TABELA HASHING

Prof. Marcelo Dib

Hashing
(Tabelas de Dispersão)
Fonte de consulta: Szwarcfiter, J.; Markezon, L.
Estruturas deDados e seus Algoritmos, 3a. ed.
Ednaldo Pizzolato

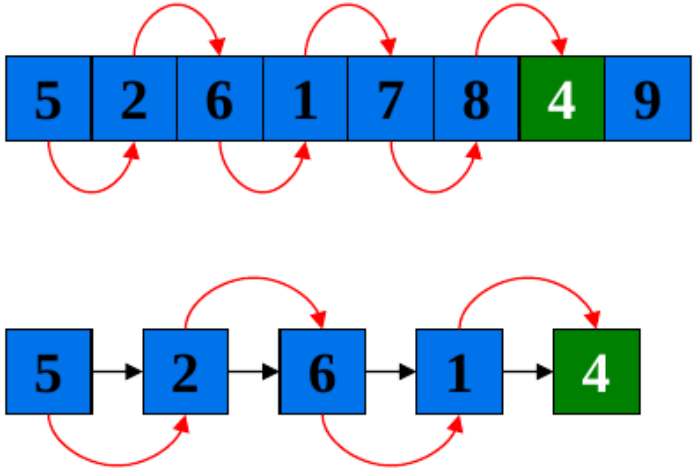
Hashing

- **Motivação**
  - Os Métodos de pesquisa vistos até agora, buscam informações armazenadas com base na comparação de suas chaves;
  - Se quisermos acessar uma informação (e não sua posição) temos que procura-la ;
  - Dada uma tabela com uma chave e vários valores por linha,quero rapidamente procurar, inserir e apagar registros baseados nas suas chaves;
  - Estruturas de busca sequencial/binária levam tempo até encontrar o elemento desejado.

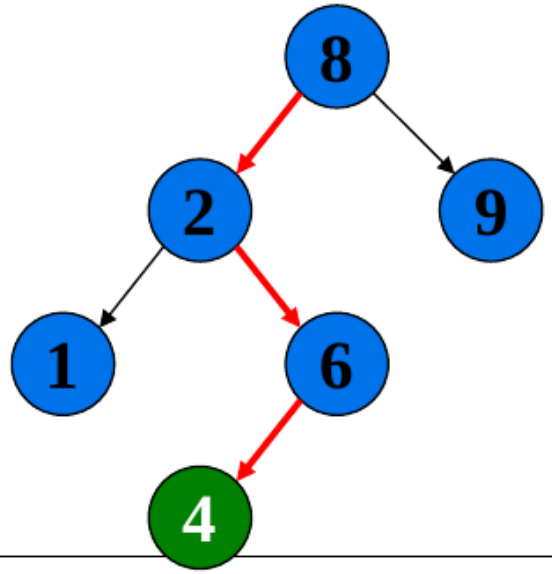
Hashing

- **Motivação**

Ex: Arrays e listas



Ex: Árvores



Hashing

- **Ideal**
  - Parte da informação de uma determinada chave poderia ser utilizada para recuperar a informação;
  - A idéia central é utilizar uma função aplicada sobre parte da informação da chave, para retornar o índice onde a informação deve ou deveria estar armazenada;
  - **OBJETIVO:** a partir de uma chave, fazer uma busca rápida e obter o valor desejado ;

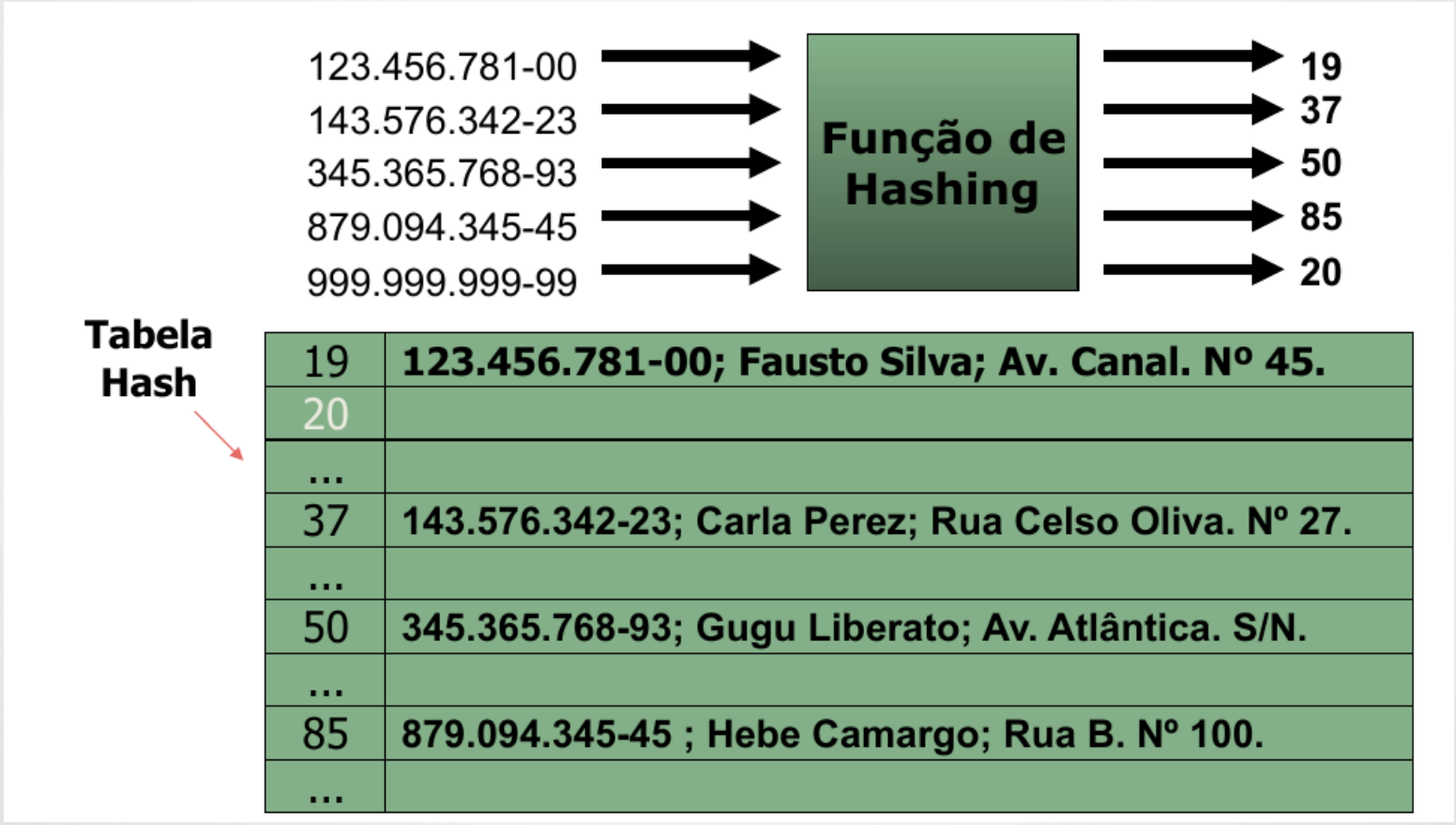
Hashing

- **Proposta**
  - Em lugar de organizarmos a tabela segundo o valor relativo de cada chave em relação às demais, a tabela hash leva em conta somente seu valor absoluto, interpretado como um valor numérico.
  - Através de uma função conveniente, a chave é transformada em um endereço de uma tabela.

Hashing

- **Princípio de Funcionamento**
  - Suponha que existam n chaves a serem armazenadas em uma tabela T, seqüêncial e de dimensão m (i.e. [0..m-1]). Cada posição corresponde a um endereço e pode armazenar r nós distintos.
  - Obs.: quando a tabela se encontra em memória, é comum que cada endereço armazene um único nó. Já quando a tabela está em disco a organização pode ser diferente.

Hashing

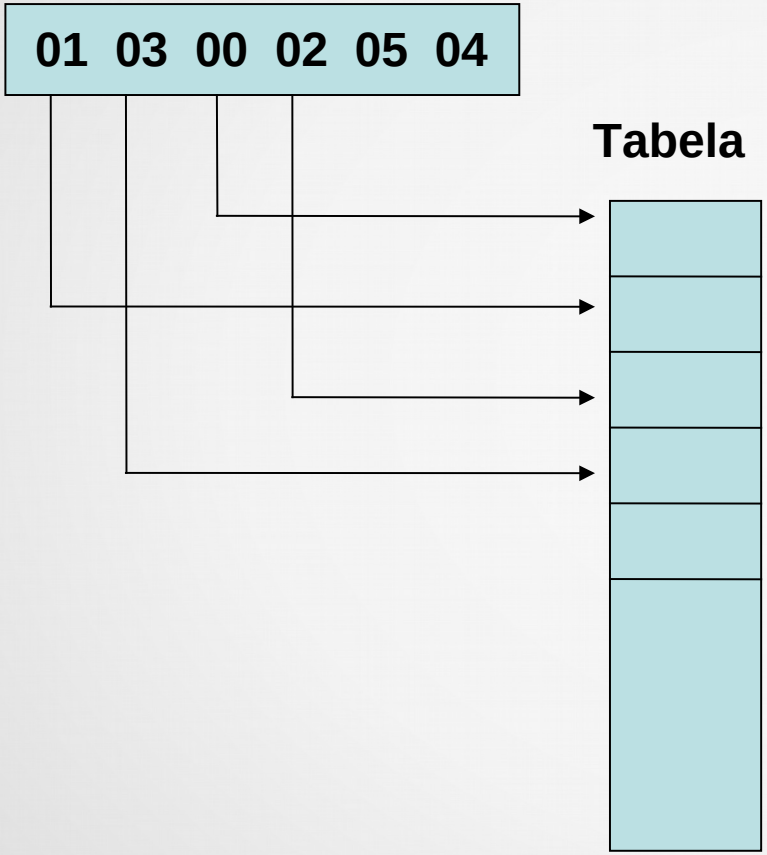
- 
123.456.781-00
143.576.342-23
345.365.768-93
879.094.345-45
999.999.999-99
Função de Hashing
19
37
50
85
20
Tabela Hash
19 | 123.456.781-00; Fausto Silva; Av. Canal. Nº 45.
20 |
... |
37 | 143.576.342-23; Carla Perez; Rua Celso Oliva. Nº 27.
... |
50 | 345.365.768-93; Gugu Liberato; Av. Atlântica. S/N.
... |
85 | 879.094.345-45 ; Hebe Camargo; Rua B. Nº 100.
... |

Hashing

- **Acesso Direto**
  - Quando possuímos uma tabela de tamanho m e o número de chaves é n, tal que n ≤ m e mais, cada chave x é armazenada na posição x da tabela, dizemos que x serve de índice para sua busca, e que o acesso à informação é direto.

Hashing

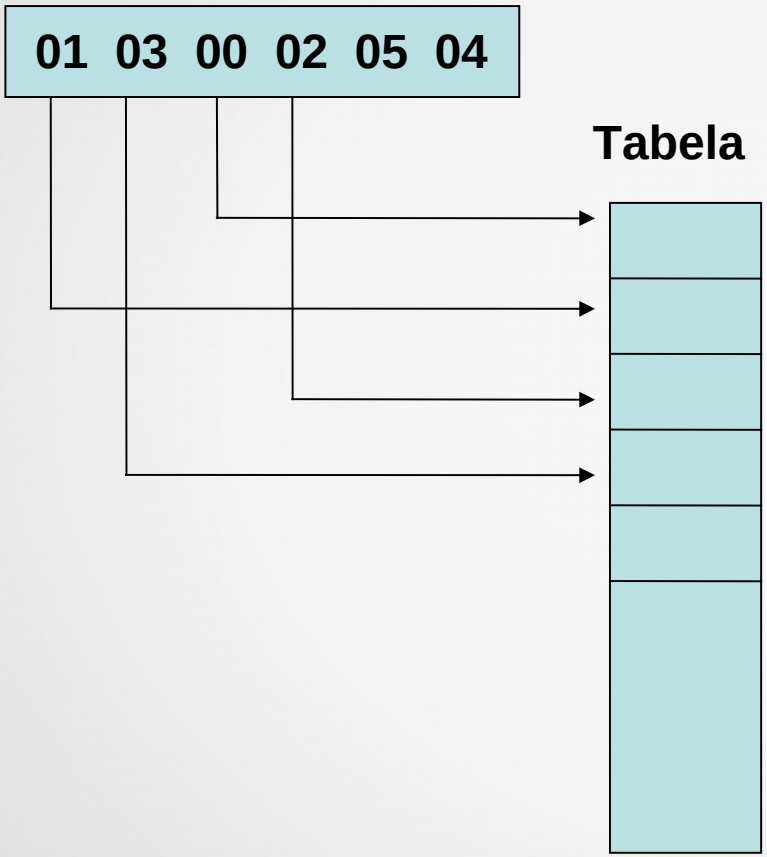
chaves



Problema ?

Hashing

chaves



Como seria o armazenamento de chaves cujos valores estivessem entre 1 e 999999?

Hashing

- **Função Hash**
  - Nem sempre as chaves estão no intervalo [0..m-1], mesmo sendo o número de chaves menor que m.
  - Por isso, devemos transformar, de alguma forma, nossa chave x para um valor no intervalo [0..m-1].
  - A função h(x), qual que $0 \leq h(x) \leq m-1$, responsável por esta transformação é chamada de função hash.

Hashing

- **Colisão**
  - Infelizmente, a função h(x) não pode garantir injetividade...
    - Isso significa que podem existir chaves x e y , onde $x \neq y$ tal que $h(x) = h(y)$.
    - Este fenômeno é chamado *colisão*.
    - Dizemos que x e y são sinônimas em relação a h.

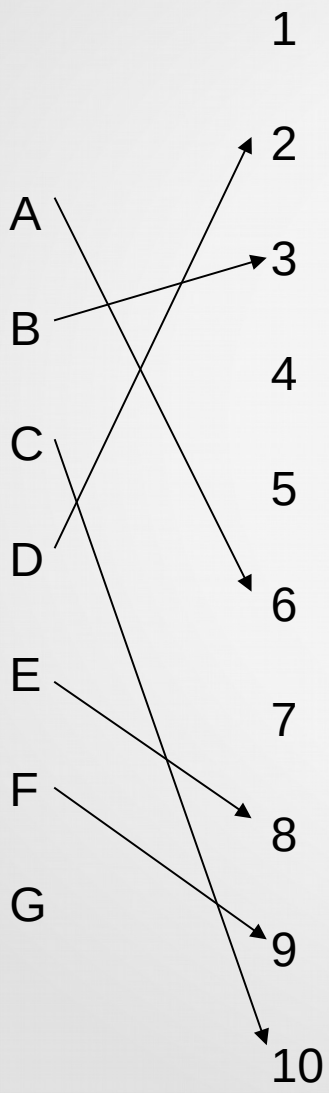
Hashing

- **Custo**
  - Pelo fato de não conhecermos profundamente a função h e, principalmente, não conhecermos previamente as chaves que serão inseridas em uma tabela, podemos apenas determinar limites inferiores e superiores para a complexidade da busca.
  - Estes limites são O(1) (acesso direto) para o melhor caso, sendo o pior caso, porém O(n).

Hashing

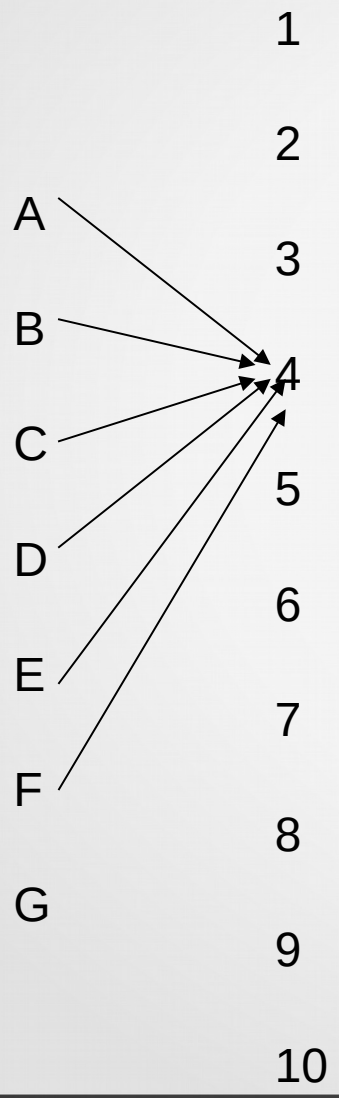
- **Propriedades da Função Hash**
  - Produzir um número baixo de colisões;
  - Ser facilmente computável;
  - Ser uniforme (distribuição equilibrada de probabilidade)

Hashing



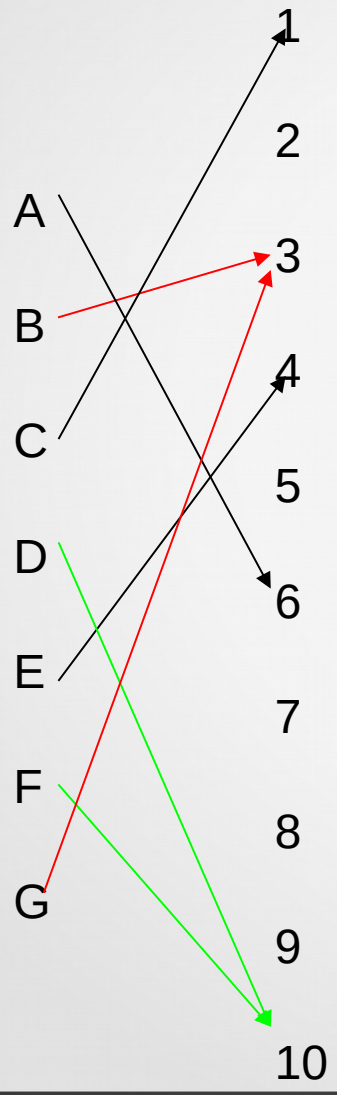
- Caso 1 – Melhor caso.
  - A função hash é perfeita.

Hashing



- Caso 2 – Pior caso.
  - A função hash apresenta muitas colisões.

Hashing



- Caso 3 – Aceitável.
  - A função hash apresenta algumas colisões.

Hashing

- **Funções HASH**
  - Método da divisão : divide-se a chave x pelo tamanho da tabela e o resto da divisão é usado como endereço-base.
    - H(x) = x % m

Hashing

- **Funções HASH**

Exemplo: busca de registro por chave

h(x) = x mod 7

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 exitem registros adicionais: registro não está armazenado na tabela

Hashing

- **Funções HASH**
  - Método da dobra (XOR) : Este método transforma uma chave de k bits ($2^k > m$) em uma chave de j elementos, tal que $2^j \leq m$. Para isso pegamos os k/2 bits da esquerda e realizamos um XOR com os k/2 bits da direita, resultando em uma nova chave de k/s bits. Repetimos o processo até que a chave gerada tenha j bits.

    Exemplo: m = 32, chaves de 10 bits (quanto vale j neste caso?)

    71 = 00010 00111 → 00010 XOR 00111 = 00101 = 5

    46 = 00001 01110 → 00001 XOR 01110 = 01111 = 15

Hashing

- Método da Multiplicação

Multiplicar a chave por ela mesma

- Armazenar o resultado numa palavra de b bits
- Descartar os bits das extremidadades direita e esquerda, um a um, até que o resultado tenha o tamanho de endereço desejado

Hashing

- 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)

Hashing

- **Tratamento de Colisão**

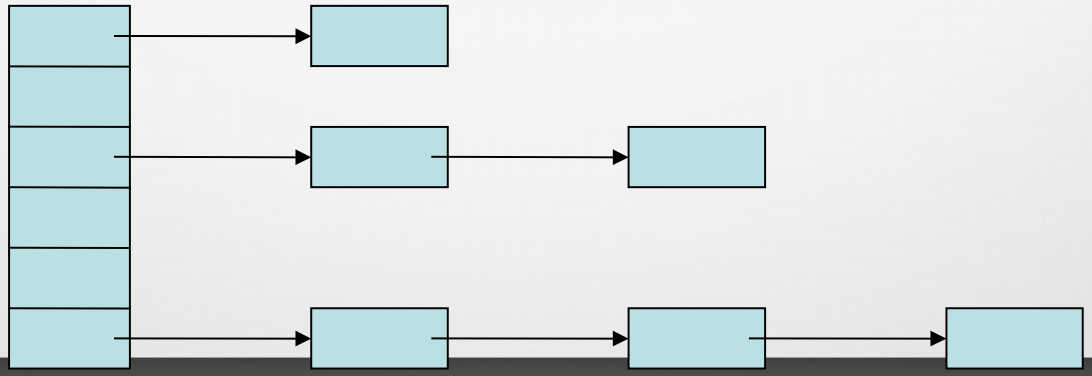
O fator de carga de uma tabela hash é $\alpha = n/m$, onde n é o número de registros armazenados na tabela

- 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
- Como tratar as colisões?

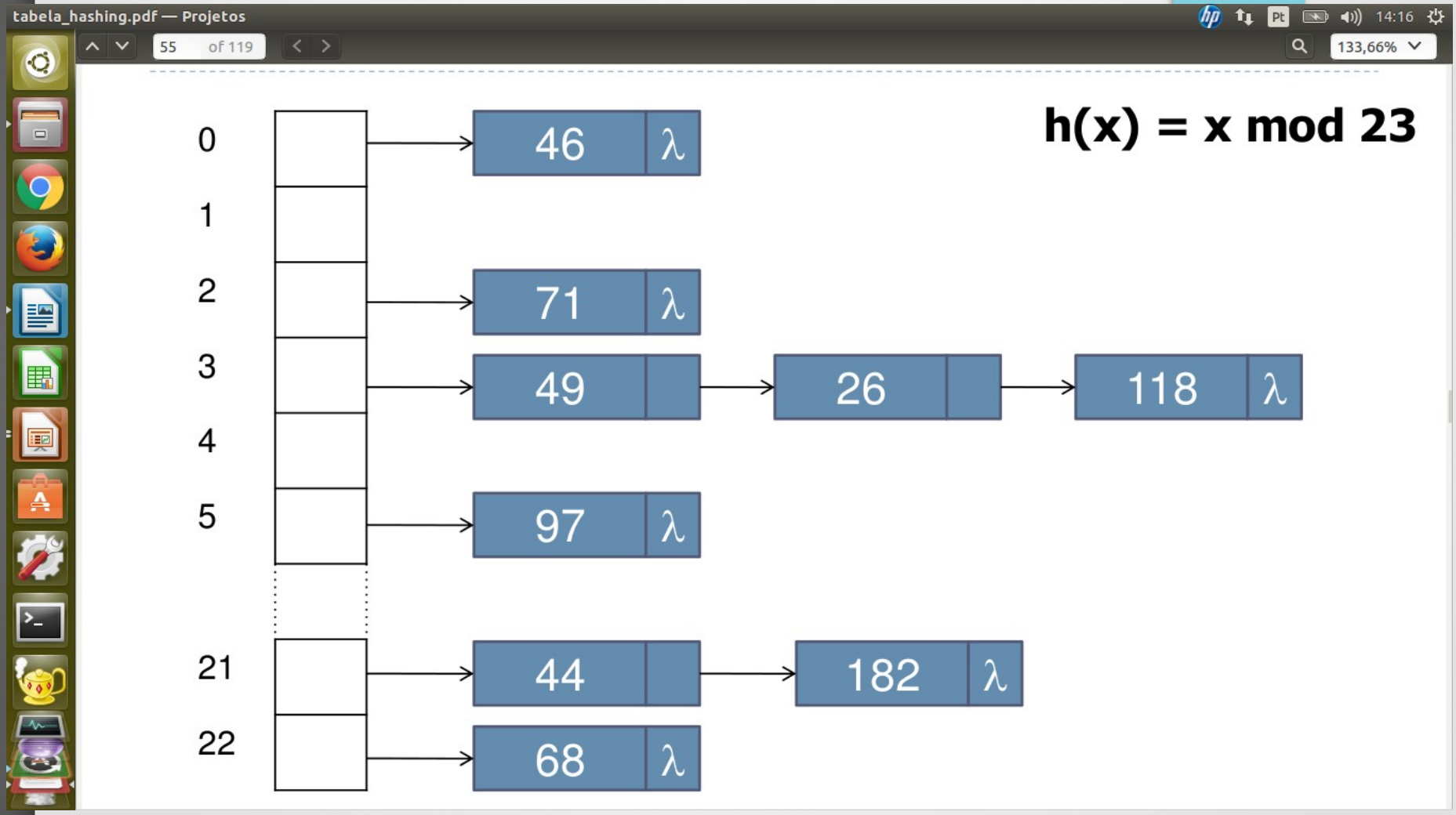
Hashing

- **Encadeamento exterior**

  Cada endereço guarda um ponteiro para uma lista (inicialmente vazia). Cada entrada é colocada na posição final da fila referente a seu endereço. Algoritmos de inserção, remoção e busca são semelhantes aos de uma lista comum.



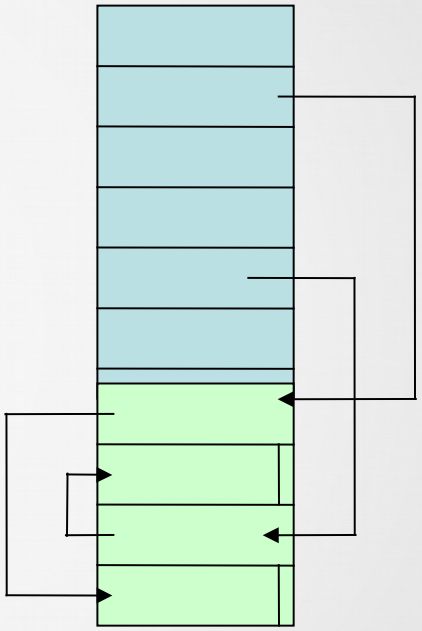
Hashing


tabela_hashing.pdf — Projetos
14:16
55 of 119
133,66%
h(x) = x mod 23
0
1
2
3
4
5
21
22
46
λ
71
λ
49
26
118
λ
97
λ
44
182
λ
68
λ

Hashing

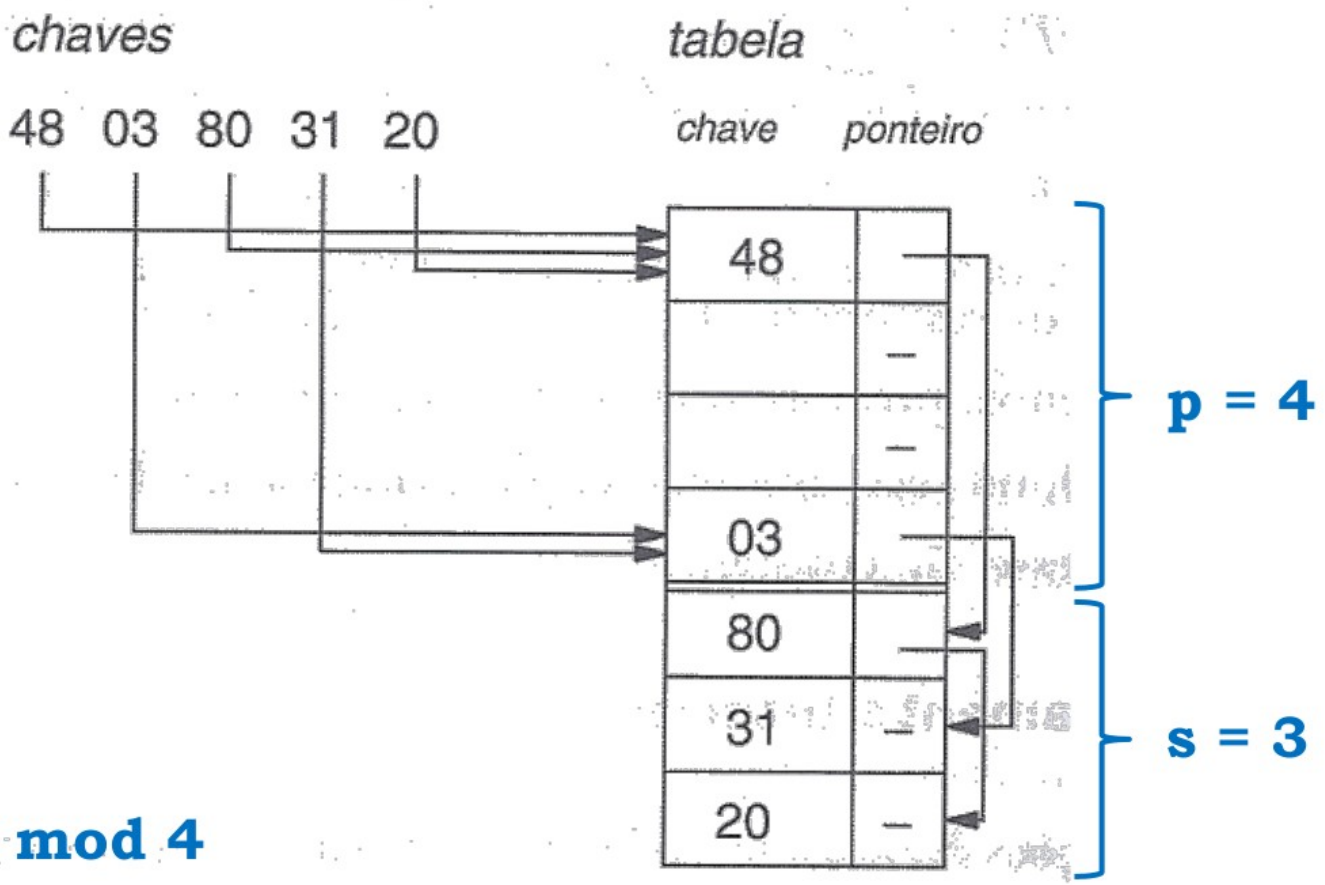
- **Encadeamento interior**

  Em algumas aplicações, não é desejável a manutenção de uma estrutura exterior à tabela hash. Neste caso podemos implementar listas encadeadas, desde que estas compartilhem o mesmo espaço de memória da tabela hash. Desta forma, dividimos a tabela T em uma área de endereços, de tamanho p, e uma área de sinônimos, de tamanho s (s+p = m). Assim, a função h deve gerar valores em [0..p-1] e n/m deve ser menor ou igual a 1.



Problema ?

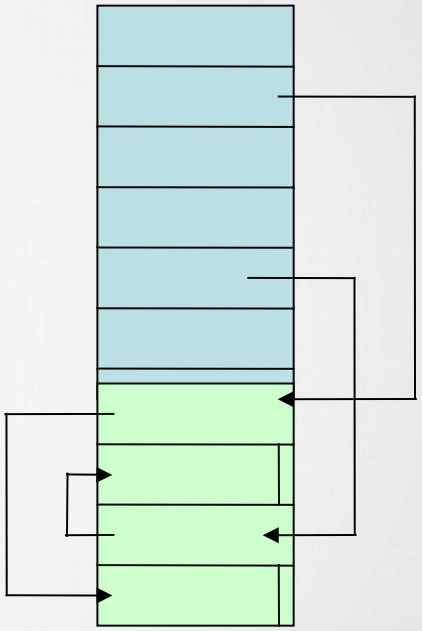
Hashing


chaves
48 03 80 31 20
tabela
chave ponteiro
48
03
80
31
20
p = 4
s = 3
h(x) = x mod 4

Hashing

- **Encadeamento interior**

  Em algumas aplicações, não é desejável a manutenção de uma estrutura exterior à tabela hash. Neste caso podemos implementar listas encadeadas, desde que estas compartilhem o mesmo espaço de memória da tabela hash. Desta forma, dividimos a tabela T em uma área de endereços, de tamanho p, e uma área de sinônimos, de tamanho s (s+p = m). Assim, a função h deve gerar valores em [0..p-1] e n/m deve ser menor ou igual a 1.

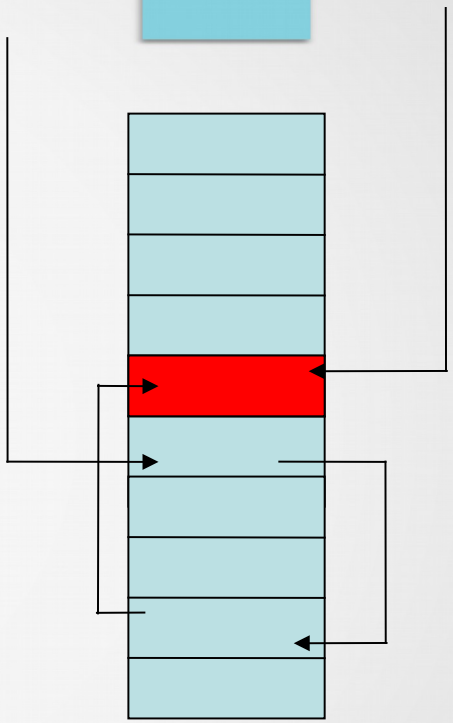


Overflow

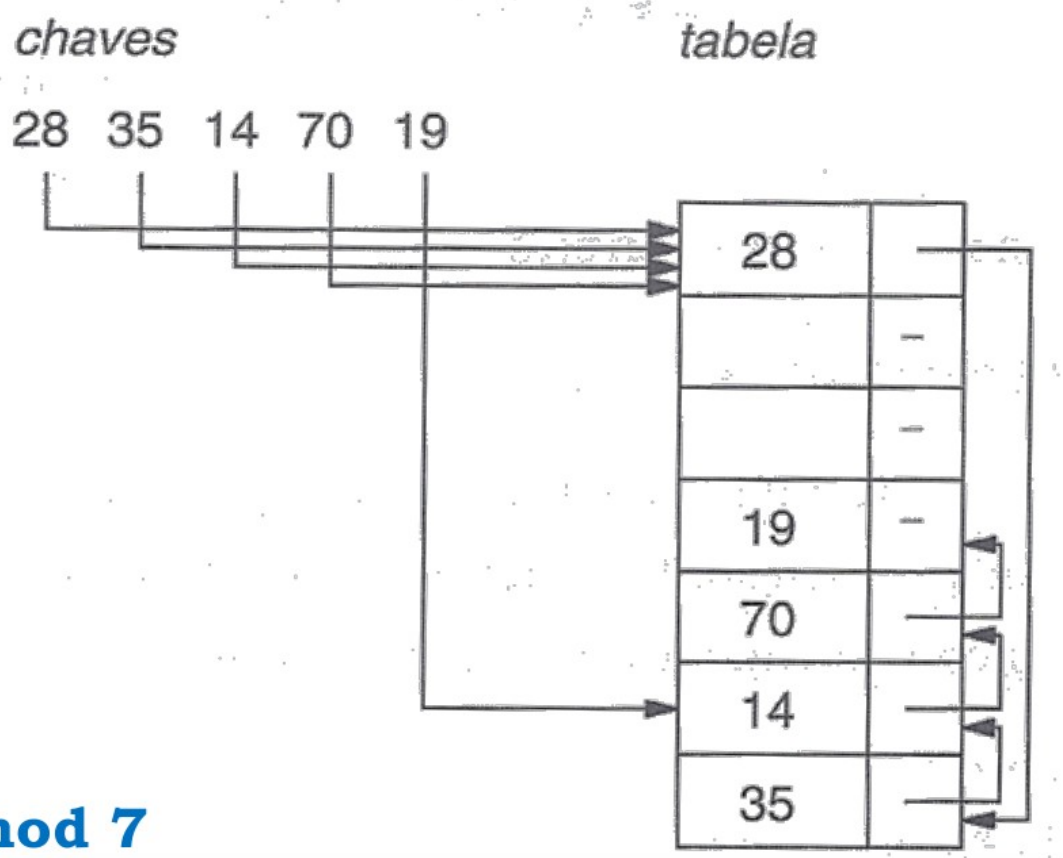
Hashing

- **Encadeamento interior**

  Outra forma de utilizarmos o encadeamento interno é através da não reserva de nenhuma área para sinônimos. Esta técnica elimina a ocorrência de overflow (uma vez que n/m < 1), mas produz um efeito indesejado que é a possibilidade de colisão de duas chaves x e y onde h(x) ≠ h(y). Este tipo de colisão é chamado de colisão secundária.



Hashing

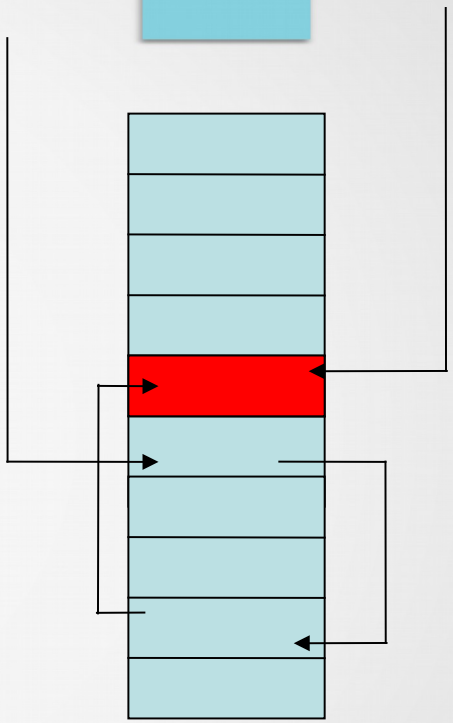


h(x) = x mod 7

Hashing

- **Remoção**

  Outro problema sério em uma tabela hash é a remoção de uma chave. Neste método, sentiremos ainda mais os efeitos da eliminação de uma chave da tabela. Notem que não podemos simplesmente abrir um buraco na lista encadeada, o que criaria problemas (quais?). Como resolver?



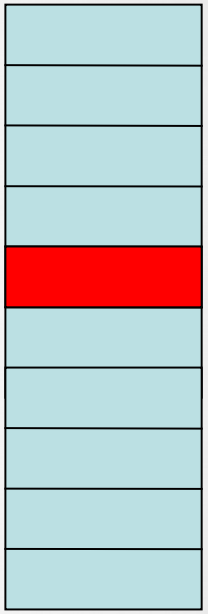
Hashing

- **Endereçamento aberto**

  Neste método, as colisões da tabela não são encadeadas como no método de encadeamento interior... Não há ponteiros. Quando houver alguma colisão, determina-se, através de uma nova função, qual o próximo endereço a ser examinado. Desta forma, teremos uma função h que leva em conta qual tentativa estamos realizando:

  h(x,k)

  onde k indica a k-ésima tentativa.

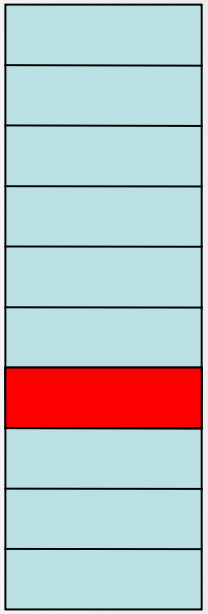


K = 0

Hashing

- **Endereçamento aberto**

  Para encontrar a chave x, deve-se tentar o endereço-base h(x,0). Caso esteja ocupado por alguma outra chave, calcula-se o endereço h(x,1), e assim por diante, até que a chave seja encontrada ou h(x,k) aponte para uma posição vazia.

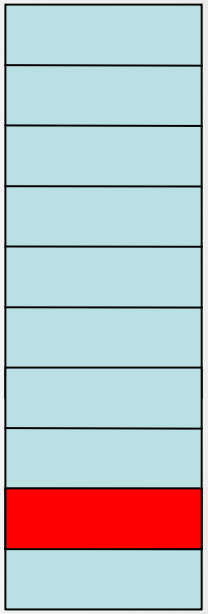


K = 1

Hashing

- **Endereçamento aberto - remoção**

  A remoção de chaves da tabela exige cuidados especiais. No método do endereçamento aberto, a remoção apresenta um problema semelhante àquele do método do encadeamento interior. Uma chave não pode ser removida, de fato, da tabela, pois faria romper a seqüência de tentativas. A solução é semelhante à empregada no método anterior.



K = 2

Hashing

- **Endereçamento aberto – tentativa linear**

  A idéia consiste em tentar armazenar a nova chave x no endereço-base h'(x). Se não for possível, tenta-se no endereço consecutivo h'(x) + 1. Se este já estiver ocupado, tenta-se h'(x) + 2... Até que uma posição vazia seja obtida ou ...(???) Lembre-se que devemos simular uma lista circular.

  $h(x,k) = (h'(x) + k) \bmod m$

  Problema : longos trechos consecutivos de memória ocupados, o que se denomina agrupamento primário.

Hashing

- **Endereçamento aberto – tentativa quadrática**

  A idéia consiste em se obter seqüências de endereços distantes para endereços-base próximos. Para isso, utilizamos como incremento, uma função quadrática em k:

  $$h(x,k) = (h'(x) + c_1 k + c_2 k^2) \bmod m$$

  Problema: Consegue evitar os agrupamentos primários da tentativa linear, mas não impede agrupamentos de chaves com mesma tentativa inicial. Os agrupamentos obtidos neste caso são chamados de secundários.

Hashing

- **Endereçamento aberto – double hashing**

  O método calcula a seqüência de tentativas, de acordo com a seguinte equação:

  $$h(x,k) = (h'(x) + k.h''(x)) \bmod m$$

  Esse método tende a distribuir as chaves na tabela de forma mais conveniente do que os dois anteriores. Se x e y são duas chaves distintas, tais que h'(x) = h'(y), então as seqüências de tentativas obtidas pelos métodos da tentativa linear e quadrática são necessariamente idênticas, o que ocasiona agrupamentos.