ACH2024

Aula 23 – Hashing em Disco (parte 2) e Hashing Dinâmico Extensível

Profa. Ariane Machado Lima

Aulas passadas

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

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

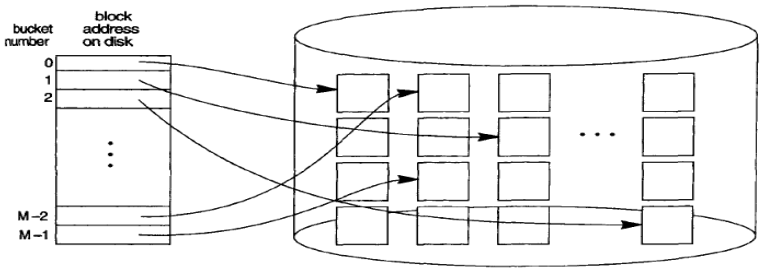


FIGURE 13.9 Matching bucket numbers to disk block addresses.

Tipos de organização de arquivos

- Sequencial
- Lista ligada (com ou sem tabela de alocação)
- Indexada
  - Um nível de índices ou índices multiníveis
- Árvores B, B+ ou B*
- Hashing

Colisões

- Se h(x) = h(y) = i → 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

2) Endereçamento aberto – vai para outro bucket
   - Ex: Sondagem linear

1) Hashing aberto

X

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 *

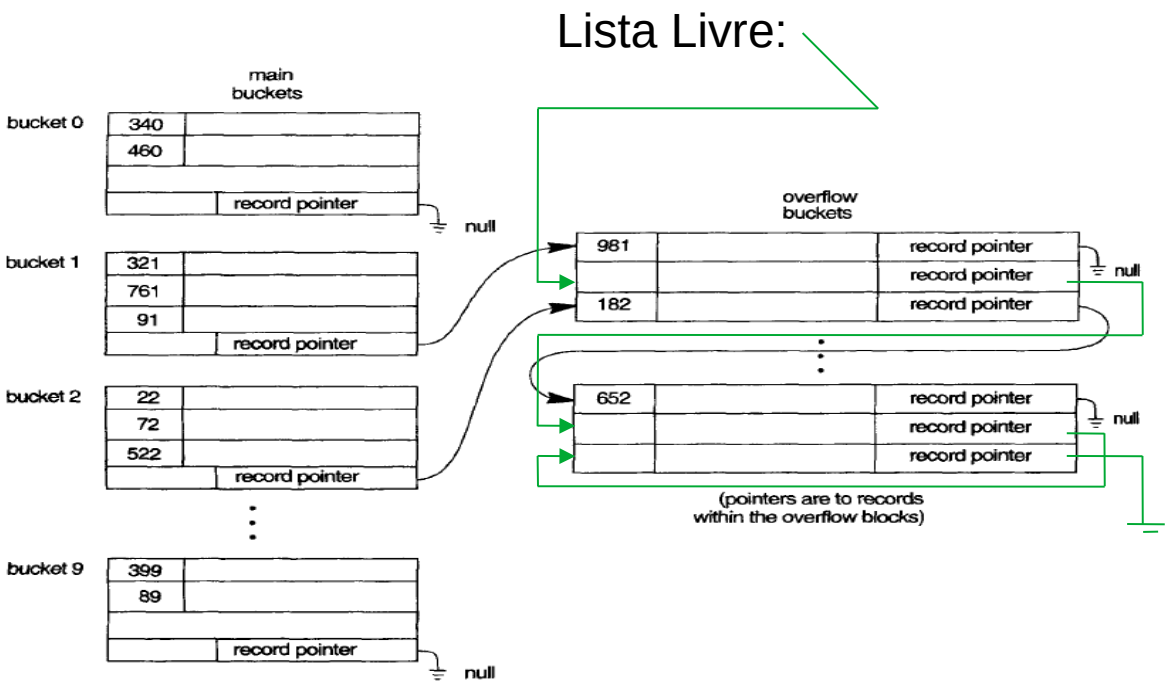


FIGURE 13.10 Handling overflow for buckets by chaining.

(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

Complexidade: ?

Ex: busca 652;
h(652) = 2

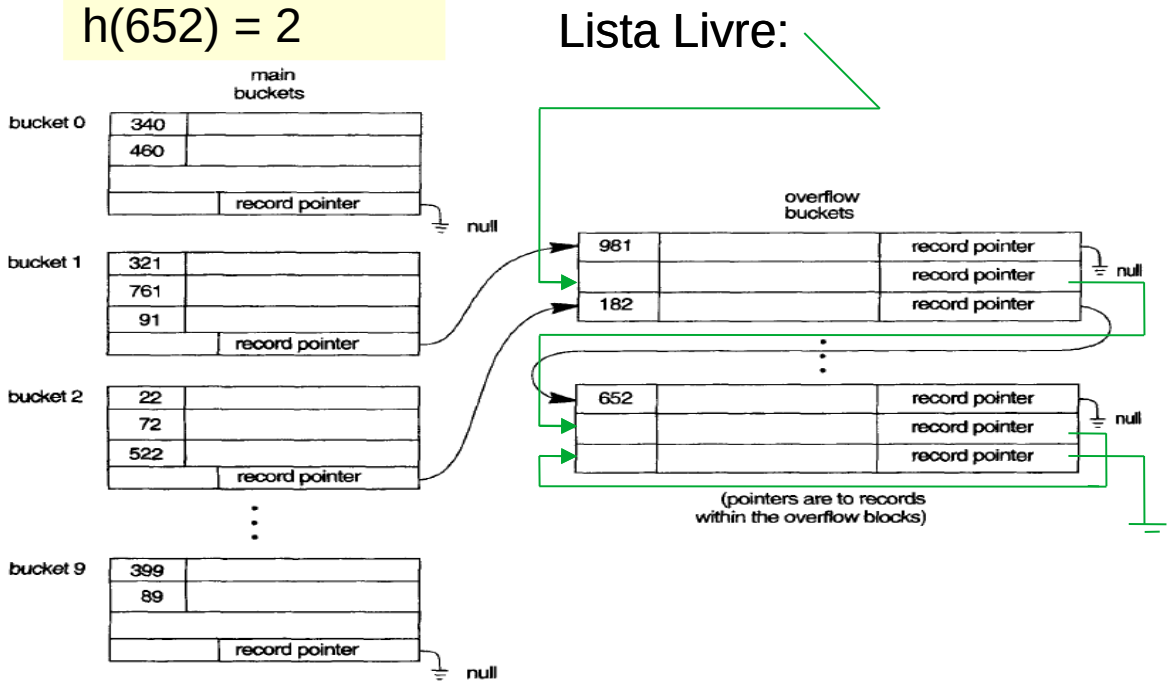


FIGURE 13.10 Handling overflow for buckets by chaining.

(ELMARIS, NAVATHE, 2004)

1.1) Buckets de overflow compartilhados

- **Busca**: procura no bucket principal (endereço-base dado pela função de hash), se não encontrar segue a lista ligada de registros

Complexidade: O(r)

r = número de registros que deveriam estar em um mesmo bucket principal mas que estão em buckets de overflow

1 seek para acessar o bloco principal, e no pior dos casos 1 seek para registro que esteja em blocos de overflow (quando cada registro está em um bloco de overflow diferente do anterior)

Ex: busca 652;
h(652) = 2

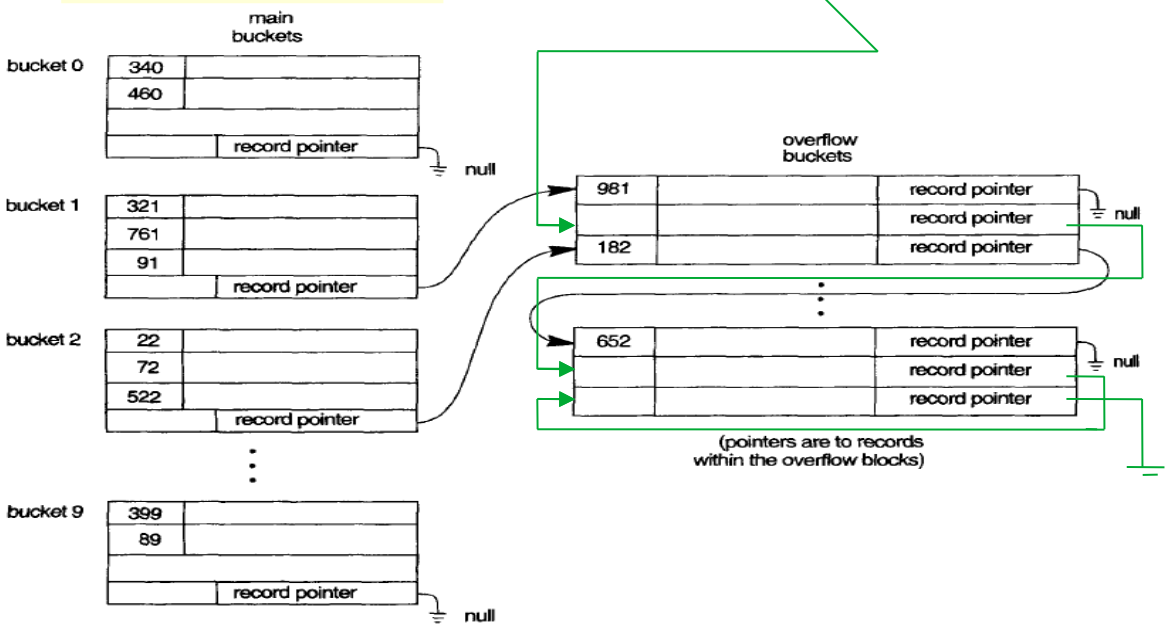


FIGURE 13.10 Handling overflow for buckets by chaining.

(ELMARIS, NAVATHE, 2004)

1.1) Buckets de overflow compartilhados

- **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)

Complexidade: ?

Ex: insere 432;
h(432) = 2

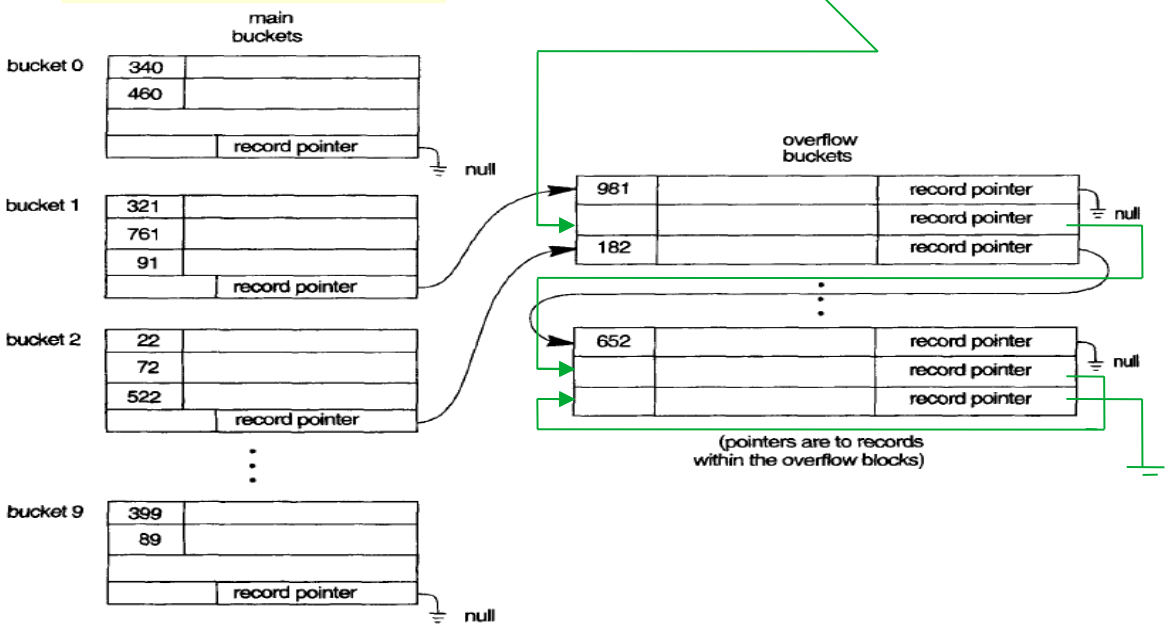


FIGURE 13.10 Handling overflow for buckets by chaining.

(ELMARIS, NAVATHE, 2004)

1.1) Buckets de overflow compartilhados

- **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)

Complexidade: ?

Ex: insere 432;
h(432) = 2

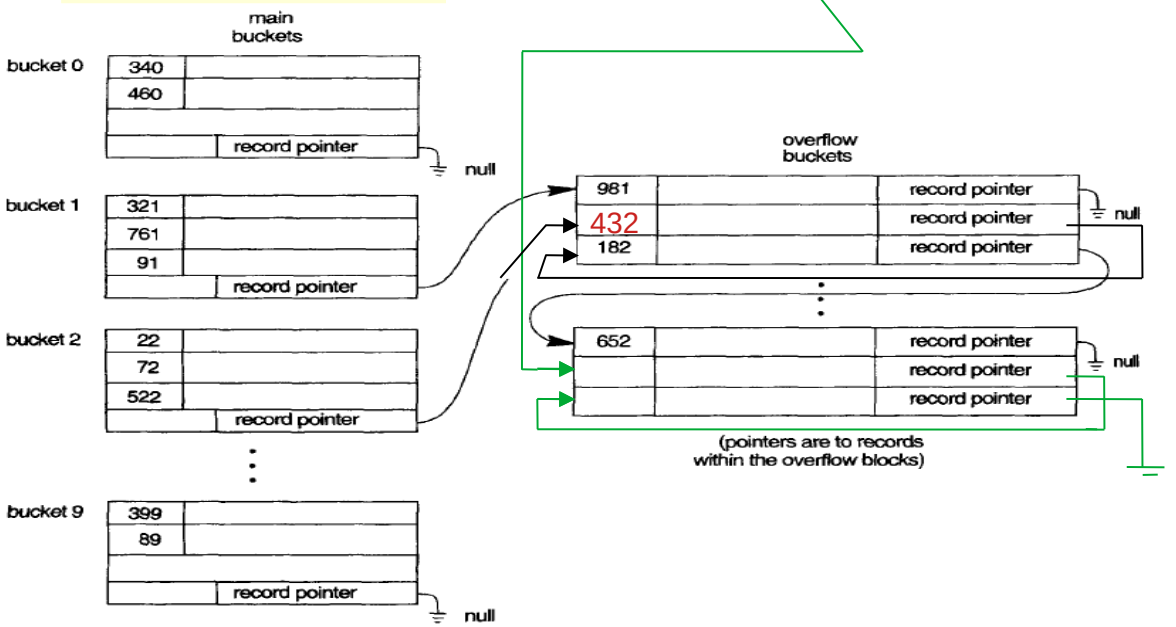


FIGURE 13.10 Handling overflow for buckets by chaining.

(ELMARIS, NAVATHE, 2004)

1.1) Buckets de overflow compartilhados

- **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)

Complexidade: O(1)

1 seek para ver se tem espaço no bucket principal

1 seek para acessar bloco contendo o primeiro registro da lista livre

Ex: insere 432;
h(432) = 2

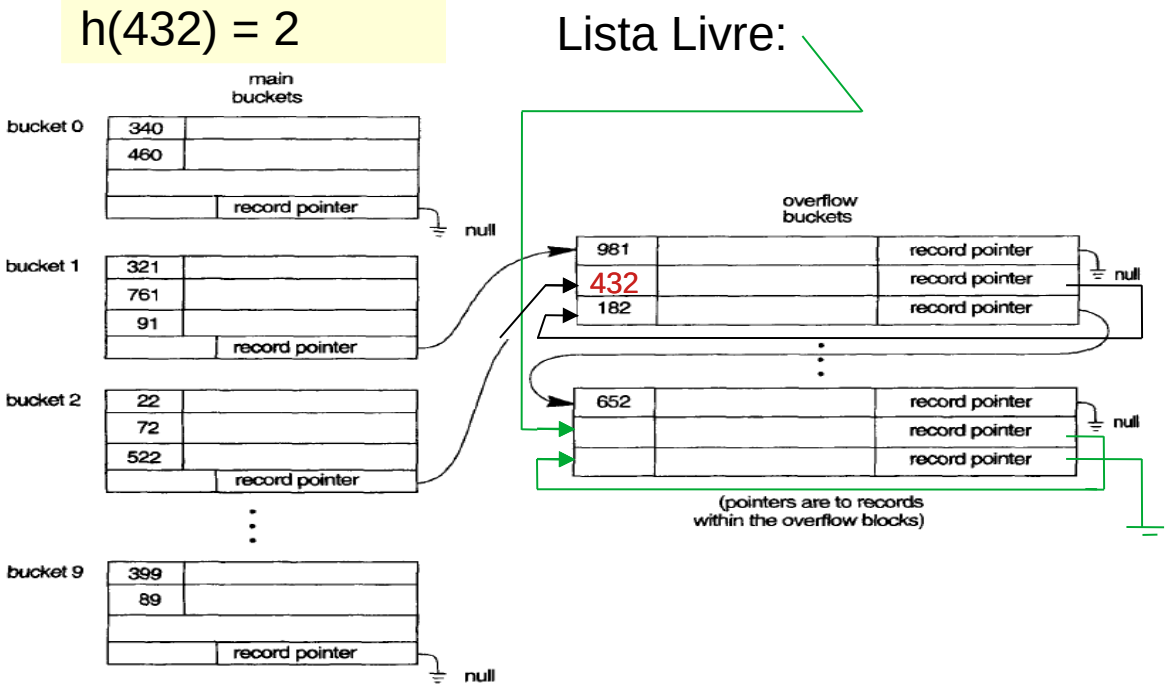


FIGURE 13.10 Handling overflow for buckets by chaining.

(ELMARIS, NAVATHE, 2004)

1.1) Buckets de overflow compartilhados

- **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)

Complexidade: ?

Ex: remove 182;
h(182) = 2

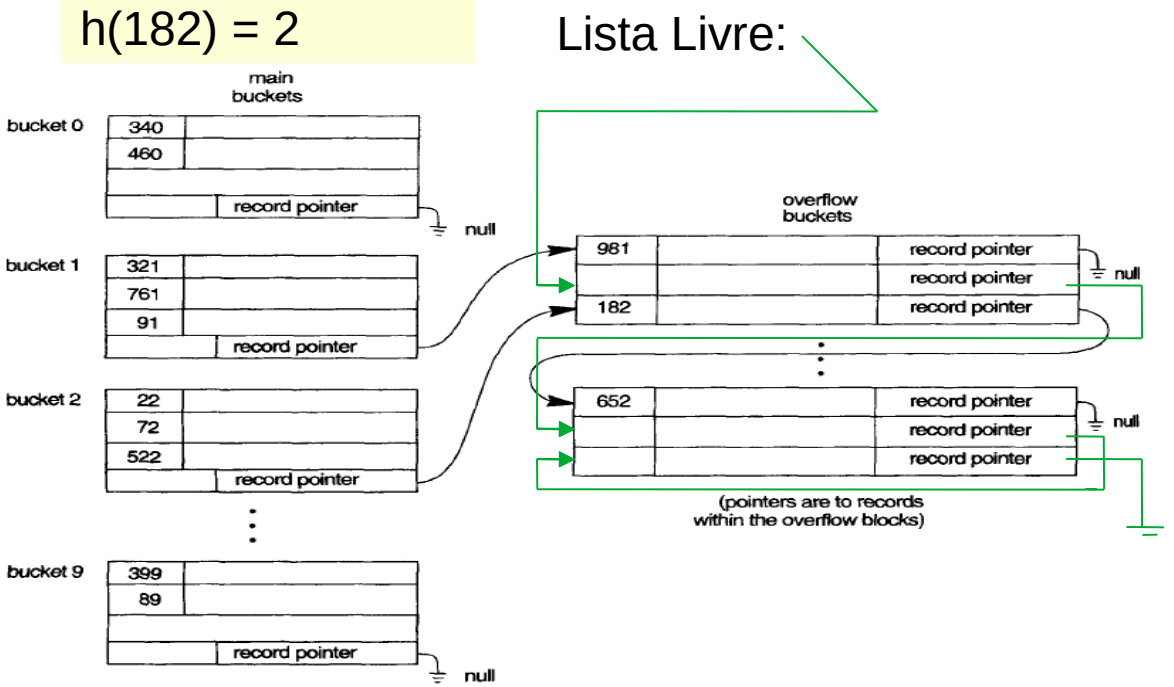


FIGURE 13.10 Handling overflow for buckets by chaining.

(ELMARIS, NAVATHE, 2004)

1.1) Buckets de overflow compartilhados

- **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)

Complexidade:
Envolve uma busca
No bucket de overflow:
nenhum seek adicional se considerar que fez seek na busca (e que também guardou o bloco do registro anterior)

Ex: remove 182;
h(182) = 2

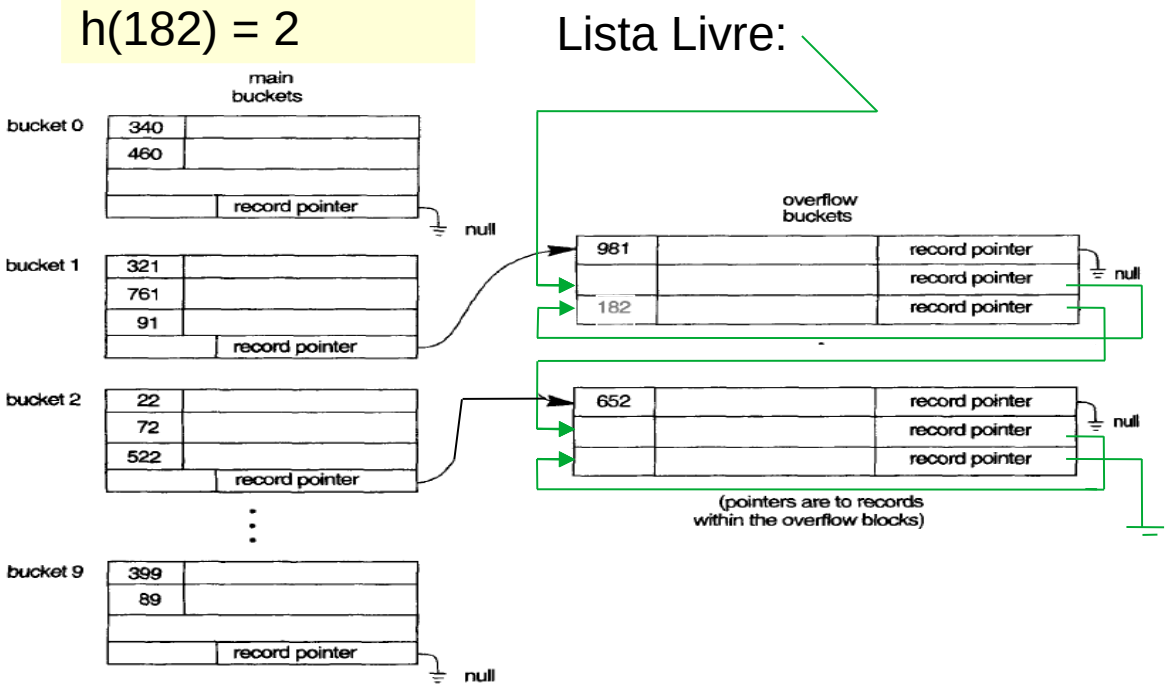


FIGURE 13.10 Handling overflow for buckets by chaining.

(ELMARIS, NAVATHE, 2004)

1.1) Buckets de overflow compartilhados

- **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)

Complexidade: ?

Ex: remove 72;
h(72) = 2

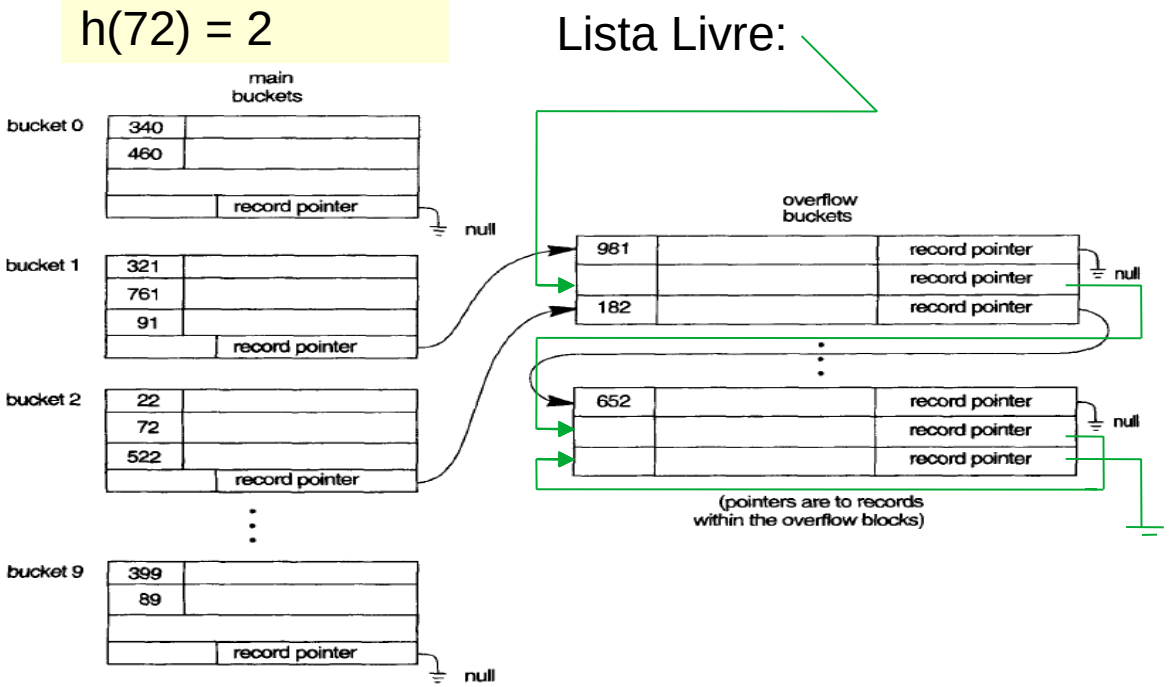


FIGURE 13.10 Handling overflow for buckets by chaining.

(ELMARIS, NAVATHE, 2004)

1.1) Buckets de overflow compartilhados

- **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)

Complexidade: O(1) neste caso
1 seek apenas realizado na busca (para achar o bucket principal)
Mais 1 seek para trazer o primeiro registro em bucket de overflow (se houver)

Ex: remove 72;
h(72) = 2

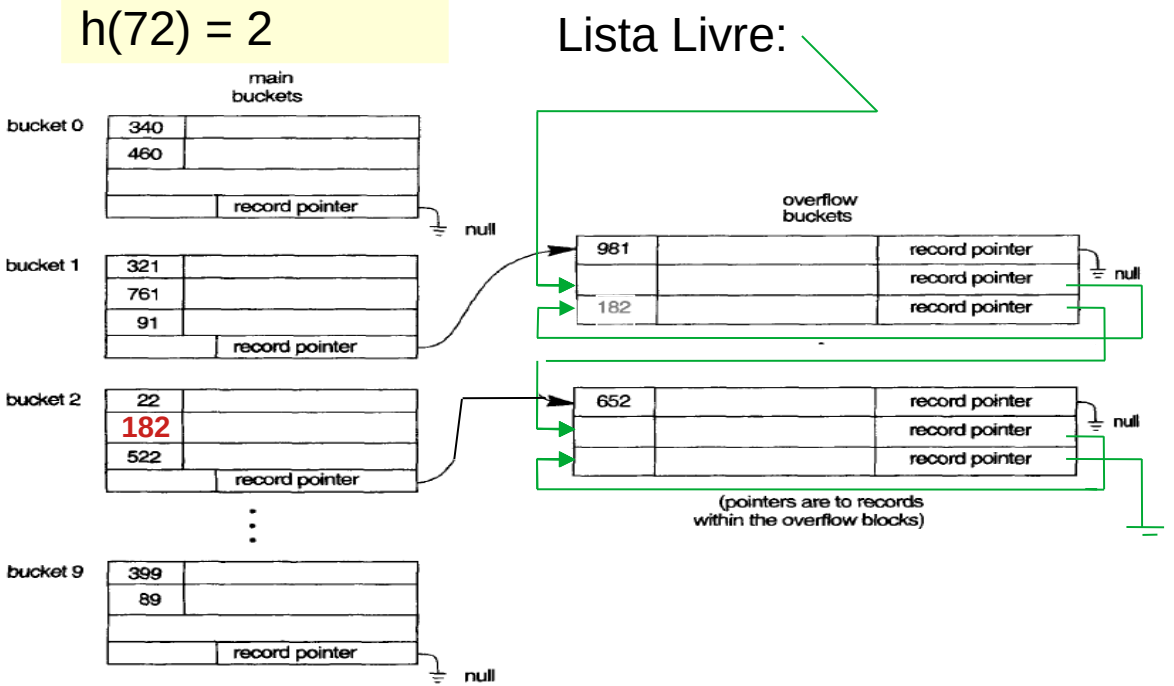


FIGURE 13.10 Handling overflow for buckets by chaining.

(ELMARIS, NAVATHE, 2004)

AULA DE HOJE:

Colisões

- Se h(x) = h(y) = i → x e y vão para o bucket i

  (h = função de hash)

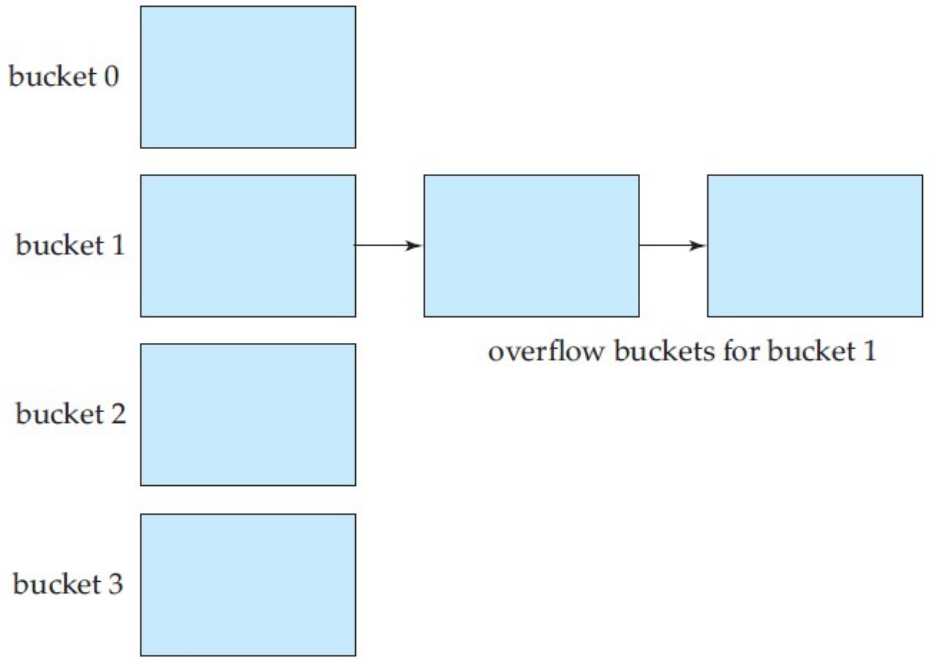
- E se o bucket i estiver lotado?

1) Encadeamento (endereçamento fechado) - Buckets de overflow !
   - Opção 1: compartilhados
   - Opção 2: exclusivos por endereço-base

2) Endereçamento aberto – vai para outro bucket
   - Ex: Sondagem linear

1.2) Buckets de overflow exclusivos

- Lista ligada de buckets de overflow para cada endereço base
- Final de buckets (principais e de overflow) lotados: ponteiro para o próximo bucket de overflow



(SILBERSCHATZ, 2011)

1.2) Buckets de overflow exclusivos

Busca: procura no bucket principal (endereço base dado pela função de hash), se não encontrar segue a lista ligada de buckets

Inserção: insere no final do bucket principal ou no último bucket de overflow

Remoção:

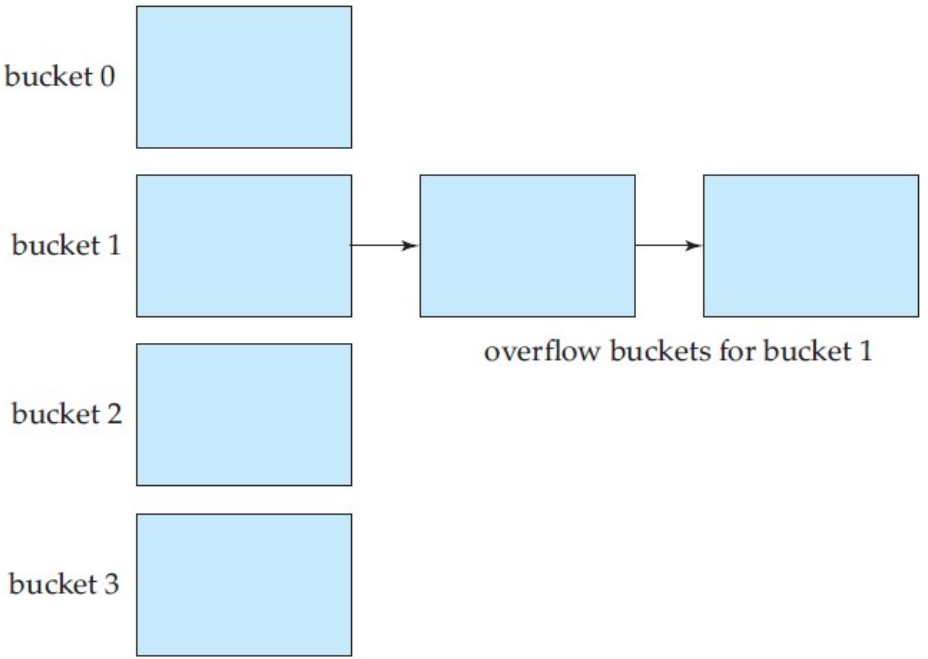
- Remove e move para esse lugar o último registro (do principal ou do último bucket de overflow se houver)

- Ou usa bit de validade e reorganiza depois (cada acesso a um bucket é um seek…) → precisa adaptar busca e inserção

Em geral: Note que essa estratégia é mais simples que a opção 1.1 (buckets de overflow compartilhados)

mas desperdiça mais espaço…

mas em média faz menos seeks do que percorrer lista ligada de registros espalhados por vários buckets de overflow...



(SILBERSCHATZ, 2011)

2) Endereçamento aberto

- Busca: segue sequência de sondagens pelos buckets da tabela (e procura registro dentro de cada bucket)
- Inserção:
  - no primeiro bucket disponível identificado por endereçamento aberto
- Remoção:
  - Enquanto o bucket não ficar vazio tudo bem. Se ficar, precisa ter os mesmos cuidados (inclusive para busca e inserção, com bit de validade) que mencionados em endereçamento aberto para memória principal

Overflow, ops...

- Overflows aumentam o tempo de busca
- O ideal é ter um M (nr de slots/buckets) que não acarrete em overflow, sem muita perda de espaço
  - M = N/r (1+d)

  N: nr de registros do arquivo

  r: número de registros que cabem em um bucket

  d = fator de *fudge* – tipicamente ao redor 0,2 (fudge = falsificação)

  aproximadamente 20% do espaço dos *buckets* será perdido
- Hashing estático: esse M é fixo!
  - Mas o que fazer quando o arquivo aumenta ou diminui de tamanho? Teremos overflows ou perda de espaço…
  - O ideal seria se M fosse dinâmico, alterando-se com o tamanho do arquivo

Hashing Dinâmico

- Para tratamento de dinamismo nos tamanhos de arquivos
- Hashing Extensível
  - Manutenção de uma estrutura adicional
- Hashing Linear
  - Não usa nenhuma estrutura adicional

Overflow, ops...

- Overflows aumentam o tempo de busca
- O ideal é ter um M (nr de slots) que não acarrete em overflow, sem muita perda de espaço
- Hashing estático: esse M é fixo!
  - Mas o que fazer quando o arquivo aumenta ou diminui de tamanho
    - Se manter o mesmo hash (M, função, etc) teremos overflows ou perda de espaço…
    - Se for reorganizar depois gasta muito tempo
  - O ideal seria se M fosse dinâmico, alterando-se com o tamanho do arquivo

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

Hashing Extensível

Hashing Extensível

Pode-se usar uma função de hash mais uniforme, que mapeie as chaves para um intervalo grande (tipicamente inteiro de 32 bits), pois não será criada inicialmente uma tabela desse tamanho...

Diretório: array de 2^i endereços de buckets

- i : **profundidade global** do diretório
- Cada posição refere-se aos i bits mais significativos de um valor de hash h(k) → todos os registros cujas chaves k possuem valores de hash h(k) com os mesmos i primeiros bits são mapeados para a mesma entrada no diretório
- Cada entrada tem um endereço de bucket que contém tais registros
- A vantagem é que diferentes entradas podem apontar para o mesmo bucket ou não
  - Registros com os mesmos i' primeiros bits, i' < i poderiam caber em um mesmo bucket
  - O valor i' depende de cada bucket b (i_b) , e deve ser armazenado com eles: i_b – **profundidade local**

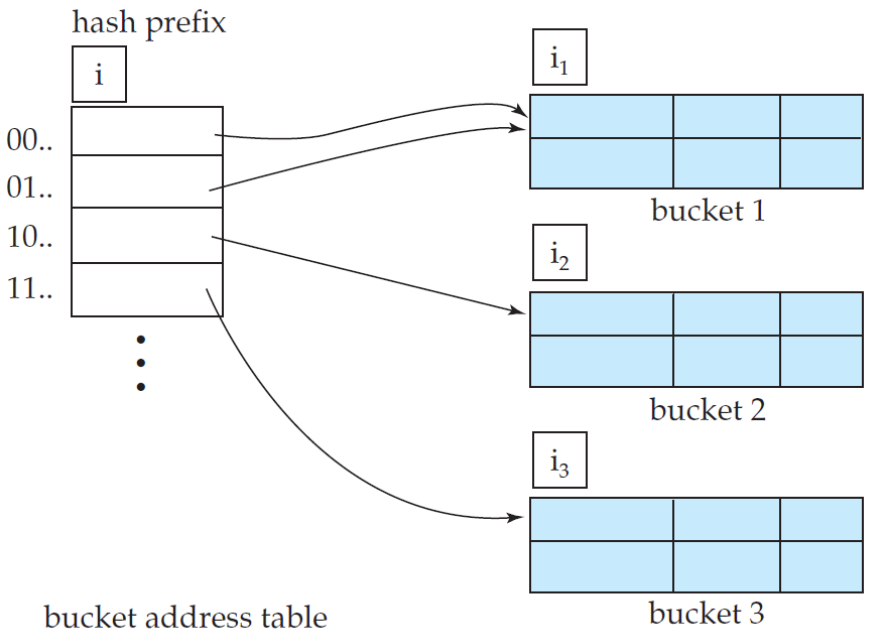


Figure 11.26 General extendable hash structure.

(SILBERSCHATZ, 2011)

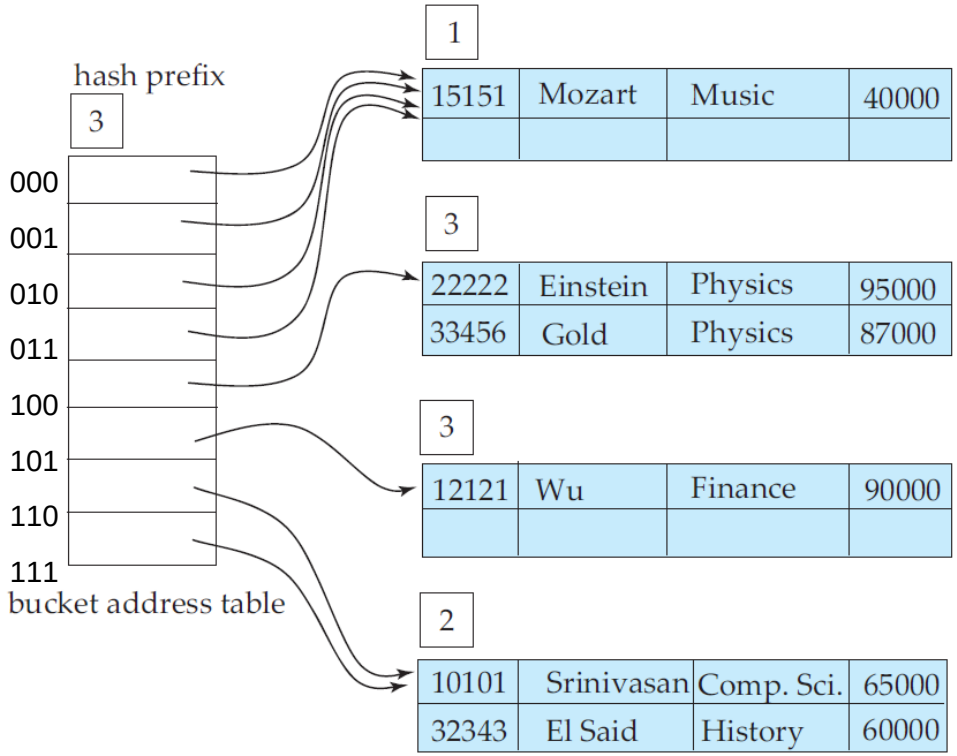
Hashing Extensível

Pode-se usar uma função de hash mais uniforme, que mapeie as chaves para um intervalo grande (tipicamente inteiro de 32 bits), pois não será criada inicialmente uma tabela desse tamanho...

Diretório: array de 2^i endereços de buckets

- i : **profundidade global** do diretório
- Cada posição refere-se aos i bits mais significativos de um valor de hash h(k) → todos os registros cujas chaves k possuem valores de hash h(k) com os mesmos i primeiros bits são mapeados para a mesma entrada no diretório
- Cada entrada tem um endereço de bucket que contém tais registros
- A vantagem é que diferentes entradas podem apontar para o mesmo bucket ou não
  - Registros com os mesmos i' primeiros bits, i' < i poderiam caber em um mesmo bucket
  - O valor i' depende de cada bucket b (i_b) , e deve ser armazenado com eles: i_b – **profundidade local**

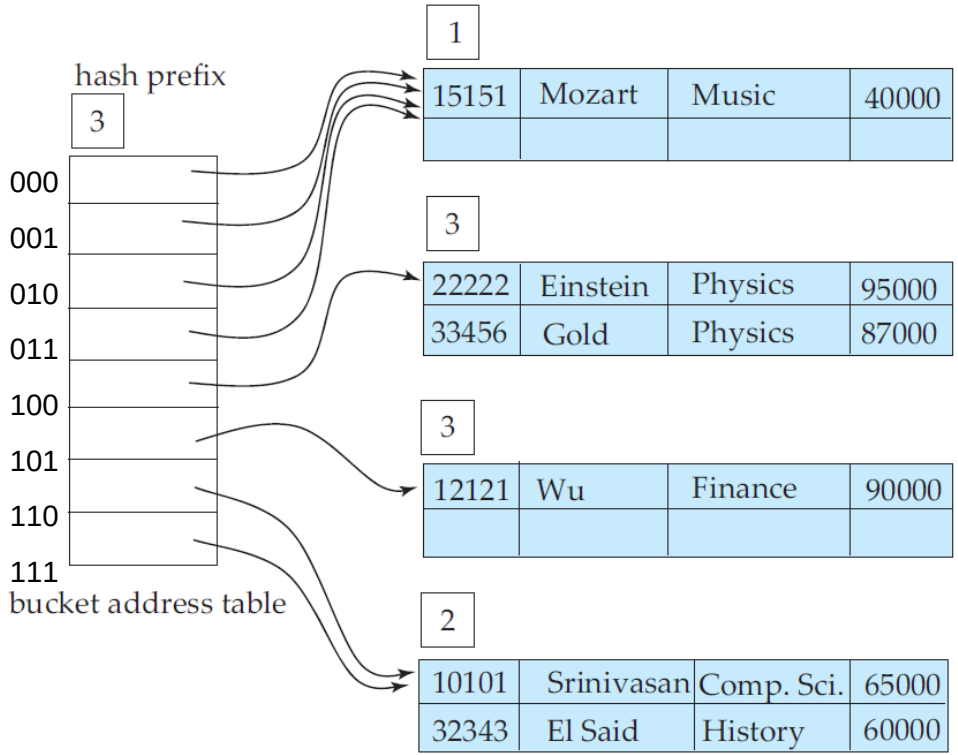
$2^{(i-ib)}$ entradas apontam para o bucket b



(SILBERSCHATZ, 2011)

Hashing Extensível

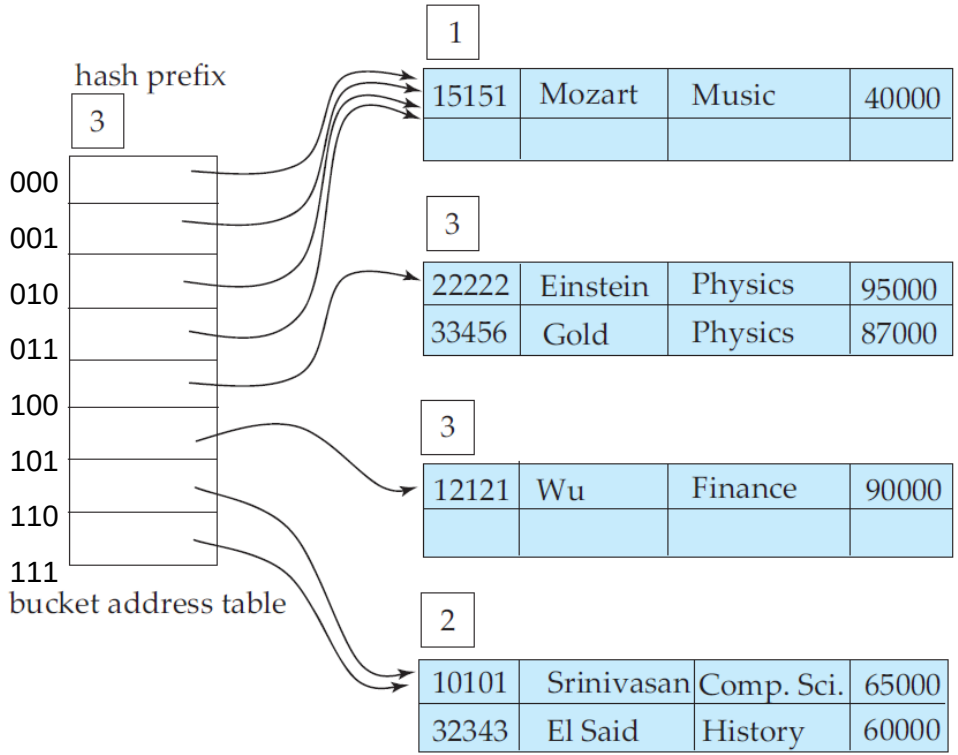
- Por que “extensível”?
  - Além de i_b poder aumentar até i, ou diminuir até 1, i também pode crescer
  - O valor i pode subir uma unidade por vez → dobra o tamanho do diretório
    - Quando?
  - O valor i pode decrescer uma unidade por vez → corta pela metade o tamanho do diretório
    - Quando?



(SILBERSCHATZ, 2011)

Hashing Extensível

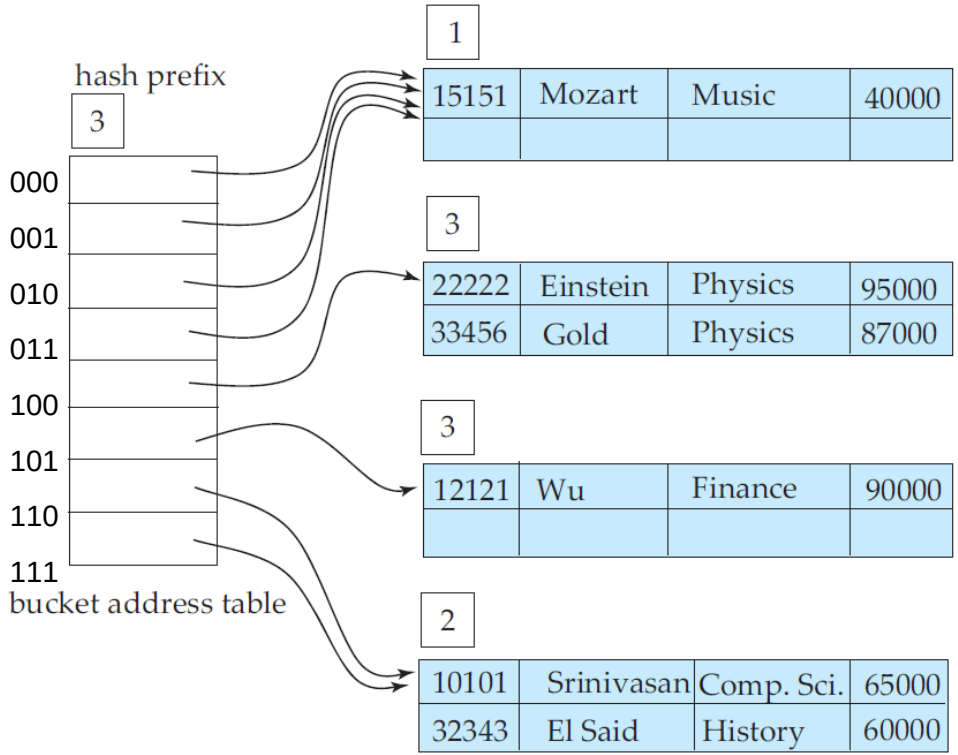
- Por que “extensível”?
  - Além de i_b poder aumentar até i, ou diminuir até 1, i também pode crescer
  - O valor i pode subir uma unidade por vez → dobra o tamanho do diretório
    - Quando há overflow de um bucket b com $i_b = i$
  - O valor i pode decrescer uma unidade por vez → corta pela metade o tamanho do diretório
    - Quando ?



(SILBERSCHATZ, 2011)

Hashing Extensível

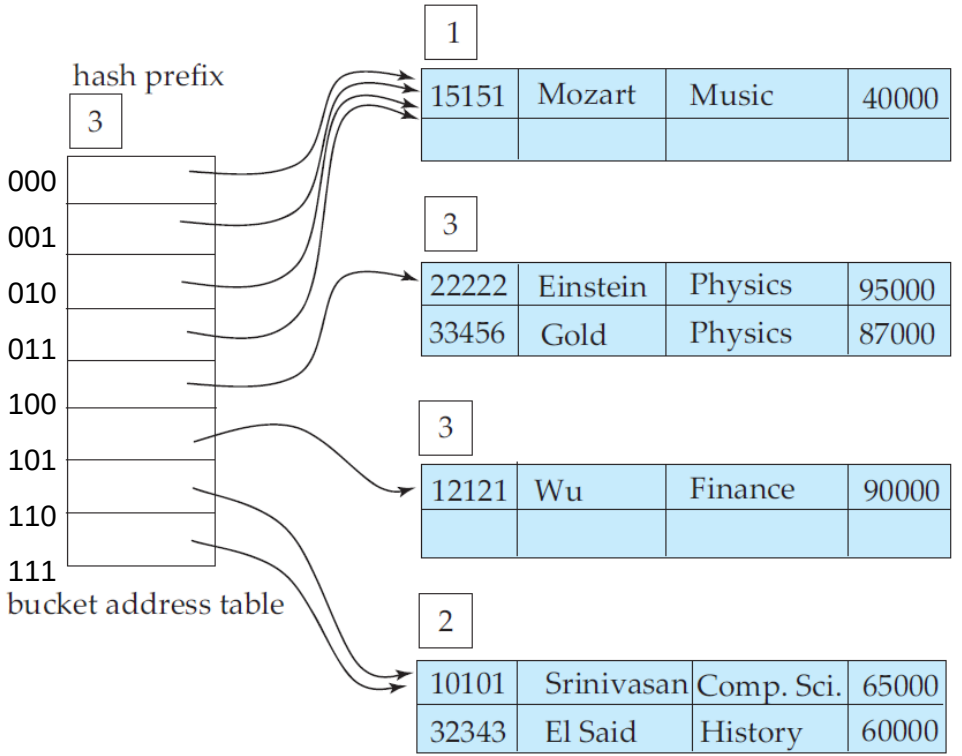
- Por que “extensível”?
  - Além de i_b poder aumentar até i, ou diminuir até 1, i também pode crescer
  - O valor i pode subir uma unidade por vez → dobra o tamanho do diretório
    - Quando há overflow de um bucket b com $i_b = i$
  - O valor i pode decrescer uma unidade por vez → corta pela metade o tamanho do diretório
    - Quando todos os bucket possuem $i_b < i$



(SILBERSCHATZ, 2011)

Hashing Extensível - Busca

- Busca(D, k):

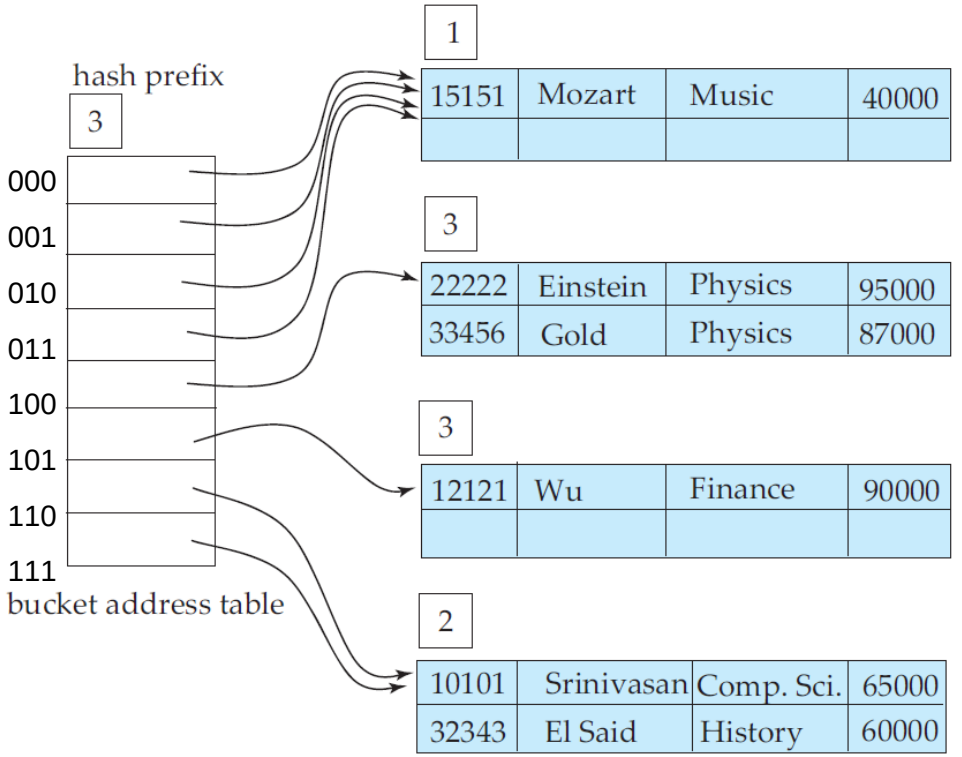


dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Figure 11.27 Hash function for *dept_name*.

Hashing Extensível - Busca

- Busca(D, k):
  - Calcula h(k) e pega os i bits mais significativos
  - Acessa o diretório nessa posição e acessa o bucket aí indicado
- Complexidade:

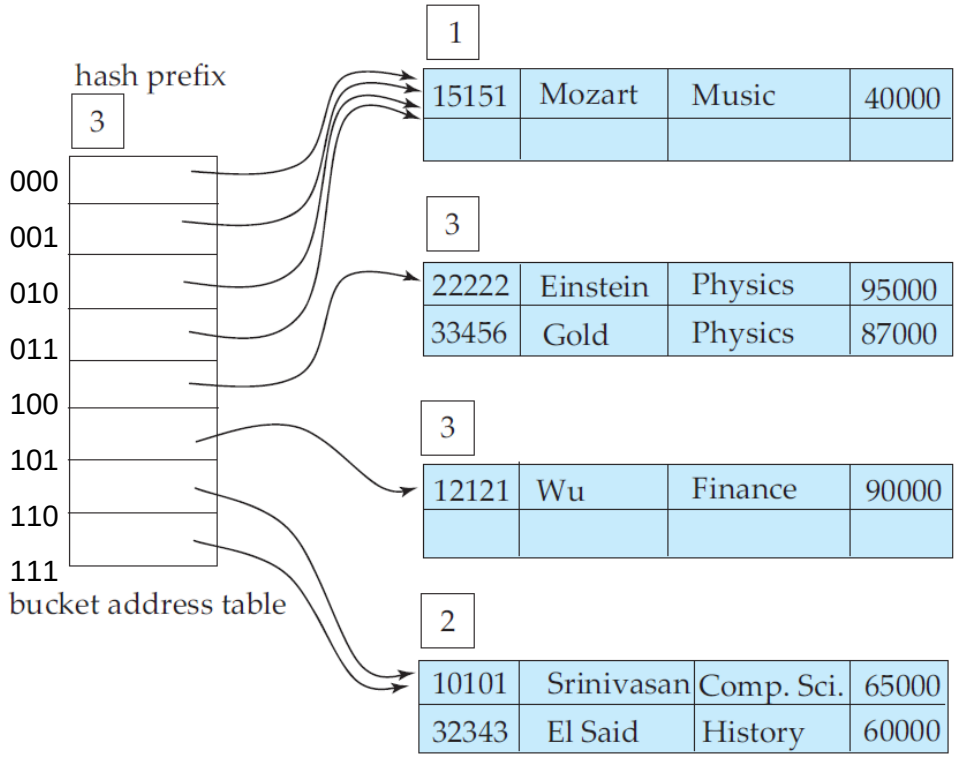


dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Figure 11.27 Hash function for *dept_name*.

Hashing Extensível - Busca

- Busca(D, k):
  - Calcula h(k) e pega os i bits mais significativos
  - Acessa o diretório nessa posição e acessa o bucket aí indicado
- Complexidade: 1 acesso ao disco (o diretório normalmente fica na memória principal)
  - Não aumenta com o aumento do tamanho do arquivo

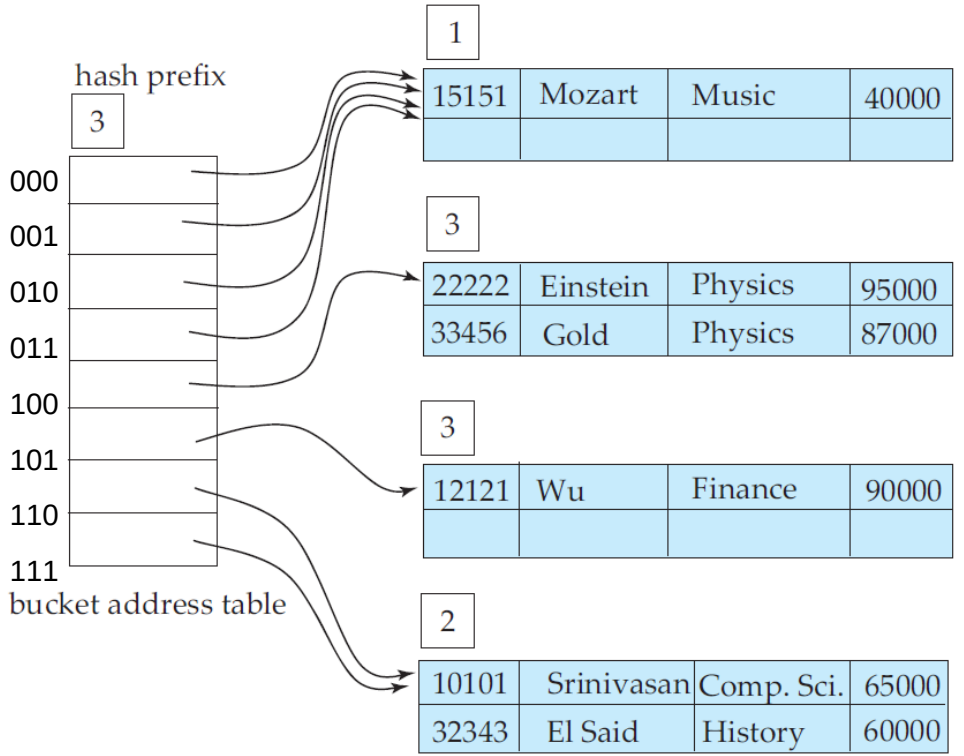


dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Figure 11.27 Hash function for *dept_name*.

Hashing Extensível - Inserção

Insere(D, k):



dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Figure 11.27 Hash function for *dept_name*.

senão cria bucket de overflow e insere nele

Hashing Extensível - Inserção

Insere(D, k):

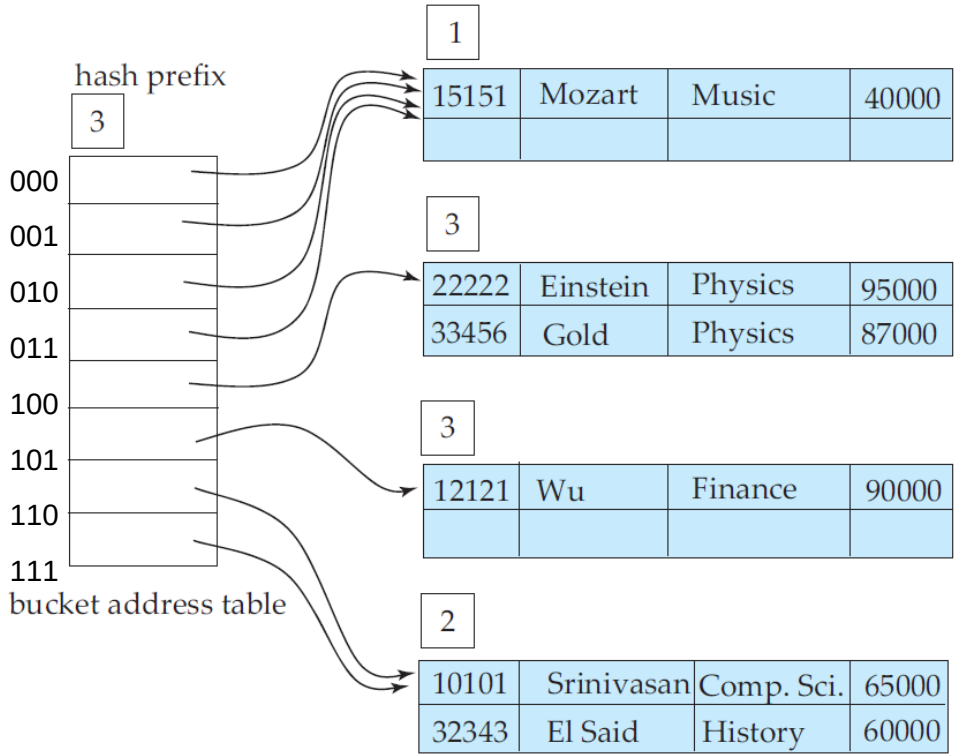
Calcula h(k) e pega os i bits mais significativos

Acessa o diretório D nessa posição e acessa o bucket aí indicado (b)

se há espaço disponível no bucket b, insere /* ex: 101 */

senão

Seeks ?



dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Figure 11.27 Hash function for *dept_name*.

senão cria bucket de overflow e insere nele

Hashing Extensível - Inserção

Insere(D, k):

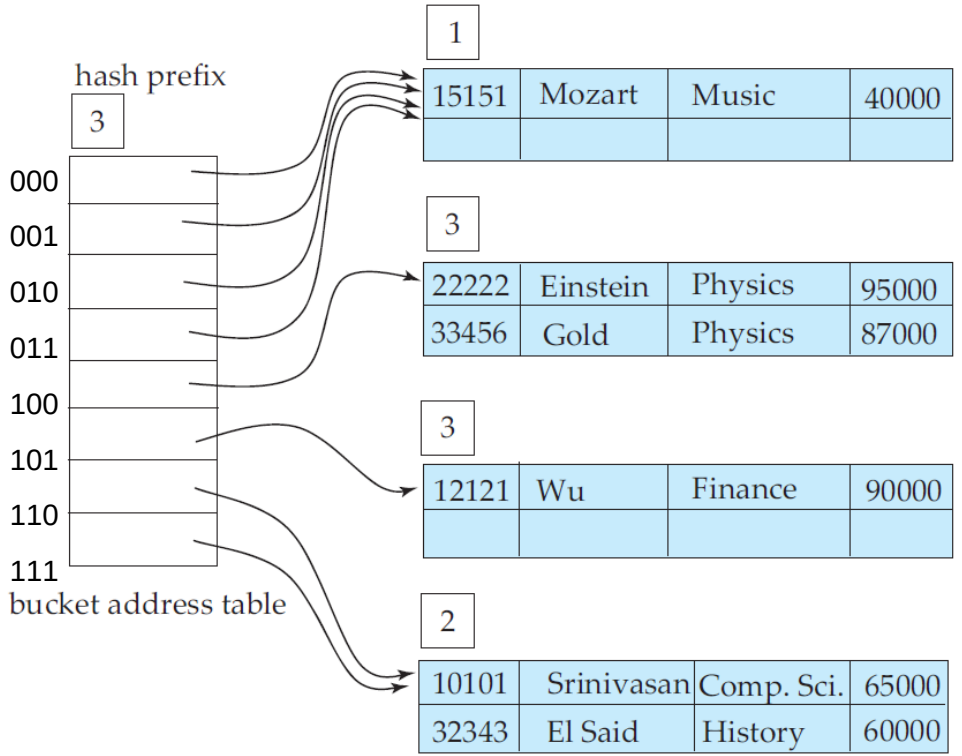
Calcula h(k) e pega os i bits mais significativos

Acessa o diretório D nessa posição e acessa o bucket aí indicado (b)

se há espaço disponível no bucket b, insere /* ex: 101 */

1 seek

senão



dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Figure 11.27 Hash function for *dept_name*.

Hashing Extensível - Inserção

Insere(D, k):

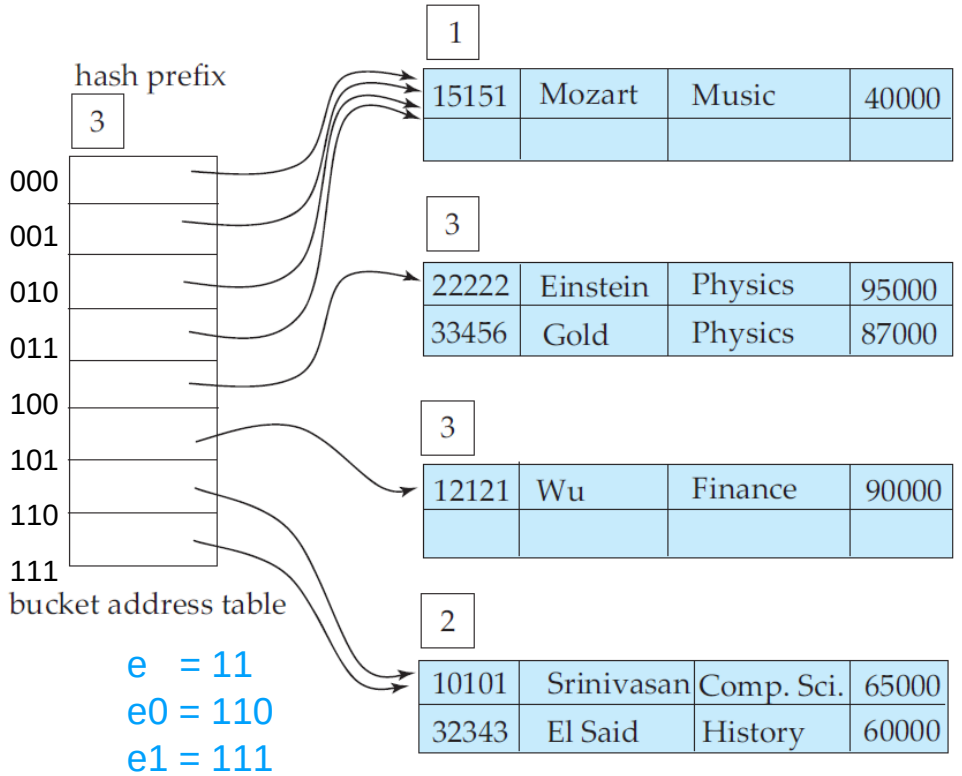
Calcula h(k) e pega os i bits mais significativos

Acessa o diretório D nessa posição e acessa o bucket aí indicado (b)

se há espaço disponível no bucket b, insere */* ex: 101 */*

senão

se i_h < i */* ex: 110 */*



dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Figure 11.27 Hash function for *dept_name*.

senão cria bucket de overflow e insere nele

Hashing Extensível - Inserção

Insere(D, k):

Calcula h(k) e pega os i bits mais significativos

Acessa o diretório D nessa posição e acessa o bucket aí indicado (b)

se há espaço disponível no bucket b, insere */* ex: 101 */*

senão

se i_b < i */* ex: 110 */*

e ← i_b bits mais significativos de h(k)

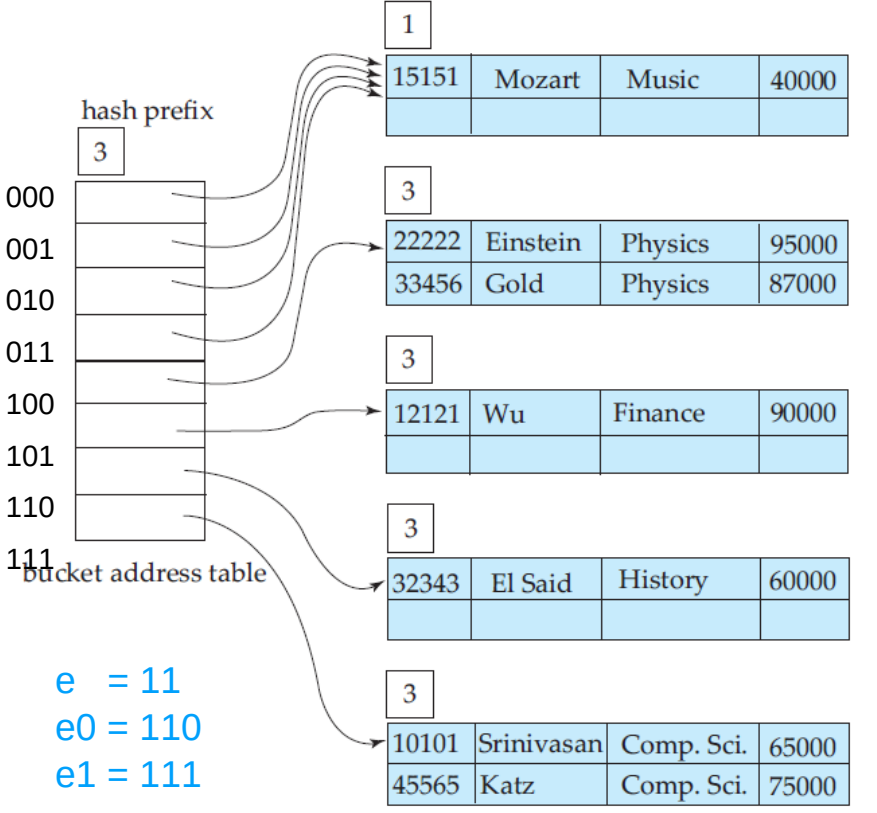
D[e0] ← novo bucket adicional b'

para cada chave k' do bucket apontado por D[e1]

move para b' se (i_b+1)-ésimo bit de h(k') = 0

insere(D,k)

2 seeks
1 novo bloco



e = 11
e0 = 110
e1 = 111

dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Figure 11.27 Hash function for *dept_name*.

senão cria bucket de overflow e insere nele

Hashing Extensível - Inserção

Insere(D, k):

Calcula h(k) e pega os i bits mais significativos

Acessa o diretório D nessa posição e acessa o bucket aí indicado (b)

se há espaço disponível no bucket b, insere /* *ex: 101* */

senão

se i_b < i /* ex: 110 */

e ← i_b bits mais significativos de h(k)

D[e0] ← novo bucket adicional b'

para cada chave k' do bucket apontado por D[e1]

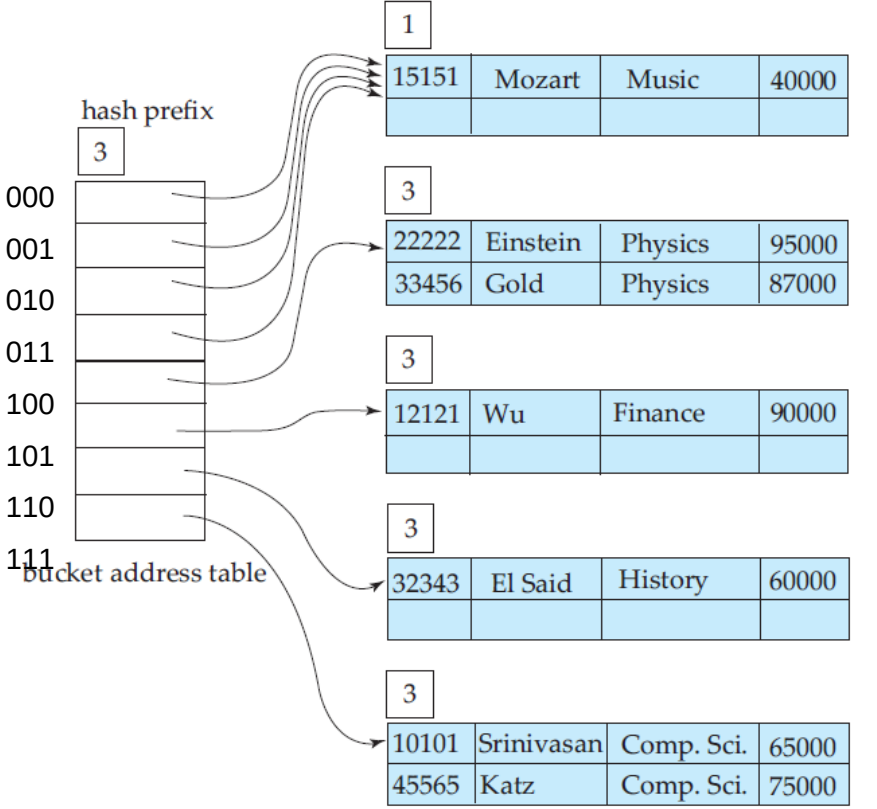
move para b' se (i_b+1)-ésimo bit de h(k') = 0

insere(D,k)

senão /* i_b = i, ex: 111 */

se b não tiver todas as chaves com os mesmos i+1 primeiro bits

dobra(D) e Insere(D, k)



dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Figure 11.27 Hash function for *dept_name*.

Hashing Extensível - Inserção

Insere(D, k):

Calcula h(k) e pega os i bits mais significativos

Acessa o diretório D nessa posição e acessa o bucket aí indicado (b)

se há espaço disponível no bucket b, insere

senão

se $i_b < i$

e ← i_b bits mais significativos de h(k)

D[e0] ← novo bucket adicional b'

para cada chave k' do bucket apontado por D[e1]

move para b' se (i_b+1)-ésimo bit de h(k') = 0

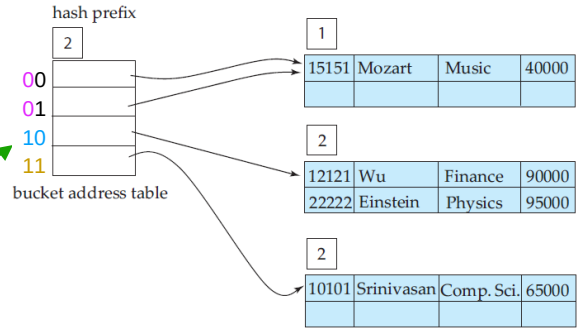
insere(D,k)

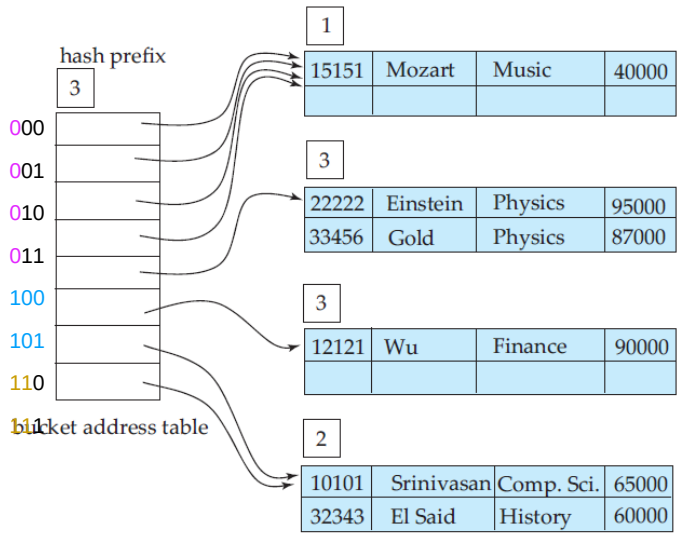
senão /* $i_b = i$, ex: 10 */

se b não tiver todas as chaves com os mesmos i+1 primeiro bits

dobra(D) e Insere(D, k)

? seeks
1 novo bloco





dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Hashing Extensível - Inserção

Insere(D, k):

Calcula h(k) e pega os i bits mais significativos

Acessa o diretório D nessa posição e acessa o bucket aí indicado (b)

se há espaço disponível no bucket b, insere

senão

se $i_b < i$

e ← i_b bits mais significativos de h(k)

D[e0] ← novo bucket adicional b'

para cada chave k' do bucket apontado por D[e1]

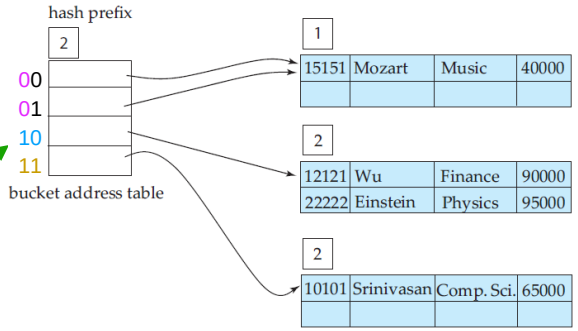
move para b' se (i_b+1)-ésimo bit de h(k') = 0

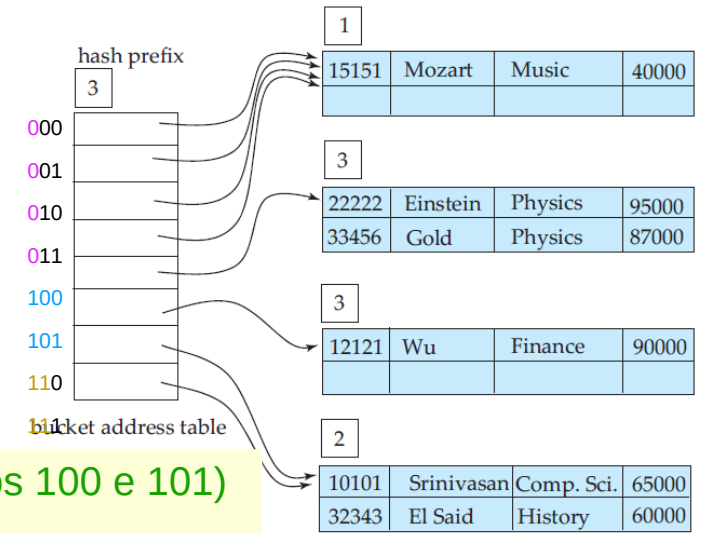
insere(D,k)

senão /* $i_b = i$, ex: 10 */

se b não tiver todas as chaves com os mesmos i+1 primeiro bits

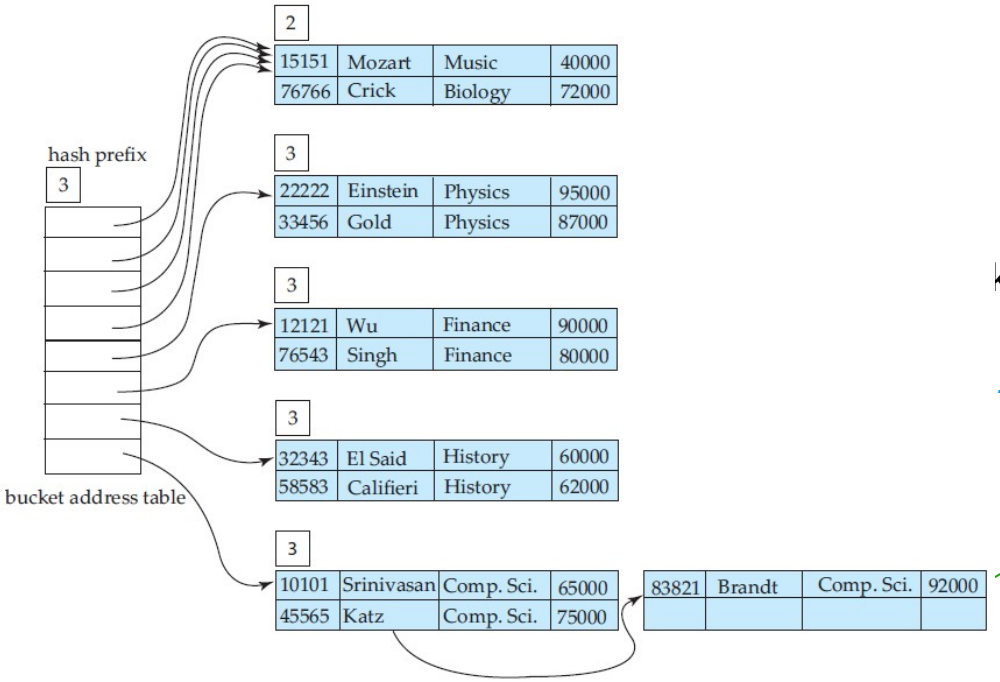
dobra(D) e Insere(D, k)





2 seeks (blocos 100 e 101)
1 novo bloco

dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001



ket aí

*101 */*

D[e1]

move para b' se (i_b+1)-ésimo bit de h(k') = 0

insere(D,k)

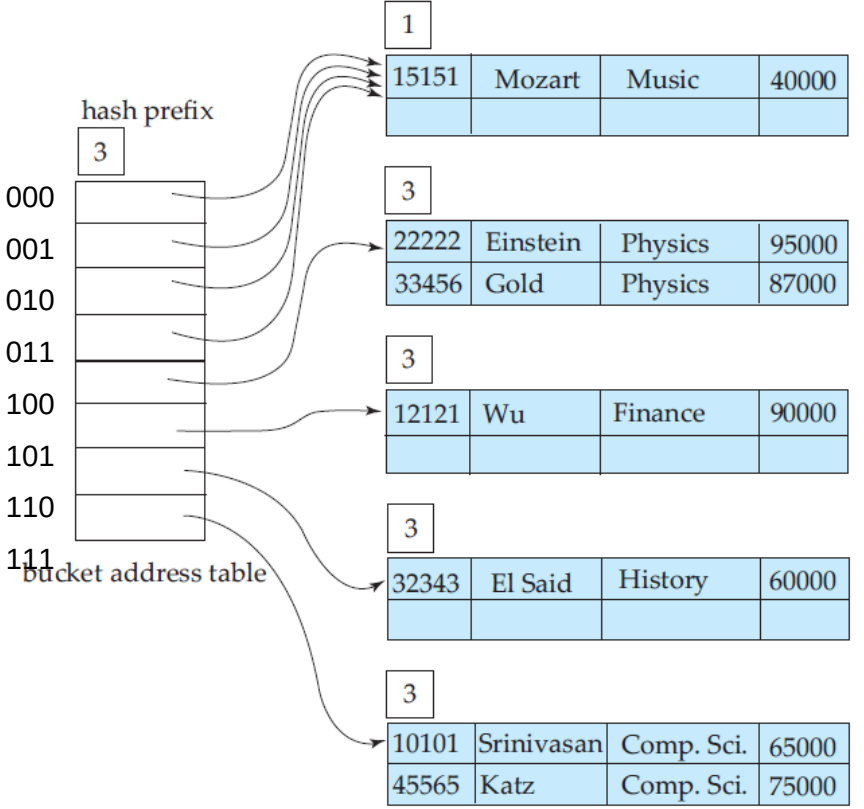
senão /* i_b = i, ex: 111 */

se b não tiver todas as chaves com os mesmos i+1 primeiro bits

dobra(D) e insere em b

senão cria bucket de overflow e insere nele

2 seeks
1 novo bloco

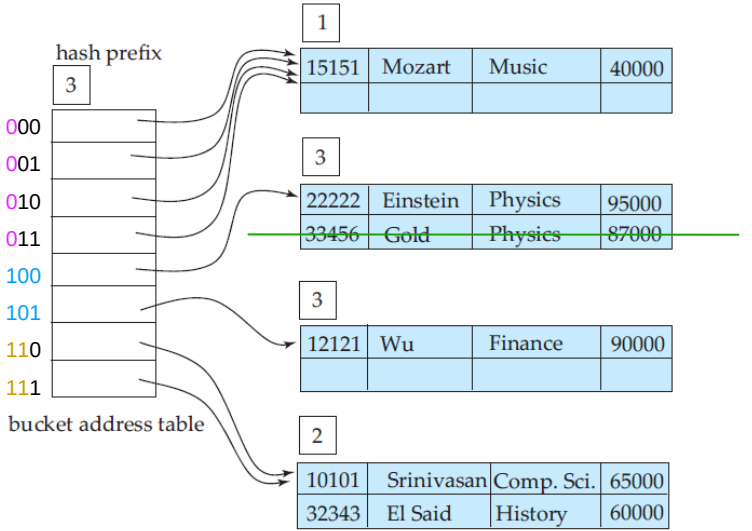


dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Figure 11.27 Hash function for *dept_name*.

ADAPTAR ROTINA DE BUSCA!

Hashing Extensível - Remoção



Remove(D, k):

Calcula h(k) e pega os i bits mais significativos /* ex: 100 */

Acessa o diretório D nessa posição e acessa o bucket aí indicado (b)

se está no bucket b principal ou de overflow

remove, traz uma chave do bucket de overflow (se houver)

e ← (i_b -1) bits mais significativos de h(k)

Se a soma do nr de registros nos blocos apontados por D[e0] e D[e1] couber em um único bloco

se |D[e0]| + |D[e1]| ≤ r

acrescenta as chaves do bloco apontado por D[e0] no bloco apontado por D[e1]

libera bloco apontado por D[e0]

D[e0] ← D[e1]

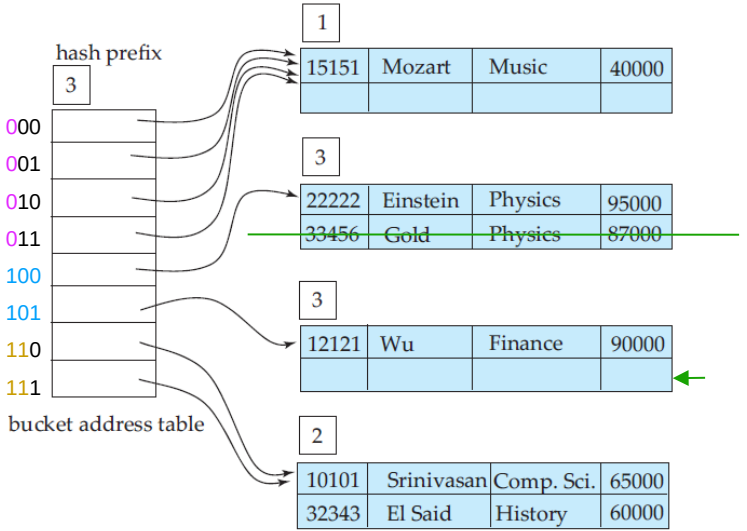
decrementa i' do bloco apontado por D[e1]

se max(i') < i

Divide D pela metade

dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Hashing Extensível - Remoção



Remove(D, k):

Calcula h(k) e pega os i bits mais significativos /* ex: 100 */

Acessa o diretório D nessa posição e acessa o bucket aí indicado (b)

se está no bucket b principal ou de overflow

remove, traz uma chave do bucket de overflow (se houver)

e ← (i_b -1) bits mais significativos de h(k)

Se a soma do nr de registros nos blocos apontados por D[e0] e D[e1] couber em um único bloco

se |D[e0]| + |D[e1]| ≤ r

acrescenta as chaves do bloco apontado por D[e0] no bloco apontado por D[e1]

libera bloco apontado por D[e0]

D[e0] ← D[e1]

decrementa i' do bloco apontado por D[e1]

se max(i') < i

Divide D pela metade

dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Hashing Extensível - Remoção

Remove(D, k):

Calcula h(k) e pega os i bits mais significativos /* ex: 100 */

Acessa o diretório D nessa posição e acessa o bucket aí indicado (b)

se está no bucket b principal ou de overflow

remove, traz uma chave do bucket de overflow (se houver)

e ← (i_b -1) bits mais significativos de h(k)

Se a soma do nr de registros nos blocos apontados por D[e0] e D[e1] couber em um único bloco

se |D[e0]| + |D[e1]| ≤ r

acrescenta as chaves do bloco apontado por D[e0] no bloco apontado por D[e1]

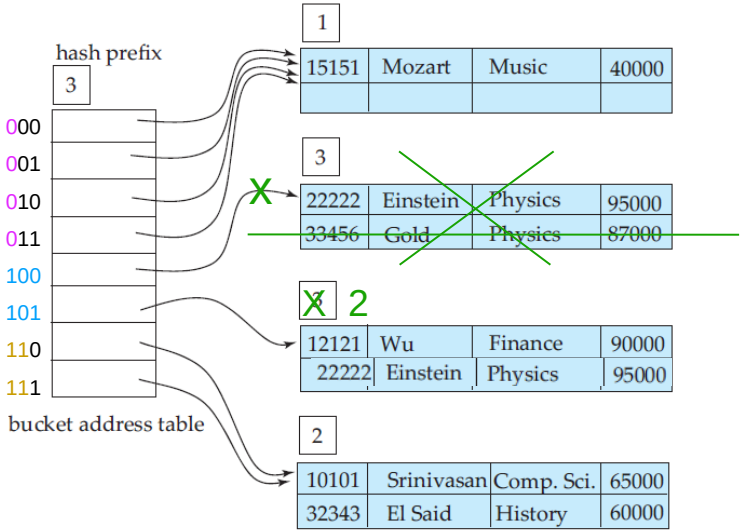
libera bloco apontado por D[e0]

D[e0] ← D[e1]

decrementa i' do bloco apontado por D[e1]

se max(i') < i

Divide D pela metade



dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Hashing Extensível - Remoção

Remove(D, k):

Calcula h(k) e pega os i bits mais significativos /* ex: 100 */

Acessa o diretório D nessa posição e acessa o bucket aí indicado (b)

se está no bucket b principal ou de overflow

remove, traz uma chave do bucket de overflow (se houver)

e ← (i_b -1) bits mais significativos de h(k)

se |D[e0]| + |D[e1]| ≤ r

acrescenta as chaves do bloco apontado por D[e0] no bloco apontado por D[e1]

libera bloco apontado por D[e0]

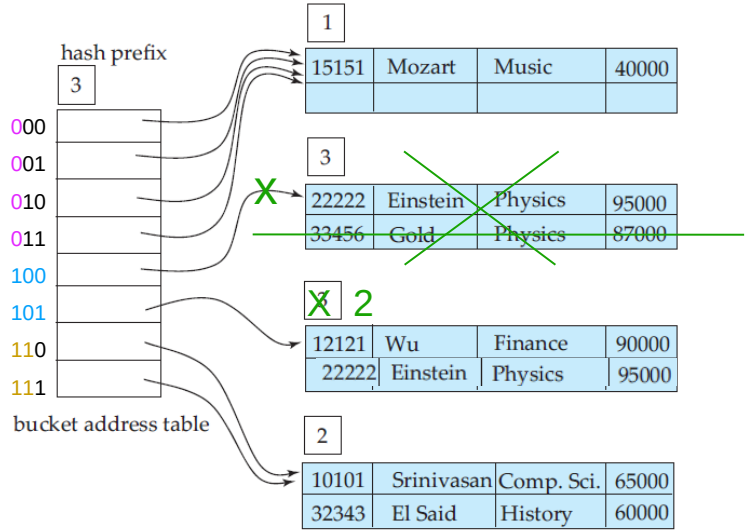
D[e0] ← D[e1]

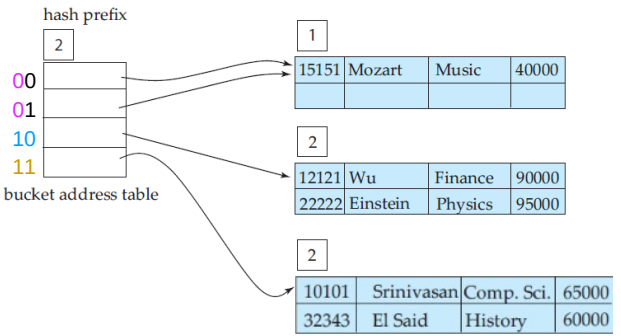
decrementa i' do bloco apontado por D[e1]

se max(i') < i

Divide D pela metade

Se a soma do nr de registros nos blocos apontados por D[e0] e D[e1] couber em um único bloco





dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Até 2 seeks
Até 1 bloco a menos

Hashing Extensível

- Tempo de busca: geralmente 1 acesso ao disco (1 no diretório que normalmente fica na memória principal e outro no bucket)
  - Não aumenta com o aumento do tamanho do arquivo (a não ser se houver buckets de overflow - raro)
- Espaço:
  - Diretório ocupa pouco espaço - no máximo 2^k, sendo k o nr de bits do valor de hash (endereço-base)
  - Dobrar o diretório só aumenta um bucket (além do diretório)
- Tempos para dobrar/cortar pela metade o diretório
  - Dobrar: além de acertar as entradas do diretório, apenas o bucket transbordado precisa ser dividido
  - Cortar pela metade: recriação do diretório (não precisa ajustar blocos)
  - Diretório fica em memória (cabeçalho do arquivo)

Exercícios

Implemente todas essas opções de Hashing Extensível (operações de busca, inserção e deleção)

Referências

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.