ACH2024

Aula 24 – Hashing em Disco (parte 3) - Hashing Dinâmico Linear

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 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**

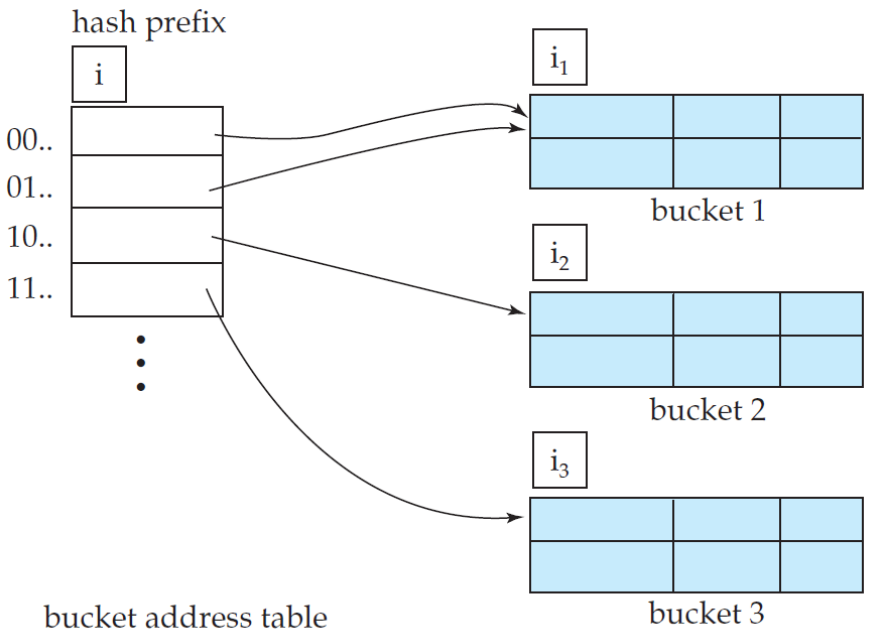


Figure 11.26 General extendable hash structure.

(SILBERSCHATZ, 2011)

Hashing Extensível - 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]

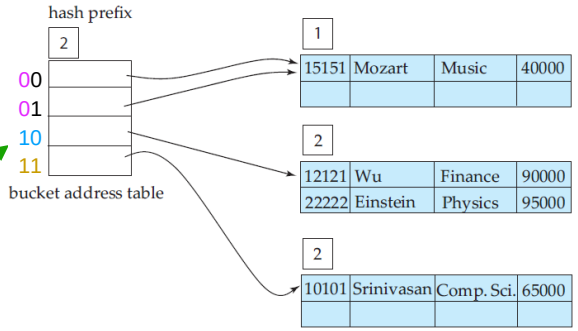
move para b' se (i_b+1)-ésimo bit de h(k') = 0

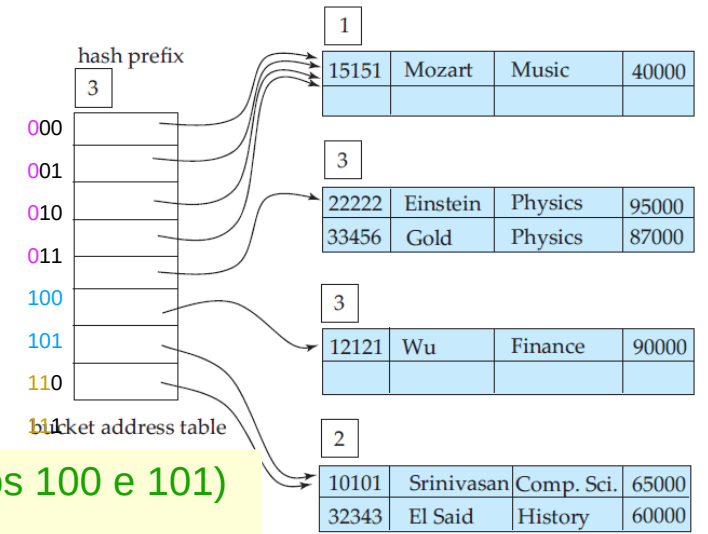
insere(D,k)

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

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]

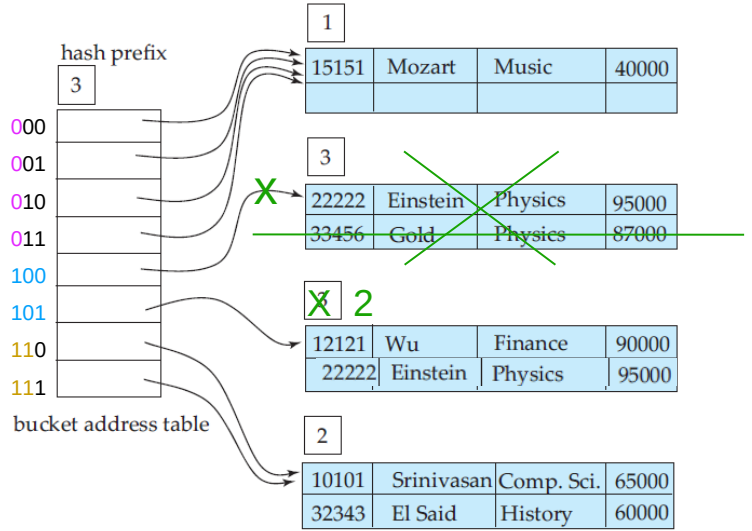
D[e0] ← D[e1]

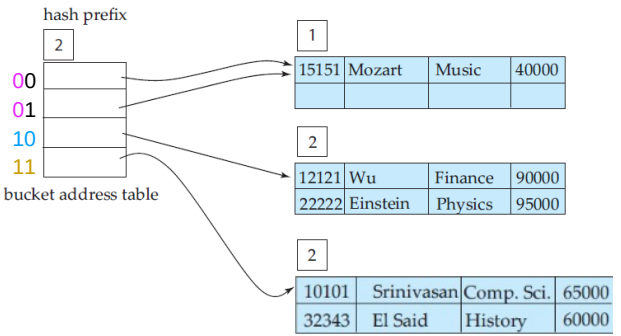
decrementa i' do bloco apontado por D[e1]

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

Aula de 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)

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

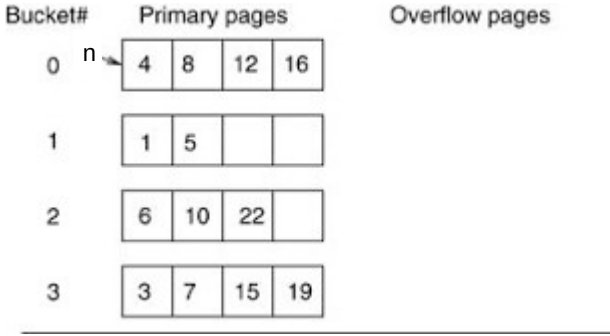
Hashing Linear

Hashing Linear (dinâmico)

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

Hashing Linear - Ideia

M = 4; n = 0; h1(k) = k mod 4



DOI: https://doi.org/10.1007/978-0-387-39940-9_742

Suponha M buckets (0 a M-1) e h1(k) = k mod M

- n = 0
- Primeiro overflow em QUALQUER bucket →
  - Aloca bucket M
  - Divide registros do bucket 0 entre os buckets 0 e M de acordo com h2(k) = k mod 2M
  - n = 1
- Segundo overflow em QUALQUER bucket →
  - Aloca bucket M+1
  - Divide registros do bucket 1 entre os buckets 1 e M+1 de acordo com h2(k) = k mod 2M
  - n = 2
- n-ésimo overflow:
  - Aloca bucket M+n-1
  - Divide registros do bucket n-1 entre os buckets n-1 e M+n-1 de acordo com h2(k) = k mod 2M
  - n ++
- (continua no próximo slide...)
- Mas nesse meio tempo… para n < M, como faço a busca?
  - Busca(k): h1(k) < n ? Se sim, use h2(k) para saber em qual bucket está, senão use h1(k)

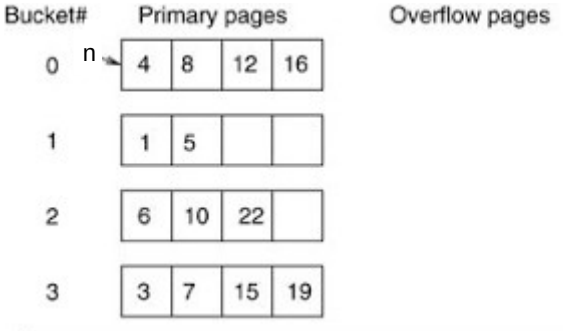
Hashing Linear - Ideia

Suponha M buckets (0 a M-1) e h1(k) = k mod M

- n = 0
- Primeiro overflow em QUALQUER bucket →
  - Aloca bucket M
  - Divide registros do bucket 0 entre os buckets 0 e M de acordo com h2(k) = k mod 2M
  - n = 1
- Segundo overflow em QUALQUER bucket →
  - Aloca bucket M+1
  - Divide registros do bucket 1 entre os buckets 1 e M+1 de acordo com h2(k) = k mod 2M
  - n = 2
- n-ésimo overflow:
  - Aloca bucket M+n-1
  - Divide registros do bucket n-1 entre os buck. n-1 e M+n-1 de acordo c/ h2(k) = k mod 2M
  - n ++
- (continua no próximo slide...)
- Mas nesse meio tempo… para n < M, como faço a busca?
  - Busca(k): h1(k) < n ? Se sim, use h2(k) para saber em qual bucket está, senão use h1(k)

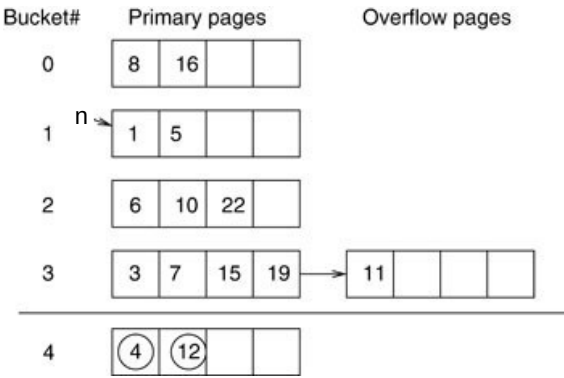
Ex: n=0→1 inserindo 11

M = 4; n = 0; h1(k) = k mod 4



M = 4; n = 1;

h1(k) = k mod 4
h2(k) = k mod 8



DOI: https://doi.org/10.1007/978-0-387-39940-9_742

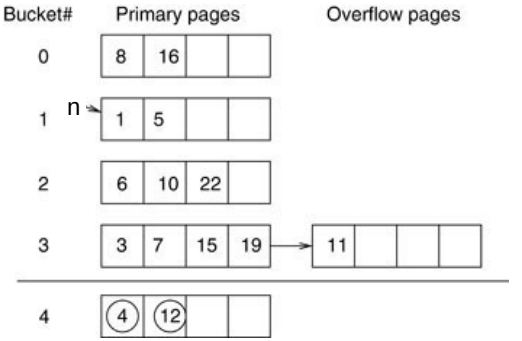
Hashing Linear - Ideia

Suponha M buckets (0 a M-1) e h1(k) = k mod M

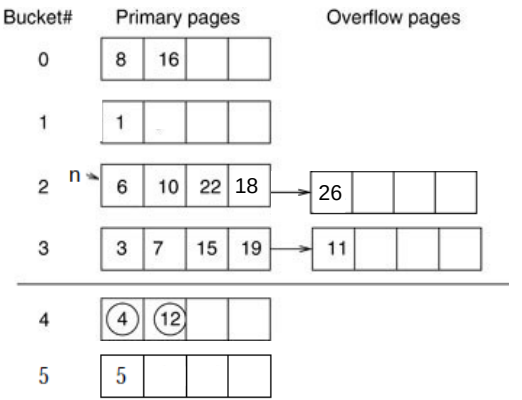
- n = 0
- Primeiro overflow em QUALQUER bucket →
  - Aloca bucket M
  - Divide registros do bucket 0 entre os buckets 0 e M de acordo com h2(k) = k mod 2M
  - n = 1
- Segundo overflow em QUALQUER bucket →
  - Aloca bucket M+1
  - Divide registros do bucket 1 entre os buckets 1 e M+1 de acordo com h2(k) = k mod 2M
  - n = 2
- n-ésimo overflow:
  - Aloca bucket M+n-1
  - Divide registros do bucket n-1 entre os buck. n-1 e M+n-1 de acordo c/ h2(k) = k mod 2M
  - n ++
- (continua no próximo slide...)
- Mas nesse meio tempo… para n < M, como faço a busca?
  - Busca(k): h1(k) < n ? Se sim, use h2(k) para saber em qual bucket está, senão use h1(k)

Ex: n1→2 inserindo 18, 26

M = 4; n = 1; h1(k) = k mod 4 h2(k) = k mod 8



M = 4; n = 2; h1(k) = k mod 4 h2(k) = k mod 8



Hashing Linear - Ideia

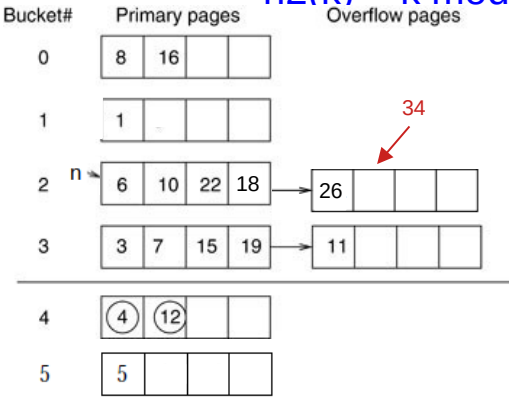
Suponha M buckets (0 a M-1) e h1(k) = k mod M

- n = 0
- Primeiro overflow em QUALQUER bucket →
  - Aloca bucket M
  - Divide registros do bucket 0 entre os buckets 0 e M de acordo com h2(k) = k mod 2M
  - n = 1
- Segundo overflow em QUALQUER bucket →
  - Aloca bucket M+1
  - Divide registros do bucket 1 entre os buckets 1 e M+1 de acordo com h2(k) = k mod 2M
  - n = 2
- n-ésimo overflow:
  - Aloca bucket M+n-1
  - Divide registros do bucket n-1 entre os buck. n-1 e M+n-1 de acordo c/ h2(k) = k mod 2M
  - n ++

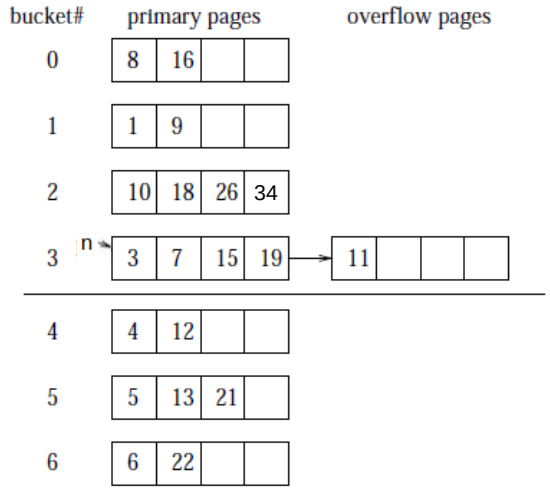
Ex: n=2→3 inserindo 9, 13, 21, 34

- (continua no próximo slide...)
- Mas nesse meio tempo… para n < M, como faço a busca?
  - Busca(k): h1(k) < n ? Se sim, use h2(k) para saber em qual bucket está, senão use h1(k)

M = 4; n = 2; h1(k) = k mod 4 h2(k) = k mod 8

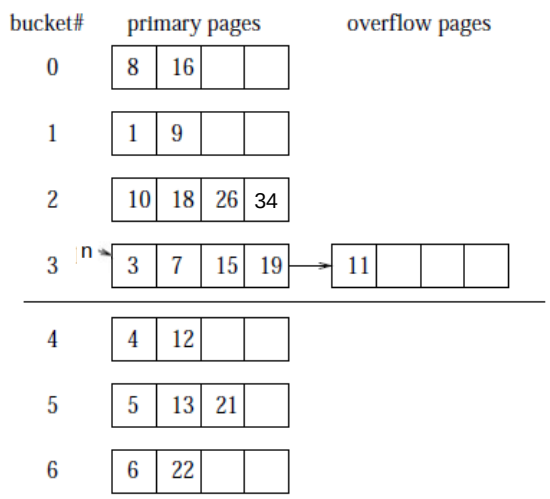


M = 4; n = 3; h1(k) = k mod 4 h2(k) = k mod 8



Hashing Linear - Ideia

M = 4; n = 3;
h1(k) = k mod 4
h2(k) = k mod 8

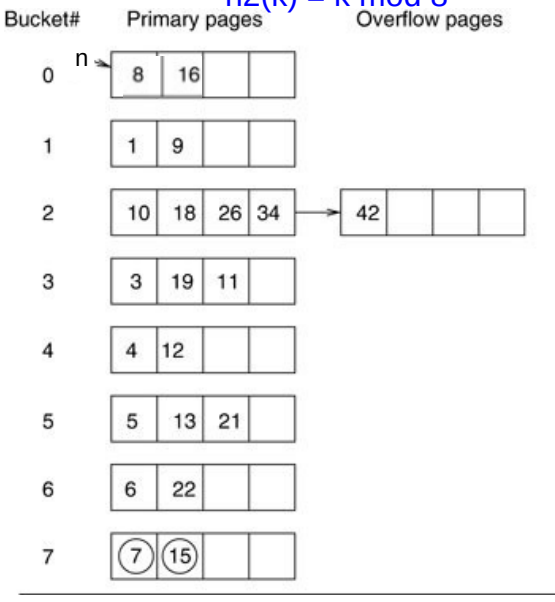


Suponha M buckets (0 a M-1) e h1(k) = k mod M

- n = 0
- Primeiro overflow em QUALQUER bucket →
  - Aloca bucket M
  - Divide registros do bucket 0 entre os buckets 0 e M de acordo com h2(k) = k mod 2M
  - n = 1
- Segundo overflow em QUALQUER bucket →
  - Aloca bucket M+1
  - Divide registros do bucket 1 entre os buckets 1 e M+1 de acordo com h2(k) = k mod 2M
  - n = 2
- n-ésimo overflow:
  - Aloca bucket M+n-1
  - Divide registros do bucket n-1 entre os buckets n-1 e M+n-1 de acordo c/ h2(k) = k mod 2M
  - n ++

Ex: n=3→4 inserindo 42

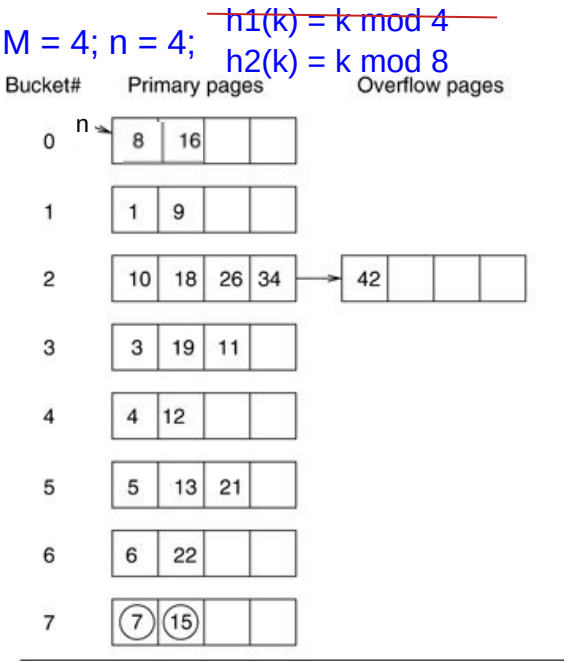
M = 4; n = 4;
h1(k) = k mod 4
h2(k) = k mod 8



- (continua no próximo slide...)
- Mas nesse meio tempo… para n < M, como faço a busca?
  - Busca(k): h1(k) < n ? Se sim, use h2(k) para saber em qual bucket está, senão use h1(k)

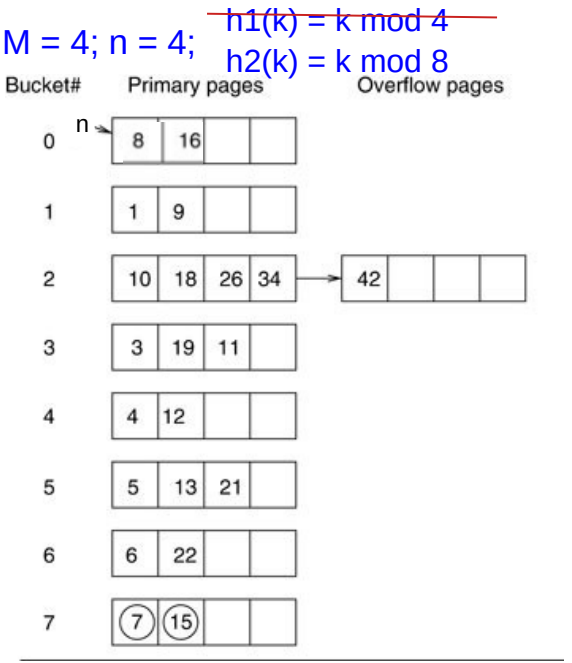
Hashing Linear

- Processo continua até que n = M:
  - todos os M buckets 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
- …
- Se d = nr de vezes que a tabela foi inteiramente dobrada:
  - Funções de hash ativas: $h_{d+1}(k)$ e $h_{d+2}(k)$, sendo $h_j(k) = k \bmod 2^{j-1}M$
  - Busca(k):



Hashing Linear

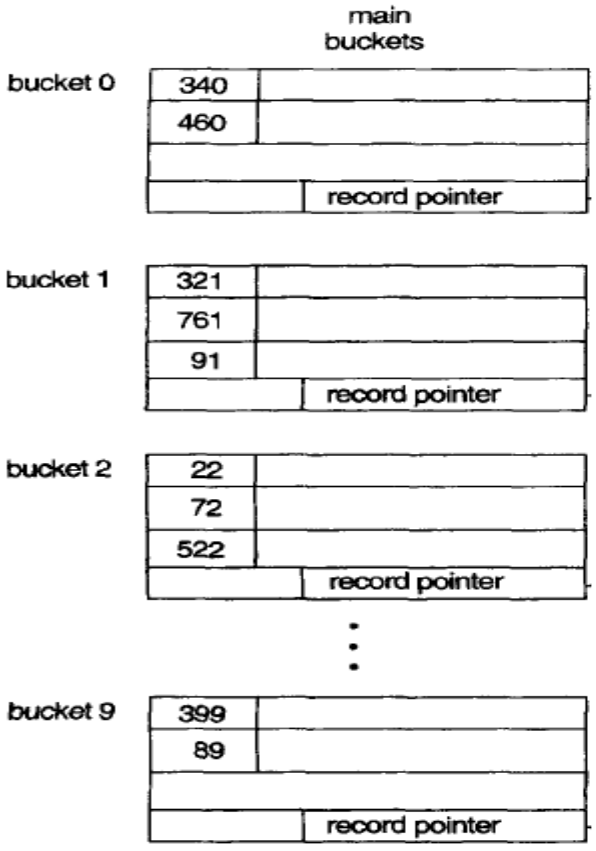
- Processo continua até que n = M:
  - todos os M buckets foram divididos
  - Tabela de Hash tem tamanho = 2M
  - h1 não é mais necessária
  - n ← 0 e recomeça nova etapa de divisões
  - Funções de hash ativas:
    - h2(k) = k mod 2M
    - h3(k) = k mod 4M
  - Processo continua até que n = 2M
- …
- Se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

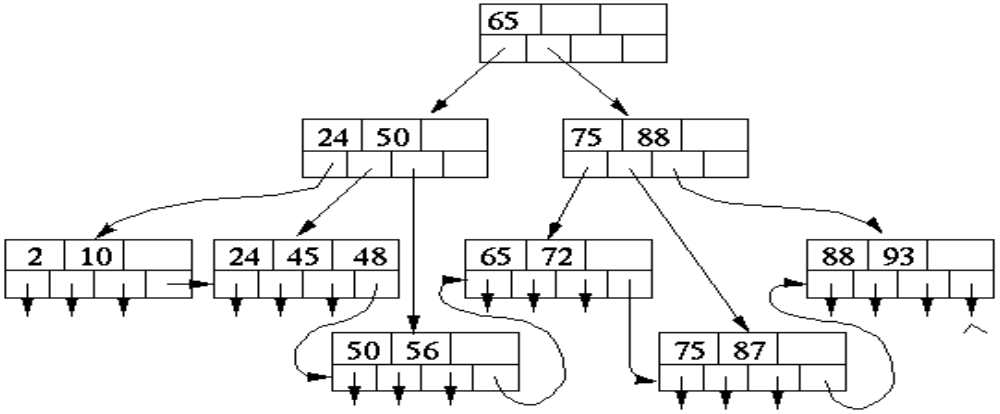


Hashing Linear

- Estratégia alternativa para controlar dinamismo (ao invés de dividir a cada colisão):
  - Fator de carga **α** = N/(b*r), N = nr de registros, b = nr de buckets, r = número de registros que cabem em um bucket
  - Definir intervalo aceitável do fator de carga (ex: 0.7 <= **α** <= 0.9)
  - Se **α** acima do limite superior, divide buckets
  - Se **α** abaixo do limite inferior, junta buckets (em um processo inverso, decrementando n)

Hash x árvores B+: diferenças





Hash x árvores B+

- 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)?
    -
  - Precisa ordenar os dados ou buscar conjunto de valores de acordo com uma relação entre os mesmos (<, <=, >, >=)?
    -

Hash x árvores B+

- 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)?
    - Use hash
  - Precisa ordenar os dados ou buscar conjunto de valores de acordo com uma relação entre os mesmos (<, <=, >, >=)?
    - Use árvore B+

Exercícios

Implemente todas essas opções de Hashing (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.