Algoritmos e Estruturas de Dados II
Aula 27 – Hashing Dinâmico

Prof. Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2024

Hashing - Tratamento de Colisões

- Hashing estático (tamanho da tabela é constante)
  - Encadeamento ou endereçamento fechado – colisões vão para uma lista ligada
  - Endereçamento aberto (chaves dentro da tabela, sem ponteiros)
- Hashing dinâmico (tabela pode expandir/encolher)
  - Hashing extensível (estrutura de dados adicional)
  - 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 bloco (um bloco ou cluster de blocos em disco)
  - Colisões vão preenchendo o bloco
  - Tabela de hash fica no cabeçalho do arquivo

Tipos de organização de arquivos

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

Colisões

- Se h(x) = h(y) = i: x e y vão para o bloco i
- E se o bloco i estiver lotado?

Colisões

- Se h(x) = h(y) = i: x e y vão para o bloco i
- E se o bloco i estiver lotado?

1. Encadeamento - blocos de overflow
   - Opção 1: compartilhados
   - Opção 2: exclusivos por endereço-base

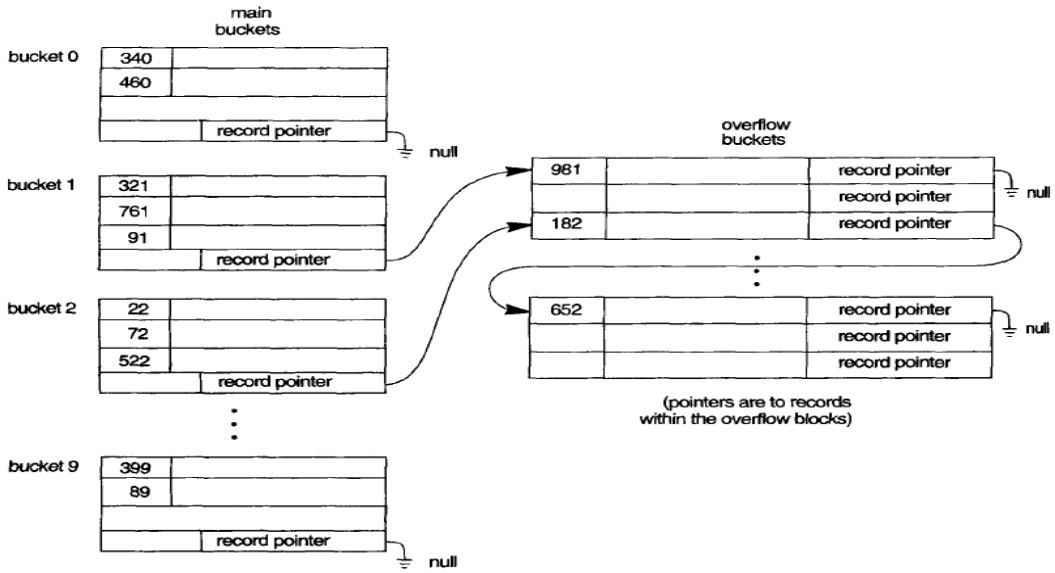
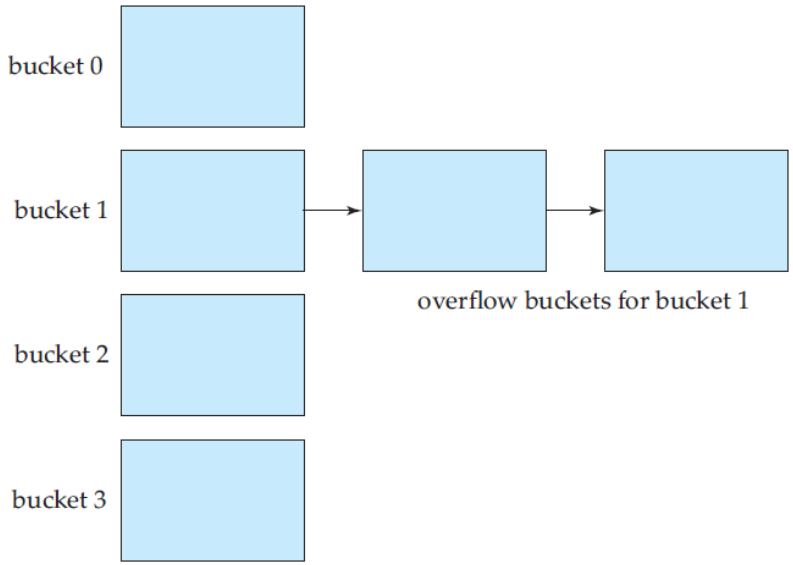


FIGURE 13.10 Handling overflow for buckets by chaining.



Colisões

- Se h(x) = h(y) = i: x e y vão para o bloco i
- E se o bloco i estiver lotado?

1. Encadeamento - blocos de overflow
   - Opção 1: compartilhados
   - Opção 2: exclusivos por endereço-base
2. Endereçamento aberto – vai para outro bloco
   - Com sondagem (linear ...)

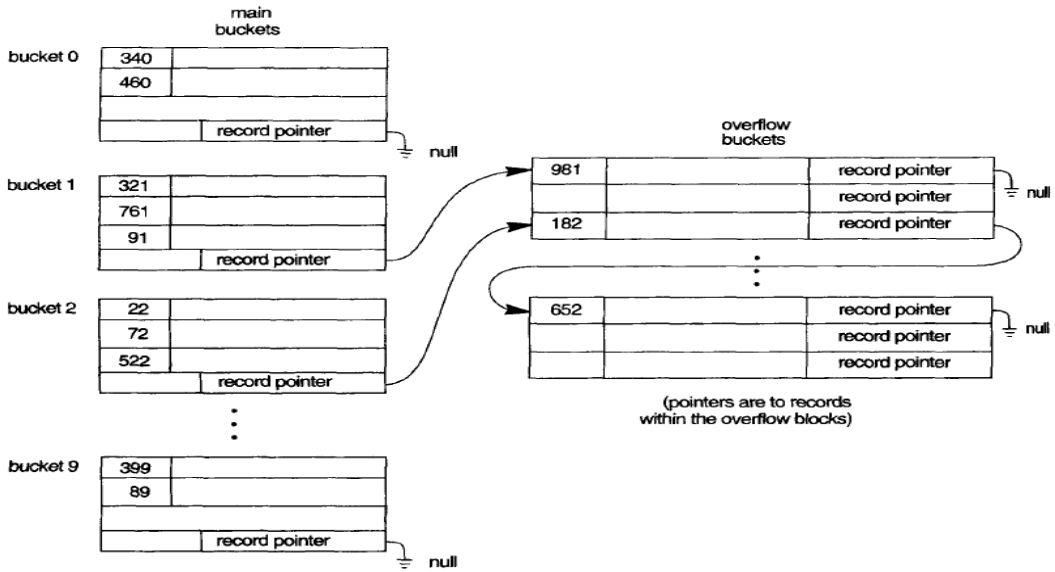
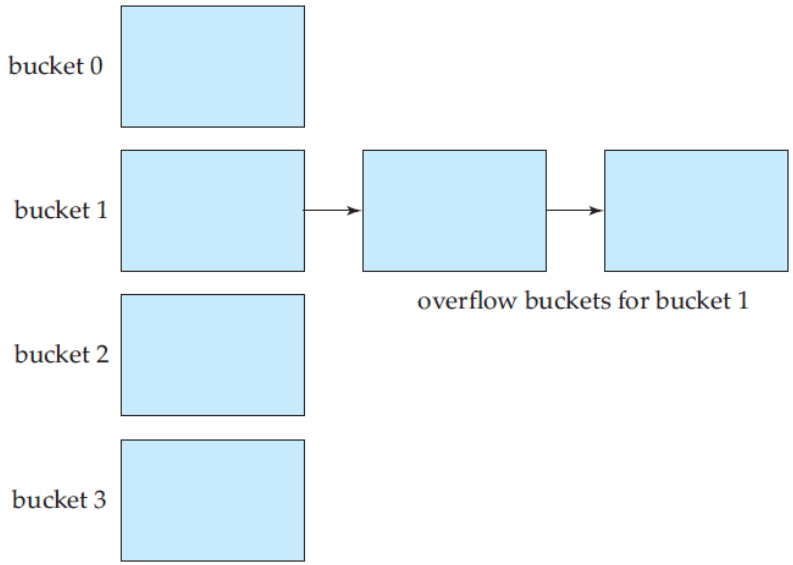


FIGURE 13.10 Handling overflow for buckets by chaining.



Overflows

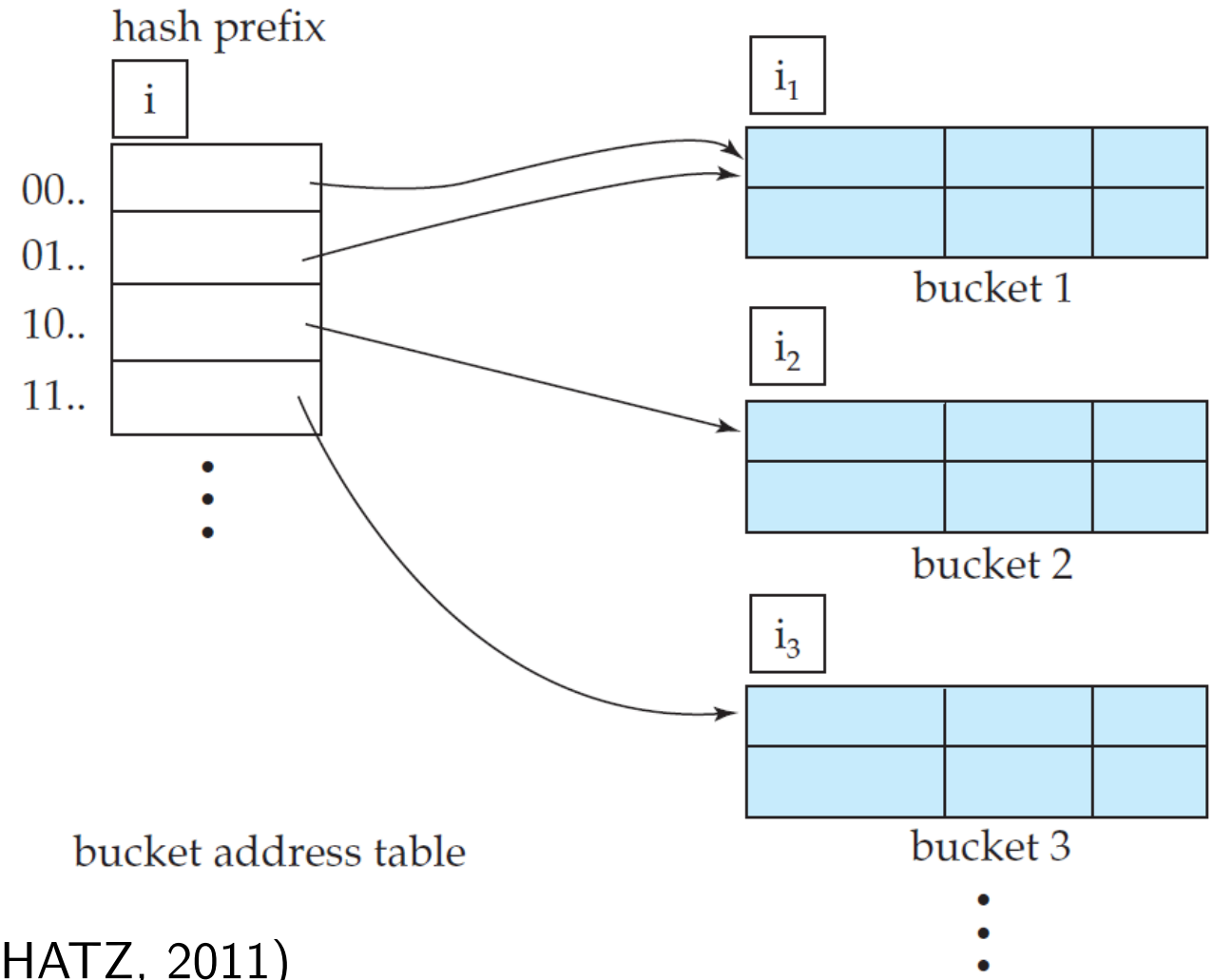
- Overflows aumentam o tempo de busca
- O ideal é ter um M (número de slots) que não acarrete em overflow, sem muita perda de espaço
- **Hashing estático:** o M é fixo!
  - Mas o que fazer quando o arquivo aumenta ou diminui de tamanho
  - Se manter o mesmo hash (M, função, etc) teremos overflows ou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

Hashing Dinâmico - Extensível

Diretório: arranjo de 2^i endereços de blocos

- 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 bloco que contém tais registros
- A vantagem é que diferentes entradas podem apontar para o mesmo bloco ou não
  - Registros com os mesmos i' primeiros bits, i' $<$ i poderiam caber em um mesmo bloco
  - O valor i' depende de cada bloco b (i_b), e deve ser armazenado com ele: i_b – profundidade local

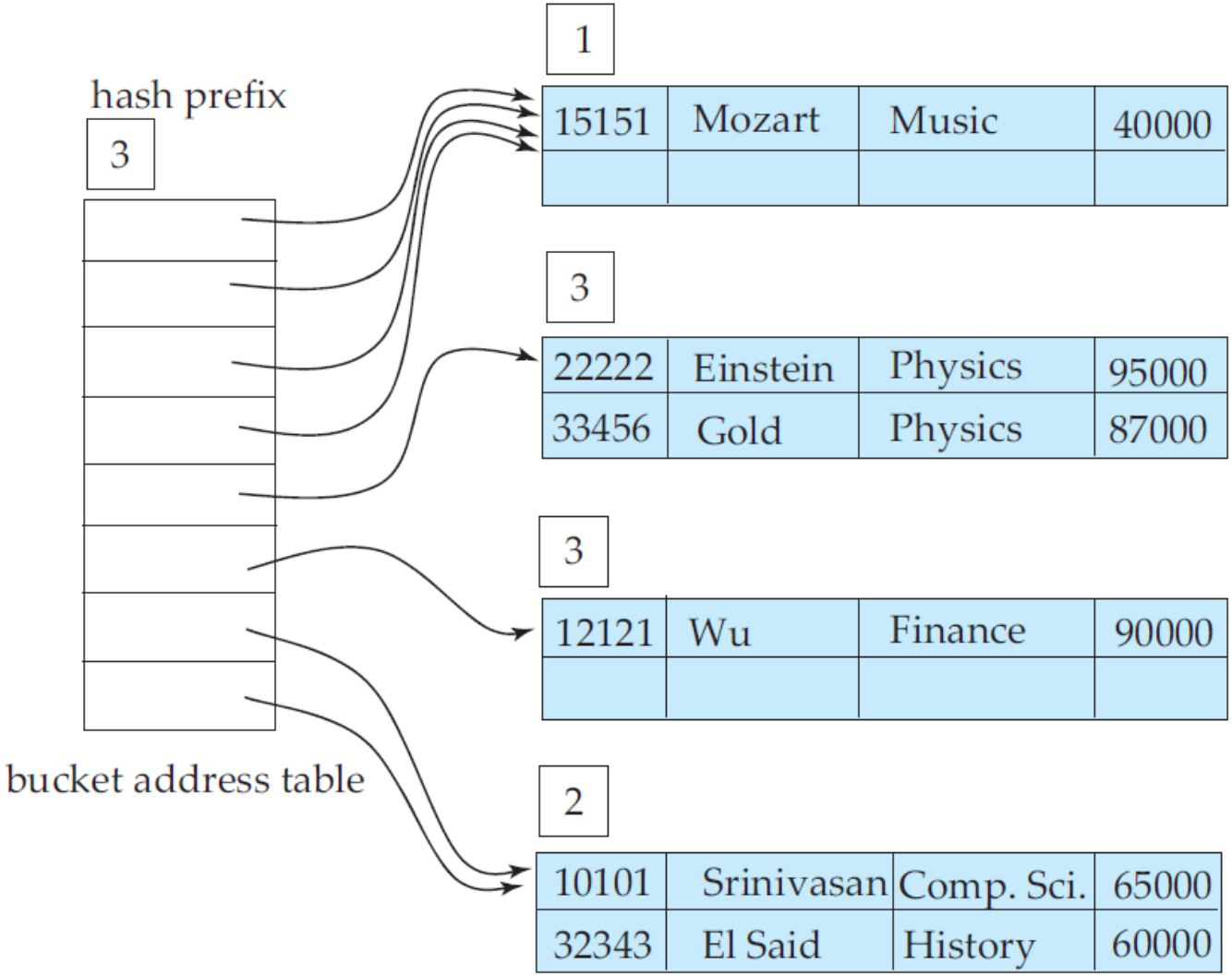
Hashing Dinâmico - Extensível



(SILBERSCHATZ, 2011)

Figure 11.26 General extendable hash structure.

Hashing Dinâmico - Extensível



(SILBERSCHATZ, 2011)

Hashing Dinâmico - Extensível

Por que **extensível**?

- Além de i_b poder aumentar até i, ou diminuir até 1, i também pode crescer

Hashing Dinâmico - 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?

Hashing Dinâmico - 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? Quando há overflow de um bloco b com $i_b = i$

Hashing Dinâmico - 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? Quando há overflow de um bloco b com $i_b = i$
- O valor i pode decrescer uma unidade por vez: corta pela metade o tamanho do diretório
  - Quando?

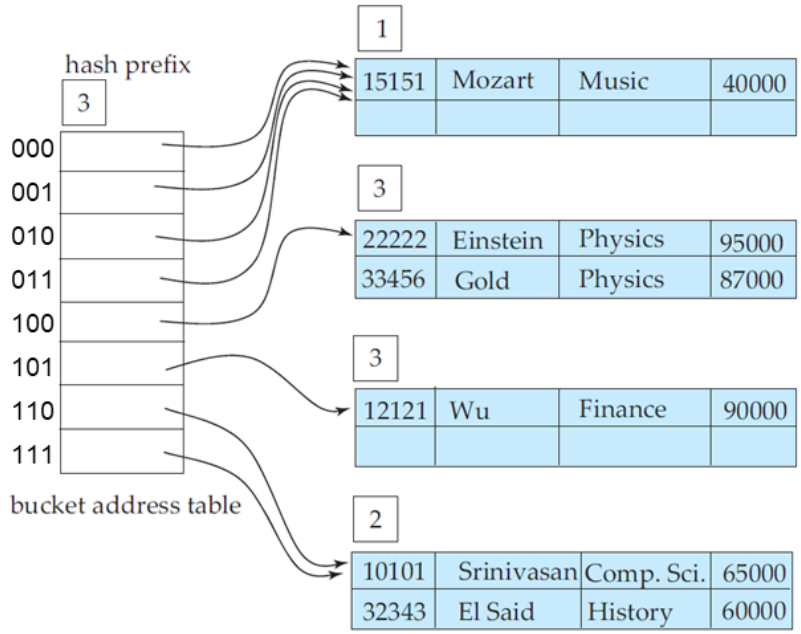
Hashing Dinâmico - Extensível

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? Quando há overflow de um bloco b com $i_b = i$
- O valor i pode decrescer uma unidade por vez: corta pela metade o tamanho do diretório
  - Quando? Quando todos os bloco possuem $i_b < i$

Hashing Dinâmico - Extensível

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

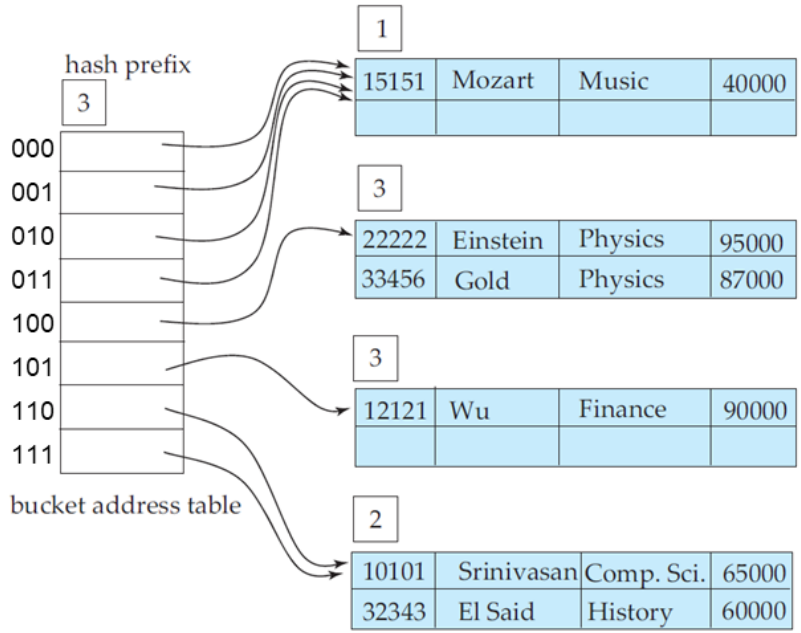


dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100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

Figure 11.27 Hash function for *dept_name*.

Hashing Dinâmico - Extensível

- Busca(D, k):
  - Calcula h(k) e pega os i bits mais significativos
  - Acessa o diretório nessa posição e acessa o bloco 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 Dinâmico - 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 *b*

se há espaço disponível no bucket b, insere

senão

se $i_b < i$

e ← i_b dígitos 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

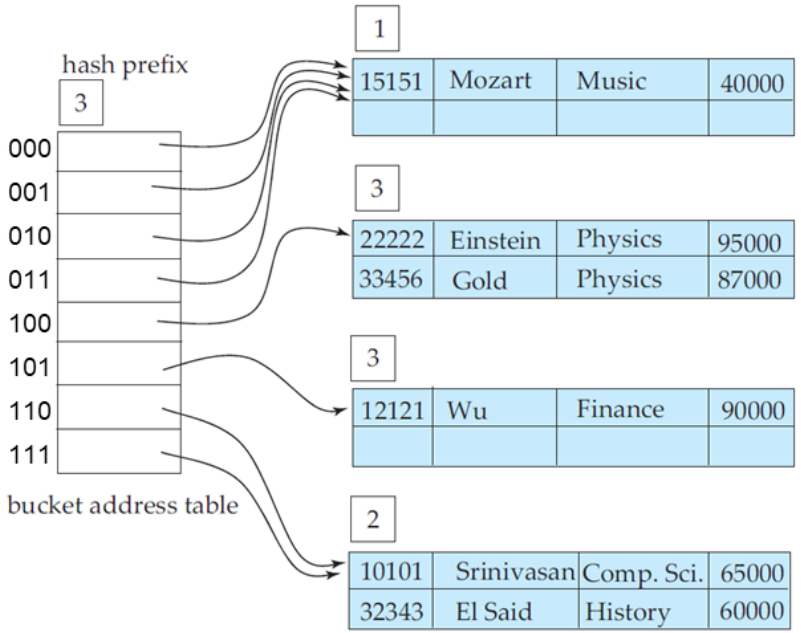
insere(D,k)

senão

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

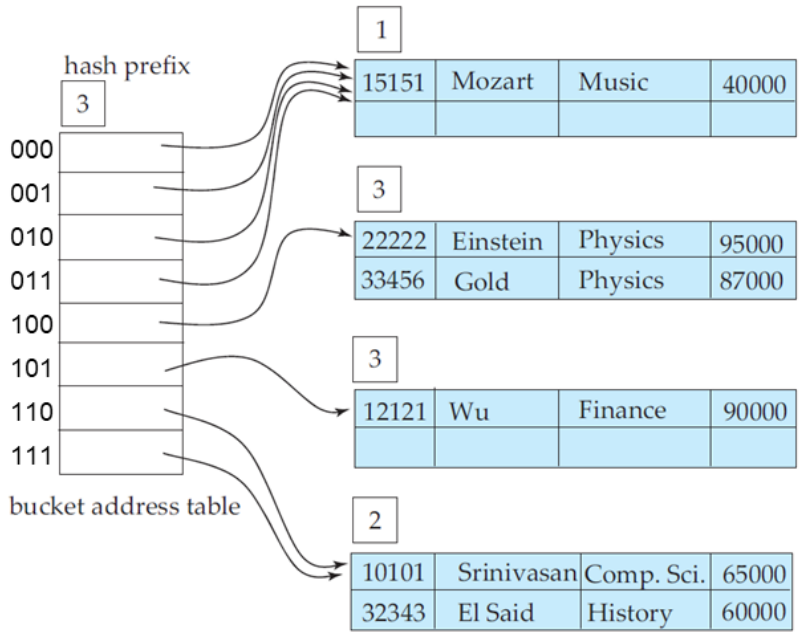


dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100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

Figure 11.27 Hash function for *dept_name*.

Hashing Dinâmico - 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 *b*
se há espaço disponível no bucket b, insere
senão
se $i_b < i$
e ← i_b dígitos 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
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



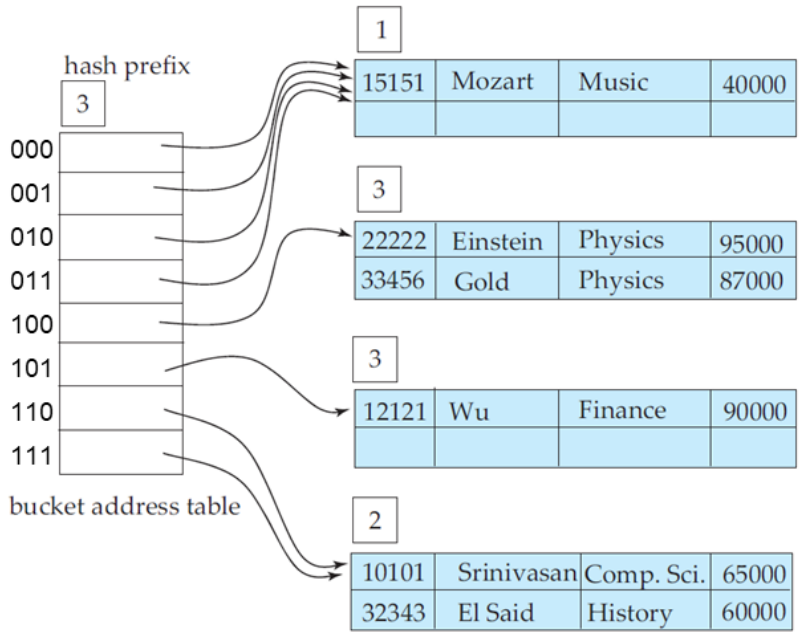
dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100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

Figure 11.27 Hash function for *dept_name*.

Inserir registro com hash 101

Hashing Dinâmico - 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 *b*
se há espaço disponível no bucket b, insere
senão
se $i_b < i$
e ← i_b dígitos 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
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



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*.

Inserir registro com hash 101: 1 seek

Hashing Dinâmico - 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 *b*

se há espaço disponível no bucket b, insere

senão

se $i_b < i$

e ← i_b dígitos 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

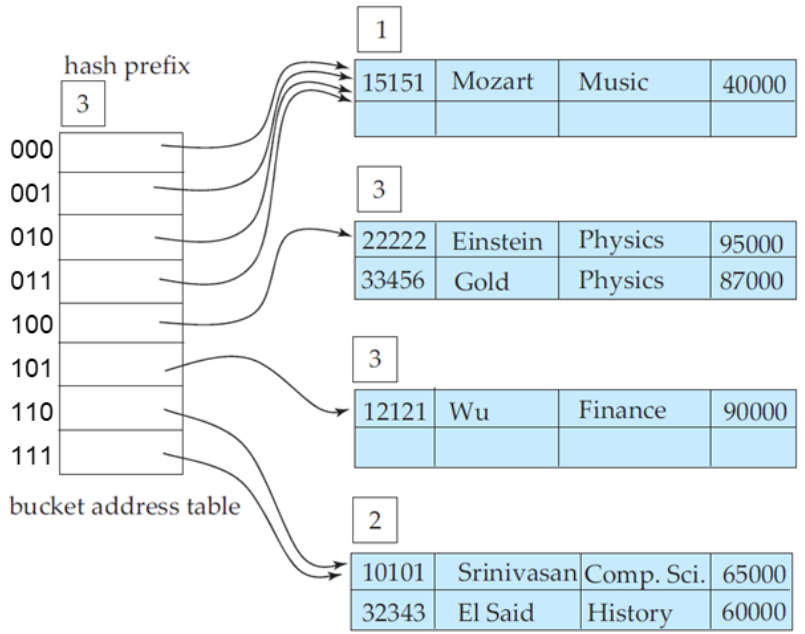
insere(D,k)

senão

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



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*.

Inserir registro com hash 111

Hashing Dinâmico - 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 *b*

se há espaço disponível no bucket b, insere

senão

se $i_b < i$

e ← i_b dígitos 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

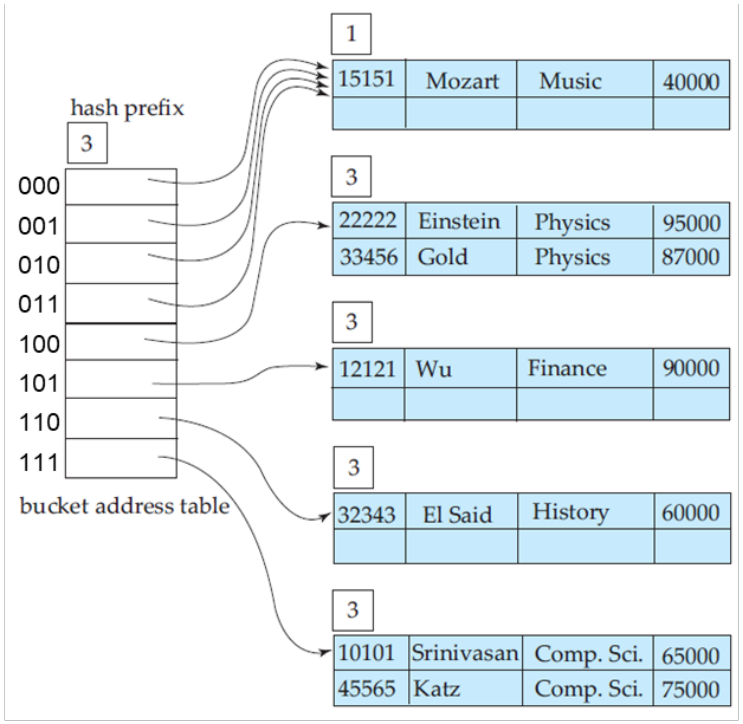
insere(D,k)

senão

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



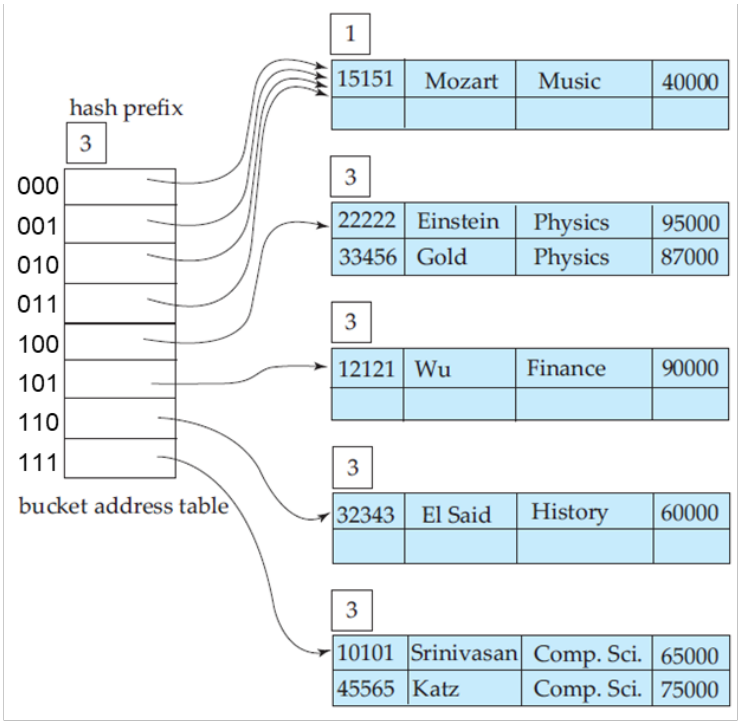
dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100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

Figure 11.27 Hash function for *dept_name*.

Inserir registro com hash 111: 2 seeks, 1 novo bloco

Hashing Dinâmico - 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 *b*
se há espaço disponível no bucket b, insere
senão
se $i_b < i$
e ← i_b dígitos 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
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



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*.

Inserir outro registro com hash 111

Hashing Dinâmico - 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 *b*

se há espaço disponível no bucket b, insere

senão

se $i_b < i$

e ← i_b dígitos 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

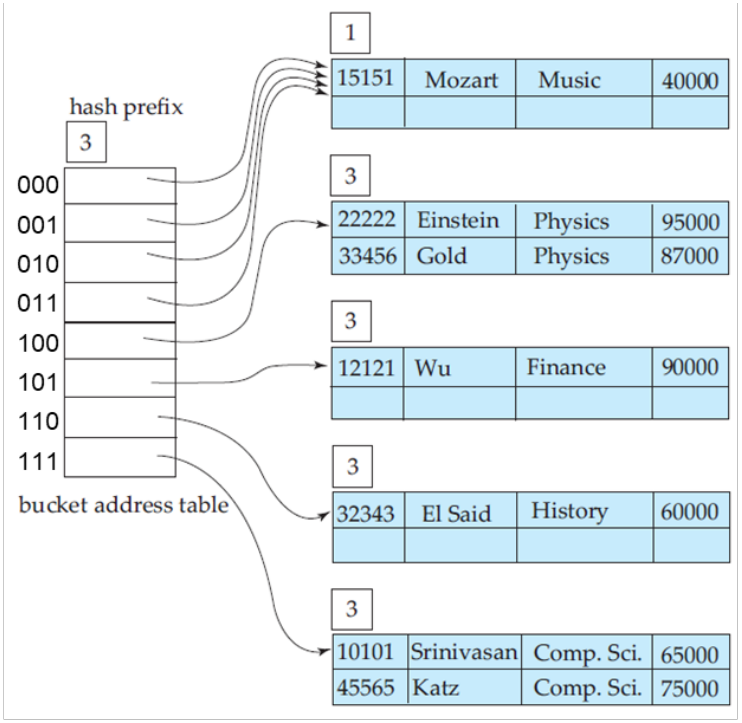
insere(D,k)

senão

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



←

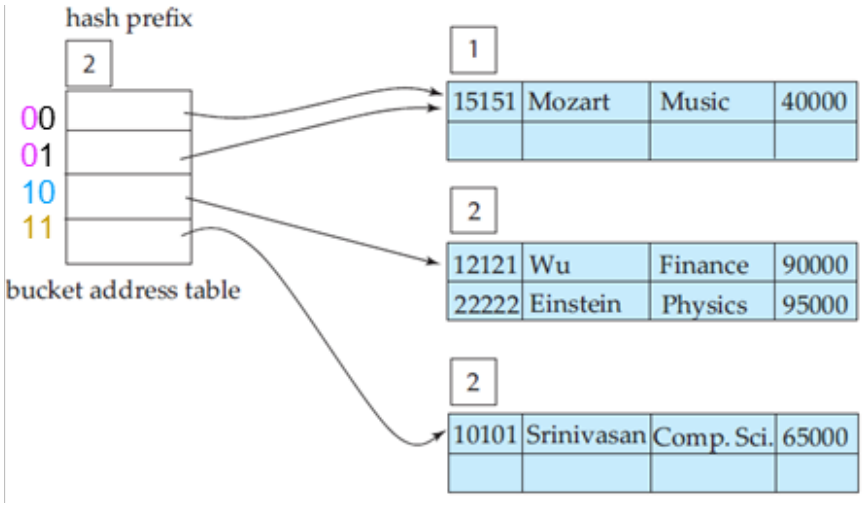
dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100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

Figure 11.27 Hash function for *dept_name*.

Inserir outro registro com hash 111: 2 seeks, 1 novo bloco (como?)

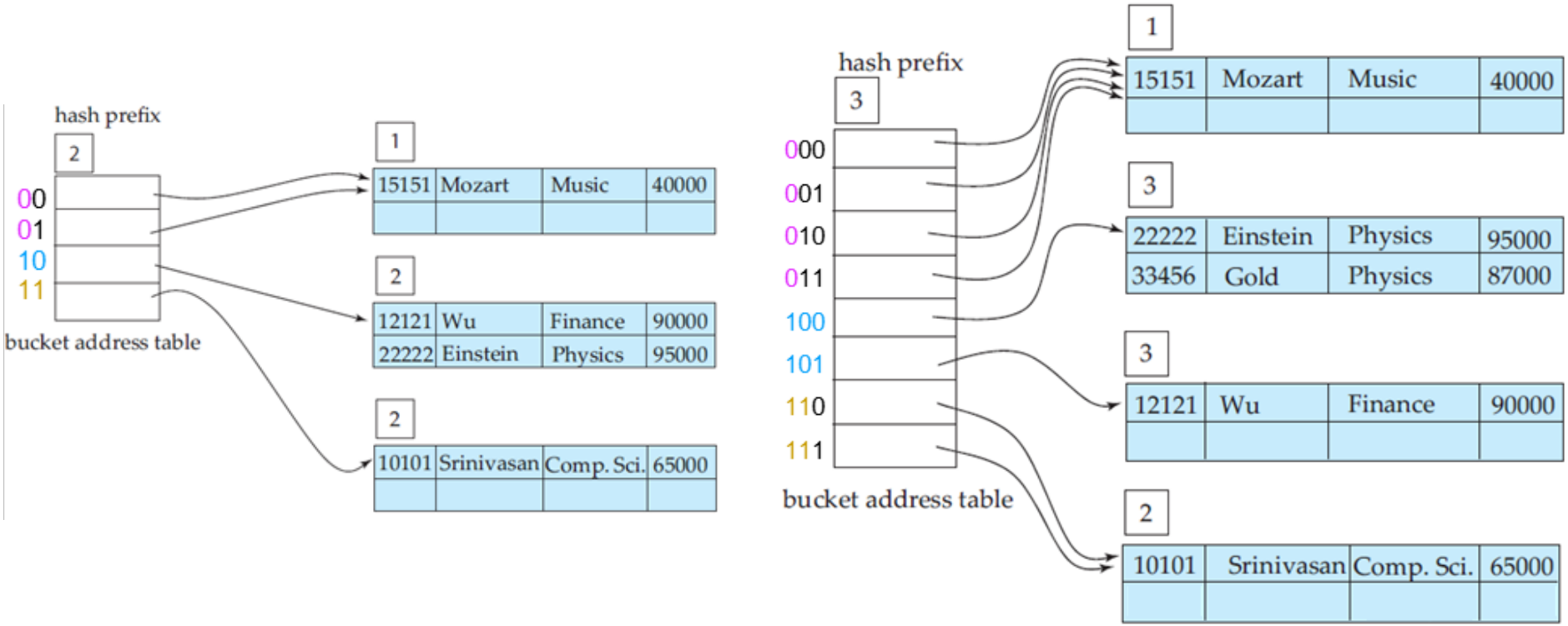
Hashing Dinâmico - Extensível - Inserção

Insere(D, k): Inserir registro com hash 10



Hashing Dinâmico - Extensível - Inserção

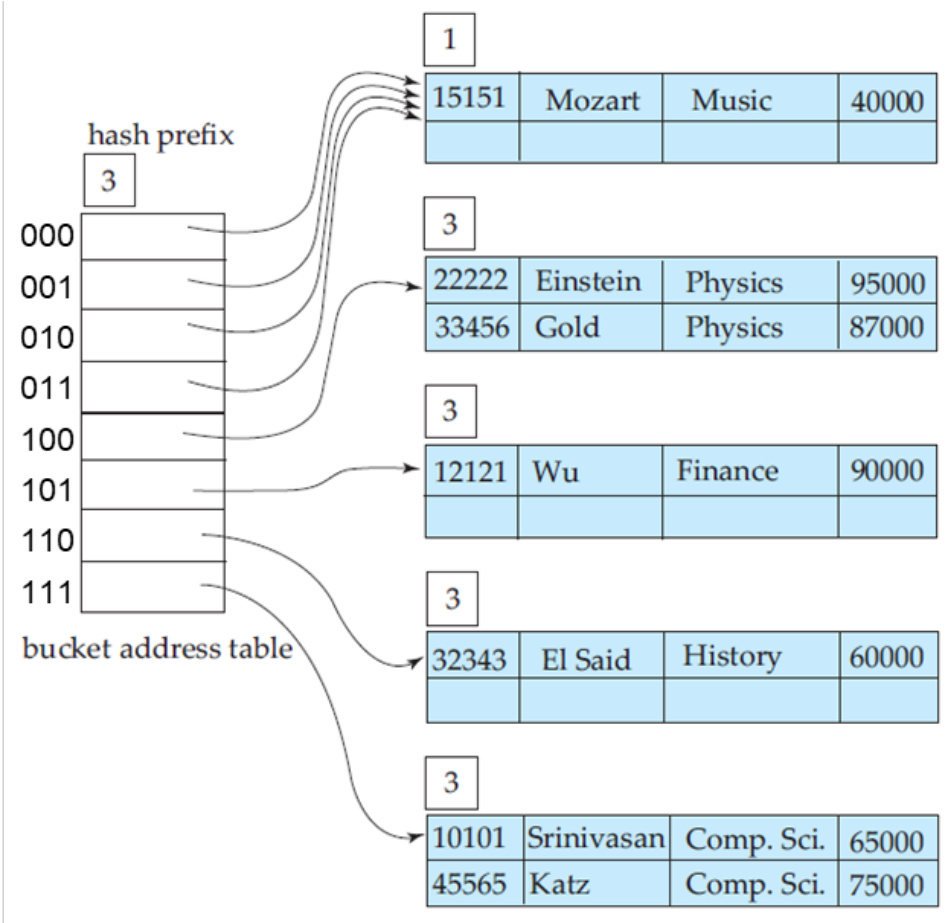
Insere(D, k): Inserir registro com hash 10



2 seeks, 1 novo bloco

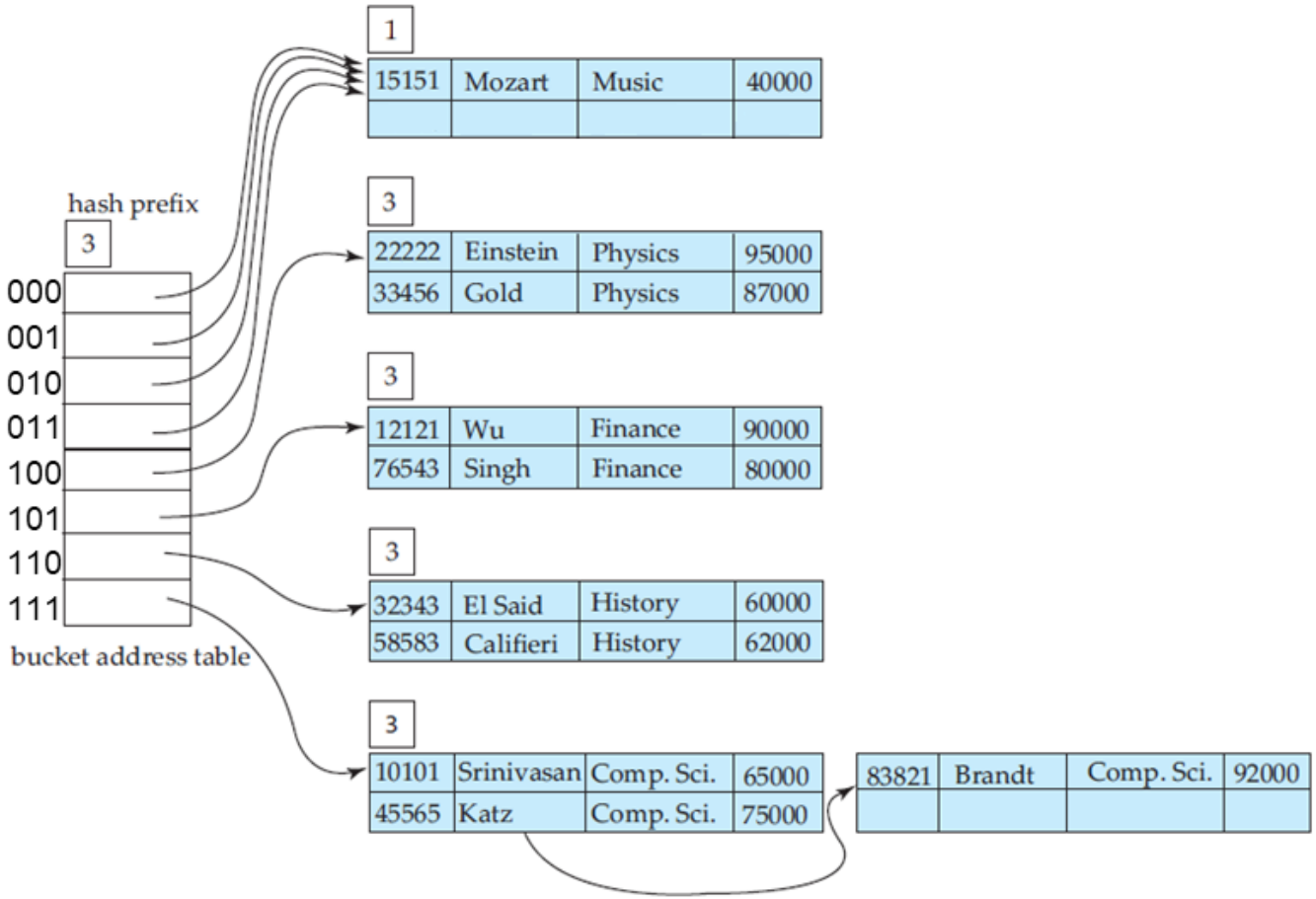
Hashing Dinâmico - Extensível - Inserção

Insere(D, k): Inserir registro com hash 111



Hashing Dinâmico - Extensível - Inserção

Insere(D, k): Inserir registro com hash 111



2 seeks, 1 novo bloco

Hashing Dinâmico - Extensível - Remoção

Remove(D, k):

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

Acessa o diretório D nessa posição e acessa o bucket 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]

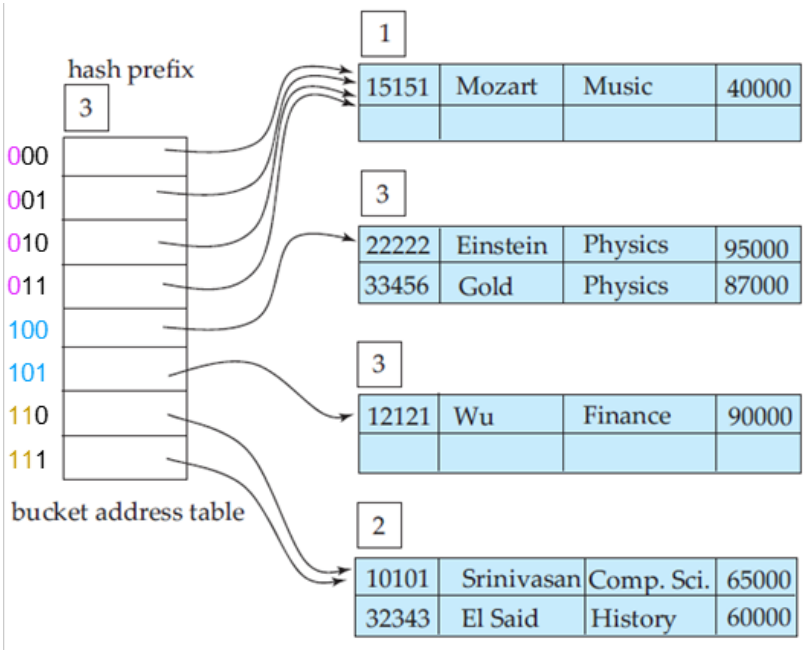
libera bloco apontado por D[e0]

D[e0] ← D[e1]

decrementa i' do bloco apontado por D[e1]

se max(i') < i

Divide D pela metade



Hashing Dinâmico - Extensível - Remoção

Remove(D, k):

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

Acessa o diretório D nessa posição e acessa o bucket 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]

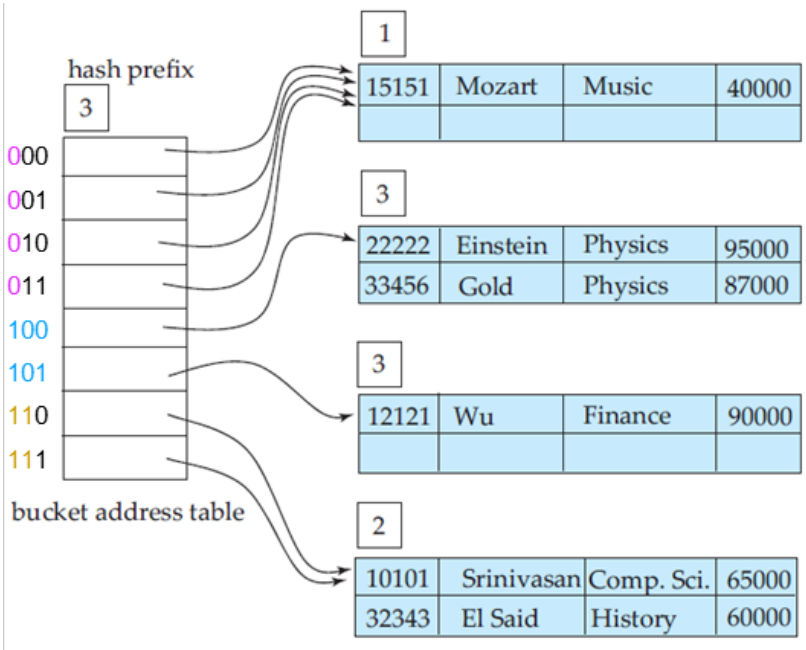
libera bloco apontado por D[e0]

D[e0] ← D[e1]

decrementa i' do bloco apontado por D[e1]

se max(i') < i

Divide D pela metade



Remoção do registro com hash 100 - *Gold*

Hashing Dinâmico - Extensível - Remoção

Remove(D, k):

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

Acessa o diretório D nessa posição e acessa o bucket 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]

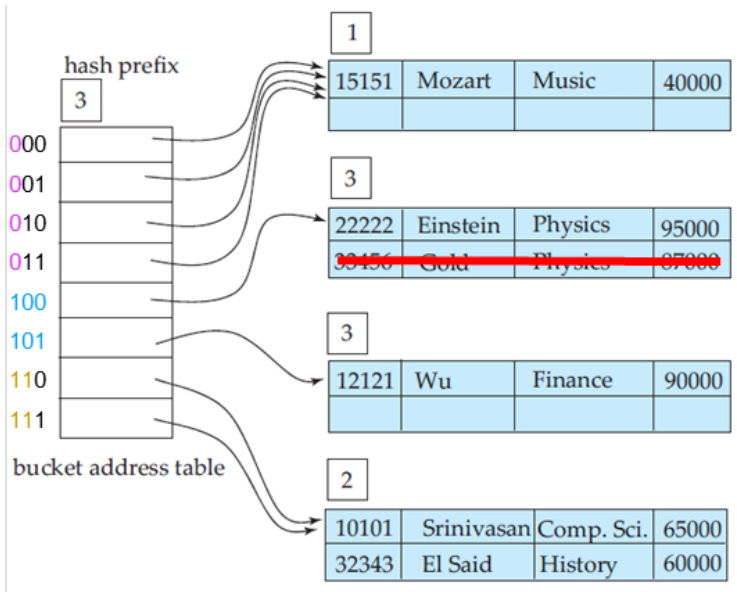
libera bloco apontado por D[e0]

D[e0] ← D[e1]

decrementa i' do bloco apontado por D[e1]

se max(i') < i

Divide D pela metade



Remoção do registro com hash 100 - *Gold*

Hashing Dinâmico - Extensível - Remoção

Remove(D, k):

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

Acessa o diretório D nessa posição e acessa o bucket 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]

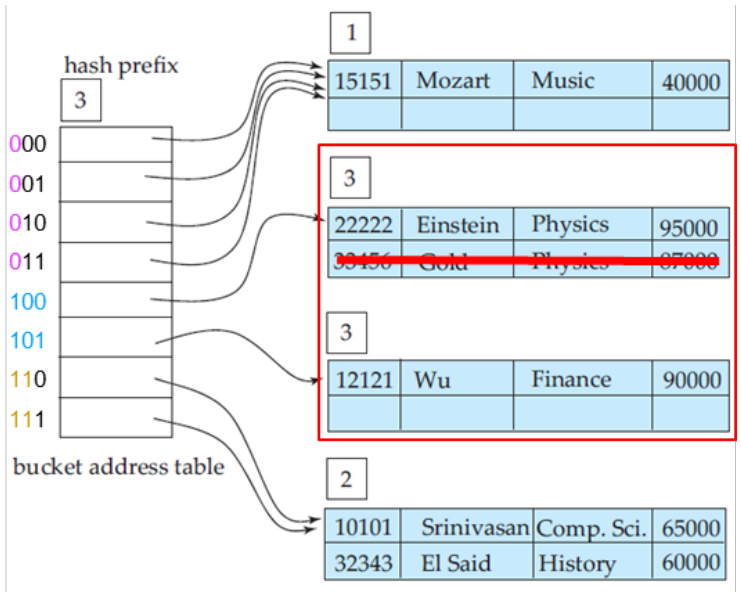
libera bloco apontado por D[e0]

D[e0] ← D[e1]

decrementa i' do bloco apontado por D[e1]

se max(i') < i

Divide D pela metade



Remoção do registro com hash 100 - *Gold*

Hashing Dinâmico - Extensível - Remoção

Remove(D, k):

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

Acessa o diretório D nessa posição e acessa o bucket 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]

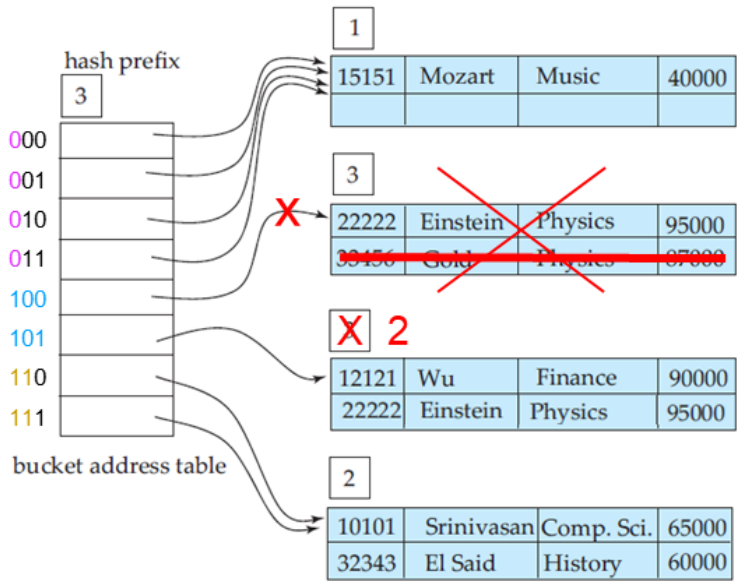
libera bloco apontado por D[e0]

D[e0] ← D[e1]

decrementa i' do bloco apontado por D[e1]

se max(i') < i

Divide D pela metade



Remoção do registro com hash 100 - *Gold*

Hashing Dinâmico - Extensível - Remoção

Remove(D, k):

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

Acessa o diretório D nessa posição e acessa o bucket 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]

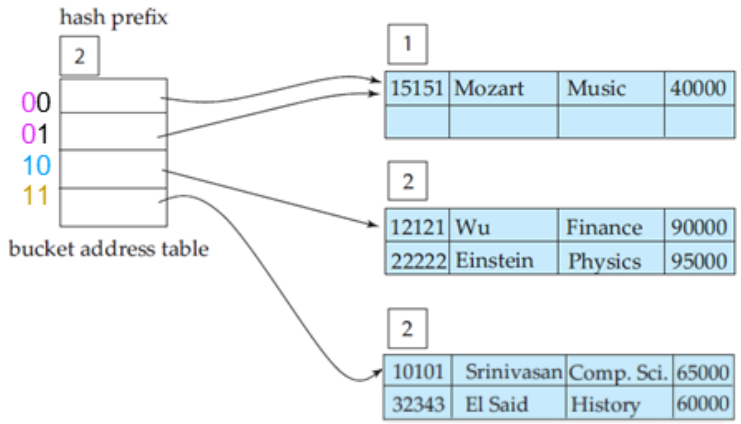
libera bloco apontado por D[e0]

D[e0] ← D[e1]

decrementa i' do bloco apontado por D[e1]

se max(i') < i

Divide D pela metade



Remoção do registro com hash 100 - *Gold*

Hashing Dinâmico - Extensível - Remoção

Remove(D, k):

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

Acessa o diretório D nessa posição e acessa o bucket 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]

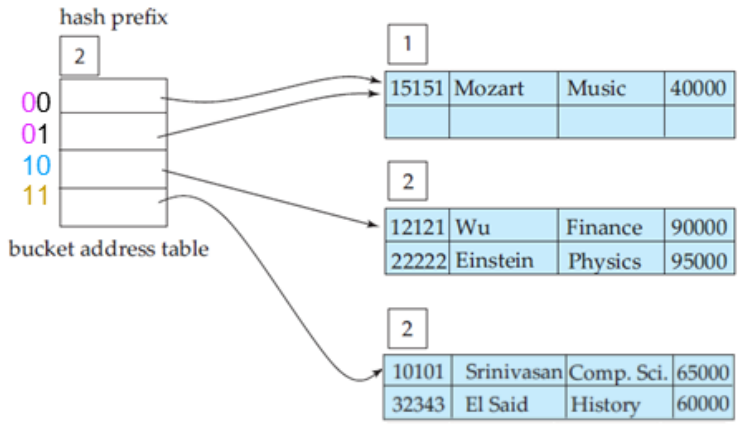
libera bloco apontado por D[e0]

D[e0] ← D[e1]

decrementa i' do bloco apontado por D[e1]

se max(i') < i

Divide D pela metade



Remoção do registro com hash 100 - *Gold*

Até 2 seeks, exclusão de até 1 bloco

Hashing Dinâmico - Extensível

- Tempo de busca: geralmente 1 acesso ao disco (1 no diretório que normalmente fica na memória principal e outro no bloco)
  - Não aumenta com o aumento do tamanho do arquivo (a não ser se houver blocos de overflow - raro)

Hashing Dinâmico - Extensível

- Tempo de busca: geralmente 1 acesso ao disco (1 no diretório que normalmente fica na memória principal e outro no bloco)
  - Não aumenta com o aumento do tamanho do arquivo (a não ser se houver blocos de overflow - raro)
- Espaço:
  - Diretório ocupa pouco espaço - no máximo 2^k, sendo k o número de bits do valor de hash (endereço-base)
  - Dobrar o diretório só aumenta um bloco (além do diretório)

Hashing Dinâmico - Extensível

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

Hashing Dinâmico - Linear

- M blocos e função de hash h1: x $\rightarrow$ 0, ..., M-1
- Colisões que causarem overflow vão para uma lista ligada de registros em blocos de overflow (até agora parece igual ao Hashing Estático..., mas as semelhanças param aqui)
- Uso de um contador (n) de overflows (colisões em blocos lotados)
- À medida que overflows forem ocorrendo (em quaisquer blocos), vou partindo em dois os blocos 0, 1, 2, ... linearmente.

Hashing Dinâmico - Linear - Ideia

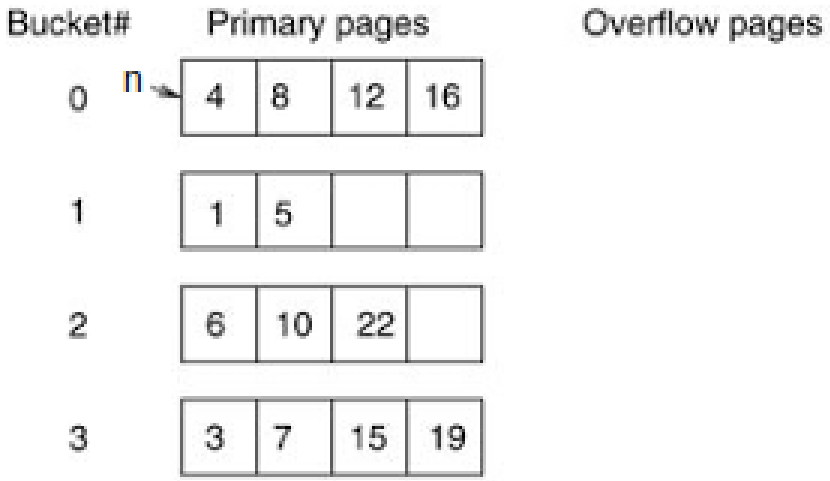
- Suponha M blocos (0 a M-1) e h1(k) = k mod M
  - n = 0
  - Primeiro overflow em QUALQUER bloco →
    - Aloca bloco M
    - Divide registros do bloco 0 entre os blocos 0 e M de acordo com h2(k) = k mod 2M
    - n = 1
  - Segundo overflow em QUALQUER bloco →
    - Aloca bloco M+1
    - Divide registros do bloco 1 entre os blocos 1 e M+1 de acordo com h2(k) = k mod 2M
    - n = 2
  - n-ésimo overflow:
    - Aloca bloco M+n-1
    - Divide registros do bloco n-1 entre os blocos n-1 e M+n-1 de acordo com h2(k) = k mod 2M
    - n++
  - ...

Hashing Dinâmico - Linear - Ideia

- Processo continua até que n = M:
  - todos os M blocos foram divididos
  - Tabela de Hash tem tamanho = 2M
  - h1 não é mais necessária
  - n = 0 e recomeça nova etapa de divisões
  - Funções de hash ativas:
    - h2(k) = k mod 2M
    - h3(k) = k mod 4M
- Processo continua até que n = 2M ...
- Sendo d = número de vezes que a tabela foi inteiramente dobrada:
  - Funções de hash ativas: $h_{d+1}(k)$ e $h_{d+2}(k)$, sendo $h_j(k) = kmod2^{j-1}M$
  - Busca(k): $h_{d+1}(k) < n$? Se sim, use $h_{d+2}(k)$ para saber em qual bloco está, senão use $h_{d+1}(k)$

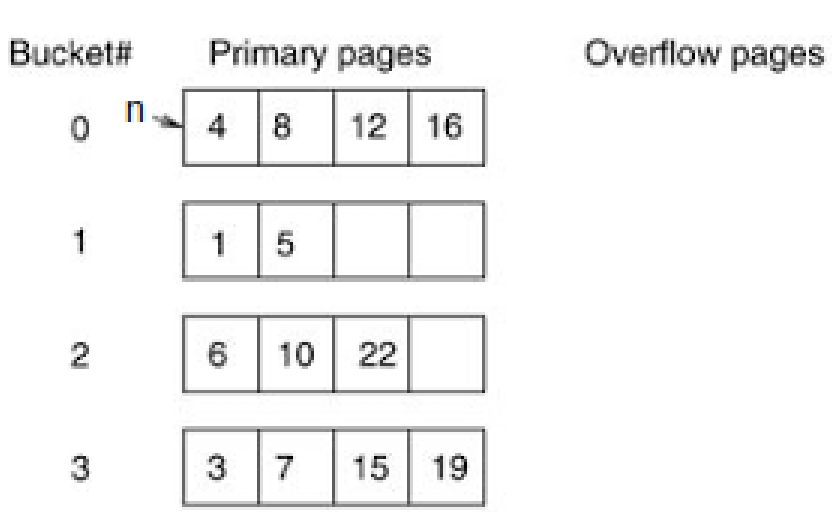
Hashing Dinâmico - Linear

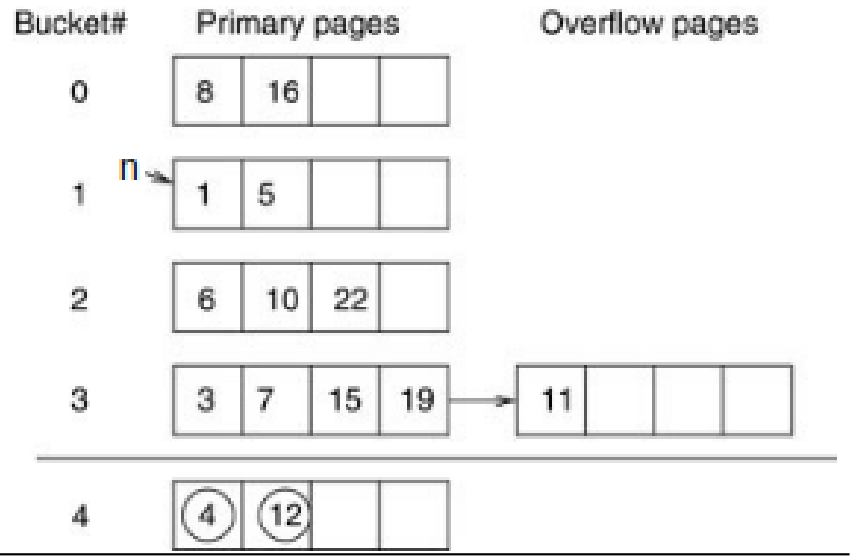
Inserir 11 em 3



Hashing Dinâmico - Linear

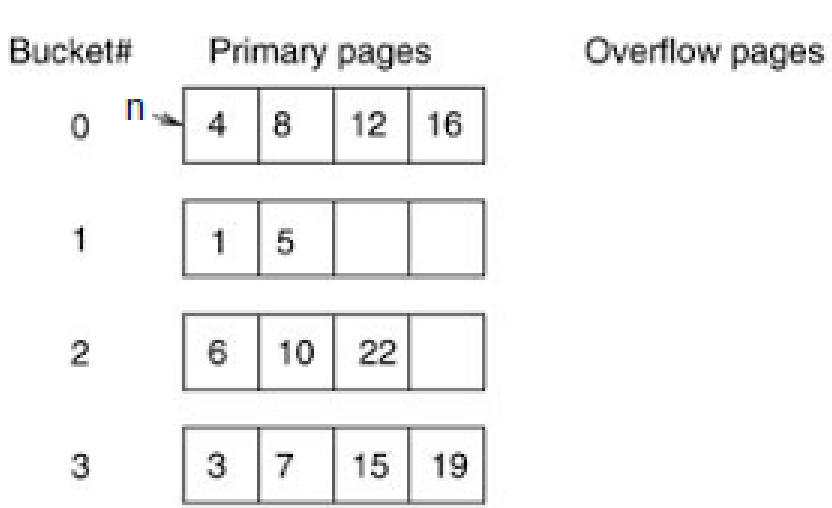
Inserir 11 em 3

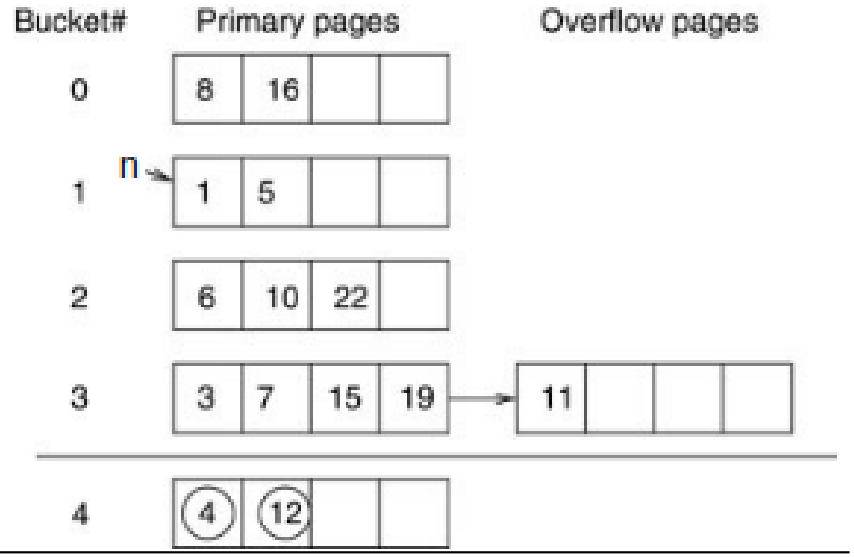




Hashing Dinâmico - Linear

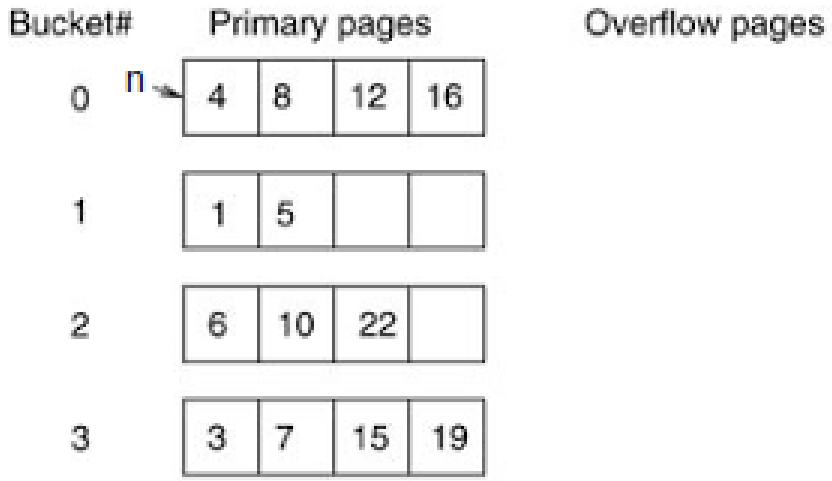
Inserir 18 e 26 em 2

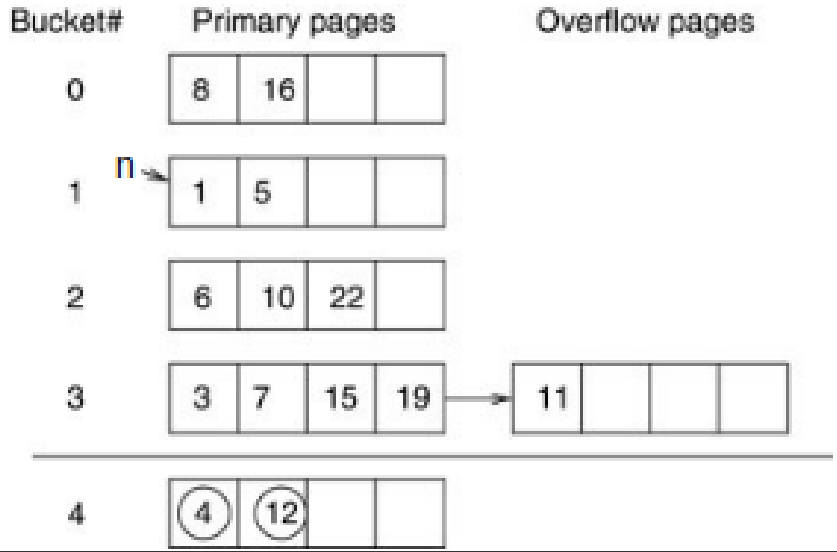


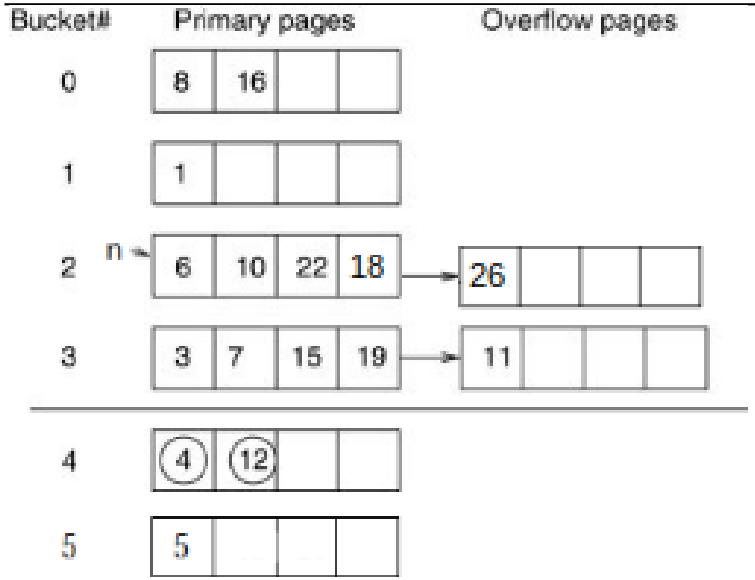


Hashing Dinâmico - Linear

Inserir 18 e 26 em 2

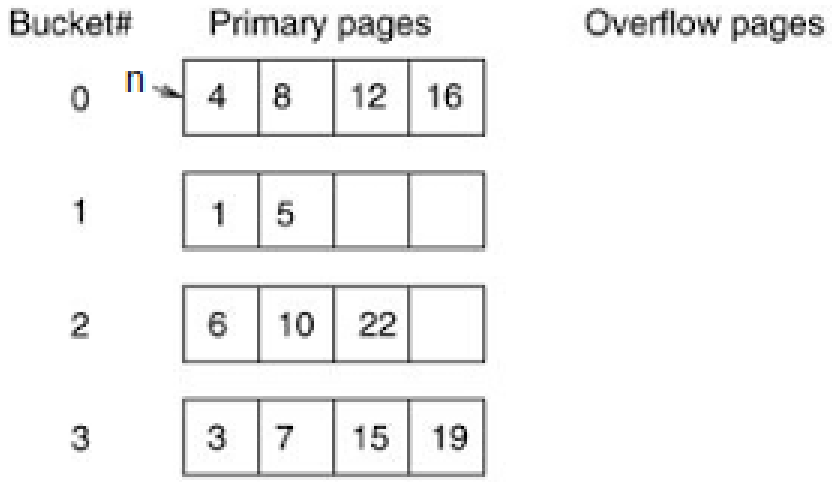


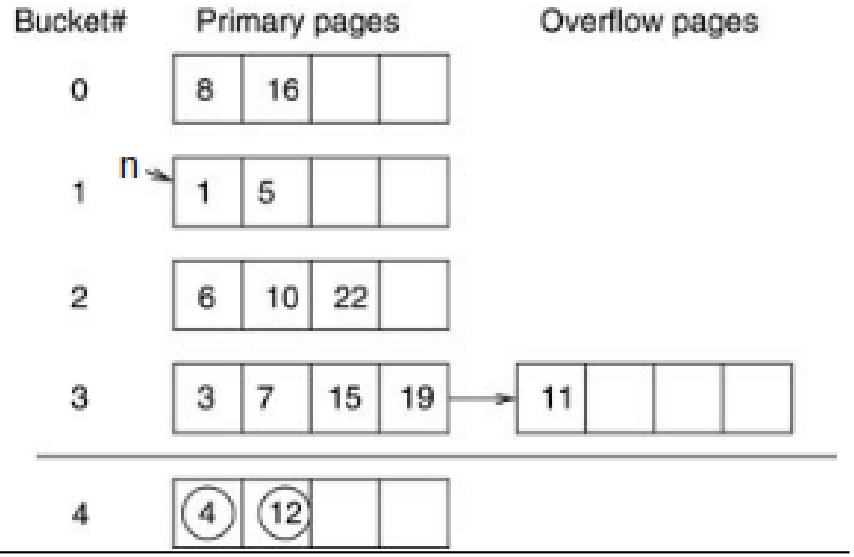


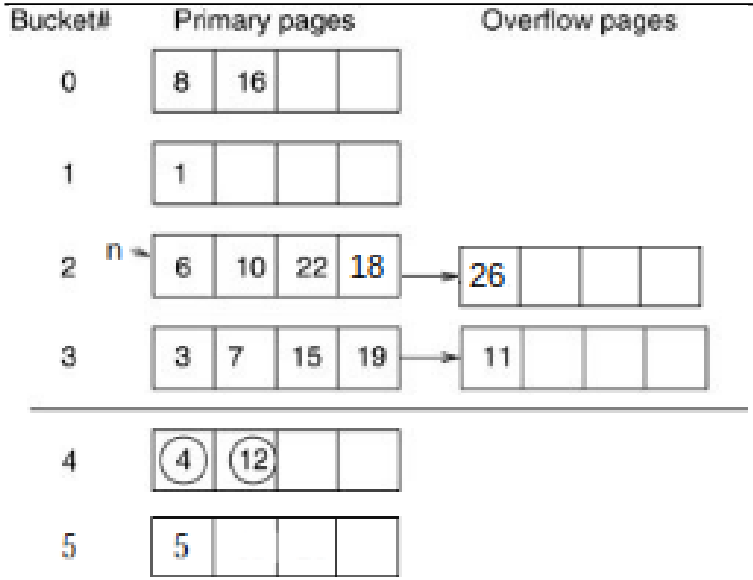


Hashing Dinâmico - Linear

Inserir 9, 13, 21 e 34

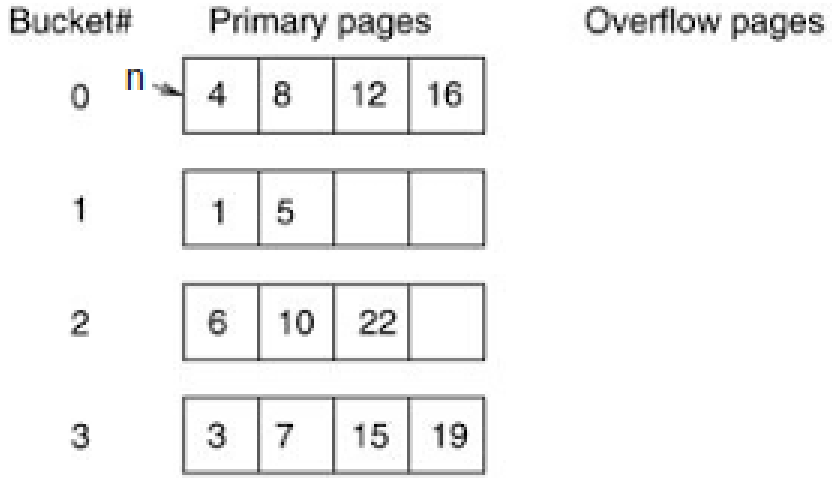


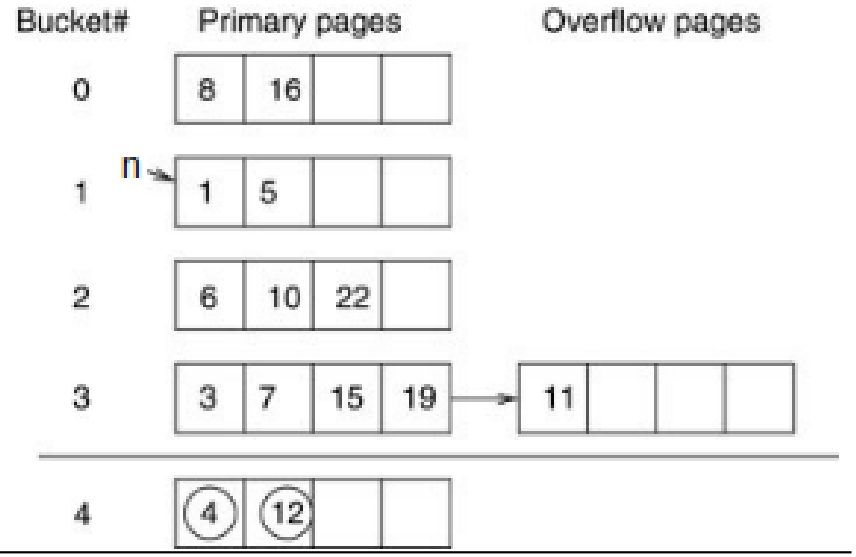


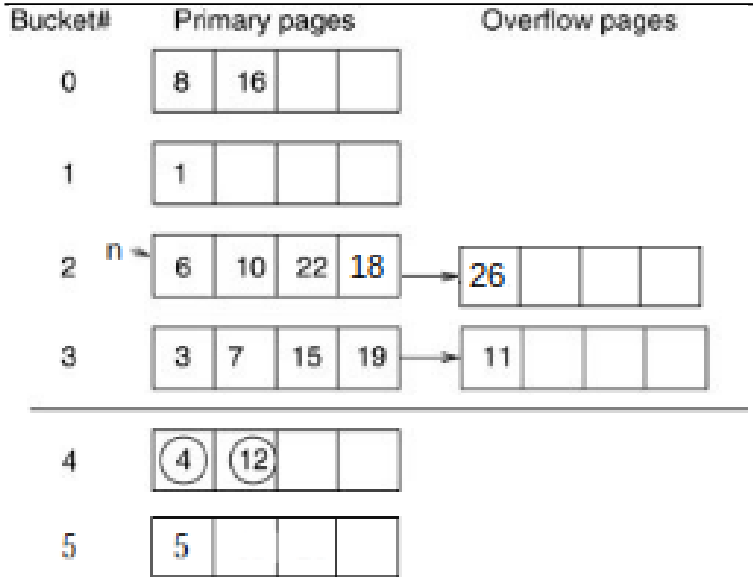


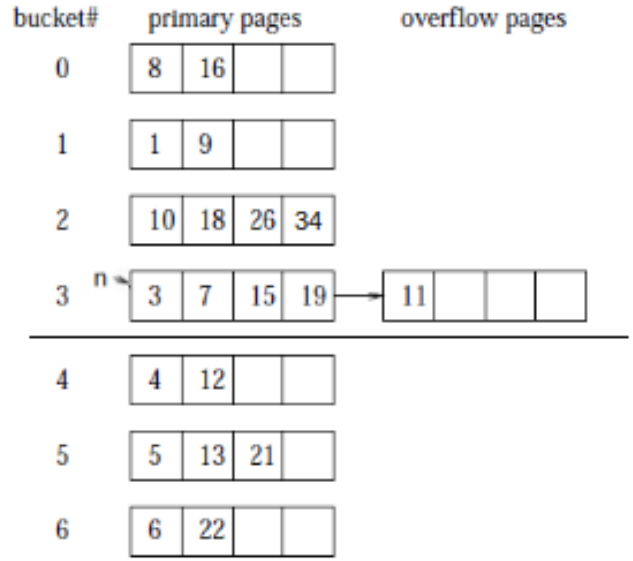
Hashing Dinâmico - Linear

Inserir 9, 13, 21 e 34



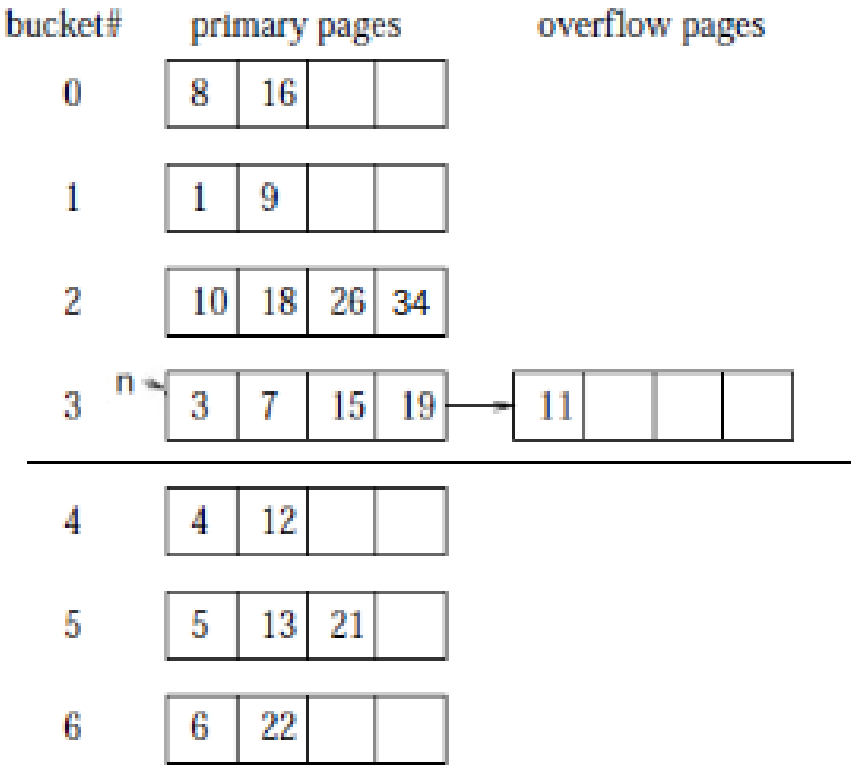






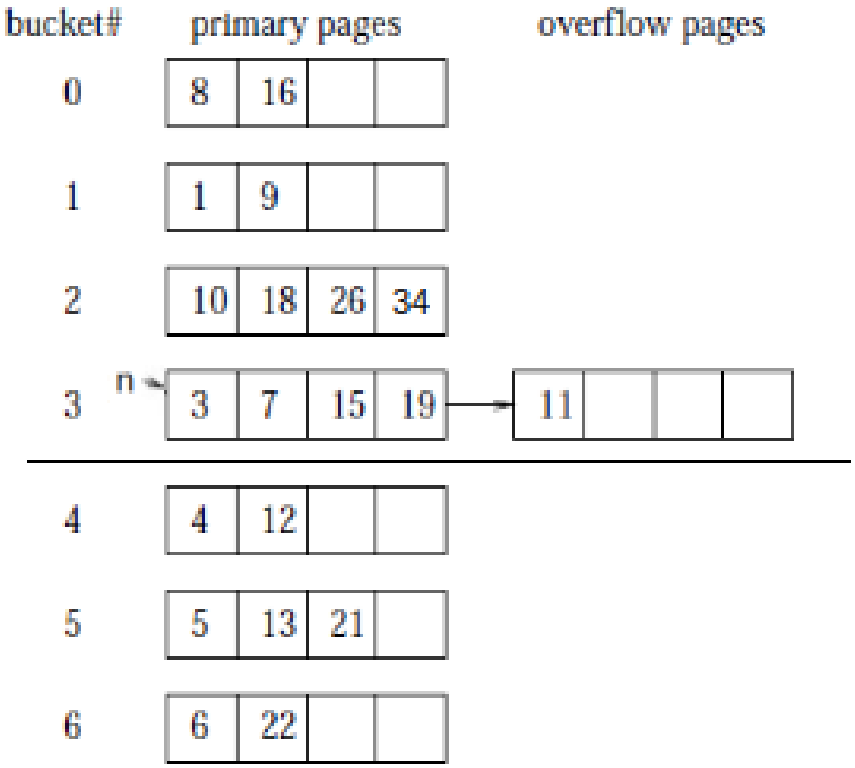
Hashing Dinâmico - Linear

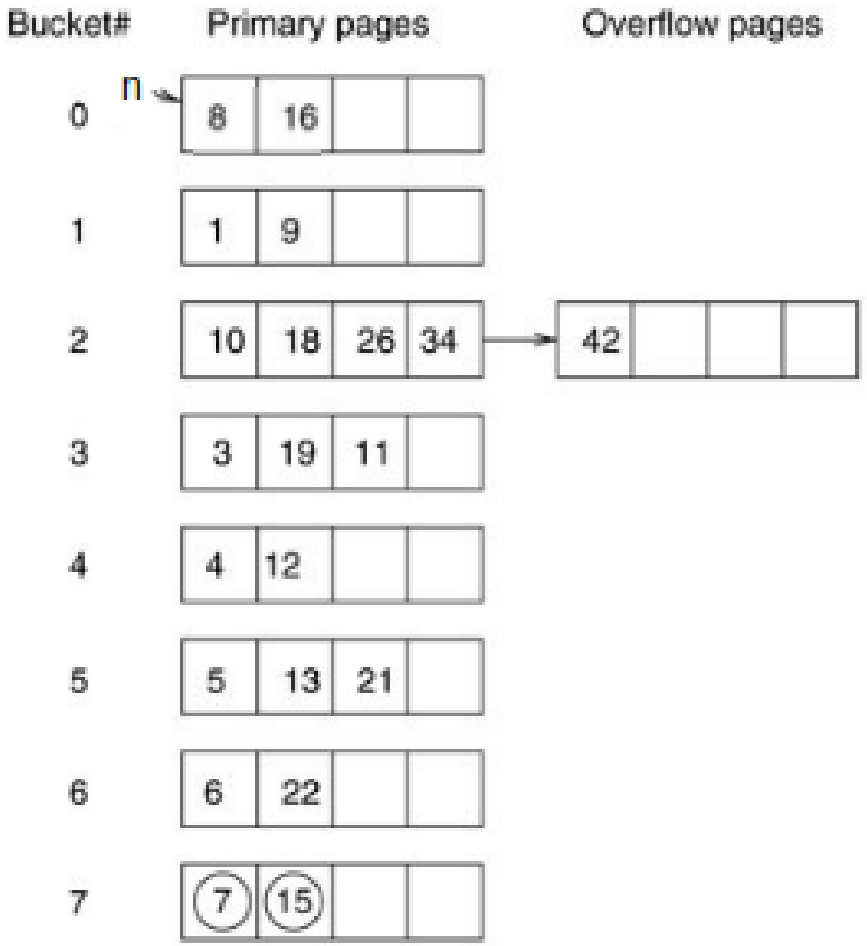
Inserir 42 em 2



Hashing Dinâmico - Linear

Inserir 42 em 2





O valor de **n volta a 0** (zero) e usamos uma única função hash: $h_2(k) = k \ mod \ 8$, reiniciando o processo.

Hashing Dinâmico - Linear - Ideia

- Estratégia alternativa para controlar dinamismo (ao invés de dividir a cada colisão):
  - Fator de carga $\alpha = N/(b * r)$, N = número de registros, b = número de blocos, r = número de registros que cabem em um bloco
  - Definir intervalo aceitável do fator de carga (ex: $0,7 <= \alpha <= 0,9$)
  - Se α acima do limite superior, divide blocos
  - Se α abaixo do limite inferior, junta blocos (em um processo inverso, decrementando n)

Hashing Dinâmico - Linear - Ideia

- Hash é muito rápido independente do tamanho do arquivo (árvore B+ depende da altura)
- Busca ordenada no hash em geral muito ruim
  - Hash mais apropriado para dados tipo dicionário ($< chave, valor >$, sendo cada entrada "independente" das demais)

Hashing Dinâmico - Linear - Ideia

- Hash é muito rápido independente do tamanho do arquivo (árvore B+ depende da altura)
- Busca ordenada no hash em geral muito ruim
  - Hash mais apropriado para dados tipo dicionário ($< chave, valor >$, sendo cada entrada "independente" das demais)
- Se precisar fazer um índice para um campo:
  - Busca apenas por valores específicos (utilizando apenas comparação de igualdade)?

Hashing Dinâmico - Linear - Ideia

- Hash é muito rápido independente do tamanho do arquivo (árvore B+ depende da altura)
- Busca ordenada no hash em geral muito ruim
  - Hash mais apropriado para dados tipo dicionário ($< chave, valor >$, sendo cada entrada "independente" das demais)
- Se precisar fazer um índice para um campo:
  - Busca apenas por valores específicos (utilizando apenas comparação de igualdade)?
    Usar hash

Hashing Dinâmico - Linear - Ideia

- Hash é muito rápido independente do tamanho do arquivo (árvore B+ depende da altura)
- Busca ordenada no hash em geral muito ruim
  - Hash mais apropriado para dados tipo dicionário ($< chave, valor >$, sendo cada entrada "independente" das demais)
- Se precisar fazer um índice para um campo:
  - Busca apenas por valores específicos (utilizando apenas comparação de igualdade)?
    Usar hash
  - Precisa ordenar os dados ou buscar conjunto de valores de acordo com uma relação entre os mesmos ($<, <=, >, >=$)?

Hashing Dinâmico - Linear - Ideia

- Hash é muito rápido independente do tamanho do arquivo (árvore B+ depende da altura)
- Busca ordenada no hash em geral muito ruim
  - Hash mais apropriado para dados tipo dicionário ($< chave, valor >$, sendo cada entrada "independente" das demais)
- Se precisar fazer um índice para um campo:
  - Busca apenas por valores específicos (utilizando apenas comparação de igualdade)?
    Usar hash
  - Precisa ordenar os dados ou buscar conjunto de valores de acordo com uma relação entre os mesmos ($<, <=, >, >=$)?
    Usar árvore B+

Referência

- Slides baseados no material da profa. Ariane Machado Lima - ACH2024
- 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.
- ZHANG, D., MANOLOPOULOS, Y., THEODORIDIS, Y., TSOTRAS, V.J. (2009). Linear Hashing. In: LIU, L., ÖZSU, M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-39940-9_742

Algoritmos e Estruturas de Dados II
Aula 27 – Hashing Dinâmico

Prof. Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2024