INF 1010
Estruturas de Dados Avançadas

Tabelas de Dispersão (*Hash tables*)

tabelas de dispersão

Uma tabela de dispersão/espalhamento (hash table) é um método muito rápido de se encontrar registros.

A busca é feita em uma tabela indexada

tabelas de dispersão - objetivo

acesso rápido (O(1)) a um elemento, dada sua chave

tabelas de dispersão - características

Chaves podem ser muito grandes e não servem como índice de vetor

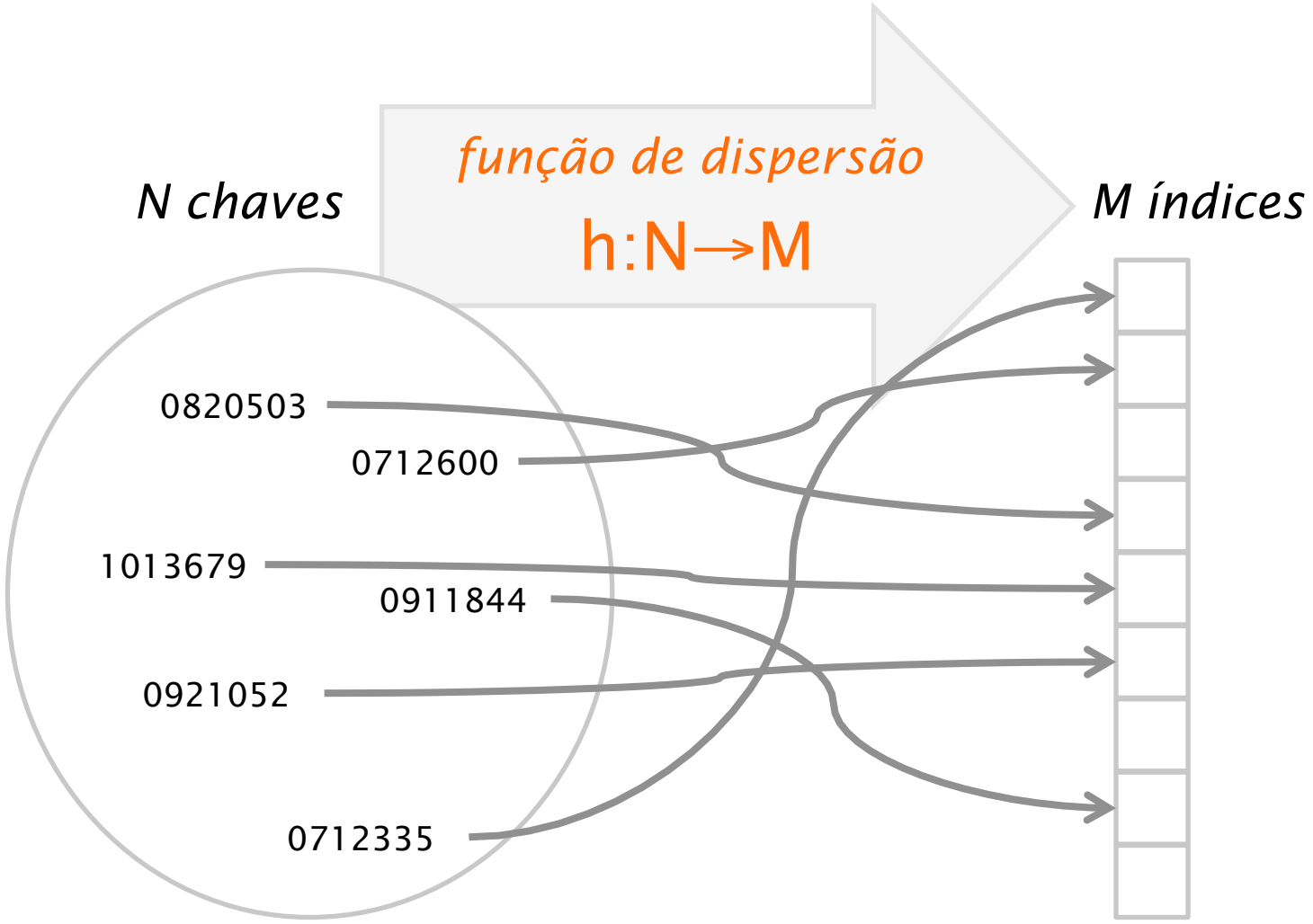
ex.: matrícula de alunos da PUC
(turmas de até 50 alunos; números de matrícula entre 0000001 a 9999999)

ex.: dicionário com o nome das funções utilizadas em um programa

Como resolver

criar uma função de dispersão **h** que,
a partir de uma chave muito grande **x**,
gera uma chave menor **h(x)**,
que pode ser utilizada como índice em um vetor

tabelas de dispersão



funções de dispersão – exemplo

N é o conjunto de todas as cadeias alfabéticas

M é um inteiro entre 65 e 90

h é o código ASCII do primeiro caractere da cadeia

h(‘AMORA’) = 65

h(‘ZEBRA’) = 90

Essa é uma boa função de dispersão?

funções de dispersão - discussão

Idealmente, para $x \neq y$, $h(x) \neq h(y)$

(h é função injetiva)

Quando isto é possível?

$|N| \leq |M|$

Mas, em geral, $|N| >> |M|$

tabelas de dispersão - problemas

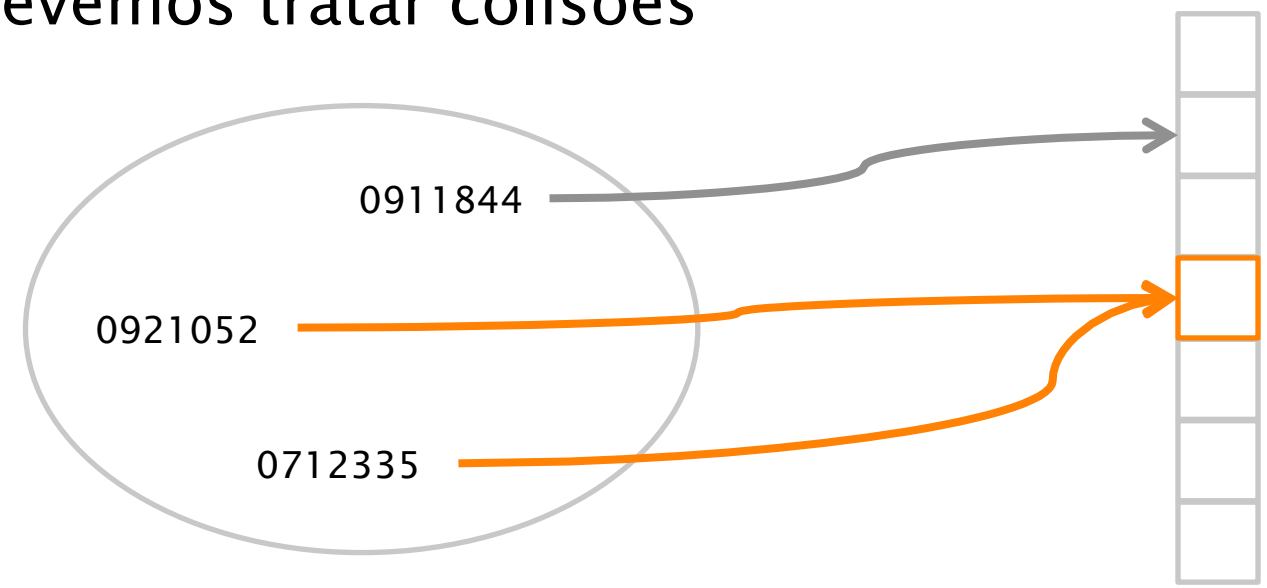
Podem ocorrer colisões

se a função não for injetiva, ou seja,

se, para duas chaves $x \neq y$, $h(x) = h(y)$

→ devemos evitar colisões

→ devemos tratar colisões



funções de dispersão – discussão

Considere o conjunto S de **nomes** dos alunos deste curso:

- |S| < 100
- O domínio de h é o conjunto de todas as cadeias alfabéticas (de até 40 caracteres)
- Qual seria uma boa função de dispersão para esse domínio?

funções de dispersão
> características desejáveis

produzir poucas colisões

- depende de se conhecer algo sobre a distribuição das chaves sendo acessadas
- ex.: se as chaves começam sempre por ‘A’ ou ‘B’, usar o primeiro caractere das chaves vai levar a muitas colisões

ser fácil de computar

- tipicamente, conter poucas operações aritméticas

ser uniforme

- idealmente, o número máximo de chaves mapeadas num mesmo índice deve ser |N|/|M|

funções de dispersão
> método da divisão

Assumindo N = {0 .. n – 1} e M = {0 .. m – 1}, a função de dispersão é dada por

h(x) = f(x) mod m

Qual deve ser o valor de m?

não deve ser uma potência de 2

se m = 2^k, h(x) = k bits menos significativos de x

não deve ser um número par

se m é par, então h(x) é par ⇔ x é par

na prática, bons resultados são obtidos com:

m = número primo não próximo a uma potência de 2

m = número sem divisores primos menores que 20

funções de dispersão
> método da divisão (cont.)

Não é bom que chaves sucessivas sejam mapeadas em índices sucessivos. Por isso, comumente se multiplica a chave por uma constante k antes de se fazer a divisão (m e k devem ser primos entre si):

$$h(x) = (k \bullet f(x)) \bmod m$$

Exemplo:

$$h(\text{"fulano"}) = ((((((6)*26+21)*26+12)*26+0)*26+14)*26+15) \bmod 257 = 81096043 \bmod 257 = 207$$

funções de dispersão
> método da multiplicação

Assume-se $m = 2^k$.

Multiplica-se a chave por ela mesma ou por alguma constante c.

Se o resultado cabe numa palavra com b bits, toma-se os k bits do meio da palavra, descartando os $(b - k)/2$ bits mais e menos significativos

$$h(x) = (x^2 \text{ div } 2^{(b-k)/2}) \bmod 2^k$$

ou

$$h(x) = ((x \bullet c) \text{ div } 2^{(b-k)/2}) \bmod 2^k$$

funções de dispersão
> método da multiplicação

Exemplos:

- $h(x) = (x^2 \text{ div } 2^{(b-k)/2}) \bmod 2^k$

b=8,k=2: $h(1001_b) = (1001_b^2 \text{ div } 2^{(8-2)/2}) \bmod 2^2 =$
$(01010001_b / 2^3) \bmod 4 = 01010_b \bmod 4 = 2$

b=8,k=2: $h(1111_b) = (1111_b^2 \text{ div } 2^{(4-2)/2}) \bmod 2^2 =$
$(11100001_b / 2^3) \bmod 4 = 11100_b \bmod 4 = 0$

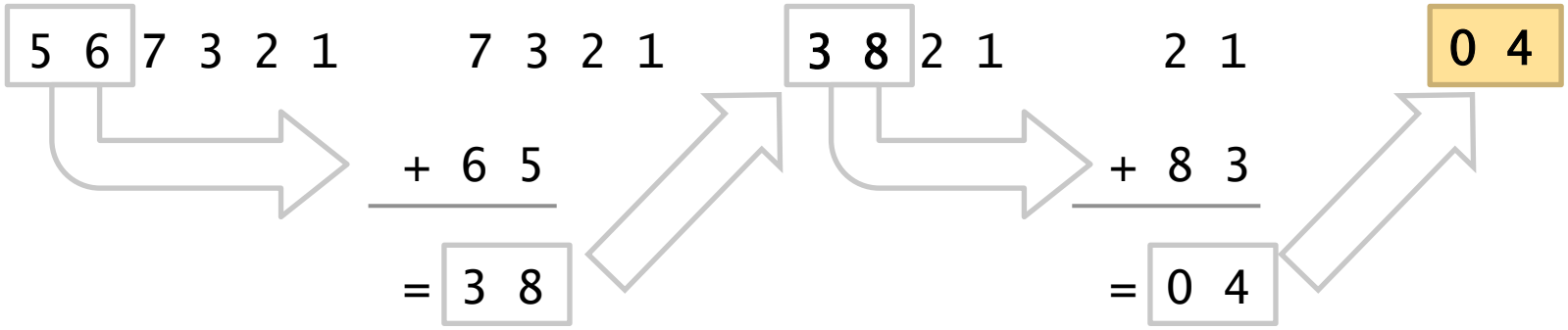
- $h(x) = ((x \bullet c) \text{ div } 2^{(b-k)/2}) \bmod 2^k$

b=6,k=4,c=3: $h(111_b) = (3 \bullet 111_b \text{ div } 2^{(6-4)/2}) \bmod 2^4$
$= (010101_b / 2^1) \bmod 16 = 01010_b \bmod 16 = 10$

B=6,k=4: $h(10101_b) = (3 \bullet 10101_b \text{ div } 2^{(6-4)/2}) \bmod 2^4$
$= (111111_b / 2^1) \bmod 16 = 11111_b \bmod 16 = 15$

funções de dispersão
> método da dobra

Suponha que a chave seja dada por uma seqüência de dígitos escritos numa folha de papel. O método consiste em dobrar sucessivamente a folha de papel após o j-ésimo dígito, somando os dígitos que se superpõem (sem fazer o “vai um”)



funções de dispersão
> método da dobra (variação)

A dobra pode ser feita de k em k bits, ou seja, considerando os "dígitos" 0 e 1 da representação binária do número. O resultado é um índice entre 0 e $2^k - 1$

Em vez de somar os bits, utiliza-se uma operação de ou-exclusivo "$\oplus$" entre os bits

Não se usa "e" (nem "ou"), pois estes produzem resultados menores (maiores) que os operandos

Exemplo: Suponha $k = 5$

$71 = 0001000111_2$

$h(71) = 01000_2$ *xor* $00111_2 = 01111_2 = 15$

funções de dispersão
> método da análise dos dígitos

Usado em casos especialíssimos

É preciso conhecer todos os valores de antemão

gperf [Schmidt 90]: função de hash "perfeita", ajustada para uma coleção particular de chaves (não gera colisões)

[Schmidt 90]: Douglas C. Schmidt, GPERF: A Perfect Hash Function Generator, in Proceedings of the 2 nd C++ Conference, 1990, pp 87--102.

tratamento de colisões

Por que tratar colisões?

Mesmo com boas funções de dispersão, à medida que o fator de carga α aumenta, a probabilidade de haver colisões aumenta.

$\alpha = N / M$, onde

N é o número de chaves armazenadas

M é o número de índices na tabela

$(0 \leq \alpha \leq 1)$

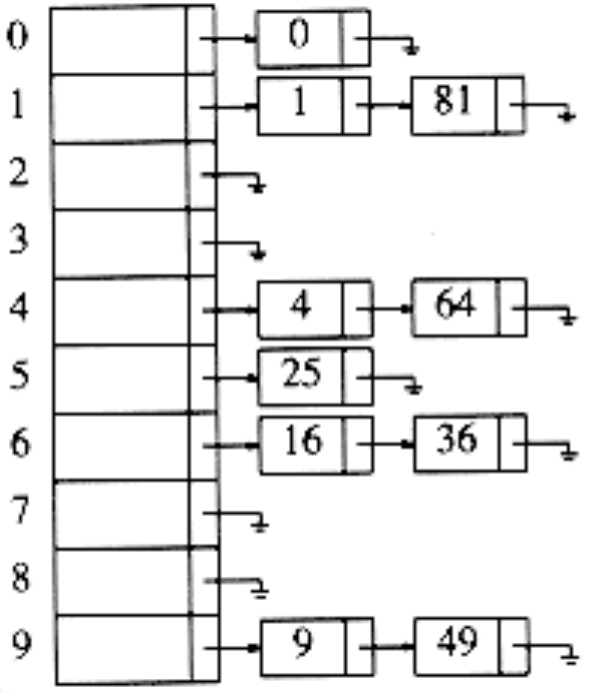
Tipos de tratamento de colisões

- encadeamento exterior (separado)
- encadeamento interior
- endereçamento aberto

Encadeamento exterior

cada posição da tabela pode ser ocupada por mais de uma chave

ex.: uma lista de chaves



Encadeamento exterior (cont.)

Quantas comparações podemos esperar em média para um acesso a chaves ausentes (buscas sem sucesso)?

- Supondo que h é uma função uniforme, que o fator de carga da tabela é α e que as listas não são ordenadas
- Então a probabilidade de h computar cada índice i é uniforme e igual a 1/m
- O número de comparações feitas ao se acessar a entrada i da tabela é o comprimento da lista L_i
- Então,

$$\text{Custo Médio} = \frac{1}{m}\sum_{i=0}^{m-1}\left|L_i\right| = \frac{n}{m} = \alpha = \text{Fator de Carga}$$

Encadeamento exterior (cont.)

Quantas comparações podemos esperar em média para um acesso a chaves presentes (buscas bem-sucedidas)?

- Para achar uma chave x, pesquisa-se uma lista Li
- Além da comparação bem sucedida com a chave armazenada em Li , o número de comparações mal-sucedidas é o comprimento da lista Li no momento em que x foi originalmente inserida na tabela
- Se x foi a (j+1)-ésima chave a ser incluída, então o comprimento médio de Li é j/m
- Então o custo médio é dado por

$$\mathrm{CM} = \frac{1}{n}\sum_{j=0}^{n-1}(1+\frac{j}{m}) = \frac{1}{n}\left(n+\frac{1}{m}\sum_{j=0}^{n-1} j\right) = 1+\frac{n(n-1)}{2nm} = 1+\frac{\alpha}{2}-\frac{1}{2m}$$

Encadeamento exterior (cont.)

Se o fator de carga for baixo, a complexidade média da busca é O(1)

desvantagem do encadeamento exterior:

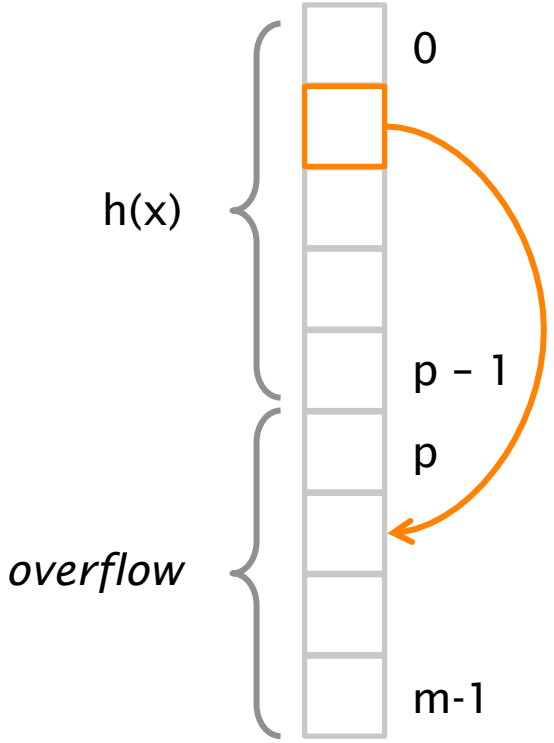
- requer o uso de estruturas externas
- → alocação dinâmica de memória

alternativas:

- encadeamento interior ou
- endereçamento aberto

Encadeamento interior (variação 1)

A idéia é usar como nós das listas as próprias entradas da tabela, numa área de *overflow*.

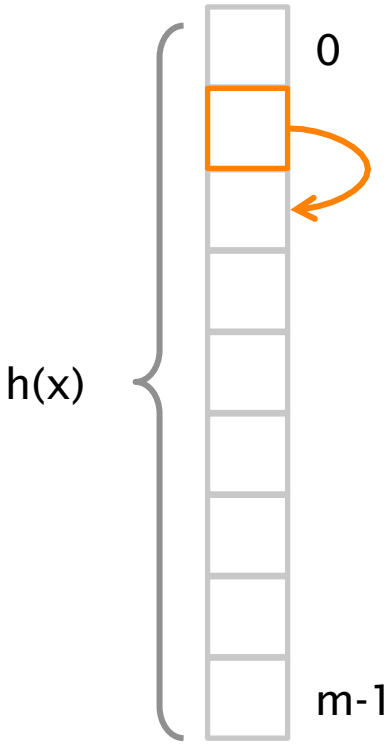


Pode acontecer que a área de *overflow* seja toda tomada sem que todas as entradas da tabela tenham sido usadas.

Pode-se aumentar a área de *overflow* diminuindo-se p, mas isso também é ineficiente. No limite, p = 1 e a tabela resume-se a uma lista encadeada.

Encadeamento interior (variação 2)

Na segunda variante, todo o espaço de endereçamento é usado.



Quando ocorre uma colisão, a chave é armazenada na primeira posição livre após h(x), a posição d, digamos.

Se agora incluirmos y tal que h(y)=d, teremos a fusão das listas correspondentes a h(x) e h(y), diminuindo a eficiência do esquema.

O maior problema dessa abordagem é que pode haver colisões secundárias, isto é colisões em que h(x) ≠ h(y).

Encadeamento interior - exclusão

Não se pode simplesmente retirar o elemento da cadeia

São necessários valores de chave especiais:

- "posição vazia"
- "elemento removido" (uma inserção posterior pode reaproveitar posições marcadas com o elemento removido, chamado de "lápide")

Na verdade, encadeamento interior com espaço de endereçamento único não é uma boa idéia, já que os problemas são os mesmos encontrados no tratamento de colisões por endereçamento aberto, sendo que nesse último temos a vantagem de não precisar de ponteiros.

Endereçamento aberto

utiliza-se uma segunda função de dispersão, que fornece o próximo índice a ser tentado

h(x, k) onde

- x é a chave
- k = 0, 1, 2, etc. é o número da tentativa

h(x, k) tem que visitar todos os m endereços em m tentativas

- No pior caso, m tentativas são feitas

Endereçamento aberto

tentativa linear

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

Tem a desvantagem de agrupar tentativas consecutivas

tentativa quadrática

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

Resolve o problema do agrupamento primário

Problema do agrupamento secundário

chaves x e y tais que h'(x) = h'(y) geram a mesma sequência de tentativas

dispersão dupla

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

Tabelas de Dispersão Dinâmica (*Dinamic Hash*)

Hash Estático

Problemas:

Alocar uma tabela hash de acordo com a quantidade estimada de registros e aceitar queda de desempenho com colisões e estouros devido a crescimento (no número de registros)

x

Alocar mais espaço que o necessário e gerar desperdício inicial de memória

Criação de Hash Estático

- Previsão insuficiente de espaço:
  - Solução:
    escolher uma nova função *hash* e
    reorganizar toda a tabela
  - Problemas:
    operação demorada
    bloqueia o acesso a estrutura de dados

Hash Dinâmico

Hash Dinâmico:

Reorganização dinâmica,
sem bloquear acesso à estrutura de dados

Técnica mais usada:

Hash Expansível (ou Extensível)
O(1) e baixa sobrecarga

Hash Expansível - Estrutura

Buckets ("Baldes" ☹):

- Armazenam os dados
- Dimensionados em função do tamanho do sistema de armazenamento

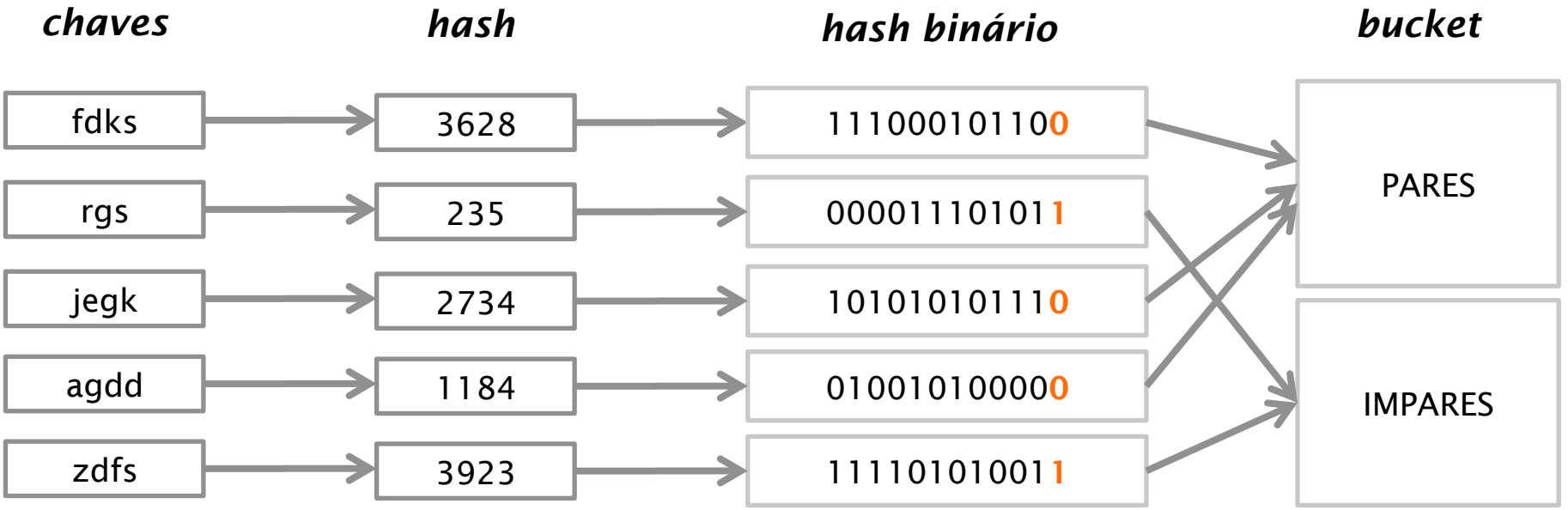
Hash Expansível - Estrutura

Separação Binária

Registros ímpares e pares em *buckets* distintos

Para localiza o *bucket*, basta verificar o bit menos significativo do valor gerado pela função de *hash* gerado para a chave

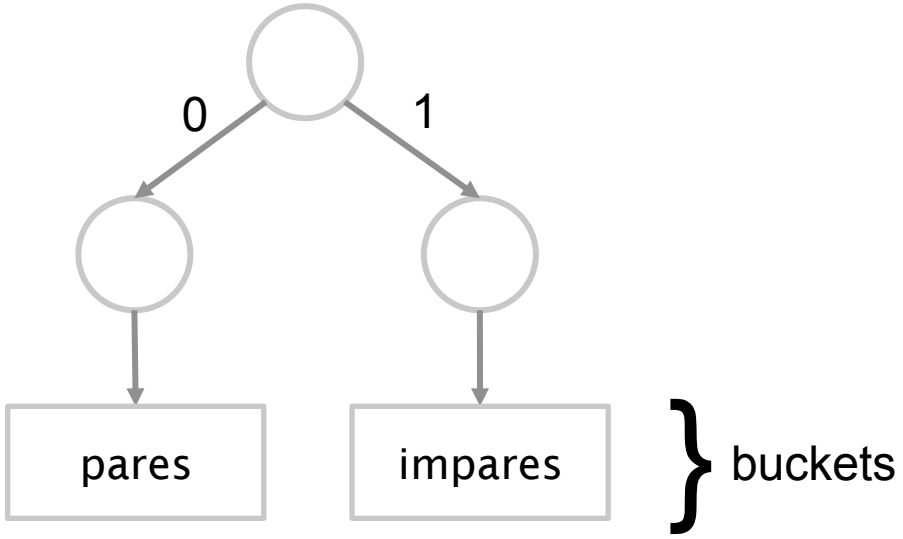
0 – para os pares e 1 – para os impares



Hash Expansível - Estrutura

Trie (*Prefix tree*):

- Forma de representar Separação Binária
- Um ramo representa os pares e outro os ímpares

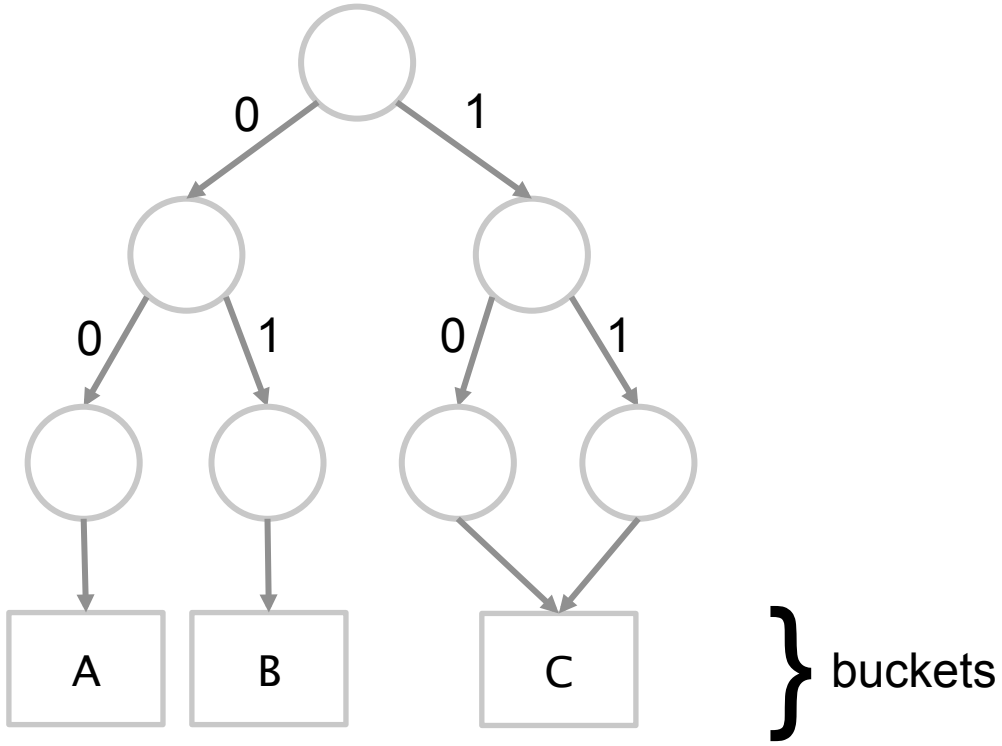


Hash Expansível - Estrutura

Bucket Completo:

O que fazer quando o bucket é preenchido?

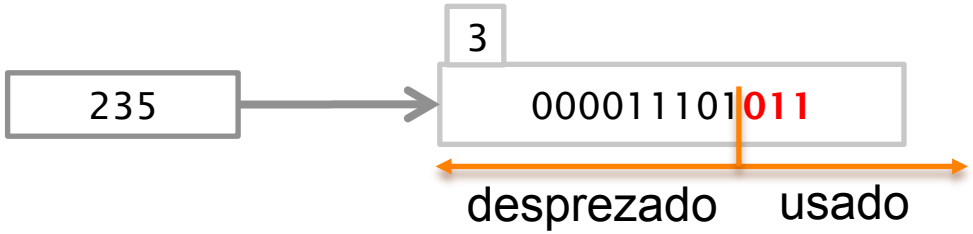
Exemplo: número de pares não cabe mais no Bucket dos pares



Hash Expansível - Estrutura

Hash Expansível

usa apenas os *bits de relevância* resultantes da função de *hash*



Hash Expansível - Estrutura

- Bits de Relevância (i)

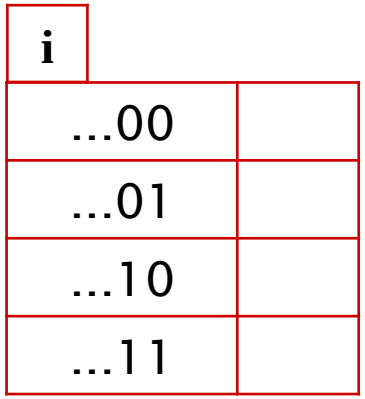
  Indica que *i* bits da função hash são necessários para determinar o *bucket* correto

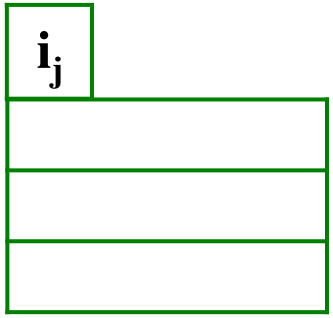
- Info adicional do Bucket

  inteiro indicando o número de bits comuns a todos os seus registros

  número de posições na tabela de endereços para o *bucket j:*

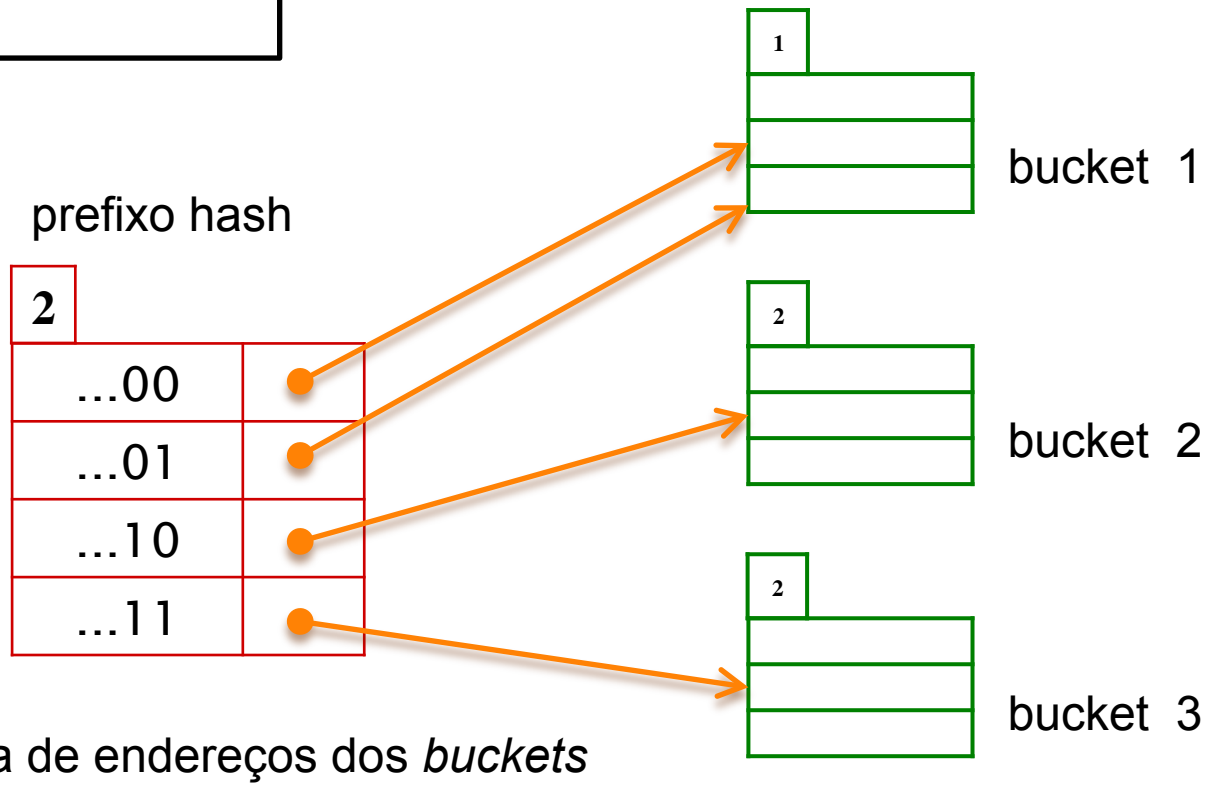
  $2^{(i-ij)}$





Hash Expansível - Estrutura

b = número de bits
0 ≤ i ≤ b



Hash Expansível - Busca

Busca

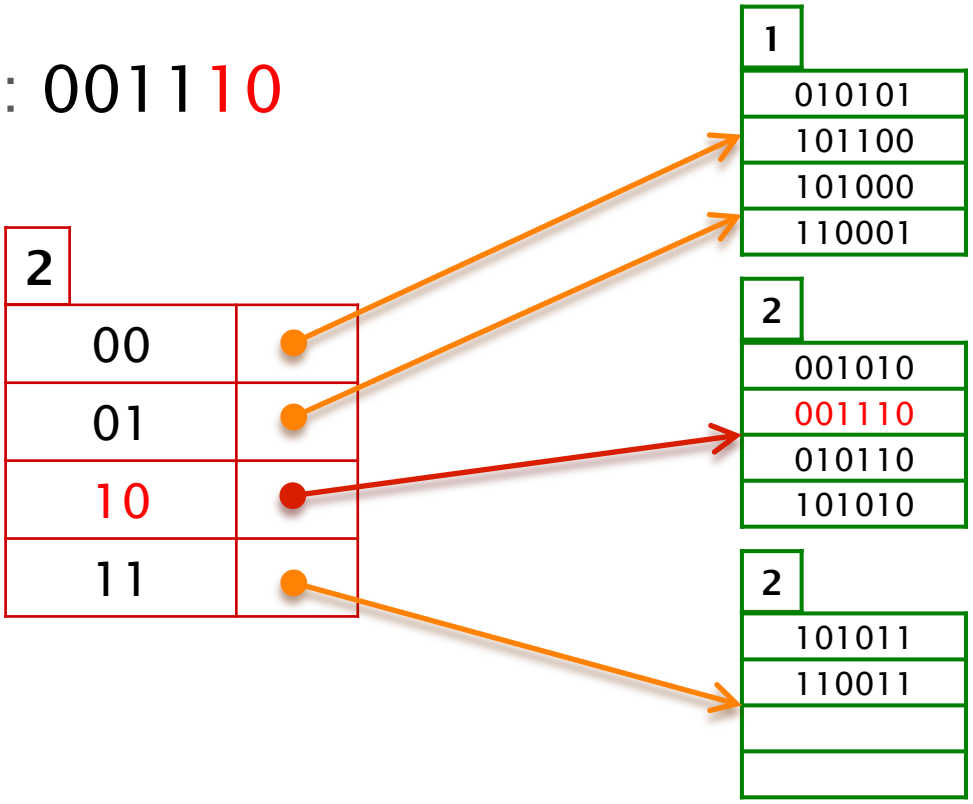
Use os *i* bits do valor da função de *hash* para localizar o *bucket*

Pesquise o *bucket* para encontrar o registro

Hash Expansível - Exemplo de busca

Exemplo

Busca por: 001110



Hash Expansível - Inserção

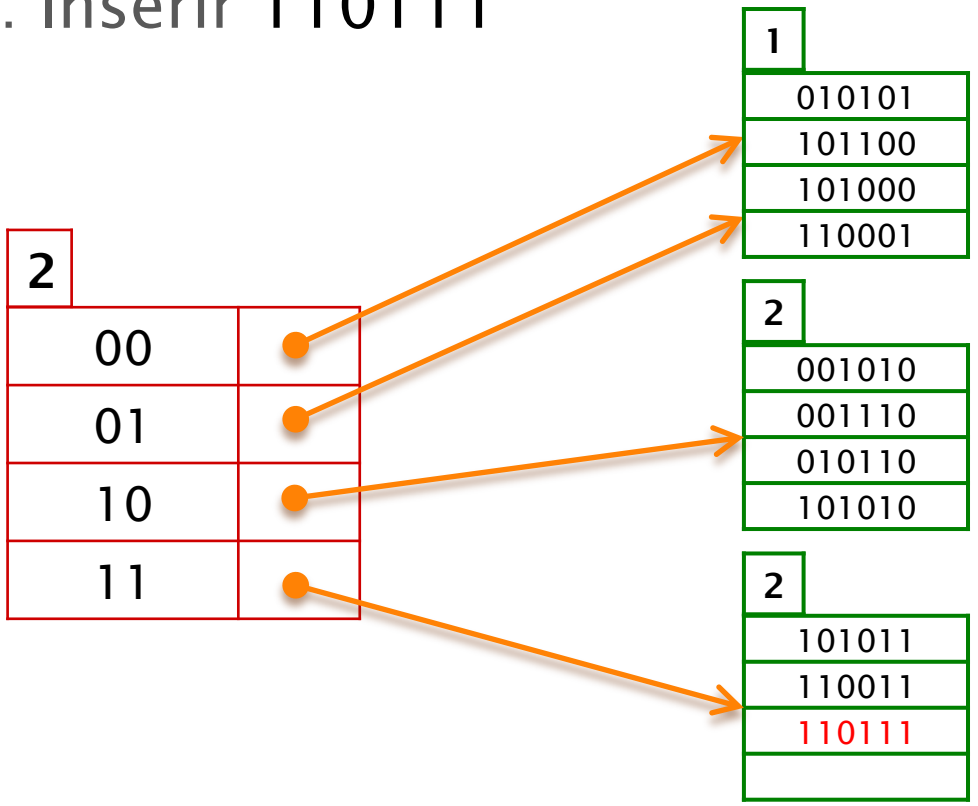
Inserção

- Localize o *bucket* a ser usado na inserção
- Considere 3 casos:
  - Caso 1: *bucket* ainda tem espaço
  - Caso 2: *bucket* está completo, mas pode ser dividido
  - Caso 3: *bucket* está completo e só possui uma referência

Hash Expansível - Inserção

Caso 1: *bucket* ainda tem espaço

Exemplo: Inserir 110111



Hash Expansível - Inserção

Caso 2:

bucket está completo, mas pode ser dividido

Se $i_j < i$, então o *bucket* pode ser dividido

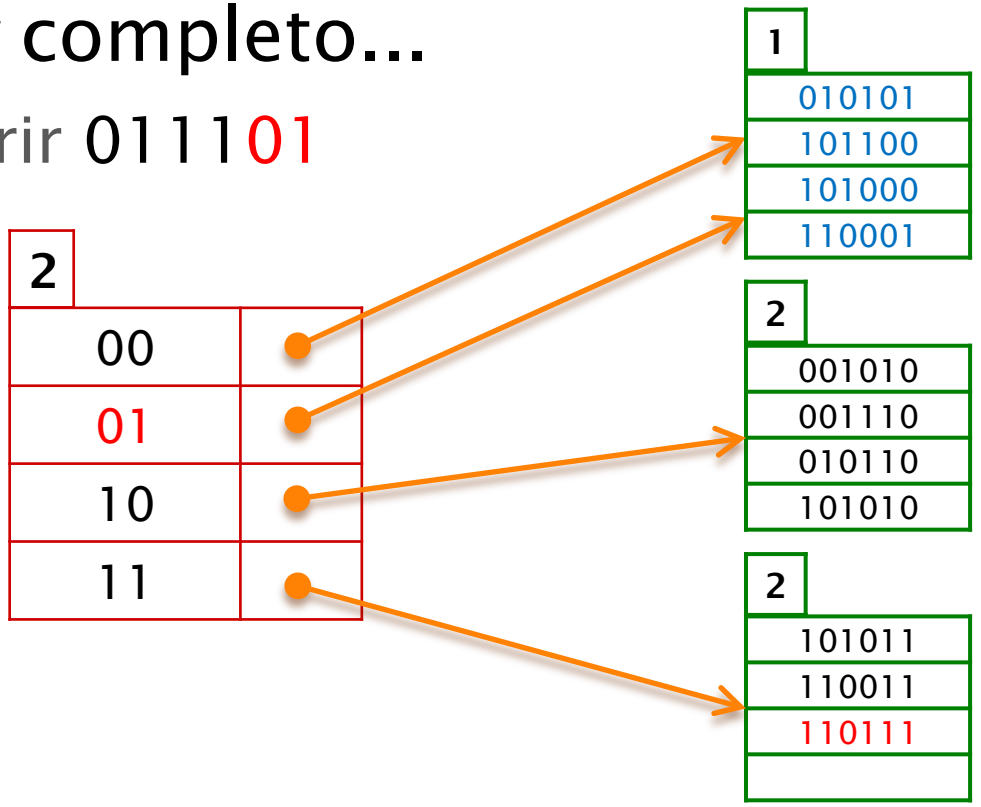
A tabela de endereçamento precisa então distinguir entre os dois *buckets*

Hash Expansível - Inserção

Caso 2: *bucket* completo...

Exemplo: Inserir 011101

(antes)

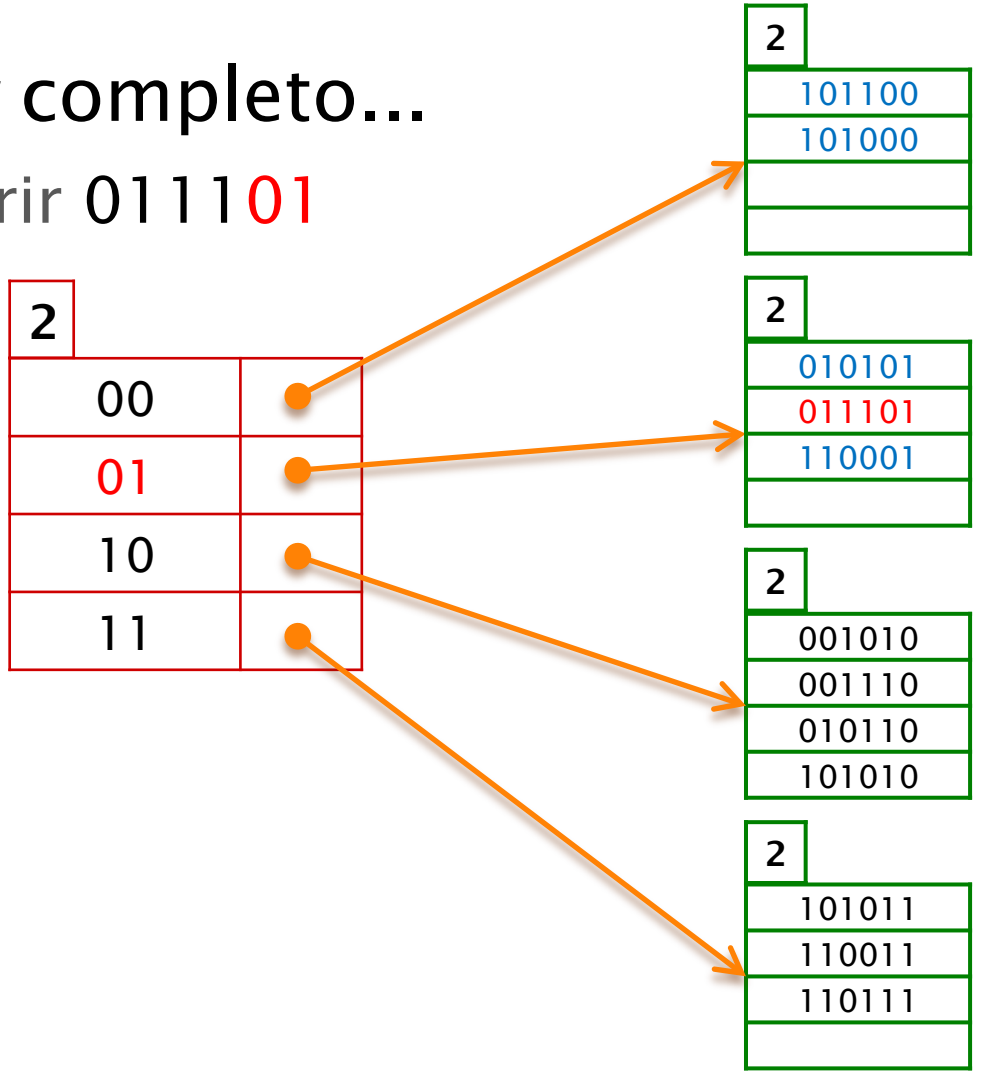


Hash Expansível - Inserção

Caso 2: *bucket* completo...

Exemplo: Inserir 011101

(depois)



Hash Expansível - Inserção

Caso 3:

bucket está completo e só possui uma referência

Se $i_j = i$ então o bucket possui só uma referencia

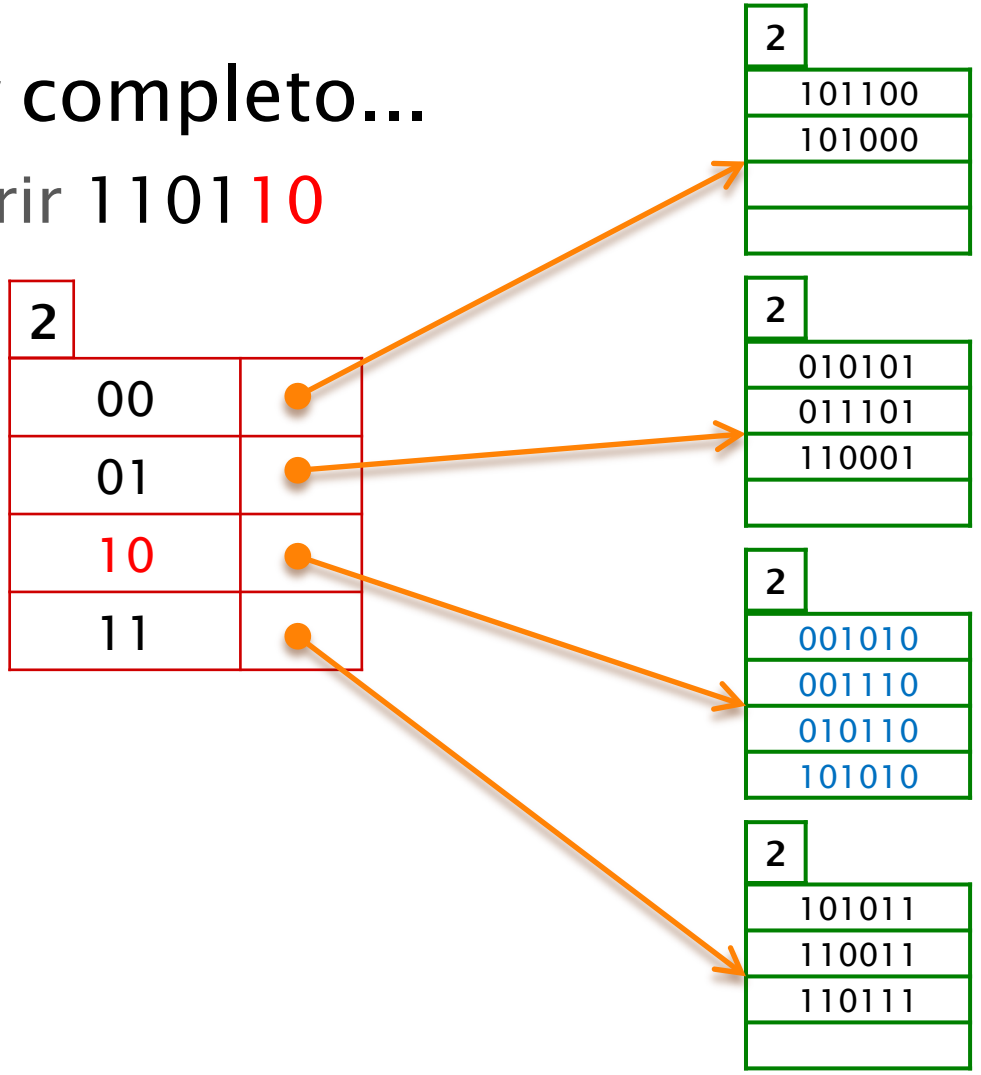
Para dividir a tabela de endereçamento
é preciso dobrá-la de tamanho e
reorganizar os ponteiros para os buckets

Hash Expansível - Inserção

Caso 3: *bucket* completo...

Exemplo: Inserir 110110

(antes)

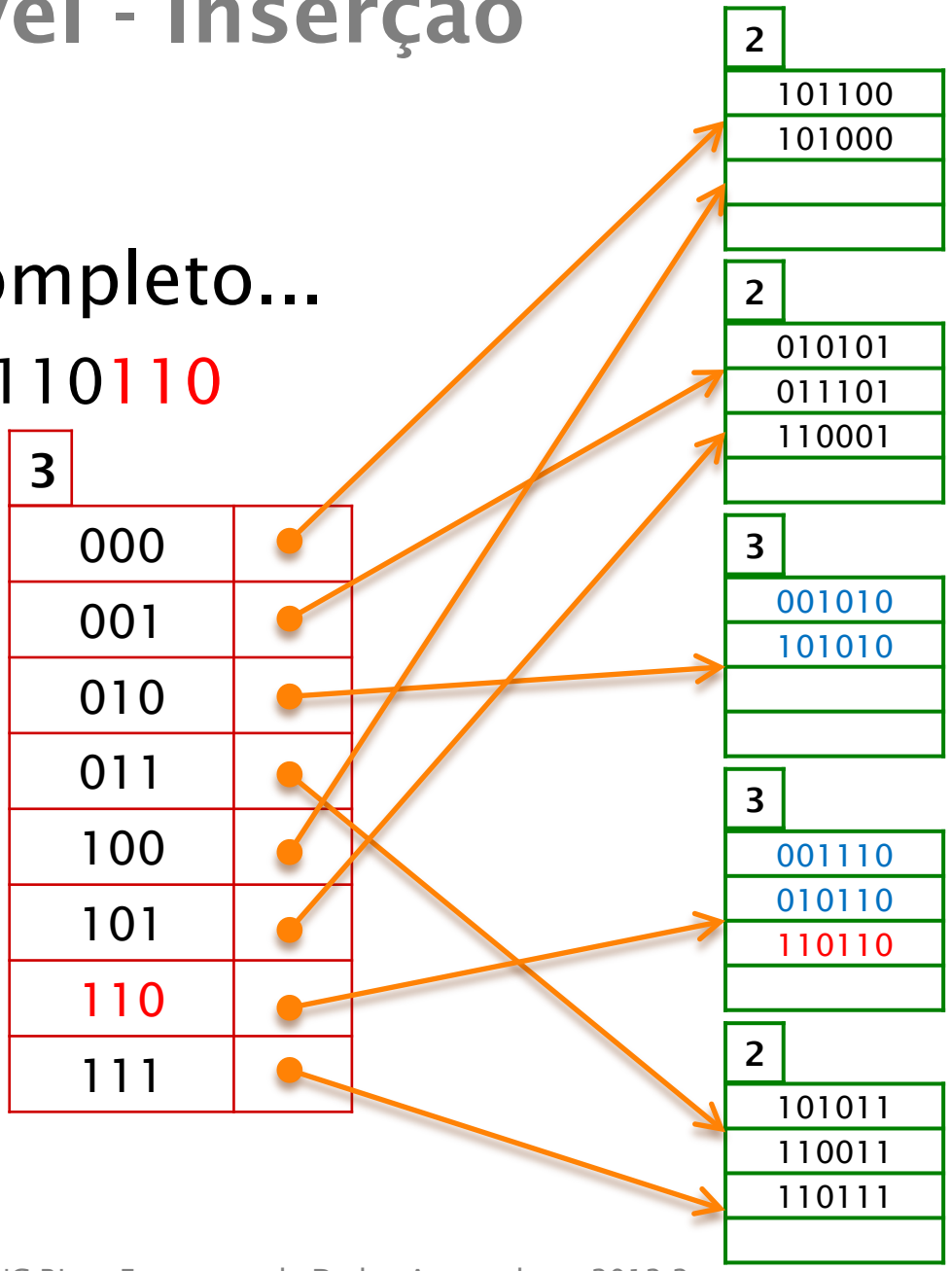


Hash Expansível - Inserção

Caso 3: *bucket* completo...

Exemplo: Inserir 110110

(depois)



Hash Expansível - Inserção

Caso particular

Se a divisão da tabela de endereçamento leva todos os registros para um mesmo *bucket*,

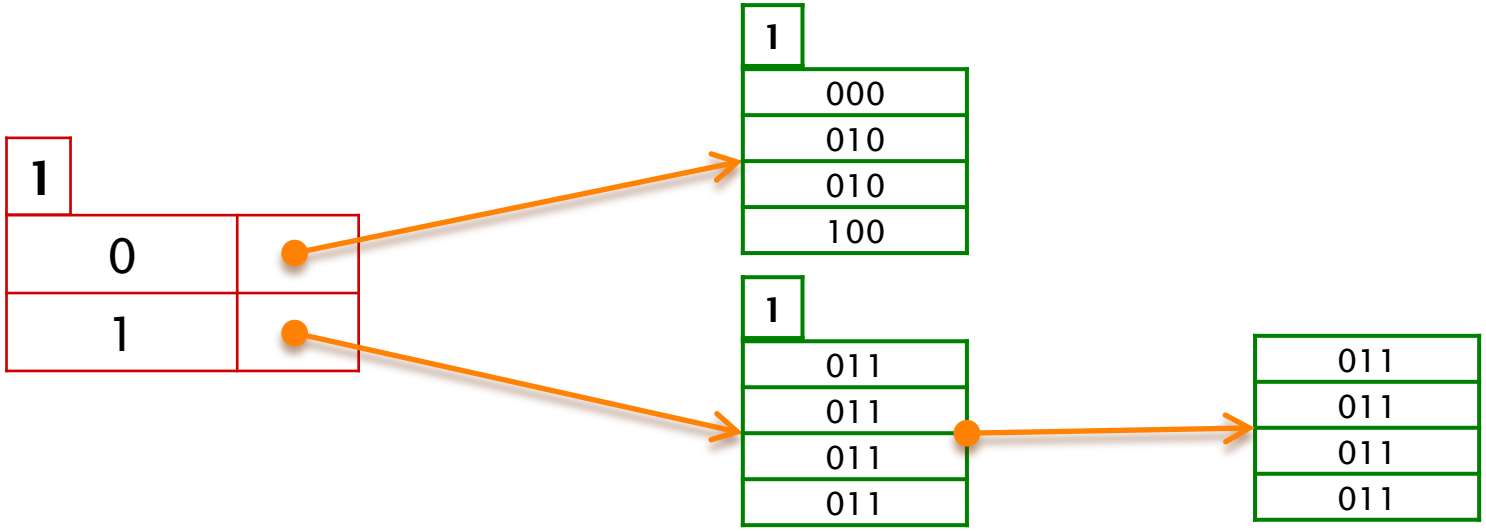
então uma nova divisão pode ser necessária se a inserção for nesse ponto

Hash Expansível - *Bucket overflow*

Bucket overflow

Colisões de chaves podem acontecer com *hash* dinâmico

Tratamento por encadeamento exterior pode ser usado



Hash Expansível - Remoção

Remoção

- Localize o *bucket* com a chave
- Considere 3 casos:
  - Caso 1: Remoção de *bucket* vazio
  - Caso 2: Combinação de *buckets*
  - Caso 3: Redução da tabela de endereços pode ser reduzida

Hash Expansível - Remoção

Caso 2: Combinação de buckets

- Combine o *bucket* que está na mesma profundidade do *bucket* sendo removido
- Os *buckets* diferem apenas no bit mais significativo

Hash Expansível - Desempenho

Considerações sobre desempenho

- Como as operações de crescimento e redução são realizadas localmente em *buckets*, a queda de desempenho por bloqueio é baixa
- A tabela de endereçamento desperdiça memória e introduz indireção

Hash Expansível - Função de hash

Escolha da função de hash

Para evitar modificação na função de *hash*
o número de bits gerado deve ser alto

Exemplo:

32 bits são suficientes para

$2^{32} \cong$ 4 bilhões de registos

Hash Expansível

Referência

Abraham Silberschatz, Henry F. Korth, S. Sudarshan.
Database System Concepts. McGraw Hill (6th Edition).

dúvidas?