

ACH2024

Aula 22

HASHING

Hashing estático – endereçamento aberto

Profa. Ariane Machado Lima

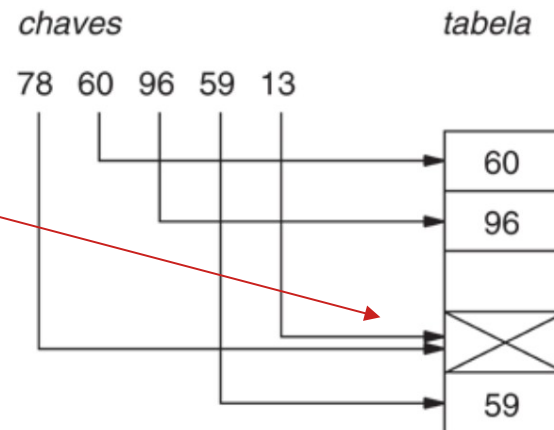
Aulas anteriores

- Organização interna de arquivos
- Acesso à memória secundária (por blocos - seeks)
- Tipos de alocação de arquivos na memória secundária:
 - Sequencial (ordenado e não ordenado)
 - Ligada
 - Indexada
 - Árvores-B
 - Hashing (veremos também hashing em memória principal)
- Algoritmos de processamento cossequencial e ordenação em disco

Motivação e Conceitos Básicos

Questões que podem surgir:

- O que fazer quando duas chaves caem na mesma posição? (**colisão**)
- **Tratamento de colisões**
- Qual **função de hash** utilizar? Como ela impacta na ocorrência de colisões?



Vamos estudar essas questões

Tratamento de colisões

Estratégias:

A) Hashing estático (tamanho da tabela é constante)

1) Encadeamento ou endereçamento fechado – colisões vão para uma lista ligada

1.1) Encadeamento exterior (fora da tabela)

Permite $\alpha = n/m > 1$

1.2) Encadeamento interior (dentro da tabela)

Necessariamente $\alpha = n/m \leq 1$

2) Endereçamento aberto (chaves dentro da tabela, sem ponteiros)

2.1) Tentativa/Sondagem linear

2.2) Tentativa/Sondagem quadrática

2.3) Dispersão dupla / Hash duplo

B) Hashing dinâmico (tabela pode expandir/encolher)

3) Hashing extensível (estrutura de dados adicional)

4) Hashing linear

Hoje

Estratégias:

A) Hashing estático (tamanho da tabela é constante)

1) Encadeamento ou endereçamento fechado – colisões vão para uma lista ligada

1.1) Encadeamento exterior (fora da tabela) Permite $\alpha = n/m > 1$

1.2) Encadeamento interior (dentro da tabela) Necessariamente $\alpha = n/m \leq 1$

2) Endereçamento aberto (chaves dentro da tabela, sem ponteiros) Necessariamente $\alpha = n/m \leq 1$

2.1) Tentativa/Sondagem linear

2.2) Tentativa/Sondagem quadrática

2.3) Dispersão dupla / Hash duplo

B) Hashing dinâmico (tabela pode expandir/encolher)

3) Hashing extensível (estrutura de dados adicional)

4) Hashing linear

Tudo isso para hashing interno (em memória) quanto para externo (em disco).

Primeiro assumiremos hashing interno e depois discutiremos mudanças para hashing externo.

Tratamento de colisões

2) Endereçamento aberto

Conceitos gerais

Tratamento de colisões

2) Endereçamento aberto

Características:

- todas as chaves dentro da tabela (**espaço constante**)
- sem uso de ponteiros (não há listas): **economiza espaço**
- endereço de uma mesma chave pode ser diferente dependendo de quando $h(x)$ é calculada (cálculo em **aberto**)
- pode ficar cheia inviabilizando novas inserções (assim como no encadeamento interno)

Tratamento de colisões

2) Endereçamento aberto

Vantagens:

- evita por completo o uso de listas encadeadas;
- ao invés de seguir os ponteiros nas listas, *calculamos* a seqüência de posições a serem examinadas;
- uso mais eficiente do espaço alocado para a tabela hash;
- o espaço não alocado para as listas pode ser usado para aumentar o tamanho da tabela hash, o que implica menor número de colisões.

Tratamento de colisões

2) Endereçamento aberto

Inserção:

- É feita uma *sondagem*, isto é, um exame sucessivo, da tabela hash até encontrarmos uma posição vazia na qual seja possível inserir a chave.
- Ao invés de fazer a sondagem na ordem 0, 1, ..., m - 1 (o que exige tempo $\Theta(n)$), a seqüência de posições examinadas depende da *chave que está sendo inserida*.
- O que vem a ser isto?

Tratamento de colisões

2) Endereçamento aberto

Inserção:

- Estendemos a função hash com o objetivo de incluir o número de sondagens (a partir de 0) como uma segunda entrada. Desse modo, a função hash se torna:

$$h : C \times \{0, 1, \dots, m - 1\} \rightarrow \{0, 1, \dots, m - 1\}$$

onde C é o universo de chaves.

Ex: sondagem linear (ver adiante)

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

- Com o endereçamento aberto, exige-se que, para toda chave k , a *seqüência de sondagem* seja uma permutação de $\langle 0, 1, \dots, m - 1 \rangle$. Por quê?

Tratamento de colisões

2) Endereçamento aberto

Inserção:

- Estendemos a função hash com o objetivo de incluir o número de sondagens (a partir de 0) como uma segunda entrada. Desse modo, a função hash se torna:

$$h : C \times \{0, 1, \dots, m - 1\} \rightarrow \{0, 1, \dots, m - 1\}$$

$$\text{Ex: } m = 5, h(x, i) =$$

$$x : i = 0, 1, 2, 3, 4$$

$$h(11, i) = 1, 2, 3, 4, 0$$

$$h(8, i) = 3, 4, 0, 1, 2$$

onde C é o universo de chaves.

Ex: sondagem linear (ver adiante)

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

- Com o endereçamento aberto, exige-se que, para toda chave k , a *seqüência de sondagem* seja uma permutação de $\langle 0, 1, \dots, m - 1 \rangle$. Por quê?

Tratamento de colisões

2) Endereçamento aberto

Inserção:

- Estendemos a função hash com o objetivo de incluir o número de sondagens (a partir de 0) como uma segunda entrada. Desse modo, a função hash se torna:

$$h : C \times \{0, 1, \dots, m - 1\} \rightarrow \{0, 1, \dots, m - 1\}$$

$$\text{Ex: } m = 5, h(x, i) =$$

$$x : i = 0, 1, 2, 3, 4$$

$$h(11, i) = 1, 2, 3, 4, 0$$

$$h(8, i) = 3, 4, 0, 1, 2$$

onde C é o universo de chaves.

Ex: sondagem linear (ver adiante)

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

- Com o endereçamento aberto, exige-se que, para toda chave k , a *seqüência de sondagem* seja uma permutação de $\langle 0, 1, \dots, m - 1 \rangle$. Por quê? Cada chave deve seguir um caminho diferente, para evitar o $O(n)$

Tratamento de colisões

2) Endereçamento aberto

Inserer(T, k) */* retorna a posição onde a chave foi inserida */*

$i \leftarrow 0$ */* nr da sondagem */*

$j \leftarrow h(k, i)$

enquanto $i \neq m$

se $T[j] = \text{NIL}$

$T[j] \leftarrow k$

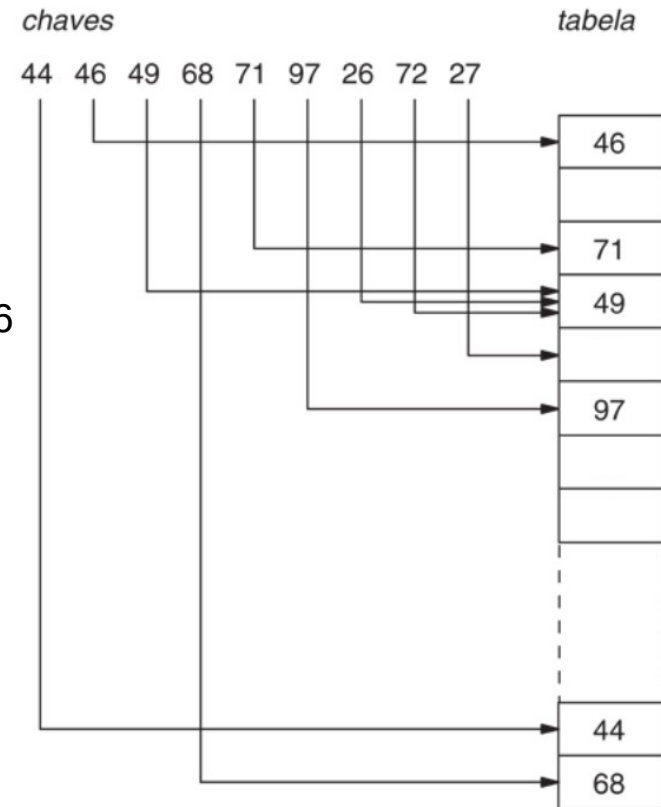
retorna j

senão $i \leftarrow i + 1$

$j \leftarrow h(k, i)$

retorna -1 */* ESTOURO DA TABELA */*

Ex:
primeira
colisão: 26



Tratamento de colisões

2) Endereçamento aberto

Inserir(T, k) */* retorna a posição onde a chave foi inserida */*

$i \leftarrow 0$ */* nr da sondagem */*

$j \leftarrow h(k, i)$

enquanto $i \neq m$

se $T[j] = \text{NIL}$

$T[j] \leftarrow k$

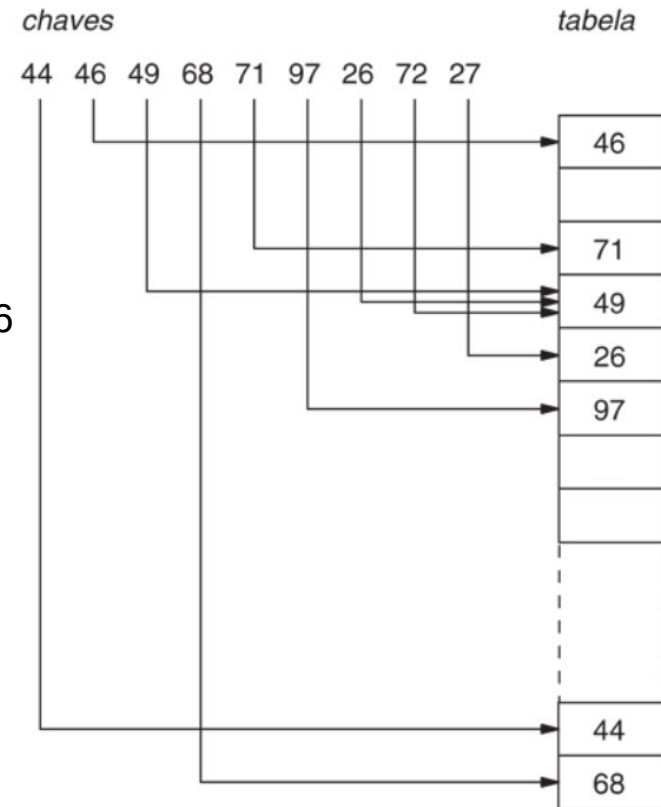
retorna j

senão $i \leftarrow i + 1$

$j \leftarrow h(k, i)$

retorna -1 */* ESTOURO DA TABELA */*

Ex:
primeira
colisão: 26



Tratamento de colisões

2) Endereçamento aberto

Inserir(T, k) /* retorna a posição onde a chave foi inserida */

$i \leftarrow 0$ /* nr da sondagem */

$j \leftarrow h(k, i)$

enquanto $i \neq m$

se $T[j] = \text{NIL}$

$T[j] \leftarrow k$

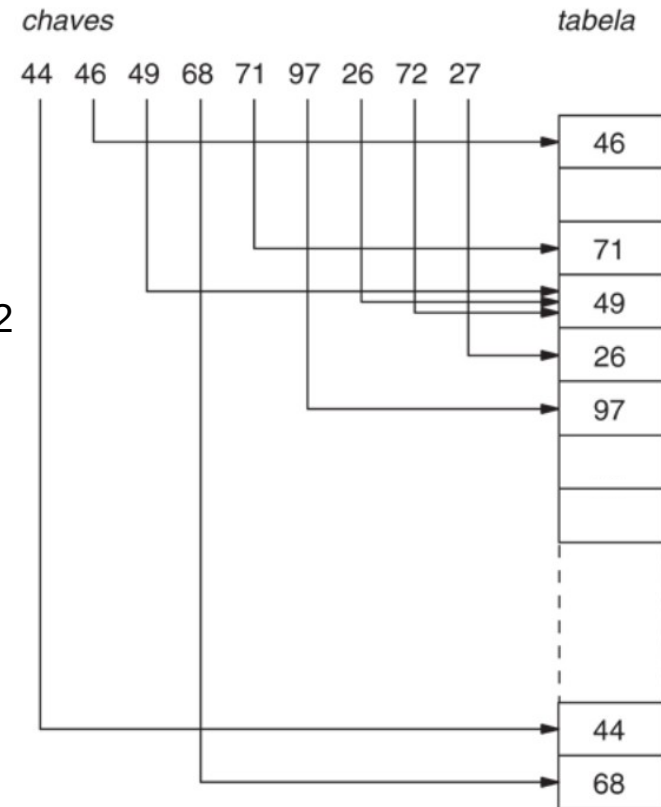
retorna j

senão $i \leftarrow i + 1$

$j \leftarrow h(k, i)$

retorna -1 /* ESTOURO DA TABELA */

Ex:
segunda
colisão: 72



Tratamento de colisões

2) Endereçamento aberto

Inserir(T, k) /* retorna a posição onde a chave foi inserida */

$i \leftarrow 0$ /* nr da sondagem */

$j \leftarrow h(k, i)$

enquanto $i \neq m$

se $T[j] = \text{NIL}$

$T[j] \leftarrow k$

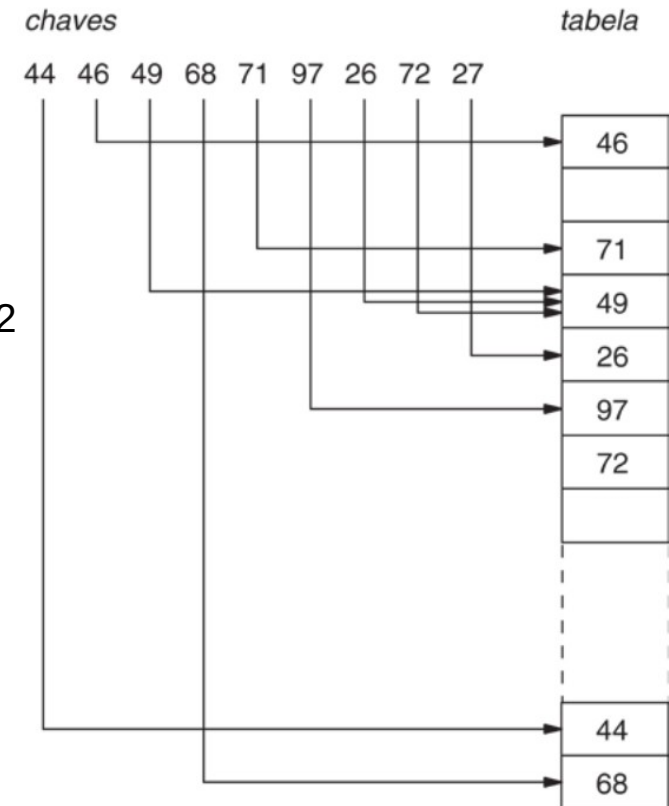
retorna j

senão $i \leftarrow i + 1$

$j \leftarrow h(k, i)$

retorna -1 /* ESTOURO DA TABELA */

Ex:
segunda
colisão: 72



Tratamento de colisões

2) Endereçamento aberto

Inserer(T, k) */* retorna a posição onde a chave foi inserida */*

$i \leftarrow 0$ */* nr da sondagem */*

$j \leftarrow h(k, i)$

enquanto $i \neq m$

se $T[j] = \text{NIL}$

$T[j] \leftarrow k$

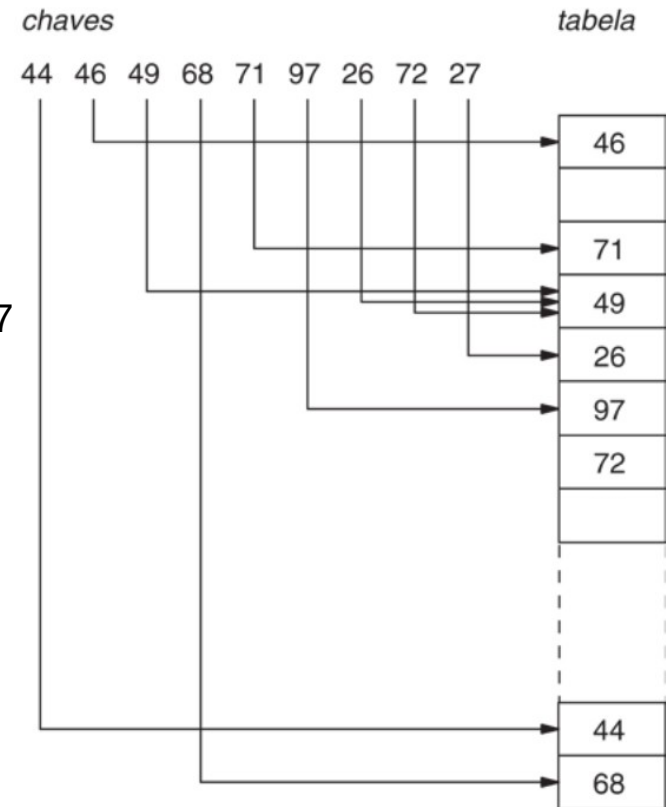
retorna j

senão $i \leftarrow i + 1$

$j \leftarrow h(k, i)$

retorna -1 */* ESTOURO DA TABELA */*

Ex:
terceira
colisão: 27



Tratamento de colisões

2) Endereçamento aberto

Inserer(T, k) */* retorna a posição onde a chave foi inserida */*

$i \leftarrow 0$ */* nr da sondagem */*

$j \leftarrow h(k, i)$

enquanto $i \neq m$

se $T[j] = \text{NIL}$

$T[j] \leftarrow k$

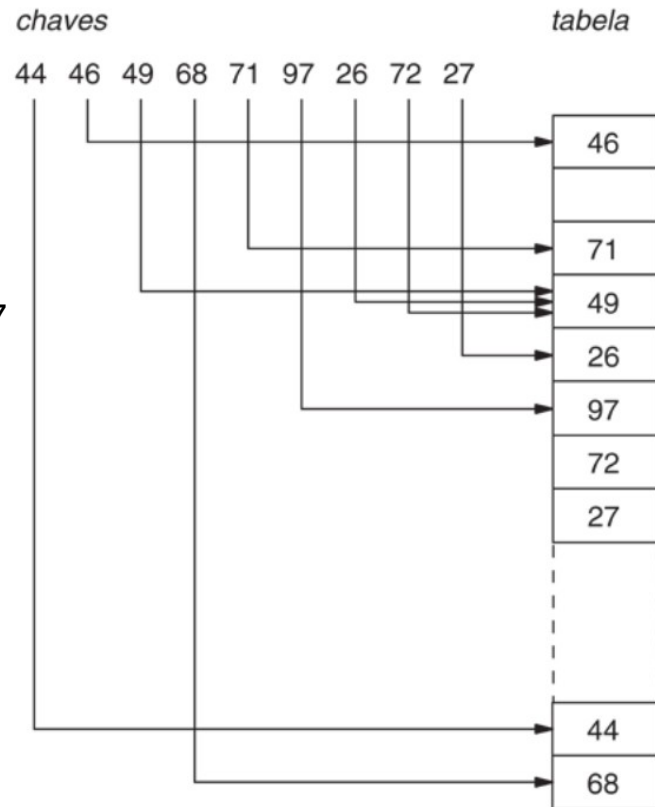
retorna j

senão $i \leftarrow i + 1$

$j \leftarrow h(k, i)$

retorna -1 */* ESTOURO DA TABELA */*

Ex:
terceira
colisão: 27



Tratamento de colisões

2) Endereçamento aberto

Busca(T,k) /* retorna a posição onde a chave foi encontrada */

$i \leftarrow 0$ /* nr da sondagem */

$j \leftarrow h(k, i)$

enquanto $i \neq m$ e $T[j] \neq \text{NIL}$

se $T[j] = k$

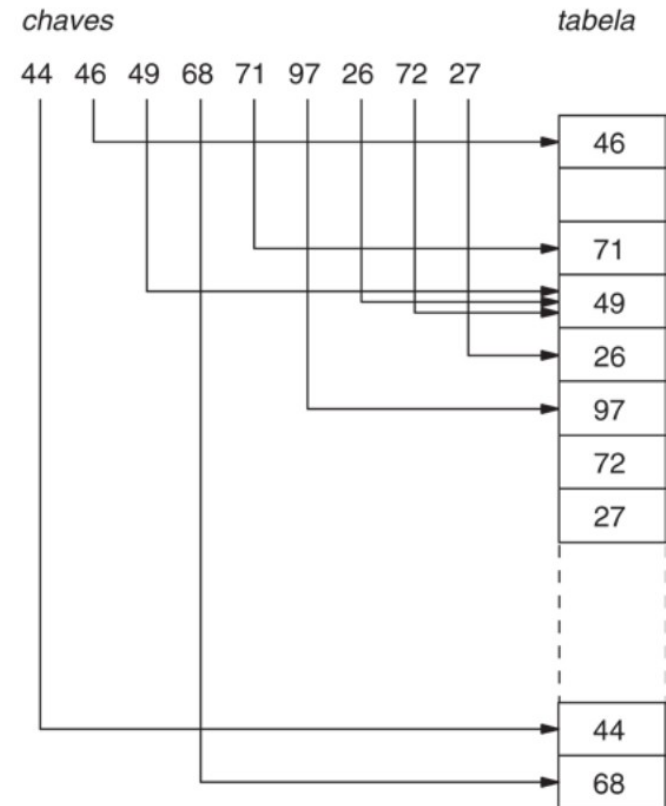
retorna j

$i \leftarrow i + 1$

$j \leftarrow h(k, i)$

retorna -1 /* CHAVE NÃO ENCONTRADA */

Ex:
busca 72



Tratamento de colisões

2) Endereçamento aberto

Remove(T,k)

$i \leftarrow 0$ /* nr da sondagem */

$j \leftarrow h(k, i)$

enquanto $i \neq m$ e $T[j] \neq \text{NIL}$

se $T[j] = k$

Elimina T[j]

$i \leftarrow i + 1$

$j \leftarrow h(k, i)$

retorna -1 /* CHAVE NÃO ENCONTRADA */

Como eliminar $T[j]$?

$T[j] \leftarrow \text{NIL}$? Isso iria romper a sequência de sondagens...

Tratamento de colisões

2) Endereçamento aberto

Remove(T,k)

$i \leftarrow 0$ /* nr da sondagem */

$j \leftarrow h(k, i)$

enquanto $i \neq m$ e $T[j] \neq \text{NIL}$

se $T[j] = k$

Elimina $T[j]$

$i \leftarrow i + 1$

$j \leftarrow h(k, i)$

retorna -1 /* CHAVE NÃO ENCONTRADA */

Como eliminar $T[j]$?

$T[j] \leftarrow \text{NIL}$? Isso iria romper a sequência de sondagens...

Solução similar à adotada no encadeamento interno (bit de validade)

Também precisará adaptar as funções de busca e inserção dos dois slides anteriores

(EXERCÍCIO!!!)

- Problema: tempo de pesquisa não depende mais somente do número elementos presentes na tabela, mas também do número de elementos eliminados.

Tratamento de colisões

2) Endereçamento aberto

Uma questão ainda fica em aberto. Como devem ser criadas as funções hash que recebem dois parâmetros:

- $h(k, i)$ onde $k \in C$ e $i \in \{0, 1, \dots, m-1\}$.

Três técnicas são comumente usadas:

- 1 Sondagem linear;
- 2 Sondagem quadrática;
- 3 Hash duplo.

Note que, embora os exemplos dos algoritmos anteriores (busca, inserção e remoção) usem a sondagem linear, os algoritmos admitem qualquer outra sondagem, pois baseiam-se em $h(k, i)$

Tratamento de colisões

2) Endereçamento aberto

Técnicas de sondagem

Tratamento de colisões

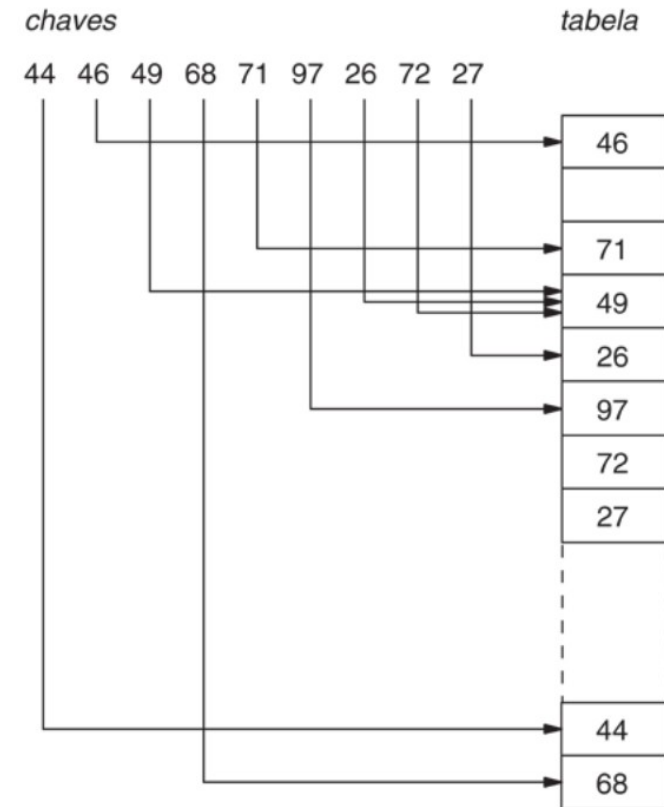
2.1) Endereçamento aberto – Sondagem Linear

Dada uma função hash comum $h' : \mathbb{C} \rightarrow \{0,1,\dots,m-1\}$, chamada de *função hash auxiliar*, o método de *sondagem linear* usa a função hash:

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

onde $i = 0,1,\dots,m-1$ e *mod* é a operação que retorna o resto de uma divisão (e.g., equivalente ao operador % do Java).

| Valor de i | Posição sondada |
|--------------|-----------------|
| 0 | $T[h'(k)]$ |
| 1 | $T[h'(k)+1]$ |
| ... | ... |
| ... | $T[m-1]$ |
| ... | $T[0]$ |
| ... | $T[1]$ |
| ... | ... |
| $m-1$ | $T[h'(k)-1]$ |



Tratamento de colisões

2.1) Endereçamento aberto – Sondagem Linear

Observações:

Ex: $m = 5$, $h(x, i) =$
 $x : i = 0, 1, 2, 3, 4$
 $h(11, i) = 1, 2, 3, 4, 0$
 $h(8, i) = 3, 4, 0, 1, 2$

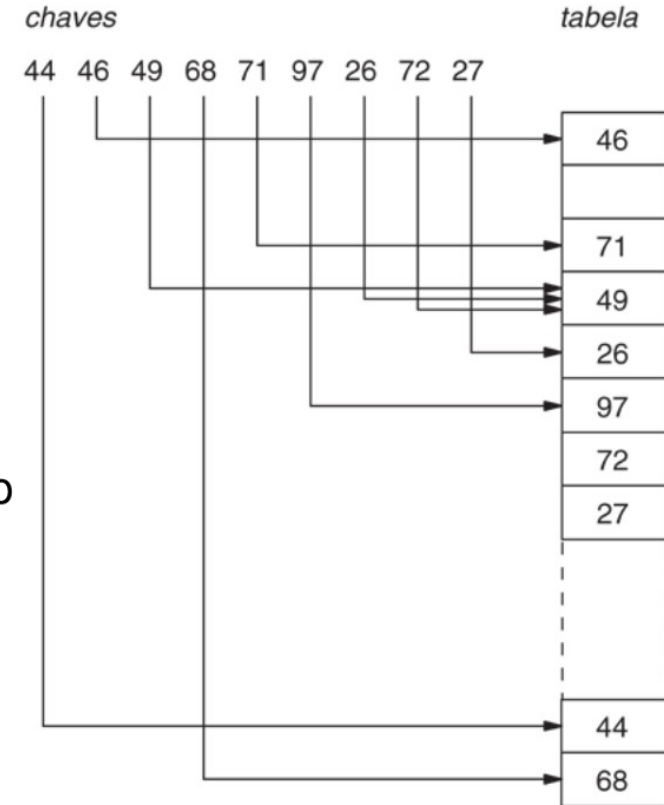
- A posição inicial $h'(k)$ de sondagem determina toda a seqüência posterior.
- Como conseqüência, só existem m seqüências de sondagem distintas.
- Fácil de implementar.
- Sofre de um problema conhecido como *agrupamento primário*.

Tratamento de colisões

2.1) Endereçamento aberto – Sondagem Linear

Agrupamento primário:

- Longas seqüências de posições ocupadas são construídas, aumentando o tempo médio de pesquisa.
- Surgem agrupamentos, pois uma posição vazia precedida por i posições completas é preenchida em seguida com probabilidade $(i+1)/m$.
- Seqüências de posições ocupadas tendem a ficar mais longas e o tempo médio de pesquisa aumenta.
- Gera no máximo m seqüências distintas, ou seja, número possível de seqüências é $\Theta(m)$.



Tratamento de colisões

2.2) Endereçamento aberto – Sondagem Quadrática

A *sondagem quadrática* utiliza uma função hash da forma:

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

onde h' é uma função hash auxiliar, c_1 e $c_2 \neq 0$ são constantes auxiliares e $i = 0, 1, \dots, m-1$.

Exemplo: $h(k, i) = (h'(k) + 0.5*i + 0.5*i*i) \bmod 17$ onde

$m = 17$, $h'(k) = k \bmod 17$, $c_1 = 0.5$ e $c_2 = 0.5$

Tratamento de colisões

2.3) Endereçamento aberto – Hash Duplo

O hash duplo é um dos melhores métodos disponíveis para endereçamento aberto, porque as permutações produzidas têm muitas características de permutações escolhidas aleatoriamente.

O *hash duplo* usa uma função hash da forma:

- $h(k, i) = (h_1(k) + ih_2(k)) \bmod m$

onde h_1 e h_2 são funções hash auxiliares.

| Valor de i | Posição sondada |
|--------------|-------------------------------------|
| 0 | $T[h_1(k)]$ |
| 1 | $T[(h_1(k) + h_2(k)) \bmod m]$ |
| 2 | $T[(h_1(k) + 2h_2(k)) \bmod m]$ |
| ... | ... |
| $m-1$ | $T[(h_1(k) + (m-1)h_2(k)) \bmod m]$ |

Tratamento de colisões

2.3) Endereçamento aberto – Hash Duplo

Observações:

- Diferentemente das sondagens quadrática e linear, a seqüência de sondagem depende da chave k de duas maneiras.
- A posição de sondagem inicial e o deslocamento, ambos, podem variar.

Questão importante: como escolher h_1 e h_2 ?

Tratamento de colisões

2.3) Endereçamento aberto – Hash Duplo

Para que a tabela hash inteira seja pesquisada, o valor de $h_2(k)$ e o tamanho m da tabela hash devem ser primos entre si (a e b são primos entre si se o máximo divisor comum for 1).

Formas de conseguir isto:

- 1 Fazer m uma potência de 2 e h_2 gerar sempre um número ímpar.
- 2 Fazer m igual a um primo e projetar h_2 para retornar um inteiro positivo sempre menor que m .

Tratamento de colisões

2.3) Endereçamento aberto – Hash Duplo

Para o caso 2, supondo m um número primo, podemos ter h_1 e h_2 :

① $h_1(k) = k \bmod m,$

② $h_2(k) = 1 + (k \bmod m'),$

onde m' é escolhido com um valor ligeiramente menor que m (digamos, $m - 1$).

Exemplo:

- Para $k = 123456$, $m = 701$ e $m' = 700$, tem-se $h_1(123456) = 80$ e $h_2(123456) = 257$.
- Portanto, a primeira posição sondada é de número 80; as demais estão separadas por 257 posições.
- Ou seja: 80, 337, 594, 150, ...

Tratamento de colisões

2.3) Endereçamento aberto – Hash Duplo

O hash duplo é um aperfeiçoamento em relação à sondagem linear e quadrática:

- o número possível de seqüências geradas é proporcional a m^2 , pois cada par $\langle h_1(k), h_2(k) \rangle$ gera uma seqüência distinta.

Neste sentido, o hash duplo é mais próximo do desempenho ideal do *hash uniforme*.

- No hash uniforme, a função $h(k, i)$ pode gerar qualquer permutação das m posições, isto é, o número possível de seqüências seria $m!$, ou seja, $\Theta(m!)$.
- O hash uniforme é difícil de implementar; na prática, utiliza-se aproximações como o hash duplo.

Tratamento de colisões

2) Endereçamento aberto – Resumo das sondagens

- 1 Sondagem linear \Rightarrow número de seqüências possíveis é $\Theta(m)$.
Problema: agrupamento primário.
- 2 Sondagem quadrática \Rightarrow número de seqüências possíveis é $\Theta(m)$.
Problema: agrupamento quadrático.
- 3 Hash duplo \Rightarrow número de seqüências possíveis é $\Theta(m^2)$.
Mais próximo do hash uniforme.

Tratamento de colisões

Estratégias:

A) Hashing estático (tamanho da tabela é constante)

1) Encadeamento ou endereçamento fechado – colisões vão para uma lista ligada

1.1) Encadeamento exterior (fora da tabela)

1.2) Encadeamento interior (dentro da tabela)

2) Endereçamento aberto (chaves dentro da tabela, sem ponteiros)

2.1) Tentativa/Sondagem linear

2.2) Tentativa/Sondagem quadrática

2.3) Dispersão dupla / Hash duplo

B) Hashing dinâmico (tabela pode expandir/encolher)

3) Hashing extensível (estrutura de dados adicional)

4) Hashing linear

Referências

Conceitos gerais de Hashing:

SZWARCFITER, J. L.; MARKENZON, L. Estruturas de Dados e Seus Algoritmos. Ed. LTC, 3^a ed, 2013. Capítulo 10 (figuras do livro)

Slides dos Profs. M. Chaim, Delano Beder e L. Digiampietri

Hashing estático em disco

Hashing Interno x Externo

- Hashing interno:
 - Hashing em memória principal
 - Cada slot da tabela de hash é um registro
 - Colisões em lista ligada (endereçamento fechado = hashing aberto) ou em outro slot (endereçamento aberto = hashing fechado)
- Hashing externo:
 - hashing em memória secundária (armazenamento e recuperação em disco)
 - Cada slot da tabela de hash é um bucket (um bloco ou cluster de blocos em disco)
 - Colisões vão preenchendo o bucket
 - Tabela de hash fica no cabeçalho do arquivo, e tem o nr do bloco ($m = \text{nr de blocos do arquivo}$)
 - Acessar um bucket (bloco) → realizar um *seek*

Hashing em disco

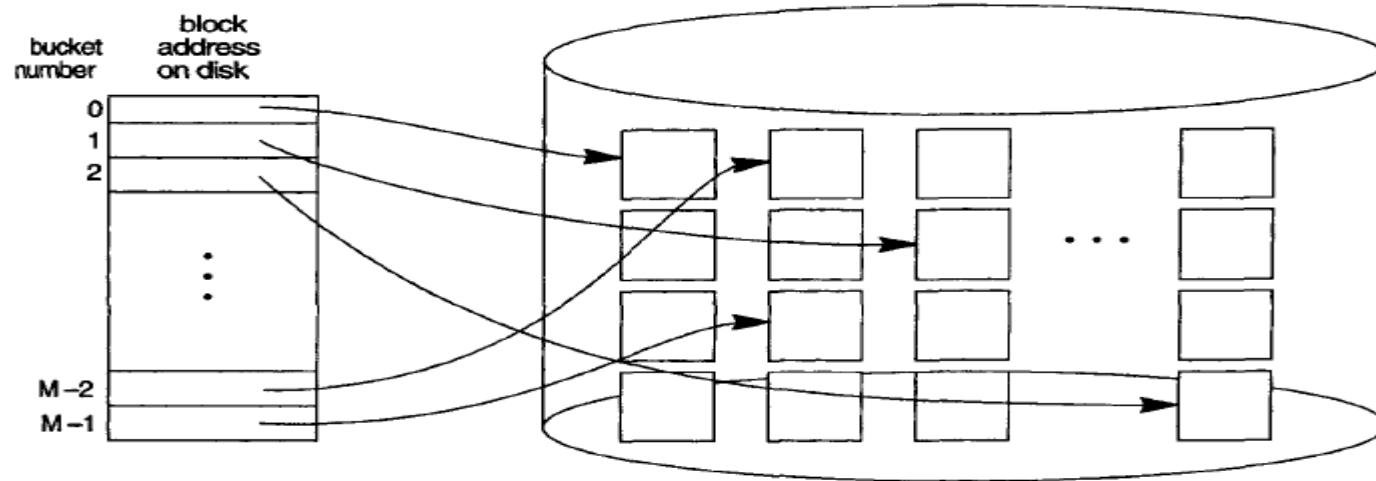


FIGURE 13.9 Matching bucket numbers to disk block addresses.

Hashing → Organização de arquivos

- ✓ Organização sequencial de um arquivo → necessária estrutura de índice ou busca binária → mais operações de I/O
- ✓ *Hashing* → permite evitar acesso a estruturas de índices
- ✓ *Hashing* também permite meio para construir índices (ex: cidades dos clientes)
- ✓ **Organização de arquivos em *Hashing***
 - obtém diretamente o endereço do bloco de disco que contém um registro desejado usando uma função sobre o valor da chave

Hashing → Organização de arquivos

Bucket 0



Bucket 1



Bucket 2



Hashing → Organização de arquivos

Bucket 0



Bucket 1



Bucket 2



Exemplo de chave

Código do cliente (valor numérico)

FUNÇÃO HASH:

$$H(\text{CODIGO}) = \text{CODIGO} \bmod 3$$

CÓDIGO = 25

Hashing → Organização de arquivos

Bucket 0



Bucket 1



Bucket 2



Exemplo de chave
Código do cliente (valor numérico)

FUNÇÃO HASH:
 $H(\text{CODIGO}) = \text{CODIGO} \bmod 3$

CÓDIGO = 25

Hashing → Organização de arquivos

Bucket 0



Bucket 1



Bucket 2



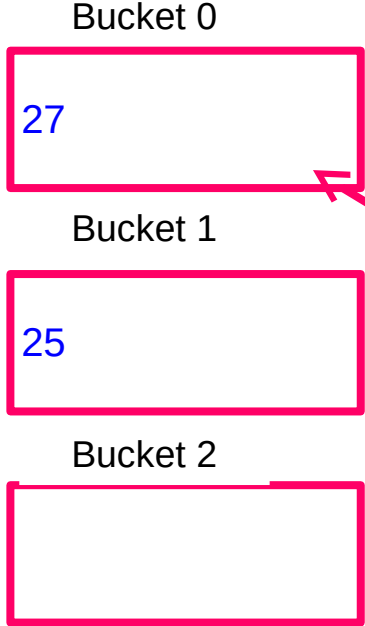
Exemplo de chave
Código do cliente (valor numérico)

FUNÇÃO HASH:

$$H(\text{CODIGO}) = \text{CODIGO} \bmod 3$$

CÓDIGO = 27

Hashing → Organização de arquivos



Exemplo de chave
Código do cliente (valor numérico)

FUNÇÃO HASH:
 $H(\text{CODIGO}) = \text{CODIGO} \bmod 3$

CÓDIGO = 27

Hashing → Organização de arquivos

Bucket 0



Bucket 1



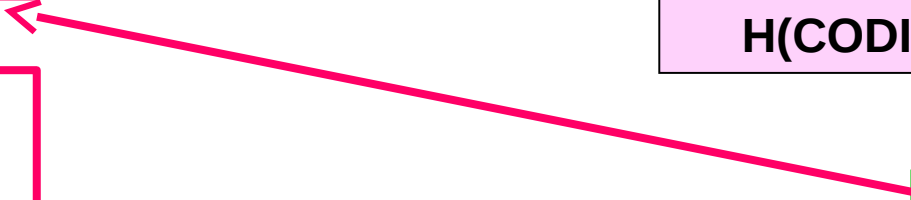
Bucket 2



Exemplo de chave
Código do cliente (valor numérico)

FUNÇÃO HASH:
 $H(\text{CODIGO}) = \text{CODIGO} \bmod 3$

CÓDIGO = 28



Fator de carga

- Hashing interno:
 - $\alpha = N/M$, $N =$ nr de registros, $M =$ nr de slots
- Hashing externo:
 - $\alpha = N/(M*r)$, $N =$ nr de registros, $M =$ nr de slots (buckets), $r =$ número de registros que cabem em um bucket
 - Isso torna os algoritmos de busca MUITO eficientes

Colisões

- Se $h(x) = h(y) = i \rightarrow x$ e y vão para o bucket i
(h = função de hash)
- E se o bucket i estiver lotado?

1) Encadeamento (endereçamento fechado) - Buckets de overflow !

- Opção 1: compartilhados
- Opção 2: exclusivos por endereço-base

1) Hashing aberto

X

2) Endereçamento aberto – vai para outro bucket

- Ex: Sondagem linear

2) Hashing fechado

(conceitos invertidos no livro do Silberchatz)

1.1) Buckets de overflow compartilhados

- Buckets de overflow possuem uma lista ligada de REGISTROS que transbordaram de seus buckets
- Final de buckets principais (não overflow) lotados: ponteiro para o próximo REGISTRO em um bucket de overflow
- Há uma lista livre: lista ligada de registros desocupados nos buckets de overflow – início da lista livre pode ficar no cabeçalho do arquivo *

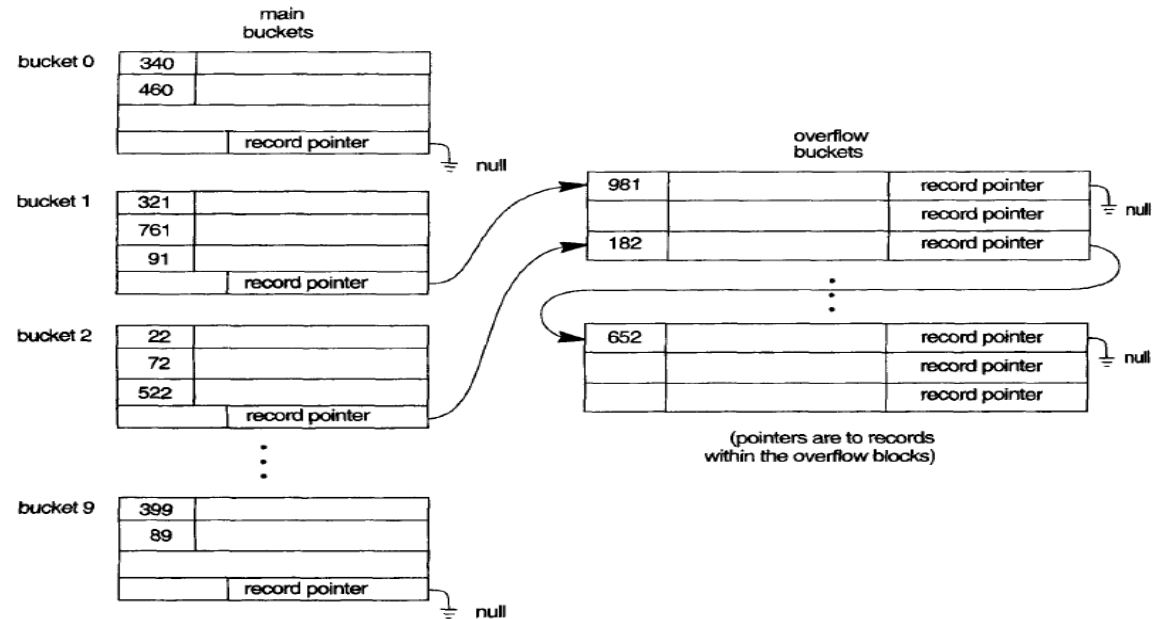


FIGURE 13.10 Handling overflow for buckets by chaining.

(ELMARIS, NAVATHE, 2004)

* Agora fica claro porque registros de tamanho fixo é mais utilizado...

1.1) Buckets de overflow compartilhados

- **Busca:** procura no bucket principal (endereço-base dado pela função de hash), se não encontrar segue a lista ligada de registros
- **Inserção:** se não houver espaço no bucket principal, “remove” um espaço da lista livre e insere no início da lista ligada de registros (nos buckets de overflow)
- **Remoção:**
 - Se em bucket de overflow, adiciona o registro à lista livre
 - se em bucket principal, traz algum registro de um bucket de overflow, se houver (o primeiro por ex)

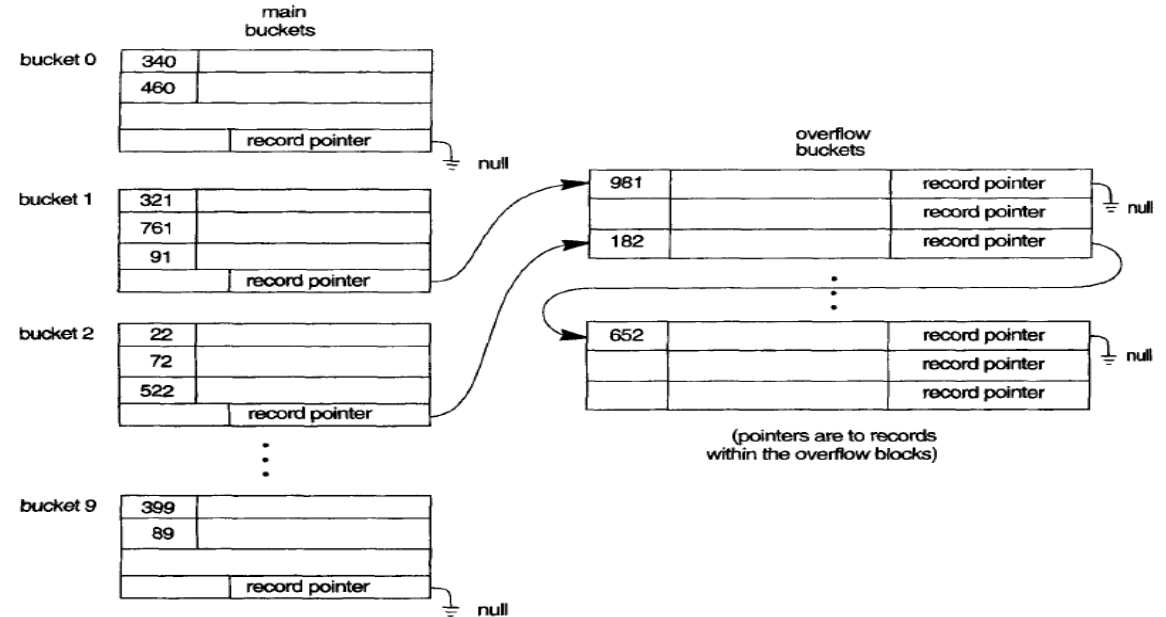


FIGURE 13.10 Handling overflow for buckets by chaining.

(ELMARIS, NAVATHE, 2004)

1.1) Buckets de overflow compartilhados

- **Busca:** procura no bucket principal (endereço-base + hash), se não ligada de reg

- **Inserção:** se bucket principal espaço da lista início da lista buckets de o

- **Remoção:**
 - Se em bucket de overflow, adiciona o registro à lista livre
 - se em bucket principal, traz algum registro de um bucket de overflow, se houver (o primeiro por ex)

EXERCÍCIO:

Proponha uma estrutura de dados para essa estratégia (buckets principais, buckets de overflow, lista livre, etc) e implemente em C as rotinas de busca, inserção e remoção.



FIGURE 13.10 Handling overflow for buckets by chaining.

(ELMARIS, NAVATHE, 2004)

Referências

Conceitos gerais de Hashing:

SZWARCFITER, J. L.; MARKENZON, L. Estruturas de Dados e Seus Algoritmos. Ed. LTC, 3ª ed, 2013. Capítulo 10 (figuras do livro)

Slides dos Profs. M. Chaim, Delano Beder e L. Digiampietri

Hash em Disco:

ELMARIS, R.; NAVATHE, S. B. Fundamentals of Database Systems. 4 ed. Ed. Pearson-Addison Wesley. Cap 13.8. 4 ed. Pearson. 2004

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. Database System Concepts, 6. ed. McGraw Hill, 2011.

Slides da Profa. Fátima L. S. Nunes.