CEFET/RJ

INDEXAÇÃO

Eduardo Ogasawara
eogasawara@ieee.org
https://eic.cefet-rj.br/~eogasawara

Indexação

- Conceitos básicos
- Índices ordenados
- Árvore B+
- Hashing estático
- Hashing dinâmico
- Comparação de indexação ordenada e hashing
- Definição de índice em SQL
- Acesso por chave múltipla

Conceitos básicos

- Os mecanismos de indexação são usados para agilizar o acesso aos dados desejados
  - Por exemplo, catálogo de autores na biblioteca
- Chave de busca: atributo ou conjunto de atributos para pesquisar registros em um arquivo
- Um arquivo de índice consiste em registros (chamados entradas de índice) na forma

- Arquivos de índice normalmente são muito menores do que o arquivo original
- Dois tipos básicos de índices:
  - Índices ordenados: chaves de busca são armazenadas em ordem
  - Índices de hash: chaves de busca são distribuídas uniformemente entre "baldes" usando uma "função de hash"

Métrica de avaliação de índice

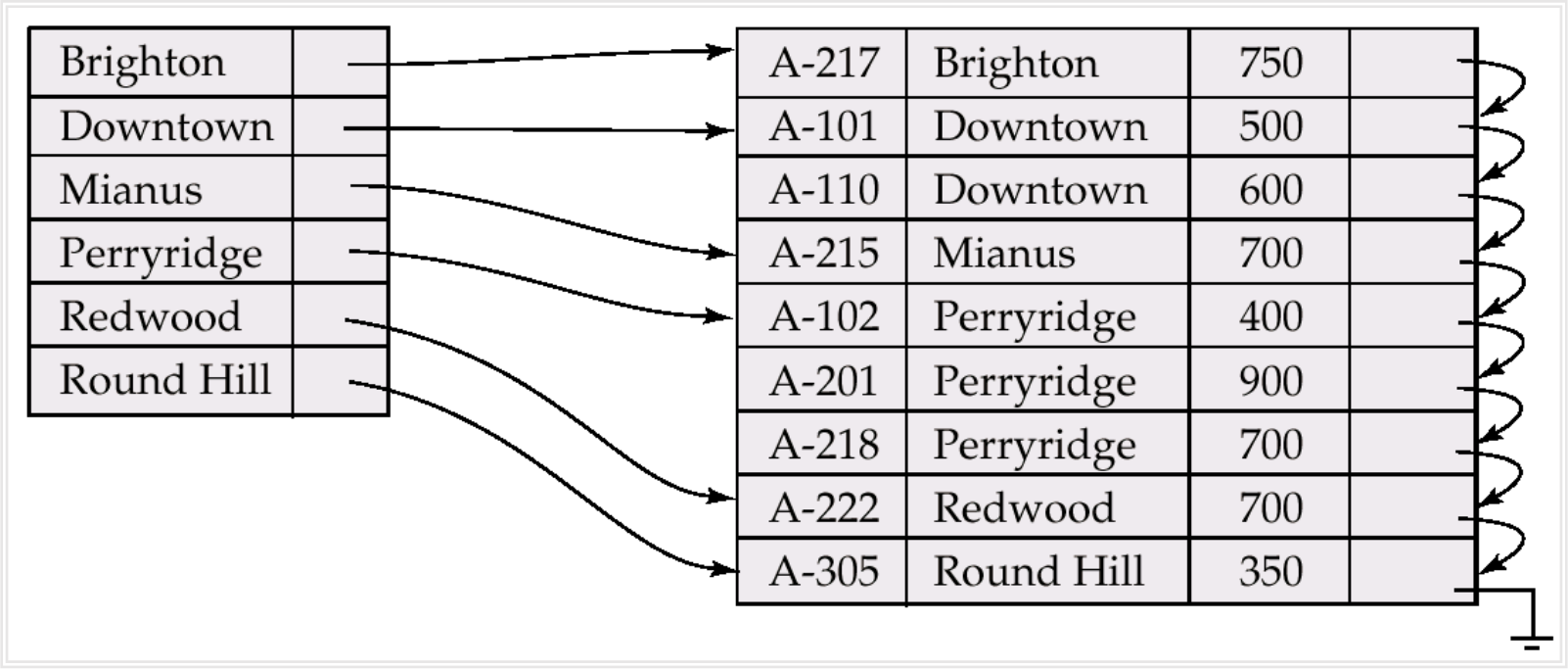
- Tipos de acesso admitidos
  - registros com um valor especificado no atributo
  - ou registros com um valor de atributo caindo em um intervalo especificado de valores
- Tempo de acesso
- Tempo de inserção
- Tempo de exclusão
- Sobrecarga de espaço

Índices ordenados

- Em um índice ordenado, as entradas de índice são armazenadas ordenadas pelo valor da chave de busca
  - Por exemplo, catálogo de autor na biblioteca
- Índice primário
  - em um arquivo ordenado sequencialmente, o índice cuja chave de busca especifica a ordem sequencial do arquivo
  - Também chamado índice de agrupamento
  - A chave de busca de um índice primário é normalmente a chave primária
- Arquivo sequencial indexado
  - arquivo sequencial ordenado com um índice primário
- Índice secundário
  - um índice cuja chave de busca especifica uma ordem diferente da ordem sequencial do arquivo
  - Também chamado índice não de agrupamento

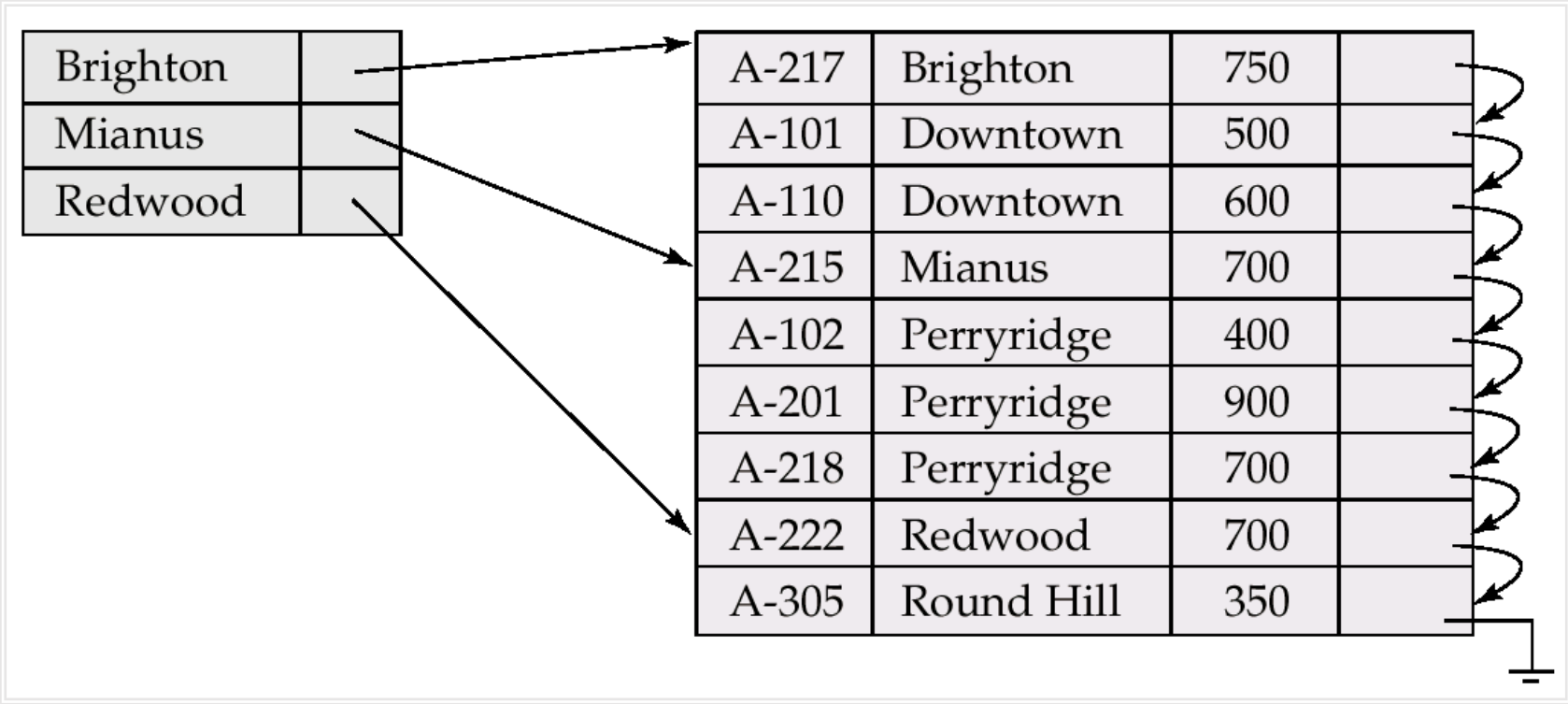
Arquivos de índice denso

- Índice denso
  - O registro de índice aparece para cada valor de chave de busca no arquivo



Arquivos de índice esparso

- Índice esparso
  - Contém registros de índice para somente alguns valores de chave de busca
  - Aplica-se quando os registros são ordenados sequencialmente por chave de busca



Para localizar um registro com o valor de chave de busca K:

Encontramos o registro de índice com o maior valor de chave de busca < K

Pesquisamos o arquivo sequencialmente, começando no registro para o qual o registro de índice aponta

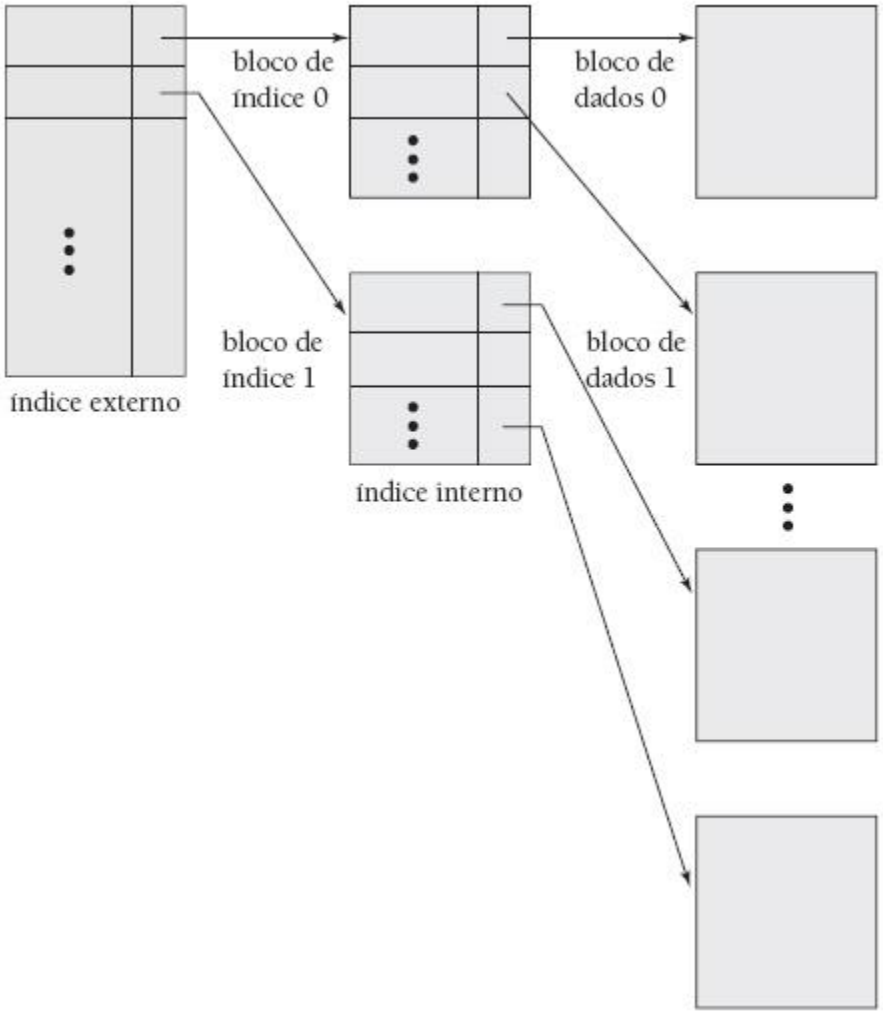
Características de arquivos de índice esparso

- Menos espaço e menos sobrecarga de manutenção para inserções e exclusões
- Geralmente mais lento do que o índice denso para localizar registros
- Boa escolha: índice esparso com uma entrada de índice para cada bloco no arquivo, correspondente ao menor valor de chave de busca no bloco

Índice multinível

- Se o índice primário não couber na memória, o acesso se torna dispendioso
- Para reduzir o número de acessos de disco aos registros de índice, trate o índice primário mantido em disco como um arquivo sequencial e construa um índice esparso sobre ele
  - índice externo – um índice esparso do índice primário
  - índice interno – o arquivo de índice primário
- Se até mesmo o índice externo for muito grande para caber na memória principal, outro nível de índice pode ser criado, e assim por diante
- Os índices, em todos os níveis, precisam ser atualizados na inserção ou exclusão no arquivo

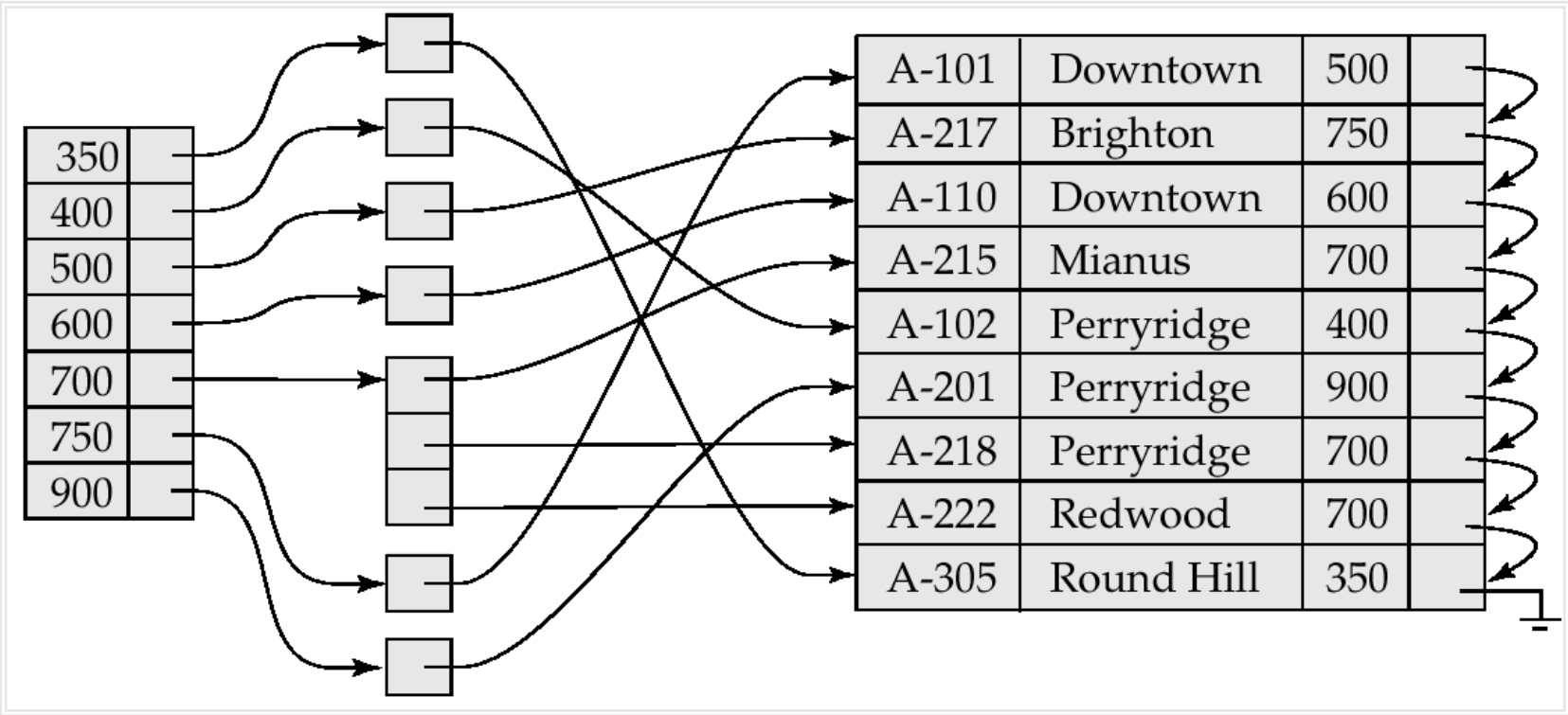
Exemplo de índice multinível



Índices secundários

- Frequentemente, alguém deseja encontrar todos os registros cujos valores em um certo campo (que não seja a chave de busca do índice primário) satisfazem alguma condição
  - Exemplo: No banco de dados, as contas são armazenadas sequencialmente por número de conta, queremos encontrar todas as contas com um saldo especificado ou intervalo de saldos
- Podemos ter um índice secundário com um registro de índice para cada valor de chave de busca
  - o registro de índice aponta para um balde que contém ponteiros para todos os registros reais com esse valor de chave de busca específico

Índice secundário sobre o campo saldo de conta



Índices primários e secundários

- Índices secundários precisam ser densos
- Índices oferecem benefícios substanciais quando procuram registros
- Quando um arquivo é modificado, cada índice no arquivo precisa ser atualizado
  - A atualização de índices impõe sobrecarga na modificação do banco de dados
- A varredura sequencial usando índice primário é eficiente, mas uma varredura sequencial usando um índice secundário é dispendiosa
  - Cada acesso a registro pode apanhar um novo bloco do disco
  - Ao mesmo tempo, entradas diferentes do índice secundário podem exigir a recarga de um bloco previamente carregado

Árvore B+

Arquivos de índice de árvore B+

- Os arquivos de índice B+ reorganizam-se automaticamente com pequenas mudanças locais, em face a inserções e exclusões
- A reorganização do arquivo inteiro não é necessária para manter o desempenho

Propriedades de arquivos de índice de árvore B+

- Uma árvore B+ é uma árvore com raiz satisfazendo as seguintes propriedades:
  - Todos os caminhos da raiz até a folha são do mesmo tamanho
  - Cada nó que não é uma raiz ou uma folha tem entre $\frac{n}{2}$ e n filhos
  - Um nó folha tem entre $\frac{n-1}{2}$ e $n-1$ valores
- Casos especiais:
  - Se a raiz não for uma folha, ela tem pelo menos 2 filhos
  - Se a raiz for uma folha (ou seja, não houver outros nós na árvore), ela pode ter entre 0 e $(n-1)$ valores

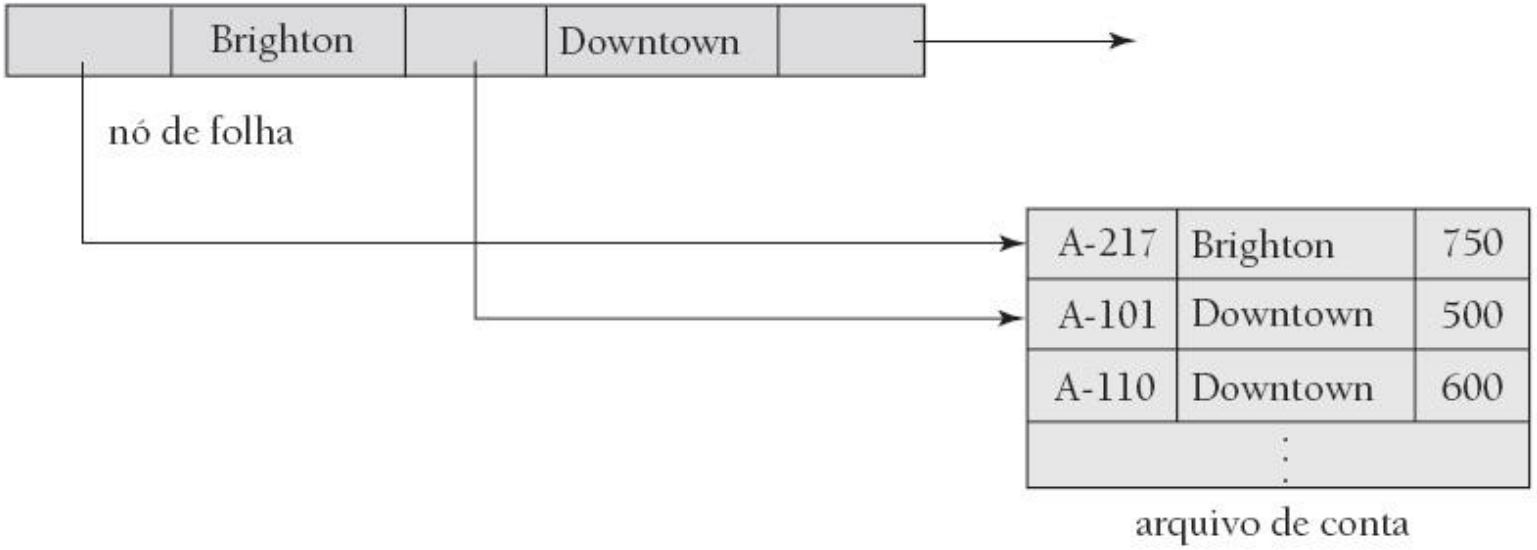
Estrutura de nós da árvore B+

- Nó típico

- $\boldsymbol{K_i}$ são os valores de chave de busca
- $\boldsymbol{P_i}$ são ponteiros para os filhos (para nós não-folha) ou ponteiros para registros ou baldes de registros (para nós de folha)
- As chaves de busca em um nó são ordenadas:
  $\boldsymbol{K_1 < K_2 < K_3 < \ldots < K_{n-1}}$

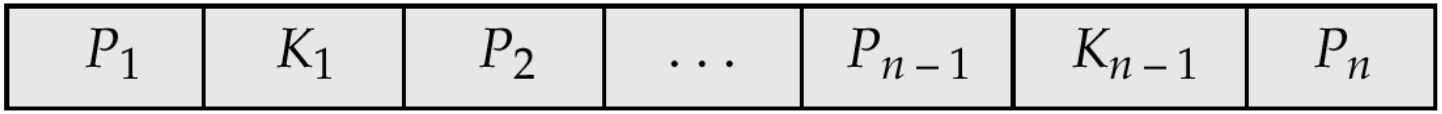
Nós folha nas árvores B+

- Para $i = 1, 2, \ldots, n-1$, o ponteiro $\boldsymbol{P_i}$ ou aponta para um registro de arquivo com valor de chave de busca $\boldsymbol{K_i}$, ou para um balde de ponteiros para registros de arquivo, cada registro tendo valor de chave de busca $\boldsymbol{K_i}$.
  - Só precisa de estrutura de balde se a chave de busca não formar uma chave primária
- Se $\boldsymbol{L_i}$, $\boldsymbol{L_j}$ forem nós de folha e $i < j$, os valores de chave de busca de $\boldsymbol{L_i}$ são menores que os valores de chave de busca de $\boldsymbol{L_j}$
- $\boldsymbol{P_n}$ aponta para o próximo nó folha na ordem da chave de busca



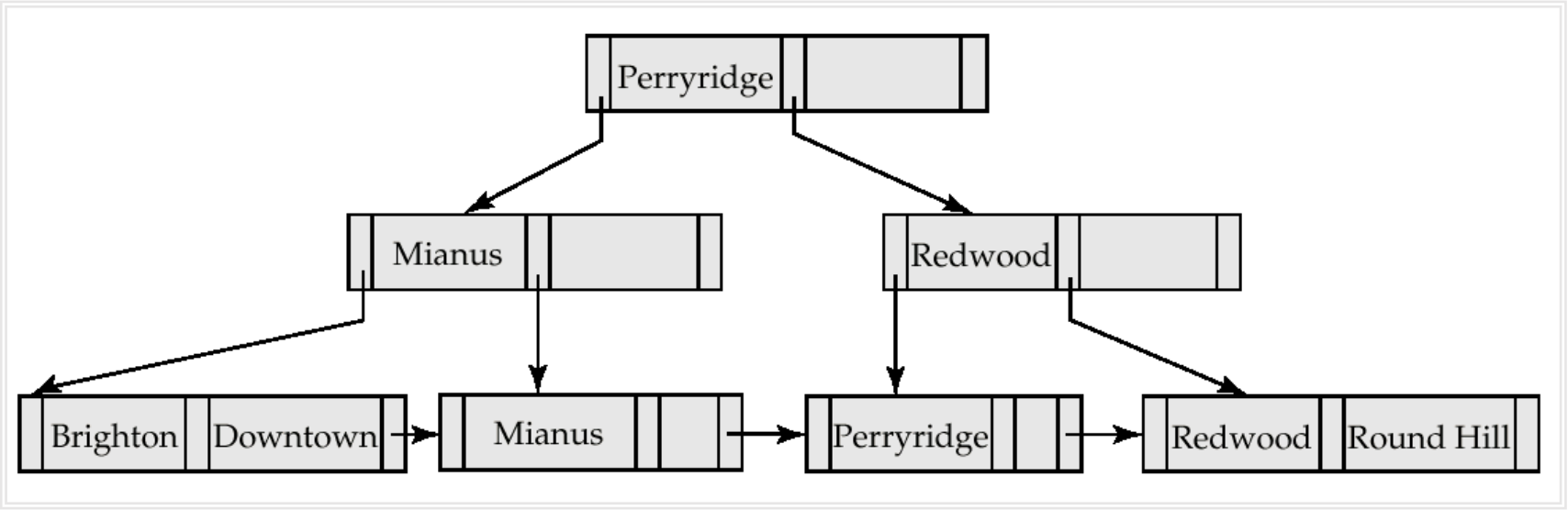
Nós não-folha nas árvores B+

- Os nós não-folha formam um índice esparso multinível sobre os nós de folha
- Para um nó não-folha com m ponteiros:
  - Todas as chaves de busca na sub-árvore à qual $\boldsymbol{P_1}$ aponta são menores que $\boldsymbol{K_1}$
  - Para $2 \leq i \quad n - 1$, todas as chaves de busca na sub-árvore à qual $\boldsymbol{P_i}$ aponta têm valores maiores ou iguais a $\boldsymbol{K_i}$ e menores que $\boldsymbol{K_{i+1}}$



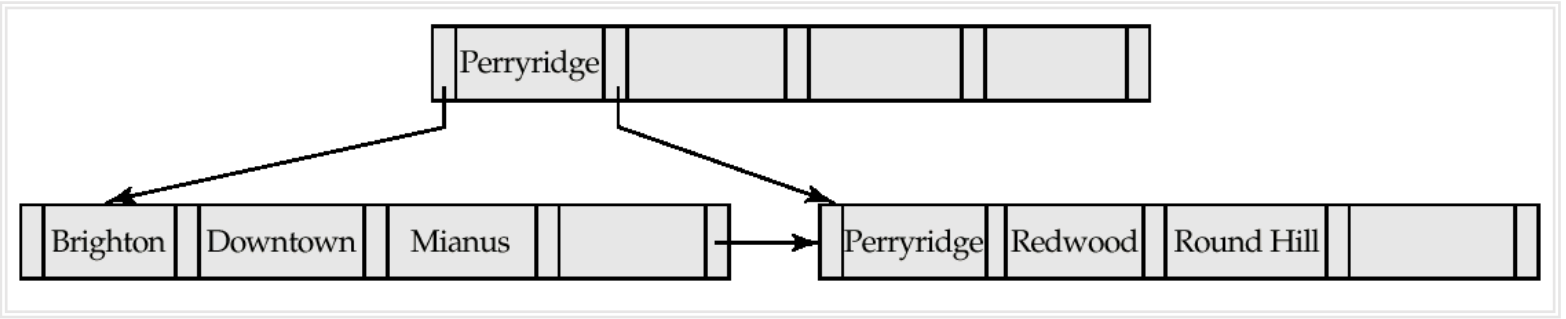
Exemplo de árvore B+

- Árvore B+ para arquivo conta (n = 3)



Exemplo de árvore B+

- Árvore B+ para arquivo conta ($n = 5$)
  - Nós de folha precisam ter entre $\left\lceil \frac{n-1}{2} \right\rceil$ e $n-1$
    Para $n = 5$, os valores ficam entre 2 e 4
  - Nós não-folha diferentes da raiz precisam ter entre $\left\lceil \frac{n}{2} \right\rceil$ e n
    Para n = 5, os valores ficam entre 3 e 5 filhos
  - Raiz precisa ter pelo menos 2 filhos



Observações sobre árvores B+

- Como as conexões entre nós são feitas por ponteiros, blocos "logicamente" próximos não precisam estar "fisicamente" próximos
- Os níveis não-folha da árvore B+ formam uma hierarquia de índices esparsos
- A árvore B+ contém uma quantidade relativamente pequena de níveis (logarítmica no tamanho do arquivo principal), de modo que as buscas podem ser conduzidas de modo eficiente
- Inserções e exclusões no arquivo principal podem ser tratadas de modo eficiente, pois o índice pode ser reestruturado em tempo logarítmico

Consultas em árvores B+

- Encontre todos os registros com um valor de chave de busca k
  - Comece com o nó raiz
    - Examine o nó em busca do menor valor de chave de busca $> k$
    - Se esse valor existir, considere que é K_j. Depois siga $\boldsymbol{P_{j-1}}$ e explore o nó filho
    - Caso contrário, $k \geq \boldsymbol{K_{m-1}}$, onde existem m ponteiros no nó, siga $\boldsymbol{P_m}$ e explore o nó filho
  - Se o nó explorado não for um nó folha, repita o procedimento sobre o nó e siga o ponteiro correspondente
  - Por fim, ao alcançar um nó folha. Se, para algum i, a chave $k = \boldsymbol{K_i}$, siga o ponteiro $\boldsymbol{P_i}$ até o registro ou balde desejado
    - Se não, não há um registro com valor de chave de busca k

Consultas em árvores B+

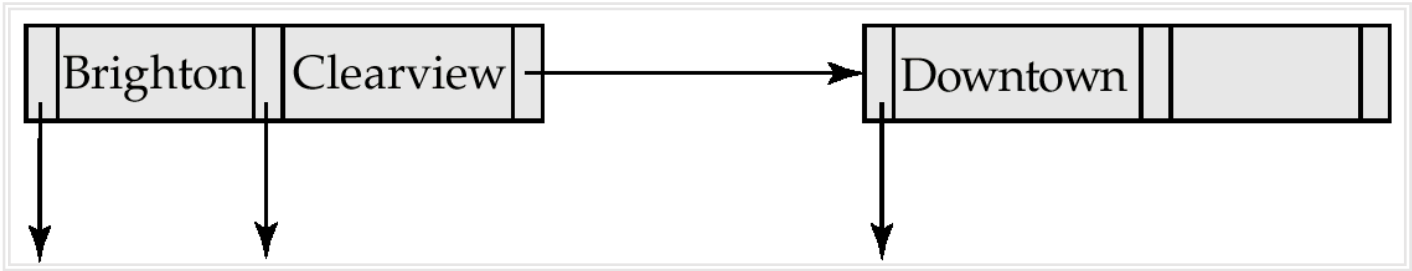
- No processamento de uma consulta, um caminho é atravessado na árvore da raiz até algum nó folha
- Se houver K valores de chave de busca no arquivo, o caminho não será maior do que $\left\lceil \boldsymbol{log}_{\lceil \frac{k}{2} \rceil}(\boldsymbol{n}) \right\rceil$
- Um nó geralmente tem o mesmo tamanho de um bloco de disco, normalmente 4 kilobytes, e n normalmente fica em torno de 100 (40 bytes por entrada de índice)
- Com 1 milhão de valores de chave de busca e $\boldsymbol{k} = 100$, no máximo, $\boldsymbol{log_{50}}(1.000.000) = \boldsymbol{4}$ nós são acessados em uma pesquisa

Atualizações em árvores B+: inserção

- Encontre o nó folha em que o valor da chave de busca apareceria
- Se o valor da chave de busca já estiver lá no nó folha, o registro é acrescentado ao arquivo e, se for preciso, um ponteiro é inserido no balde
- Se o valor da chave de busca não estive lá, então acrescente o registro ao arquivo principal e crie um balde, se for preciso. Depois:
  - Se houver espaço no nó folha, insira o par (valor de chave, ponteiro) no nó folha
  - Caso contrário, divida o nó (junto com a nova entrada (valor de chave, ponteiro))

Divisão de um nó

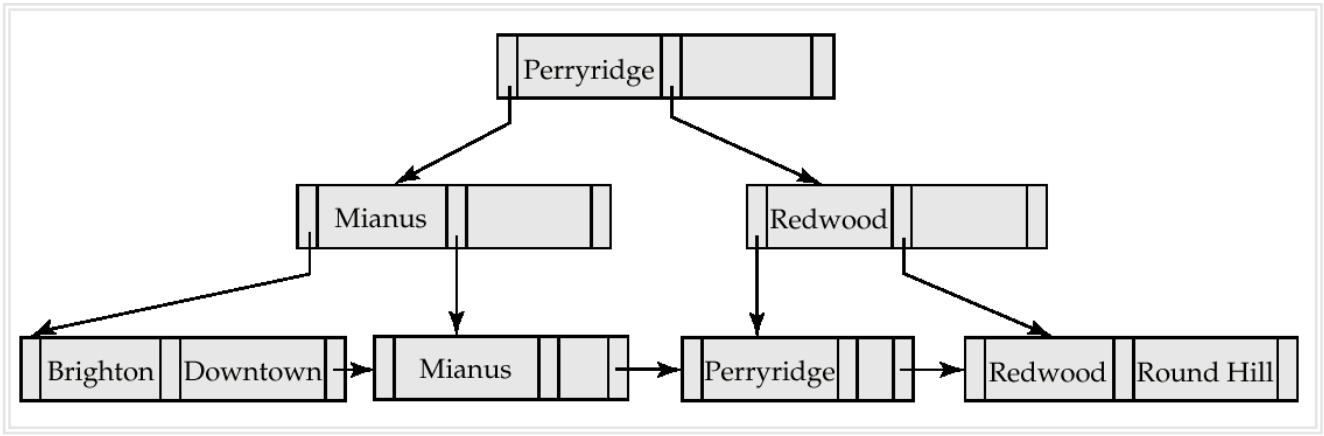
- Apanhe os n pares (valor de chave, ponteiro) (incluindo aquele sendo inserido) em ordem classificada
- Coloque os $\left\lceil \frac{n}{2} \right\rceil$ primeiros no nó original, e o restante em um novo nó
- Considere que o novo nó seja p, e considere que k é o menor valor de chave em p
- Insira (k, p) no pai do nó sendo dividido. Se o pai estiver cheio, divida-o e propague a divisão para cima
- A divisão de nós prossegue para cima até que um nó que não esteja cheio seja encontrado. No pior caso, o nó raiz pode ser dividido, aumentando a altura da árvore em um nível

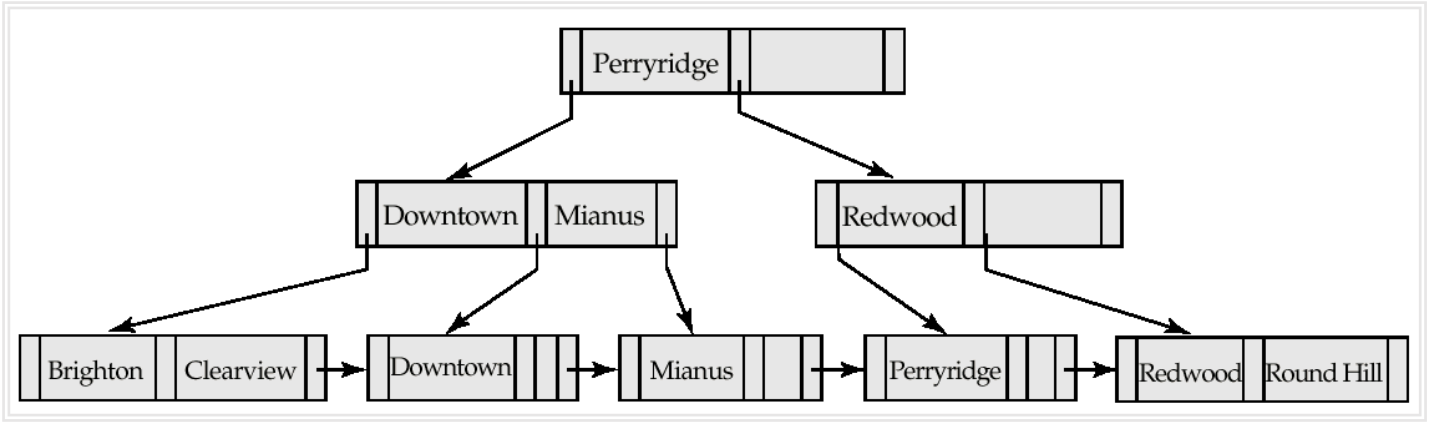


Resultado da divisão do nó contendo Brighton e Downtown na inserção de Clearview

Atualizações em árvores B+: inserção (cont.)

- Árvore B+ antes e depois da inserção de "Clearview"





Exclusão em árvores B+

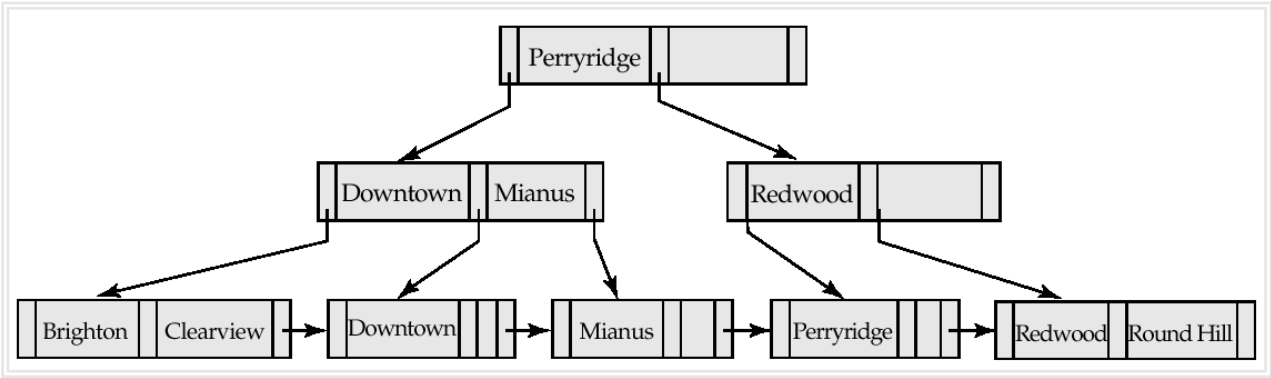
- Encontre o registro a ser excluído e remova-o do arquivo principal e do balde (se estiver presente)
- Remova (valor chave de busca, ponteiro) do nó folha se não houver balde ou se o balde tiver ficado vazio
- Se o nó tiver muito poucas entradas devido à remoção, e as entradas no nó e um irmão couberem em um único nó, então
  - Insira todos os valores de chave de busca nos dois nós em um único nó (aquele à esquerda) e exclua o outro nó
  - Exclua o par ($\boldsymbol{K_{i-1}}$, $\boldsymbol{P_i}$), onde $\boldsymbol{P_i}$ é o ponteiro para o nó excluído, a partir de seu pai, recursivamente o procedimento anterior recursivamente

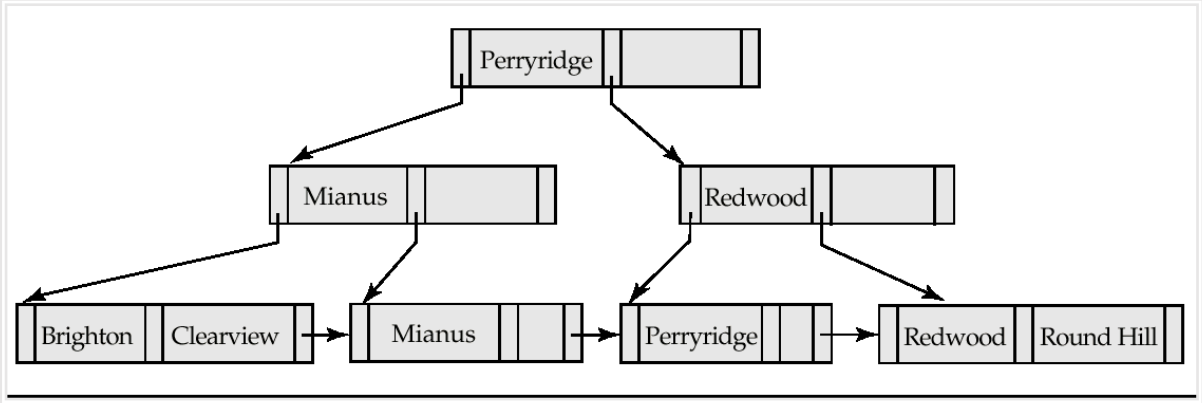
Consolidações em exclusão em árvores B+

- Caso contrário, se o nó tiver muito poucas entradas devido à remoção, e as entradas no nó e um irmão couberem em um único nó, então
  - Redistribua os ponteiros entre o nó e um irmão, de modo que ambos tenham mais do que o número mínimo de entradas
  - Atualize o valor de chave de busca correspondente no pai do nó
- As exclusões de nó podem ser propagadas para cima, até que seja encontrado um nó contendo $\left\lceil \frac{n}{2} \right\rceil$ ou mais ponteiros
  - Se o nó raiz tiver apenas um ponteiro após a exclusão, ele é excluído e o único filho se torna a raiz

Exemplos de exclusão em árvore B+

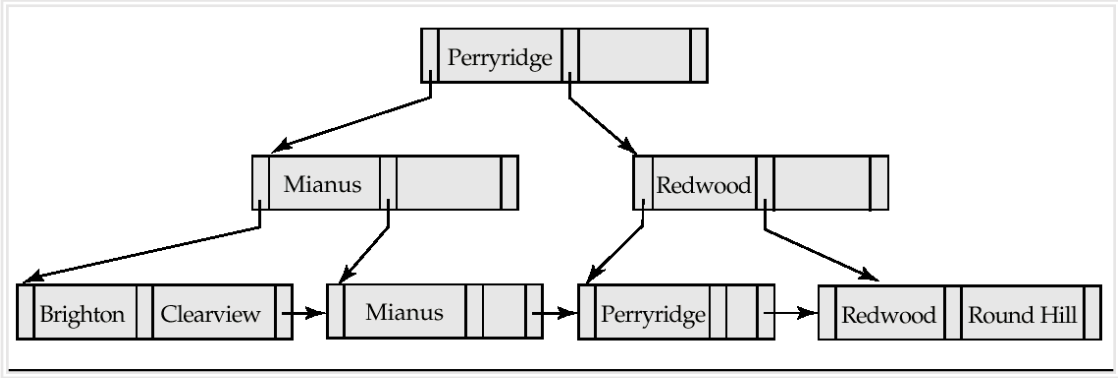
- A remoção do nó folha contendo “Downtown” não resultou em seu pai tendo muito poucos ponteiros.
  - Assim, as exclusões em cascata pararam com o pai do nó folha excluído

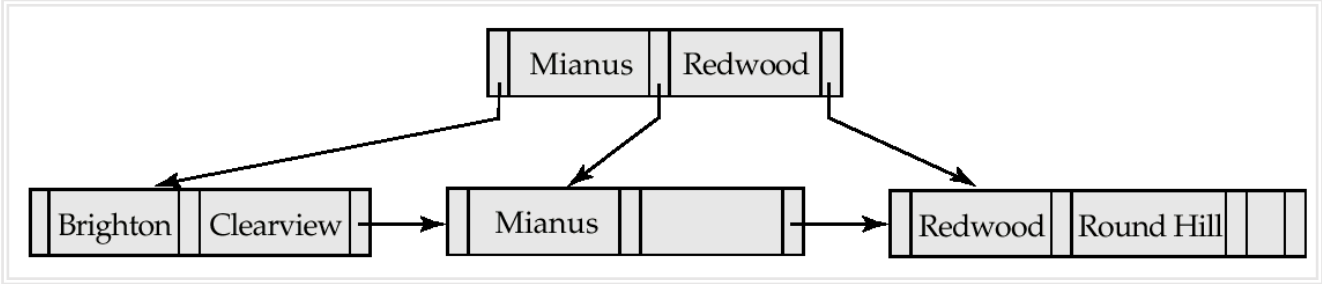




Exemplos de exclusão em árvore B+ (mesclado ao irmão da esquerda)

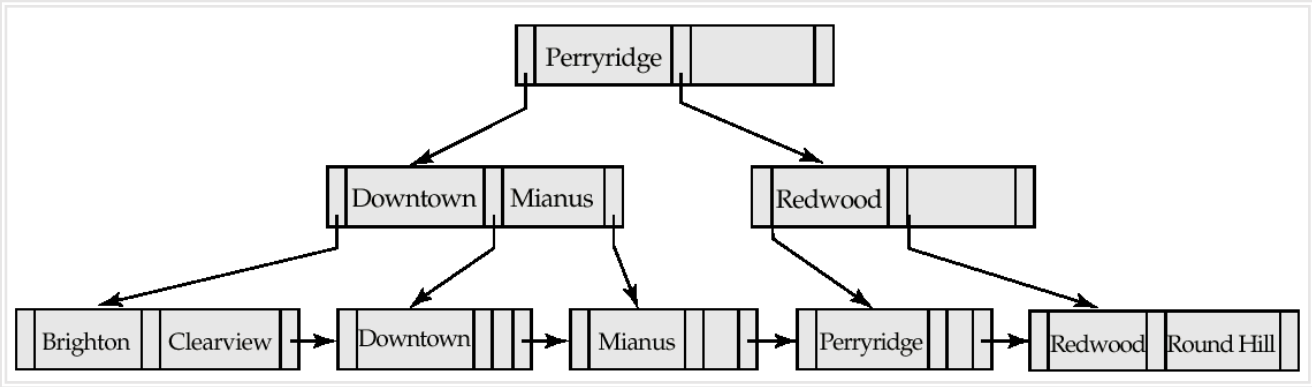
- Nó com "Perryridge" torna-se menos que cheio (na verdade, vazio, nesse caso especial) e mesclado com seu irmão
  - Como resultado, o pai do nó "Perryridge" se torna menos que cheio, e foi mesclado com seu irmão (uma entrada foi excluída do seu pai)
- O nó raiz, então, tinha apenas um filho, e foi excluído e seu filho se tornou o novo nó raiz

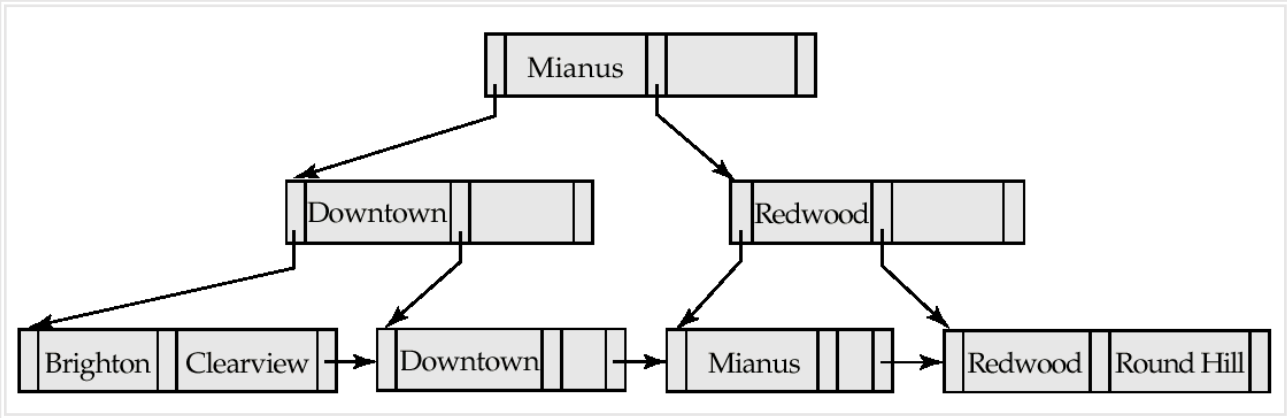




Exemplos de exclusão em árvore B+ (empréstimo do irmão a esquerda)

- Antes e depois da exclusão de "Perryridge"
  - Pai da folha contendo Perryridge tornou-se menos que cheio, e pediu emprestado um ponteiro do seu irmão da esquerda
  - Valor de chave de busca no pai do pai muda como resultado





Hash

Hashing estático

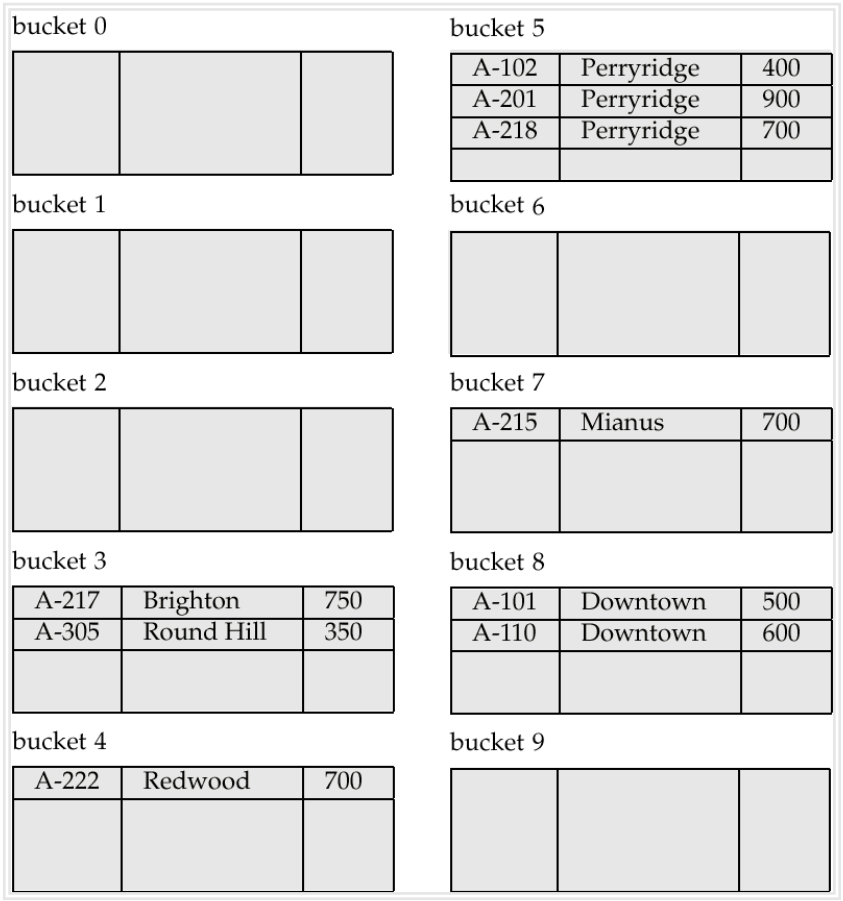
- Um balde é uma unidade de armazenamento contendo um ou mais registros (um balde normalmente é um bloco de disco)
- Em uma organização de arquivo de hash, obtemos o balde de um registro diretamente por seu valor de chave de busca, usando uma função de hash
- A função de hash h é uma função do conjunto de todos os valores de chave de busca K para o conjunto de todos os endereços de balde B
- A função de hash é usada para localizar registros para acesso, inserção e também exclusão
- Os registros com diferentes valores de chave de busca podem ser mapeados para o mesmo balde
  - Assim, o balde inteiro precisa ser pesquisado sequencialmente, para localizar um registro

Exemplo de organização de arquivo de hash

- Organização de arquivo de hash do arquivo conta, usando nome-agência como chave
- Existem 10 baldes
- A representação binária do i-ésimo caractere é considerada como o inteiro i
- A função de hash retorna a soma das representações binárias dos caracteres módulo 10
  - Por exemplo:
    - h(Perryridge) = 5
    - h(Round Hill) = 3
    - h(Brighton) = 3

Exemplo de organização de arquivo de hash

- Organização de arquivo de hash do arquivo conta, usando nome-agência como chave (detalhes no slide anterior)



Funções de hash

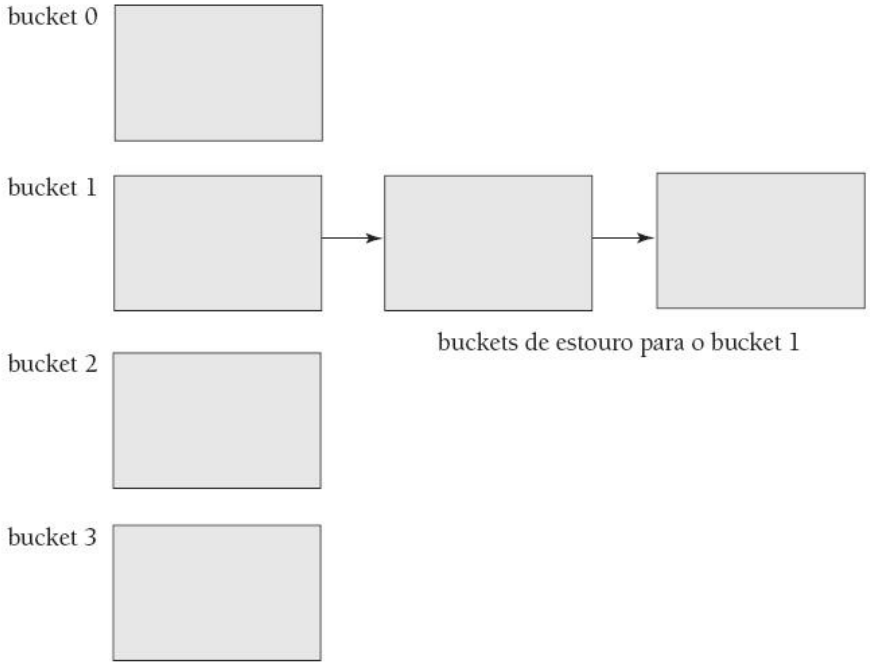
- A pior função de hash mapeia todos os valores de chave de busca para o mesmo balde
  - isso torna o tempo de acesso proporcional ao número de valores de chave de busca no arquivo
- Uma função de hash ideal é uniforme, ou seja, cada balde recebe o mesmo número de valores de chave de busca do conjunto de todos os valores possíveis
- As funções de hash típicas realizam seu cálculo sobre a representação binária interna da chave de busca
  - Por exemplo, para uma chave de busca de string, as representações binárias de todos os caracteres na string poderiam ser somadas e a soma módulo número de baldes poderia ser retornada

Tratamento de estouros de balde

- O estouro de balde pode ocorrer devido a
  - Baldes insuficientes
  - Distorção na distribuição de registros. Isso pode ocorrer por dois motivos:
    - Vários registros possuem o mesmo valor de chave de busca
    - A função de hash escolhida produz a distribuição não uniforme de valores de chave
- Embora a probabilidade de estouro de balde possa ser reduzida, ela não pode ser eliminada
  - Neste caso, faz-se uso de baldes de estouro

Tratamento de estouros de balde (cont.)

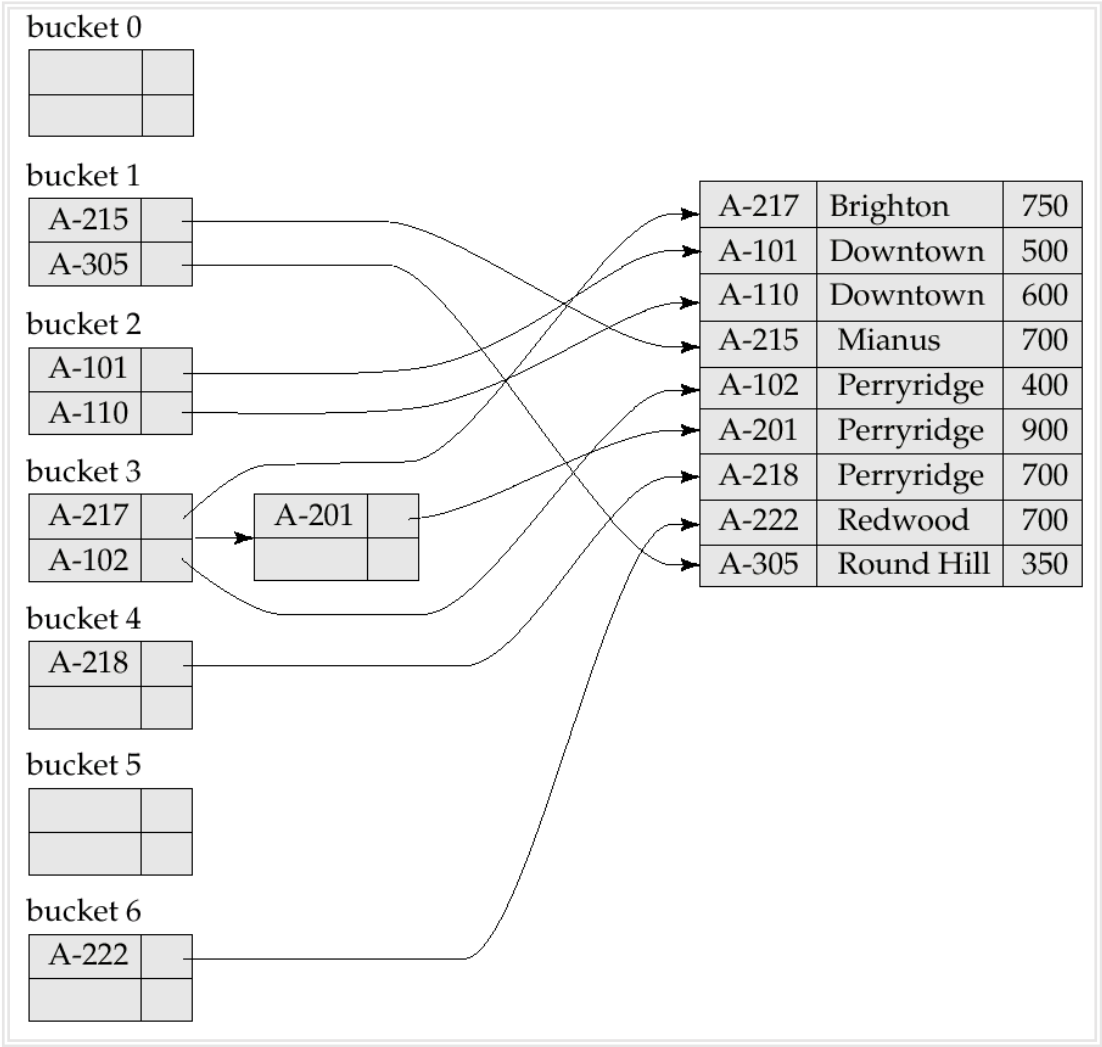
- Encadeamento de estouro – os baldes de estouro de determinado balde são encadeados em uma lista interligada
- O esquema apresentado é chamado de hashing fechado
  - Uma alternativa, chamada hashing aberto, que não usa baldes de estouro, não é apropriada para aplicações de banco de dados



Índices de hash

- O hashing pode ser usado não apenas para organização de arquivos, mas também para a criação de estrutura de índice
- Um índice de hash organiza as chaves de busca, com seus ponteiros de registro associados, em uma estrutura de arquivo de hash
- Estritamente falando, os índices de hash sempre são índices secundários
  - Se o próprio arquivo for organizado usando o hashing, um índice de hash primário separado sobre ele, usando a mesma chave de busca, é desnecessário
  - Porém, usamos o termo índice de hash para se referir às estruturas de índice secundárias e arquivos organizados em hash

Exemplo de índice de hash



Deficiências do hashing estático

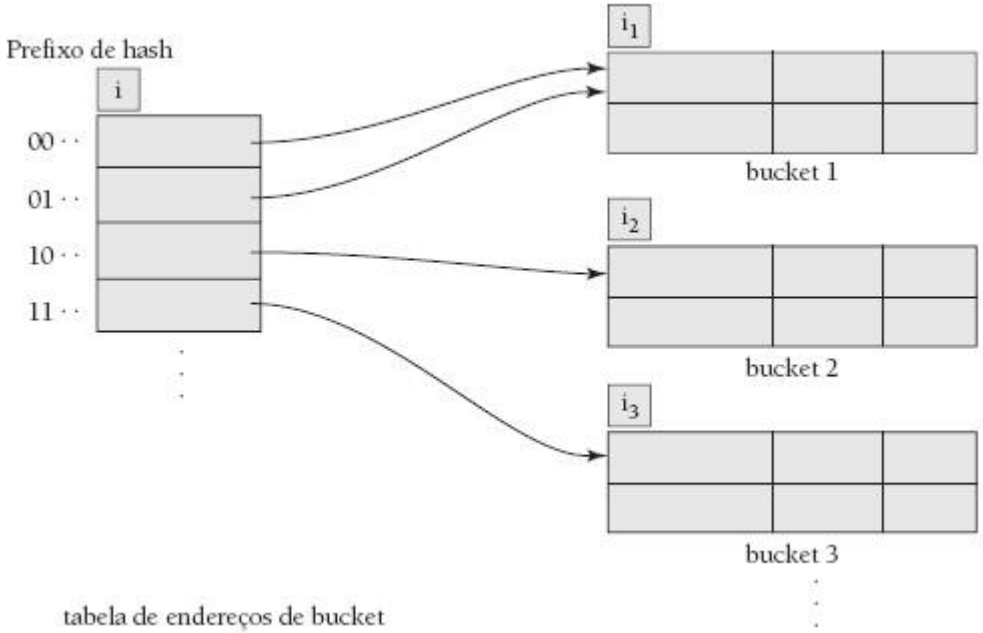
- No hashing estático, a função h mapeia os valores de chave de busca a um conjunto fixo B de endereços de balde
  - Os bancos de dados crescem como tempo
  - Se o número inicial de baldes for muito pequeno, o desempenho degradará devido a muitos estouros
  - Se o tamanho do arquivo em algum ponto no futuro for antecipado e o número de baldes for alocado de acordo, uma quantidade de espaço significativa será desperdiçada inicialmente
  - Se o banco de dados encurtar, novamente o espaço será desperdiçado
  - Uma opção é a reorganização periódica do arquivo com uma nova função de hash, mas isso é muito dispendioso
- Esses problemas podem ser evitados por meio de técnicas que permitem que o número de baldes seja modificado dinamicamente

Hashing dinâmico

- Bom para banco de dados que aumenta e diminui de tamanho
- Permite que a função de hash seja modificada dinamicamente
- Hashing extensível - uma forma de hashing dinâmico
  - Função de hash gera valores por um intervalo grande - normalmente, inteiros de b bits, com b = 32.
  - A qualquer momento, usa apenas um prefixo da função de hash para indexar em uma tabela de endereços de balde
  - Considere que o tamanho do prefixo seja i bits, $0 \leq i \leq 32$
  - Tamanho da tabela de endereços de balde $\mathbf{2}^{\boldsymbol{i}}$. Inicialmente, $i = 0$
  - O valor de i aumenta e diminui à medida que o tamanho do banco de dados aumenta e diminui
  - Várias entradas na tabela de endereços de balde podem apontar para um balde
  - Assim, o número real de baldes é $< \mathbf{2}^{\boldsymbol{i}}$
    - O número de baldes também muda dinamicamente, devido à união e divisão de baldes

Estrutura geral do hash extensível

- Nessa estrutura, $\boldsymbol{i_2 = i_3 = i}$, enquanto $\boldsymbol{i_1 = i - 1}$



Uso da estrutura de hash extensível

- Cada balde j armazena um valor $\boldsymbol{i_j}$
  - todas as entradas que apontam para o mesmo balde têm os mesmos valores nos primeiros $\boldsymbol{i_j}$ bits
- Para localizar o balde contendo a chave de busca $\boldsymbol{K_j}$:
  - 1. Calcule $h(\boldsymbol{K_j}) = X$
  - 2. Use os primeiros i bits de alta ordem de X como um deslocamento na tabela de endereços de balde, e siga o ponteiro para o balde apropriado
- Para inserir um registro com valor de chave de busca $\boldsymbol{K_j}$
  - siga o mesmo procedimento da pesquisa e localize o balde j
  - Se houver espaço no balde j, insira o registro no balde
  - Senão, o balde precisa ser dividido e a inserção tentada novamente
    - Baldes de estouro usados em alguns casos

Atualizações na estrutura de hash extensível

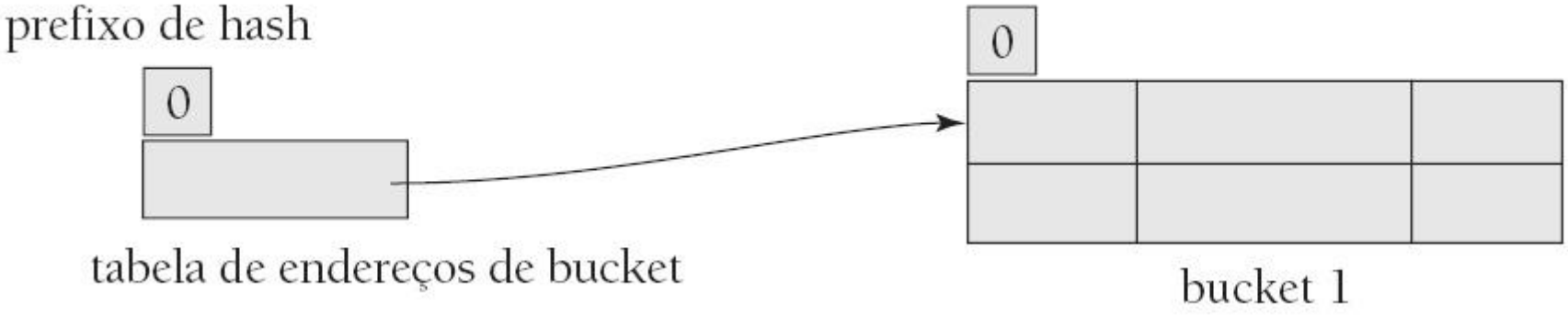
- Se $i > i_j$ (mais de um ponteiro para o balde j)
  - aloque um novo balde z, e defina i_j e i_z como o antigo i_{j+1}
  - faça com que a segunda metade das entradas da tabela de endereços de balde apontando para j apontem para z
  - remova e insira novamente cada registro no balde j
  - recalcule o novo balde para K_j e insira o registro no balde (outra divisão é necessária se o balde ainda estiver cheio)
- Se $i = i_j$ (apenas um ponteiro para o balde j)
  - incremente i e dobre o tamanho da tabela de endereços de balde
  - substitua cada entrada na tabela por duas entradas que apontam para o mesmo balde
  - recalcule a nova entrada da tabela de endereços de balde para K_j
    Agora, $i > i_j$; portanto, use o primeiro caso, acima

Atualizações na estrutura de hash extensível (cont.)

- Ao inserir um valor, se o balde estiver cheio após várias divisões (ou seja, se i alcançar algum limite b)
  - crie um balde de estouro em vez de dividir mais a tabela de entradas de balde
- Para excluir um valor de chave
  - localize-o em seu balde e remova-o
  - O próprio balde pode ser removido se ficar vazio (com as atualizações apropriadas na tabela de endereços de balde)
  - A união de baldes pode ser feita (só pode unir com um balde "companheiro" tendo o mesmo valor de $\boldsymbol{i_j}$ e o mesmo prefixo $\boldsymbol{i_{j-1}}$, se estiver presente)
  - Também é possível diminuir o tamanho da tabela de endereços de balde
    - Nota: diminuir o tamanho da tabela de endereços de balde é uma operação dispendiosa, e só deve ser feito se o número de baldes se tornar muito menor que o tamanho da tabela

Uso da estrutura de hash extensível: Exemplo

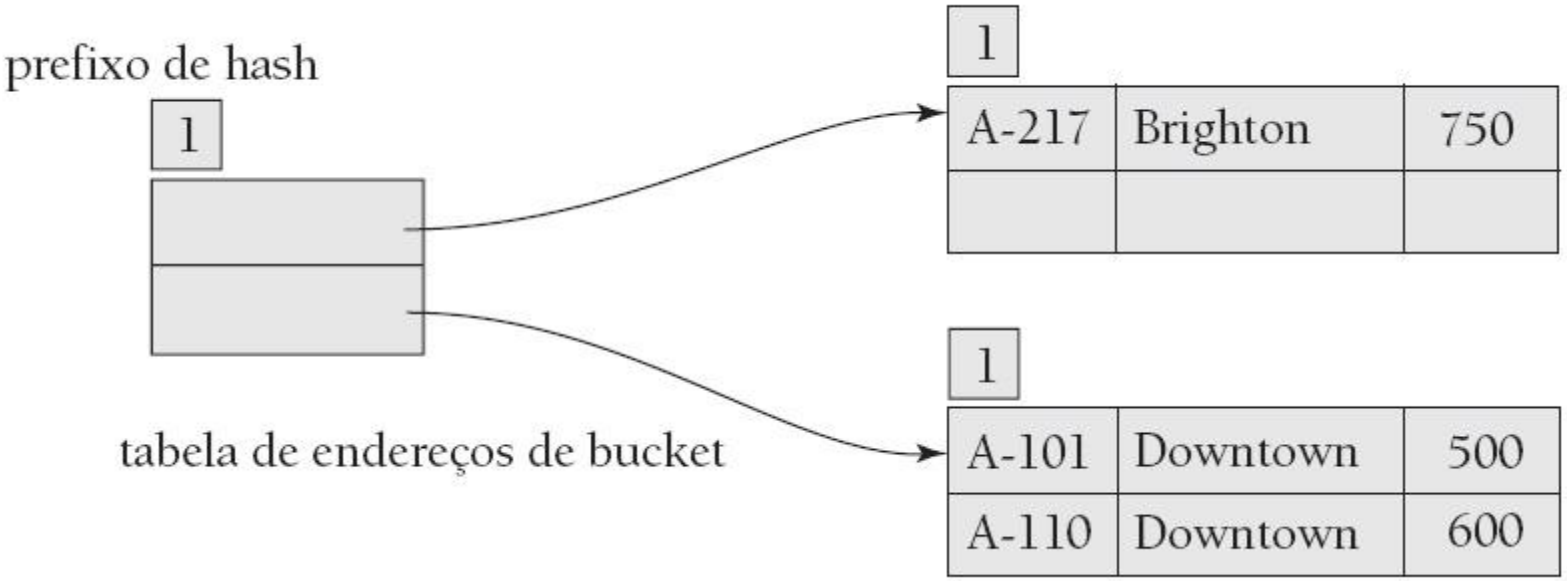
nome_agência	h(*nome_agência*)
Brighton	0010 1101 1111 1011 0010 1100 0011 0000
Downtown	1010 0011 1010 0000 1100 0110 1001 1111
Mianus	1100 0111 1110 1101 1011 1111 0011 1010
Perryridge	1111 0001 0010 0100 1001 0011 0110 1101
Redwood	0011 0101 1010 0110 1100 1001 1110 1011
Round Hill	1101 1000 0011 1111 1001 1100 0000 0001



Estrutura de hash inicial, tamanho de balde = 2

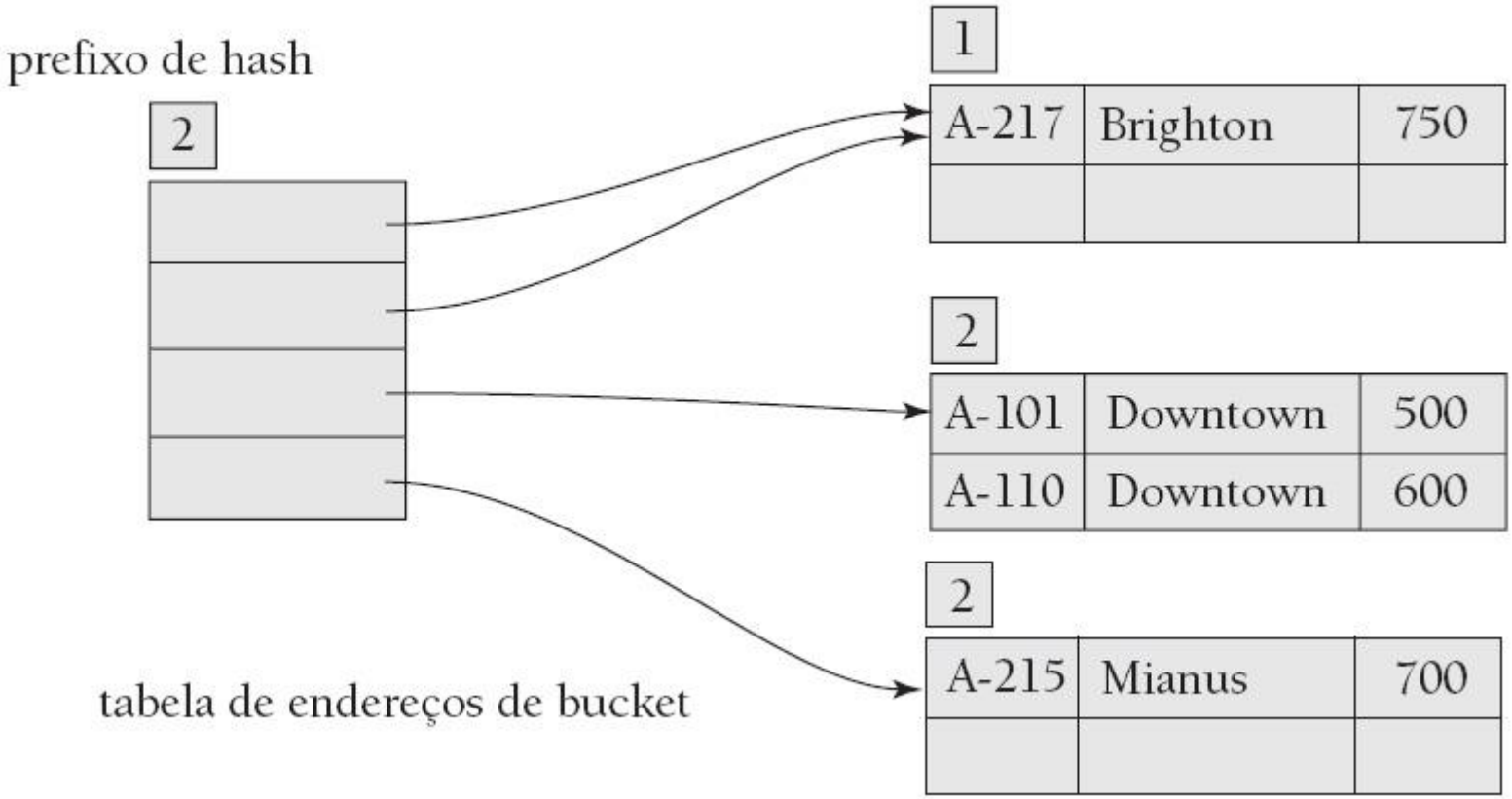
Exemplo (cont.)

- Estrutura de hash após inserção de um registro de Brighton e dois de Downtown



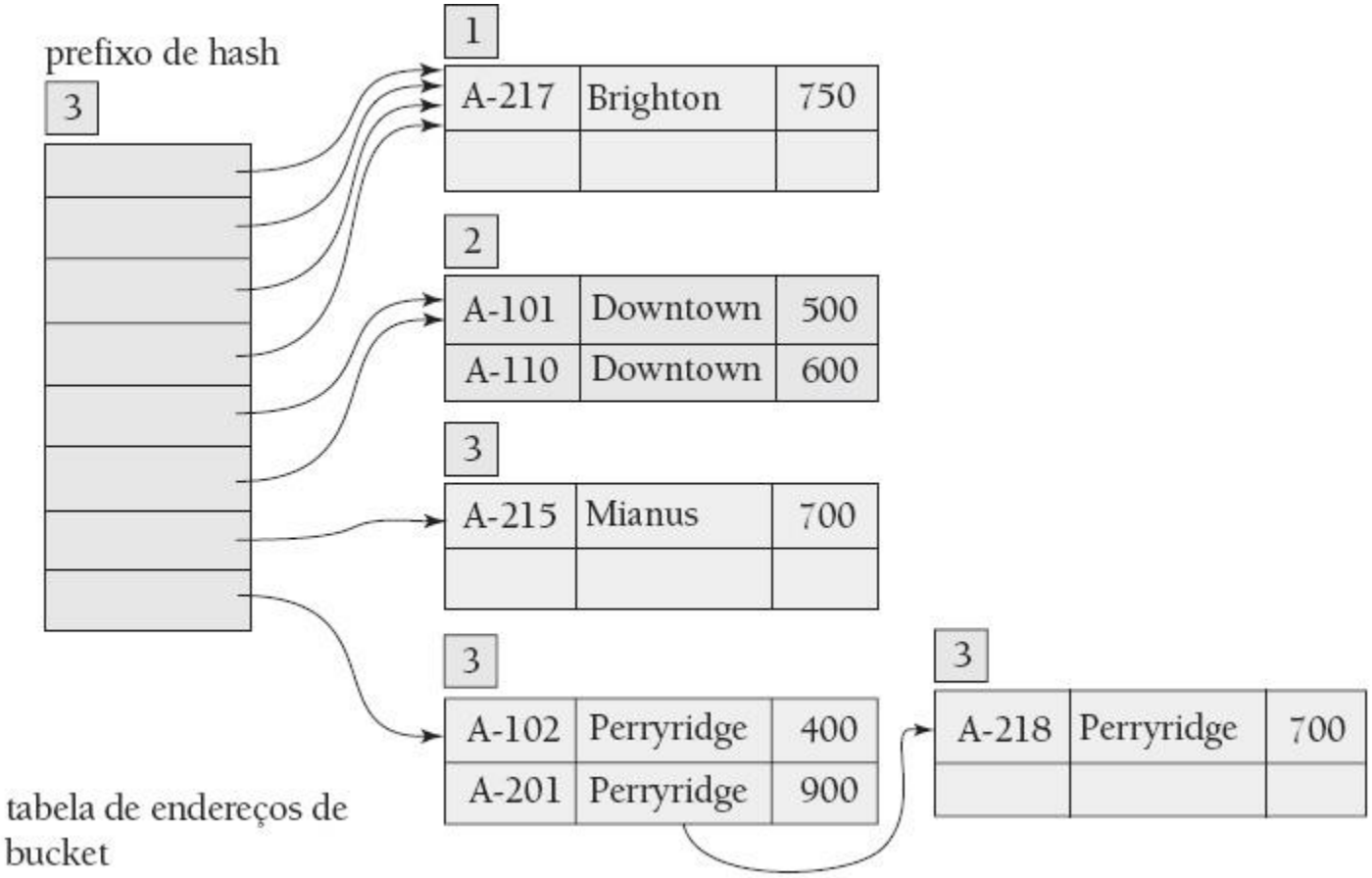
Exemplo (cont.)

- Estrutura de hash após inserção do registro de Mianus



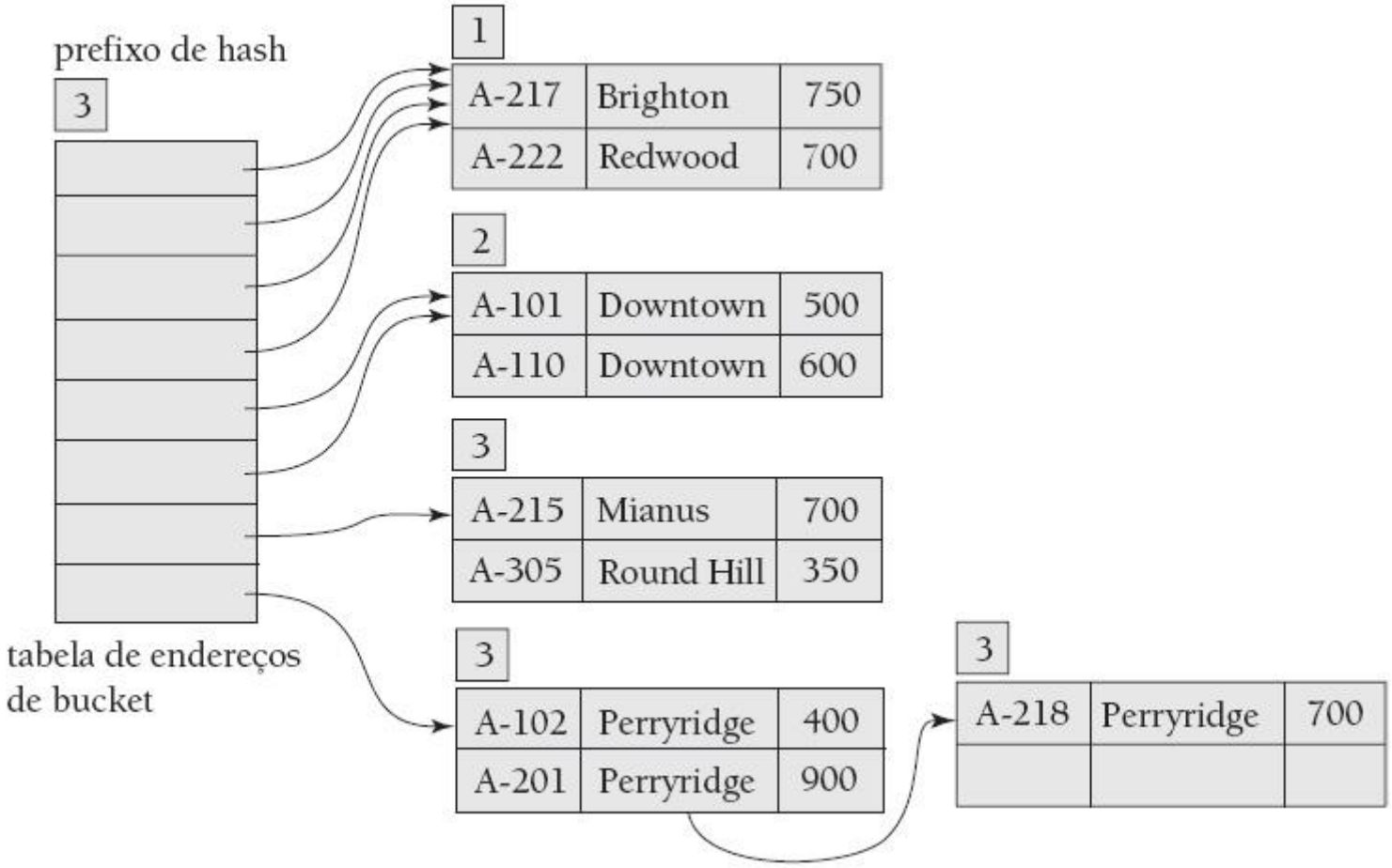
Exemplo (cont.)

- Estrutura de hash após inserção de três registros de Perryridge



Exemplo (cont.)

- Estrutura de hash após inserção de registros de Redwood e Round Hill



Hashing extensível versus outros esquemas

- Benefícios do hashing extensível:
  - Desempenho do hash não diminui com o crescimento do arquivo
  - Sobrecarga de espaço mínima
- Desvantagens do hashing extensível:
  - Nível extra de indireção para encontrar registro desejado
  - A própria tabela de endereços de balde pode se tornar muito grande (maior que a memória)
    - Precisa de uma estrutura de árvore para localizar o registro desejado na estrutura!
  - Mudar o tamanho da tabela de endereços de balde é uma operação dispendiosa
- Hashing linear é um mecanismo alternativo, que evita essas desvantagens ao custo possível de mais estouros de balde

Comparação entre indexação ordenada e hashing

- Custo da reorganização periódica
- Frequência relativa de inserções e exclusões
- É desejável otimizar o tempo de acesso médio em detrimento do tempo de acesso no pior caso?
- Tipo esperado das consultas:
  - O hashing geralmente é melhor na recuperação de registros com um valor especificado da chave
  - Se as consultas por intervalo forem comuns, os índices ordenados são preferíveis

Definição de índice em SQL

- Crie um índice
  create index <nome-índice> on <nome-relação> (<lista-atributos>)
  - Por exemplo: create index emp_pnome_unome on empregado(pnome, unome)
- Use create unique index para especificar indiretamente e impor a condição em que a chave de busca é uma chave candidata
  - Não é realmente necessário se a restrição de integridade unique da SQL for admitida
- Para remover um índice
  - drop index <nome-índice>

Acesso de chave múltipla

- Use índices múltiplos para certos tipos de consultas
- Exemplo:

```
select pnome
from empregado
where dno = 5 and salario = 30000
```

- Estratégias possíveis para processar consulta usando índices sobre atributos isolados:
  - 1. Use índice sobre dno para encontrar dno = 5
    - teste contas com salario = 30000
  - 2. Use índice sobre salario para encontrar contas com salario = 30000
    - Teste dno = 5
  - 3. Use índice dno para encontrar ponteiros para todos os registros pertencentes à ao departamento 5. De modo semelhante, use índice sobre salario = 30000 Apanhe a interseção dos dois conjuntos de ponteiros obtidos

Índices sobre atributos múltiplos

- Suponha que temos um índice sobre a chave combinada (dno, salario)
- Use a cláusula

  where dno = 5 and salario = 30000
  - o índice sobre a chave de busca combinada apanhará apenas registros que satisfazem as duas condições
  - O uso de índices separados é menos eficiente - podemos apanhar muitos registros (ou ponteiros) que satisfazem apenas uma das condições
- Também pode tratar de modo eficiente

  where dno = 5 and salario < 30000
- Mas não pode tratar de modo eficiente

  where dno <= 5 and salario = 30000
  - Pode trazer muitos registros que satisfaçam a primeira restrição, mas não a segunda condição

Índices de mapa de bits

- Índices de mapa de bits são um tipo especial de índice projetado para a consulta eficiente sobre múltiplas faixas
- Os registros em uma relação são considerados como sendo numerados sequencialmente a partir do 0
  - Dado um número n, precisa ser fácil apanhar o registro n
    - Particularmente fácil se os registros forem de tamanho fixo
- Aplicável sobre atributos que assumem uma quantidade relativamente pequena de valores distintos
  - Por exemplo: gênero, país, estado, ...
  - Por exemplo: nível-receita (entrada dividida em um pequeno número de níveis, como 0-9999, 10000-19999, 20000-50000, 50000-infinito)
- Um mapa de bits é simplesmente um array de bits

Índices de mapa de bits (cont.)

- Em sua forma mais simples, um índice de mapa de bits sobre um atributo tem um mapa de bits para cada valor do atributo
  - O mapa de bits tem tantos bits quantos registros
  - Em um mapa de bits para o valor v, o bit para um registro é 1 se o registro tiver o valor v para o atributo, e é 0 em caso contrário

número de registro	*nome*	*sexo*	*endereço*	*nível_renda*
0	John	m	Perryridge	L1
1	Diana	f	Brooklyn	L2
2	Mary	f	Jonestown	L1
3	Peter	m	Brooklyn	L4
4	Kathy	f	Perryridge	L3

Mapas de bits para *sexo*

m	10010
f	01101

Mapas de bits para *nível_renda*

L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

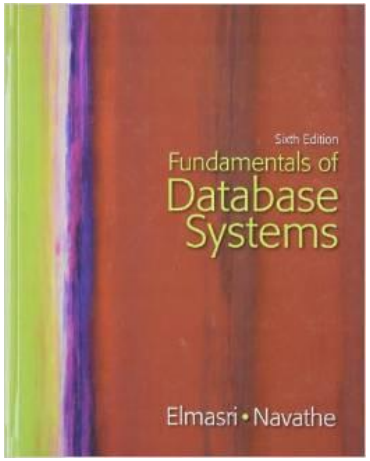
Índices de mapa de bits (cont.)

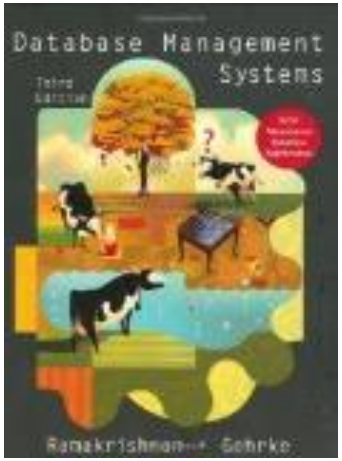
- Os índices de mapa de bits são úteis para consultas sobre múltiplos atributos
  - não é útil particularmente para consultas de atributo único
- As consultas são respondidas usando operações de mapa de bits
  - Interseção (and)
  - União (or)
  - Complementação (not)
- Cada operação usa dois mapas de bits do mesmo tamanho e aplica a operação sobre bits correspondentes para obter o mapa de bits do resultado
  - Por exemplo:
    (100110 AND 110011) = 100010
    (100110 OR 110011) = 110111
    NOT 100110 = 011001
  - Homens com nível de receita N1:
    10010 AND 10100 = 10000
    - Pode então apanhar as tuplas exigidas
    - Contar número de tuplas que combinam é ainda mais rápido

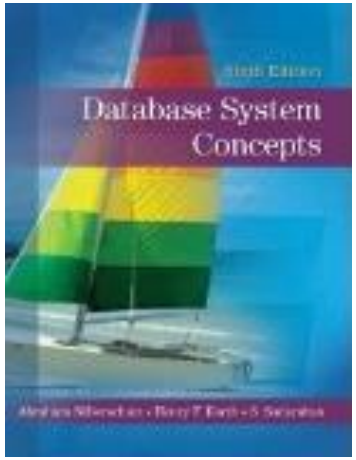
Índices de mapa de bits (cont.)

- Os índices de mapa de bits geralmente são muito pequenos em comparação com o tamanho da relação
  - Por exemplo, se o registro tiver 100 bytes, o espaço para um único mapa de bits é 1/800 do espaço usado pela relação.
    - Se o número de valores de atributo distintos for 8, o mapa de bits tem apenas 1% do tamanho da relação
- A exclusão precisa ser tratada corretamente
  - Mapa de bits de existência para anotar se existe um registro válido em um local de registro

Referências







A maioria dos slides apresenta exemplos do livro Database System Concepts