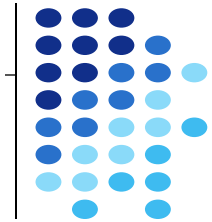
0	-1
1	-1
2	-1
3	-1
4	-1
5	-1
6	-1

`m = 7`

Estrutura dos Arquivos



Inserir 49



Arquivo `clientes.dat`
(cliente)

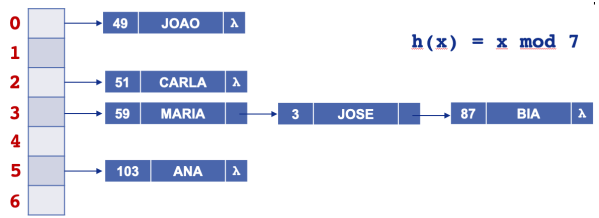
	CodCliente	Nome	Prox	Ocupado
0	49	JOAO	-1	TRUE
1				
2				
3				
4				
5				
6				
7				
...				

Arquivo `tabHash.dat`
(compartimento_hash)

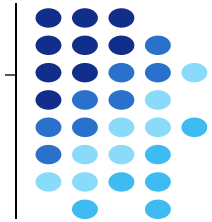
0	0
1	-1
2	-1
3	-1
4	-1
5	-1
6	-1

`m = 7`

Estrutura dos Arquivos



Inserir 59



Arquivo `clientes.dat`
(cliente)

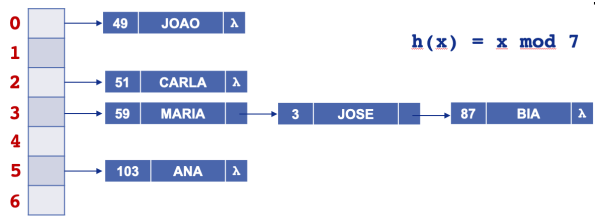
	CodCliente	Nome	Prox	Ocupado
0	49	JOAO	-1	TRUE
1				
2				
3				
4				
5				
6				
7				
...				

Arquivo `tabHash.dat`
(compartimento_hash)

0	0
1	-1
2	-1
3	-1
4	-1
5	-1
6	-1

`m = 7`

Estrutura dos Arquivos



Inserir 59

Arquivo `clientes.dat`
(cliente)

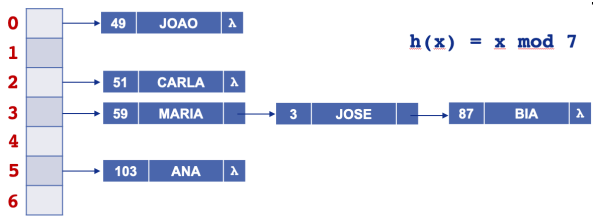
	CodCliente	Nome	Prox	Ocupado
0	49	JOAO	-1	TRUE
1	59	MARIA	-1	TRUE
2				
3				
4				
5				
6				
7				
...				

Arquivo `tabHash.dat`
(compartimento_hash)

0	0
1	-1
2	-1
3	1
4	-1
5	-1
6	-1

`m = 7`

Estrutura dos Arquivos



Inserir 103

Arquivo `clientes.dat`
(cliente)

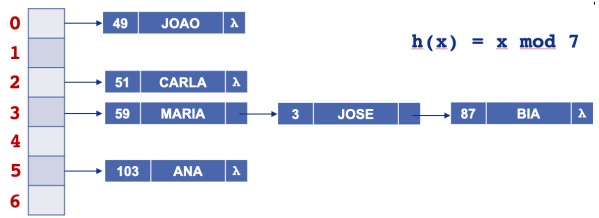
	CodCliente	Nome	Prox	Ocupado
0	49	JOAO	-1	TRUE
1	59	MARIA	-1	TRUE
2				
3				
4				
5				
6				
7				
...				

Arquivo `tabHash.dat`
(compartimento_hash)

0	0
1	-1
2	-1
3	1
4	-1
5	-1
6	-1

`m = 7`

Estrutura dos Arquivos



Inserir 103

Arquivo `clientes.dat`
(cliente)

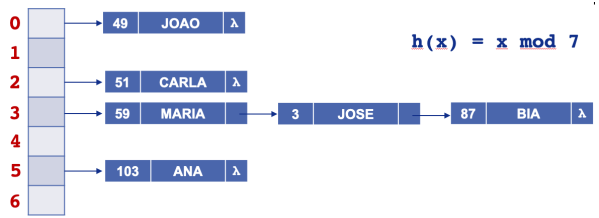
	CodCliente	Nome	Prox	Ocupado
0	49	JOAO	-1	TRUE
1	59	MARIA	-1	TRUE
2	103	ANA	-1	TRUE
3				
4				
5				
6				
7				
...				

Arquivo `tabHash.dat`
(compartimento_hash)

0	0
1	-1
2	-1
3	1
4	-1
5	2
6	-1

`m = 7`

Estrutura dos Arquivos



Inserir 3

Arquivo `clientes.dat`
(cliente)

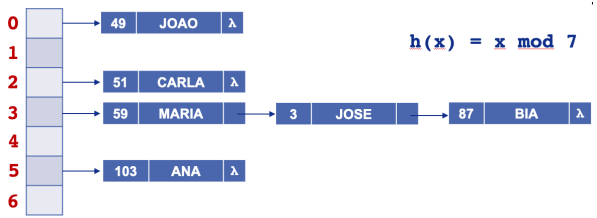
	CodCliente	Nome	Prox	Ocupado
0	49	JOAO	-1	TRUE
1	59	MARIA	-1	TRUE
2	103	ANA	-1	TRUE
3				
4				
5				
6				
7				
...				

Arquivo `tabHash.dat`
(compartimento_hash)

0	0
1	-1
2	-1
3	1
4	-1
5	2
6	-1

`m = 7`

Estrutura dos Arquivos



Inserir 3

Arquivo `clientes.dat`
(cliente)

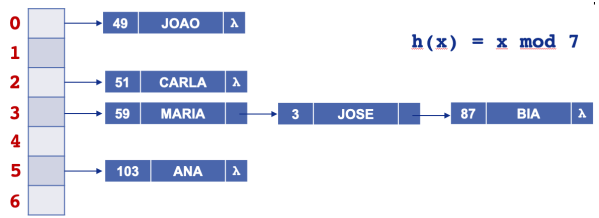
	CodCliente	Nome	Prox	Ocupado
0	49	JOAO	-1	TRUE
1	59	MARIA	3	TRUE
2	103	ANA	-1	TRUE
3	3	JOSE	-1	TRUE
4				
5				
6				
7				
...				

Arquivo `tabHash.dat`
(compartimento_hash)

0	0
1	-1
2	-1
3	1
4	-1
5	2
6	-1

`m = 7`

Estrutura dos Arquivos



Inserir 51

Arquivo `clientes.dat`
(cliente)

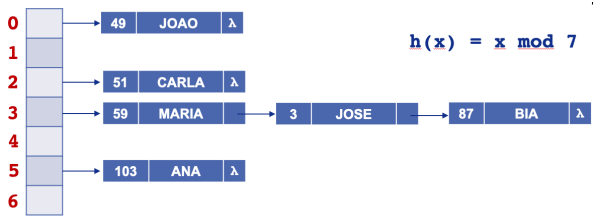
	CodCliente	Nome	Prox	Ocupado
0	49	JOAO	-1	TRUE
1	59	MARIA	3	TRUE
2	103	ANA	-1	TRUE
3	3	JOSE	-1	TRUE
4				
5				
6				
7				
...				

Arquivo `tabHash.dat`
(compartimento_hash)

0	0
1	-1
2	-1
3	1
4	-1
5	2
6	-1

`m = 7`

Estrutura dos Arquivos



Inserir 51

Arquivo `clientes.dat`
(cliente)

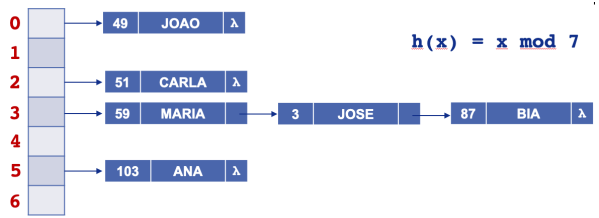
	CodCliente	Nome	Prox	Ocupado
0	49	JOAO	-1	TRUE
1	59	MARIA	3	TRUE
2	103	ANA	-1	TRUE
3	3	JOSE	-1	TRUE
4	51	CARLA	-1	TRUE
5				
6				
7				
...				

Arquivo `tabHash.dat`
(compartimento_hash)

0	0
1	-1
2	4
3	1
4	-1
5	2
6	-1

`m = 7`

Estrutura dos Arquivos



Inserir 87

Arquivo `clientes.dat`
(cliente)

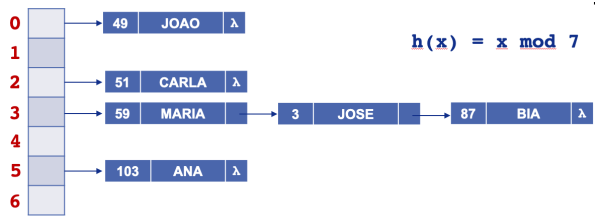
	CodCliente	Nome	Prox	Ocupado
0	49	JOAO	-1	TRUE
1	59	MARIA	3	TRUE
2	103	ANA	-1	TRUE
3	3	JOSE	-1	TRUE
4	51	CARLA	-1	TRUE
5				
6				
7				
...				

Arquivo `tabHash.dat`
(compartimento_hash)

0	0
1	-1
2	4
3	1
4	-1
5	2
6	-1

`m = 7`

Estrutura dos Arquivos



Inserir 87

Arquivo `clientes.dat`
(cliente)

	CodCliente	Nome	Prox	Ocupado
0	49	JOAO	-1	TRUE
1	59	MARIA	3	TRUE
2	103	ANA	-1	TRUE
3	3	JOSE	5	TRUE
4	51	CARLA	-1	TRUE
5	87	BIA	-1	TRUE
6				
7				
...				

Arquivo `tabHash.dat`
(compartimento_hash)

0	0
1	-1
2	4
3	1
4	-1
5	2
6	-1

`m = 7`

Uso de Flag Indicador de Status

- Para facilitar a manutenção da lista encadeada, pode-se adicionar um **flag** indicador de **status** a cada registro
- No exemplo do slide anterior, esse **flag** é chamado **ocupado**
- O **flag ocupado** pode ter os seguintes valores:
  - **`TRUE`**: quando o compartimento tem um registro
  - **`FALSE`**: quando o registro que estava no compartimento foi excluído

Reflexão:

- Como seriam os procedimentos para inclusão e exclusão?

Implementação de Exclusão

- Ao excluir um registro, marca-se o **flag** de **ocupado** como **`FALSE`** (ou seja, marca-se que o compartimento está liberado para nova inserção)

Implementação de Inserção (Opção 1)

- Para inserir novo registro
  - Inserir o registro no final da lista encadeada, se ele já não estiver na lista
  - De tempos em tempos, re-arrumar o arquivo para ocupar as posições onde o **flag** de **ocupado** é **`FALSE`**

Implementação de Inserção (Opção 2)

- Para inserir novo registro
  - Ao passar pelos registros procurando pela chave, guardar o endereço **p** do primeiro nó marcado como **LIBERADO** (**flag ocupado** = **FALSE**)
  - Se ao chegar ao final da lista encadeada, a chave não for encontrada, gravar o registro na posição **p**
  - Atualizar ponteiros
    - Nó anterior deve apontar para o registro inserido
    - Nó inserido deve apontar para nó que era apontado pelo nó anterior

Referências

- Material baseado nos slides de **Vanessa Braganholo**, Disciplina de Estruturas de Dados e Seus Algoritmos. Instituto de Computação. Universidade Federal Fluminense (UFF), Niterói, Brasil.
- Szwarcfiter, J.; Markezon, L. Estruturas de Dados e seus Algoritmos, 3a. ed. LTC. Cap. 10
- Inhaúma Neves Ferraz. Programação Com Arquivos. 2003. Editora: manole.
- Schildt, H. C Completo e Total. Ed. McGraw-Hill.