Bancos de Dados 1
Indexação

Prof. Altigran Soares da Silva
IComp/UFAM

V2018.1

Técnicas de Indexação

- Índice : Estrutura auxiliar que torna a busca baseada em alguns campos, chamados **campos de indexação**, mais eficiente
- A indexação independe da organização dos registros no arquivo de dados
- São armazenados em arquivos auxiliares chamados **arquivos de índice**

Técnicas de Indexação

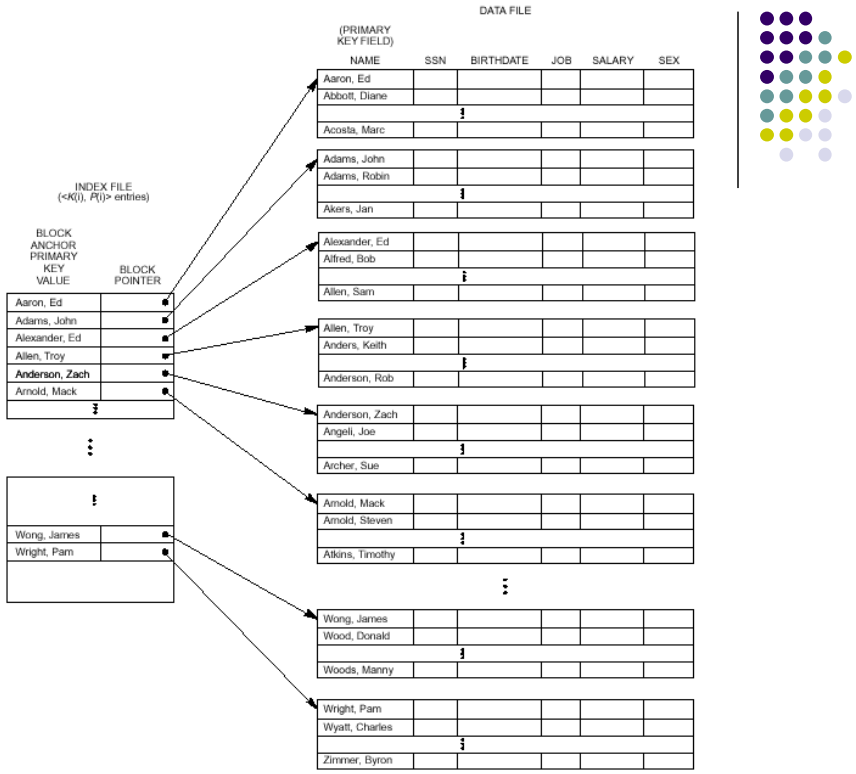
- Arquivos de índices: construídos sobre campos de indexação (CI) escolhidos entre os campos dos registros de um arquivo
- Armazenam os valores do CI associando ao endereço do registro onde o valor ocorre
- São arquivo ordenados pelo CI de tal forma que é possível buscar no índice usando busca binária

Tipos de Índices

- **Primário**: CI é uma chave e o arquivo de dados é ordenado por este campo
- **De Cluster**: CI não é uma chave mas ordena o arquivo de dados
- **Secundário**: CI é um campo qualquer
- So há um índice primário ou índice de cluster
- Podem haver vários índices secundários

Índices Primários

- Registros da forma <K_i,P_i>,
  - K_i: valor do campo de indexação – chave
  - P_i: endereço de um bloco do arquivo de dados
- Arquivo de registros fixos ordenados por K_i
- Existe um registro de índice para cada bloco do arquivo de dados – índice esparso



Índices Primários

- Ki é a chave do registro âncora do bloco
- Registros menores, menos registros que o arquivo de dados. Usa menos blocos.
- Boa parte pode estar em memória principal;
- Operação de busca envolve:
  - Busca binária no índice
  - Carga do bloco do arquivo de dados
  - Busca binária no bloco em memória

Índice Primário – Exemplo

- Parâmetros Gerais:
  - Tamanho do bloco
    - B = 1024 bytes;
  - Arquivos com registros de tamanho fixo;
  - Alocação Não-Espalhada;

Índice Primário – Exemplo

- Parâmetros do arquivo de dados:
  - Número de registros
    - r = 30.000 registros
  - Tamanho dos registros
    - R = 100 bytes
  - Fator de bloco:
    - bfr = ⌊(B/R)⌋ = ⌊1024/100⌋ = 10 registros por bloco;
  - Número de blocos:
    - b = ⌈r/bfr⌉ = ⌈30.000/10⌉ = 3.000 blocos;

Índice Primário – Exemplo

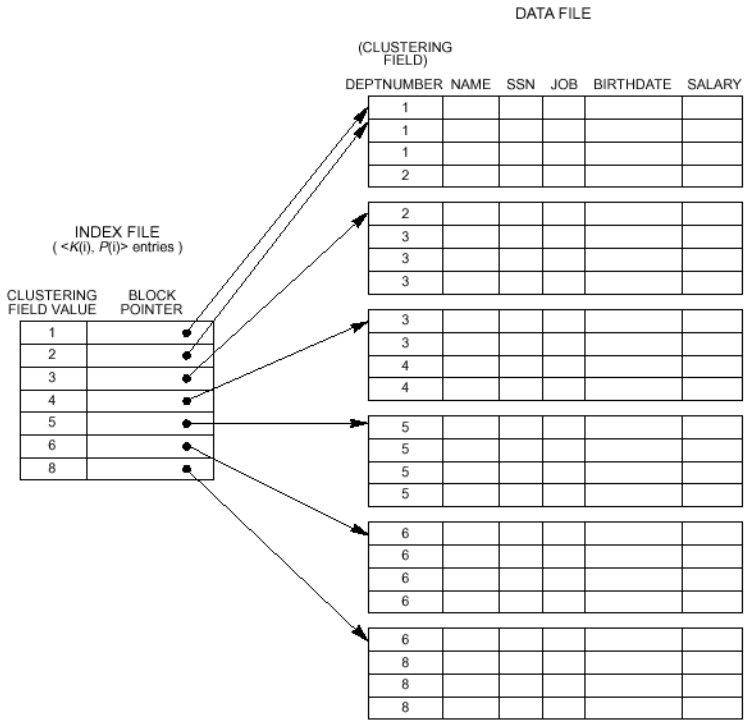
- Parâmetros do arquivo de índices:
  - Tamanho do Registro de índice
    - Ri = 15 bytes ⇒ chave 9 bytes + apontador 6 bytes
  - Fator de bloco:
    - bfri = ⌊(B/Ri)⌋ = ⌊1024/15⌋ = 68 registros por bloco
  - Número de registros:
    - ri = 3.000 (um para cada bloco do arq. de dados)
  - Número de blocos:
    - bi = ⌈ri/bfri⌉ = ⌈3.000/68⌉ = 45 blocos;

Índice Primário – Exemplo

- Sem índice:
  - Uma busca binária pode ser feita com
    - $\lceil \log_2 b \rceil = \lceil \log_2 3000 \rceil = 12$ acessos a blocos;
- Com índice:
  - Uma busca binária pode ser feita com
    - $\lceil \log_2 bi \rceil = \lceil \log_2 45 \rceil = 6$ acessos a blocos de índice
    - + 1 acesso ao bloco do arquivo de dados;

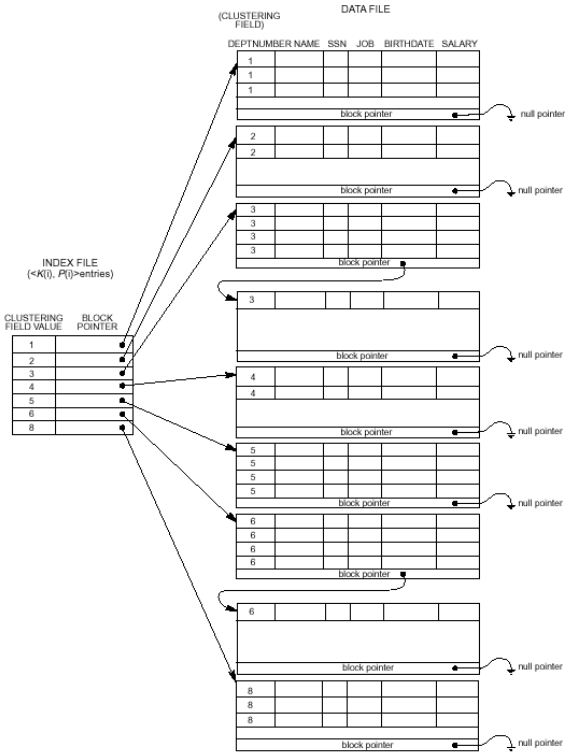
Índice de Cluster

- Campo de Cluster: Ordena o arquivo de dados, mas podem haver registros diferentes com o mesmo valor (não é uma chave);
- Índice de Cluster:
  - Arquivo ordenado com registros de dois campo:
    - campo de cluster e apontador de bloco.
  - Existe um registro de índice para cada valor distinto do campo de cluster no arquivo de dados
  - Índice Esparso



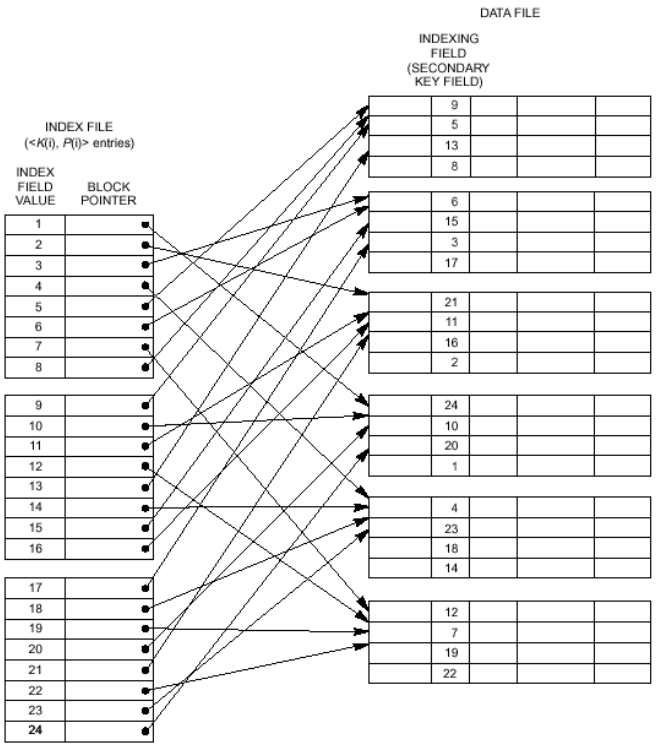
Índice de Cluster

- Para Facilitar remoções e inserções, pode ser usado um bloco de apontadores para cada valor diferente do campo de cluster, sendo encadeados blocos adicionais quando necessário.



Índices Secundários

- Campo de Indexação:
  - Não ordena o arquivo de dados
  - Pode ou não ser uma chave no arquivo de dados
- Índice secundário sobre chaves
  - Campo de indexação não ordena o arquivo de dados mas é uma chave;
  - Existe uma entrada de índice para cada registro no arquivo de dados (índice denso);
  - Provê maior ganho com relação ao acesso direto do arquivo de dados que seria sequencial;



Índice Secundário – Exemplo

- Parâmetros Gerais
  - Mesmos do exemplo anterior
- Parâmetros do Arquivo de Dados
  - Mesmos do exemplo anterior

Índice Secundário – Exemplo

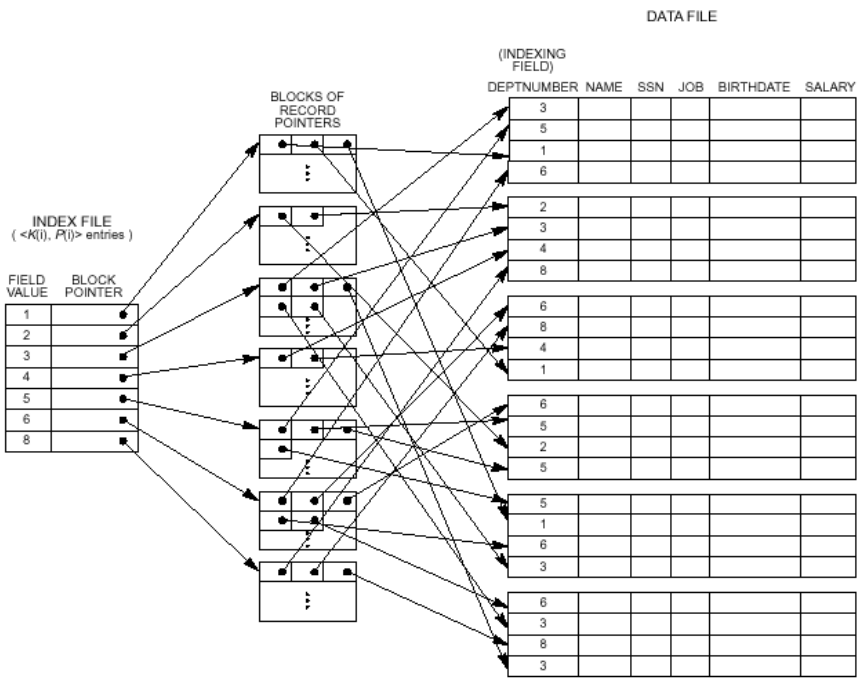
- Parâmetros do arquivo de índice
  - Tamanho do Registro de índice
    - Ri = 15 bytes ⇒ chave 9 bytes + apontador 6 bytes
  - Fator de bloco:
    - bfri = [(B/Ri)] = [1024/15] = 68 registros por bloco
  - Número de registros:
    - ri = 30.000 (um para cada registro do arq. de dados)
  - Número de blocos:
    - bi = [ri/bfri] = [30.000/68] = 442 blocos;

Índice Secundário – Exemplo

- Sem índice:
  - Busca sequencial com:
    - [b/2] = [3000/2] = 1500 acessos a blocos
- Com índice:
  - Busca binária sobre o índice com
  - $[\log_2 bi] = [\log_2 442] = 9$ acessos a bls. de índice
  - + 1 no arquivo de dados

Índices Secundários

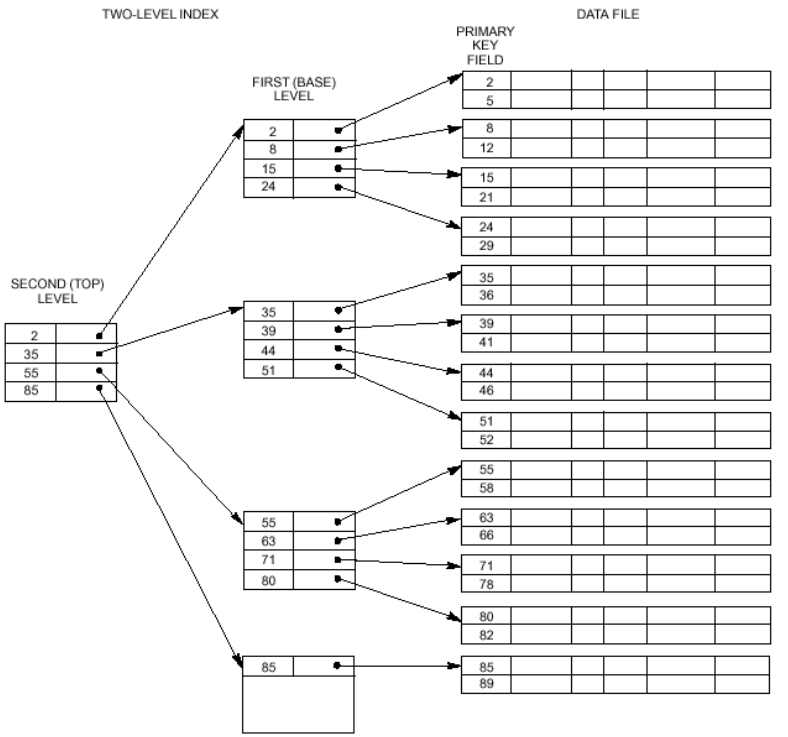
- Sobre um campo que não é chave
  - Opção 1: Uma entrada de índice para cada registro do arquivo de dados;
  - Opção 2: Vários apontadores em cada entrada do índice, um para cada posição do arquivo de dados onde o valor do campo de indexação ocorre
  - Opção 3: Cada entrada do índice aponta para um bloco que contém apontadores de blocos para o arquivo de dado



Índices Multi-nível

- Motivação: Reduzir o número de acessos a blocos de disco de $\log_2$ bi para $\log_{fo}$ bi;
- Fan-Out:
  - Fator de bloco do arquivo de índice: fo > 2

Nível	Registros	Fan-Out	Blocos
1	r1	fo	$\lceil r1/fo \rceil$
2	r2 = $\lceil r1/fo \rceil$	fo	$\lceil r1/fo^2 \rceil$
3	r3 = $\lceil r1/fo^2 \rceil$	fo	$\lceil r1/fo^3 \rceil$
...	...	...	...
t	r2 = $\lceil r1/fo^{t-1} \rceil$	fo	$\lceil r1/fo^t \rceil$



Índices Multi-nível

- O último nível ocorre quando um só bloco é usado: $[r1/fo^t] = 1$
- Assim, o número de níveis t é dado por:
  - $t = [\log_{fo} r1]$
- Pode ser de qualquer tipo: primário, de cluster ou secundário

Índices Multi-nível – Exemplo

- Fan-out: fo= bfr = Ri/B = 68
- Nível 1: b1 = 442 blocos;
- Nível 2: b2 = [b1/fo] = [442/68] = 7 blocos
- Nível 3: b3 = [b2/fo] = [7/68] = 1 bloco
- Número de Níveis: $t = [\log_{68} 30.000] = 3$
- Número de acesso:
  - t + 1 = 4 acesso a blocos

Índices Multi-nível

- Se o índice for esparso, é necessário acessar o bloco do arquivo de dados para determinar a existência do registro
- Ainda existem problemas quanto à inserção de registros, pois cada nível do índice é um arquivo ordenado fisicamente;
  - Solução: Deixar espaço livre para inserções nos índices
  - Temos então um índice dinâmico Multi-nível

Árvores de Pesquisa e Índices

- Um índice multi-nível pode ser organizado de acordo com uma Árvore de Pesquisa
- Uma árvore de pesquisa de ordem m tem os nós compostos por no máximo 2m valores de busca (chaves) e no máximo 2m + 1 apontadores ordenados como:

$$[P_1,k_1,P_2,k_2,\ldots,P_j,k_j,P_{j+1}]$$

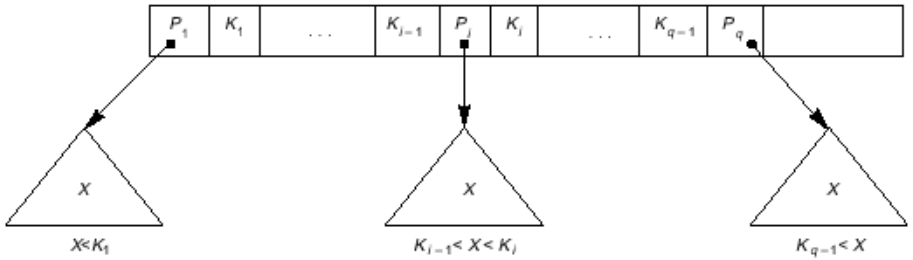
Árvores de Pesquisa e Índices

$$[P_1,k_1,P_2,k_2,...,P_j,k_j,P_{j+1}]$$

- $j \leq 2m$
- P_i é nulo ou aponta para uma sub-árvore.
- k_i é um valor de busca único.
- $k_1 < k_2 < ... < k_j$

Árvores de Pesquisa e Índices

- Para todos os valores de chave x nos nós de uma sub-árvore apontada por P_i temos:
  - $k_{i-1} < x < k_i$, para $1 < i < j+1$
  - $x < k_i$, para $i=1$
  - $k_{i-1} < x$,para $i=j+1$



Árvores de Pesquisa e Índices

- Podemos usar uma árvore de pesquisa como mecanismo de acesso aos registros de um arquivo de dados
- Cada k_i é um valor do campo de busca (indexação) do arquivo
- Pode haver um apontador para o registro (ou bloco que contém o registro) onde o valor k_i ocorre
- Um arquivo de índices é criado usando um bloco de disco para cada nodo da árvore

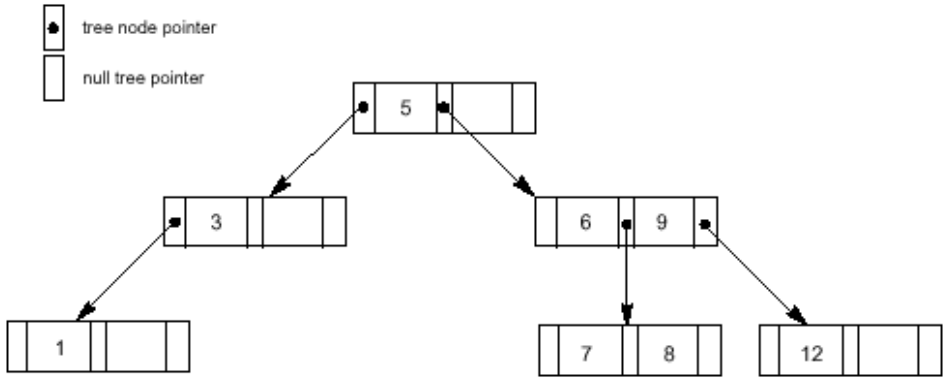
Árvores de Pesquisa e Índices

- Quando um novo registro é inserido no arquivo de dados, uma nova entrada deve ser criada no índice para inserir uma chave e uma apontador para o novo registro
- É desejável que:
  - A árvore esteja balanceada ou seja as folhas estejam aproximadamente no mesmo nível
  - Se evitem blocos/nodos excessivamente cheios ou vazios, para evitar desperdícios de espaço

Indexação com Árvores-B

- Árvore-B: Árvore de Busca com Restrições
  - A árvore é mantida balanceada
  - A carga em cada nodo/bloco esta entre 50% e 100%
  - Inserção: Quando a carga tende a ser superior a 100% (overflow) é executada uma operação de divisão de blocos (split)
  - Remoção: Quando a carga se torna inferior a 50% (underflow) é executada uma operação de concatenação de blocos (merge)

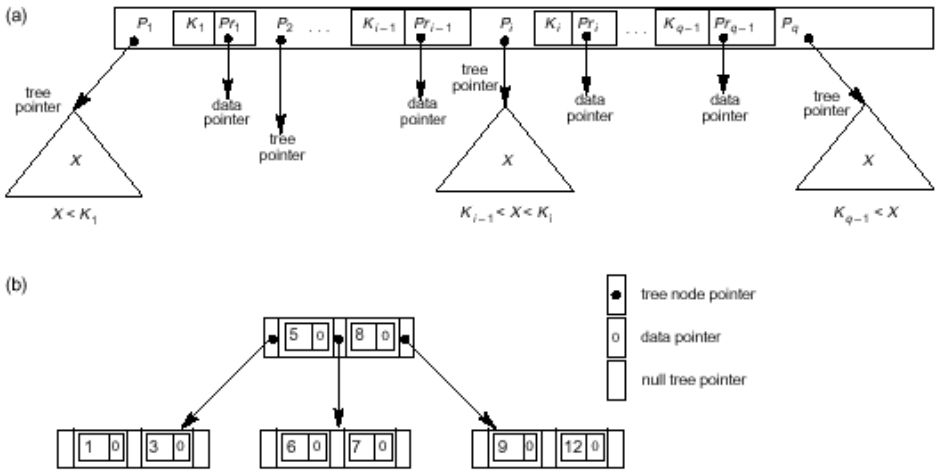
Árvore-B



Indexação com Árvores-B

- Árvore-B de ordem 2m: árvore pesquisa onde
  - Cada nó possui no máximo 2m chaves
  - Cada nó, exceto a raiz e as folhas, tem pelo menos m chaves e apontadores de dados
  - Um nodo com j ≤ 2m tem j+1 apontadores estruturais (de índice)
  - Todas as folhas estão no mesmo nível
- Os algoritmos de inserção e remoção devem observar estas restrições

Indexação com Árvores-B



Árvore-B – Exemplo

- Parâmetros:
  - Tamanho da chave: V = 9 bytes
  - Tamanho do apontador: P = 7 bytes
  - Tamanho do bloco: B = 512 bytes
- Em cada nodo devem caber
  - $(2m+1) \times 7 + 2m \times 7 + 2m \times 9$ bytes
- Assim
  - $(2m+1) \times 7 + 2m \times 7 + 2m \times 9 \leq 512$
  - $m \leq 10{,}9 \Rightarrow m = 10$

Árvore-B – Exemplo

Nível	Blocos	Chaves	Ponteiros
0	1	20	21
1	21	420	441
2	441	8.820	9.261
3	9.261	185.220	194.481
4	194.481	3.889.620	4.084.101
5	4.084.101	81.682.020	85.766.121
6	85.766.121	1.715.322.420	1.801.088.541

m = 10

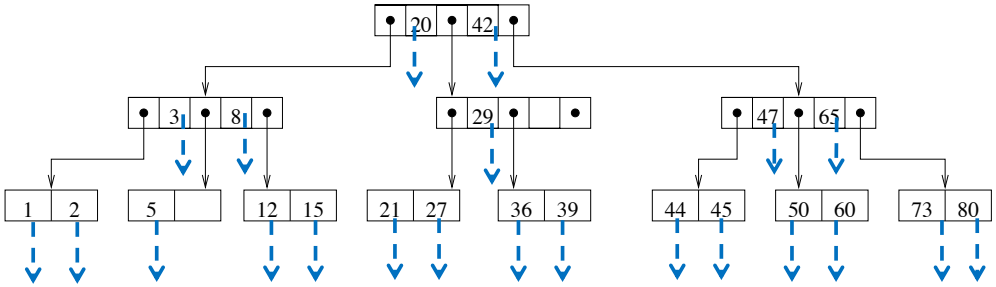
Árvore-B – Exemplo

Nível	Blocos	Chaves	Ponteiros
0	1	40	41
1	41	1.640	1.681
2	1.681	67.240	68.921
3	68.921	2.756.840	2.825.761
4	2.825.761	113.030.440	115.856.201
5	115.856.201	4.634.248.040	4.750.104.241
6	4.750.104.241	190.004.169.640	194.754.273.881

m = 20

Árvore-B – Busca

- No pior caso desce até o último nível
- Pesquisa binária em cada bloco de índice carregado na memória
- Carga do bloco de dados



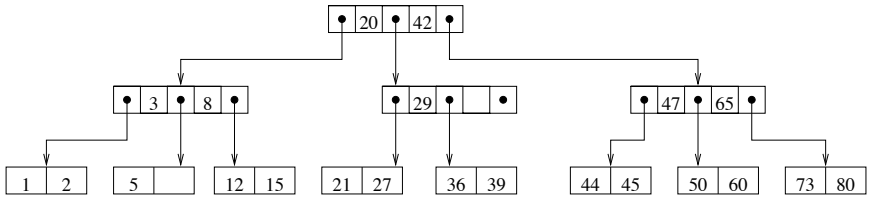
Apontador pra bloco de arquivo de dados

Árvore-B – Caminhamento

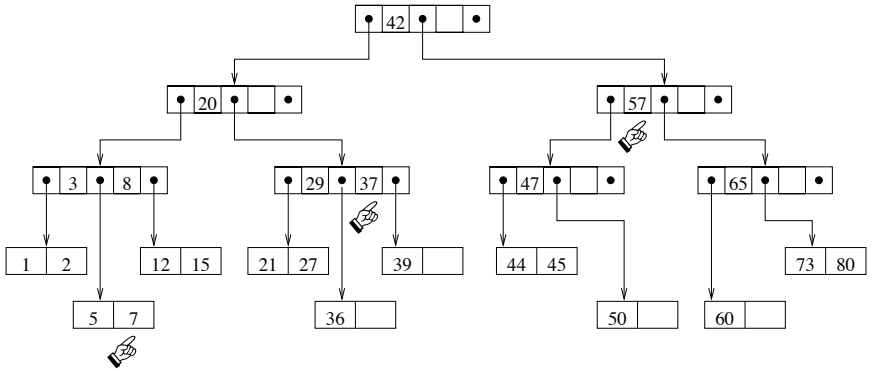
- Consiste em executar uma mesma operação (por exemplo imprimir o conteúdo) sobre todos os nós da árvore segundo uma determinada ordem
  - Pré-fixado;
  - Infixado ou central
  - Pós-fixado

Árvores-B – Inserção

- Caso 1: Há espaço na folha para inserção
- Caso 2: A folha já esta cheia (overflow)
  - A folha é dividida em duas (split), cada uma com m, chaves e uma chave é inserida no nó pai do que foi dividido (se há espaço neste nó)
- Caso 3: Se o nó pai também está cheio, também sofre divisão
  - No pior caso, esta divisão se propaga até a raiz fazendo com que a árvore-B cresça um nível ganhando um nova raiz.



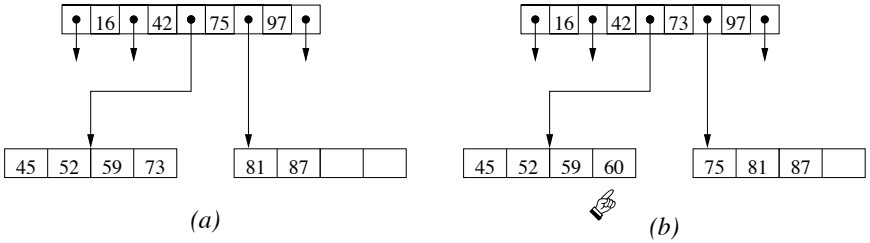
(a)



(b)

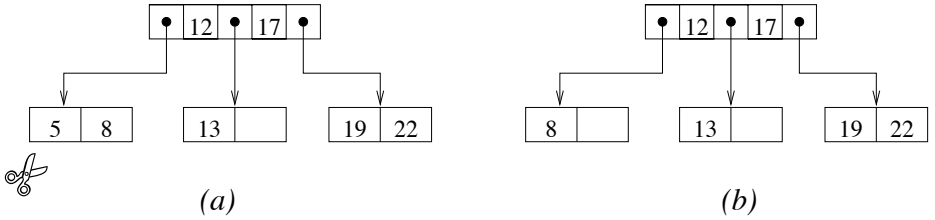
Árvores-B – Inserção

- Caso 4 (alternativa): Verifica-se os nós irmãos de um nó cheio que recebe uma nova chave, na tentativa de redistribuir as chaves, evitando assim a divisão do nó.



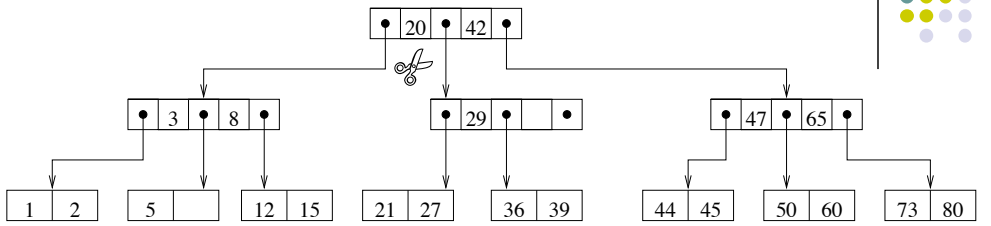
Árvores-B – Remoção

- Após a busca para encontrar o elemento a ser removido, podem existir 3 casos:
- Caso 1: Remoção de uma chave em uma folha permanecendo, a folha com pelo menos m chaves

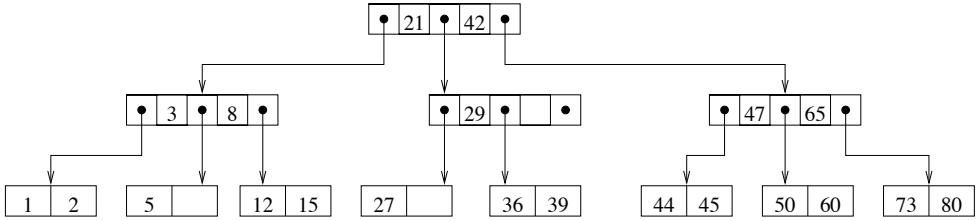


Árvores-B – Remoção

- Caso 2: Remoção de chave em um nó não folha. Para manter a restrição estrutural é necessário repor a chave removida. Para a reposição é usada a menor chave da sub-árvore à direita. Esta chave é removida da folha onde se encontra. O pior caso é aquele em que a remoção é feita na raiz.



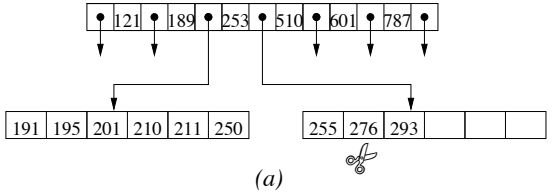
(a)



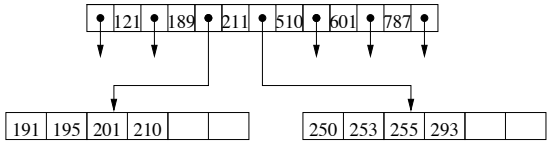
(b)

Árvores-B – Remoção

- Caso 3: A remoção de uma chave em uma folha diminui o número de chaves nesta folha para um número menor que m (underflow)
- Deverá haver transferência de chaves de um nó irmão desta folha, balanceando o número de chaves para uma quantidade aproximadamente igual nos dois nós.



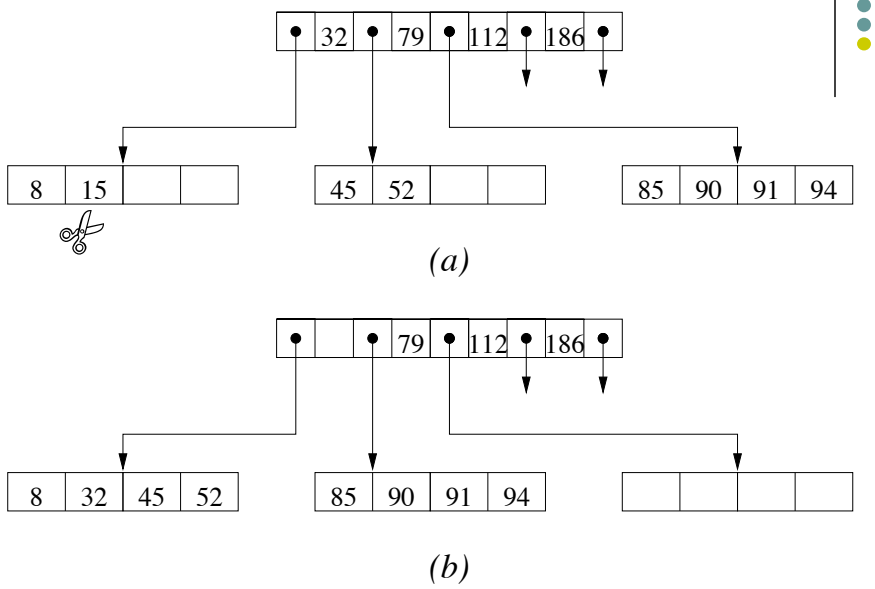
(a)



(b)

Árvores-B – Remoção

- Caso 3: Se não houverem chaves suficientes, os dois nós são concatenados, trazendo uma chave do nó pai para o novo nó resultante.
- Neste caso um dos dois nó é destruído. Se esta remoção causa underflow, a operação de concatenação pode se propagar para o nível acima, fazendo com que árvore-B diminua um nível.

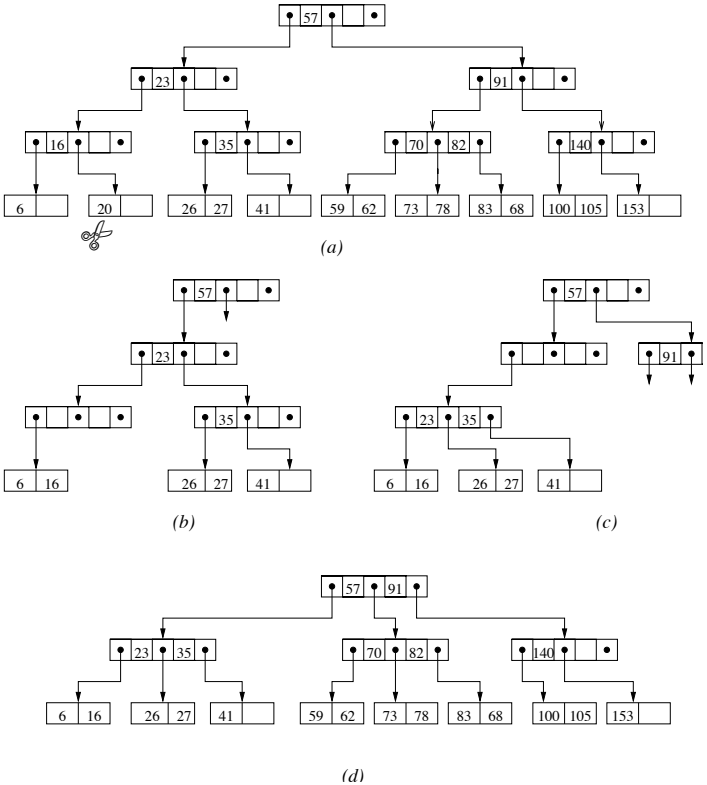


Árvores-B – Remoção

- No underflow temos os seguintes sub-casos:
  - O nó irmão de um nó folha que sofreu remoção tem chaves suficientes para “emprestar”
  - O nó irmão de um nó folha não possui chaves suficientes para “emprestar”, resultando na destruição de um dos nós e na transferência das chaves para um só nó, mais uma chave retirada do nó pai;

Árvores-B – Remoção

- Com a retirada de uma chave, o nó pai ficou com menos que m chaves e um empréstimo ou concatenação devem ser feitos.
- No pior caso, são feitas várias concatenações resultando na remoção da raiz e na diminuição de um nível da árvore-B.

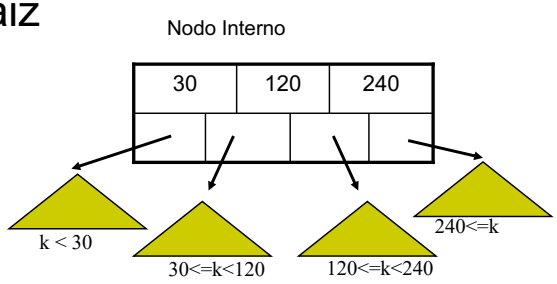


Árvores B+

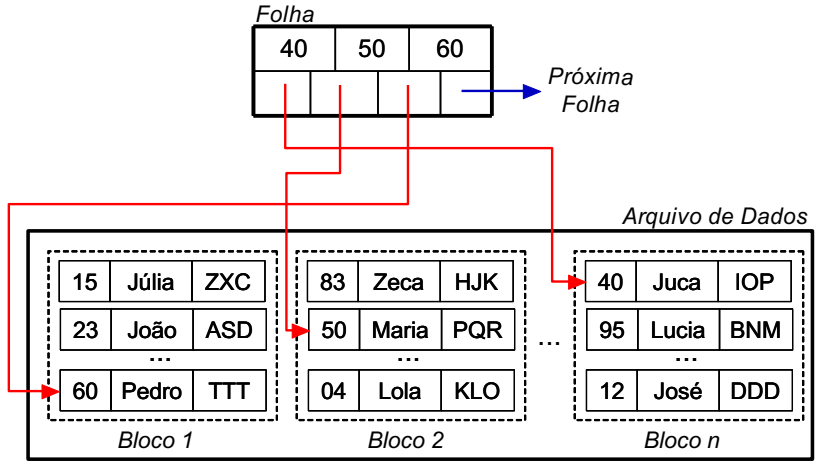
- Árvores de busca n-árias baleançadas derivadas das Árvores B tradicionais
- Página
  - Nodo da árvore
  - Corresponde a um bloco de disco
- Diferença das Árvores B
  - Todas as chaves devem ocorrer nas folhas, mesmo que ocorram também nos nodos internos
  - Folhas podem ser encadeadas
    - Facilita consultas por intervalos (range queries)
  - Não há apontadores de dados nos nódos internos
    - Isso aumenta o nr. de chaves nos blocos e ajuda a reduzir a altura final da árvore

Árvores B+ Conceitos

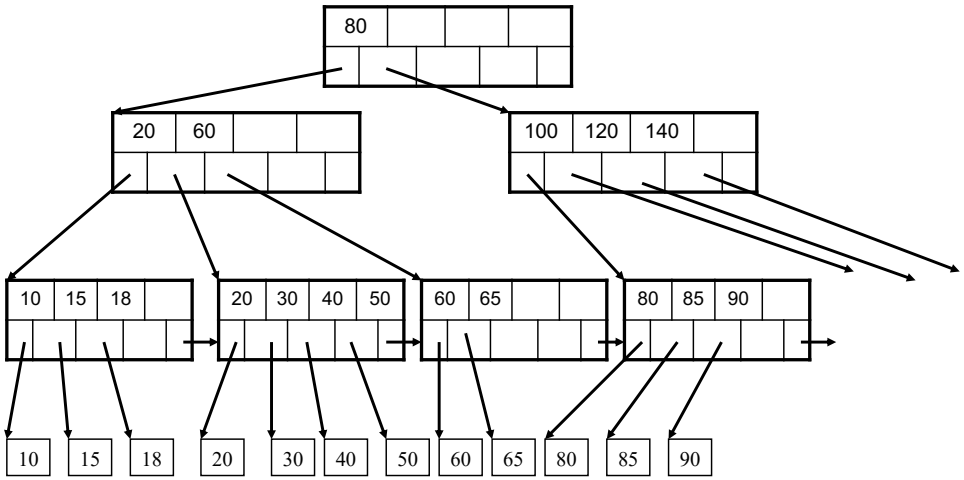
- Ordem ou grau: m
  - 2m = nr. Máximo de chaves no nodo
  - Fan-out = 2m
- Cada nodo tem n >= m e n <= 2m chaves
- Exceto o nodo raiz



Árvores B+ Conceitos



Árvores B+
Exemplo (m=2)



Árvores B+
Projeto

- Como determinar m?
- Exemplo:
  - Chave = 4 bytes
  - Apontador = 8 bytes
  - Bloco = 4096 byes
- 2m x 4 + (2m+1) x 8 ≤ 4096
- m = 170

Árvores B+
Busca

- Valores exatos de chave:
  - Ler página raiz
  - Ler blocos subseqüentes até uma folha

```
SELECT Nome
FROM   Pessoa
WHERE  Idade = 25
```

Árvores B+
Busca

- Busca por intervalo
  - Repetir os passos anteriores
  - Caminhar seqüencial pelas folhas

```
SELECT Nome
FROM   Pessoa
WHERE  20 <= idade
       AND idade <= 30
```

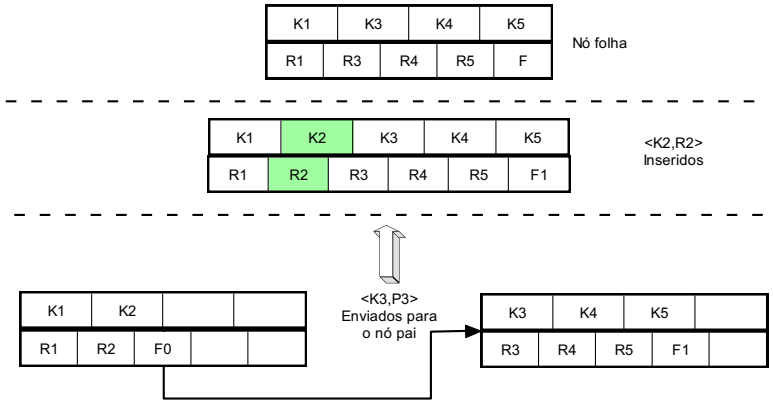
Árvores B+
Parâmetros Práticos

- Ordem típica: 100
- Taxa de ocupação típica (*fill factor*): 67%
  - Ocupação típica: 133
- Capacidades típicas
  - Altura 4: 133^4 = 312.900.700 registros
  - Altura 3: 133^3 = 2.352.637 registros
- Níveis mais altos podem ser armazenados no buffer pool:
  - Nível 1 = 1 página = 8 Kbytes
  - Nível 2 = 133 páginas = 1 Mbyte
  - Nível 3 = 17.689 páginas = 133 Mbytes

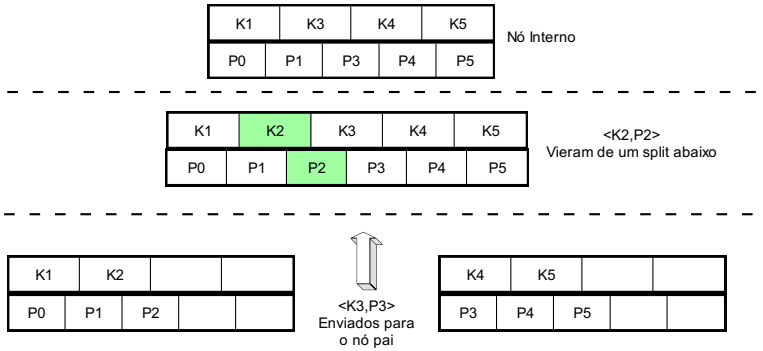
Árvores B+
Inserção

- Inserção (K,R)
  - K : Chave; R : Apontador para o bloco/registro de dados
  - Encontrar a folha onde K deve ficar e inserir (K,R)
  - Se o número de chaves ≤ 2m, terminar
  - Se não temos um *overflow*
    - Divir o nó e inserir chave extra no nó pai

Árvores B+
Inserção (2)

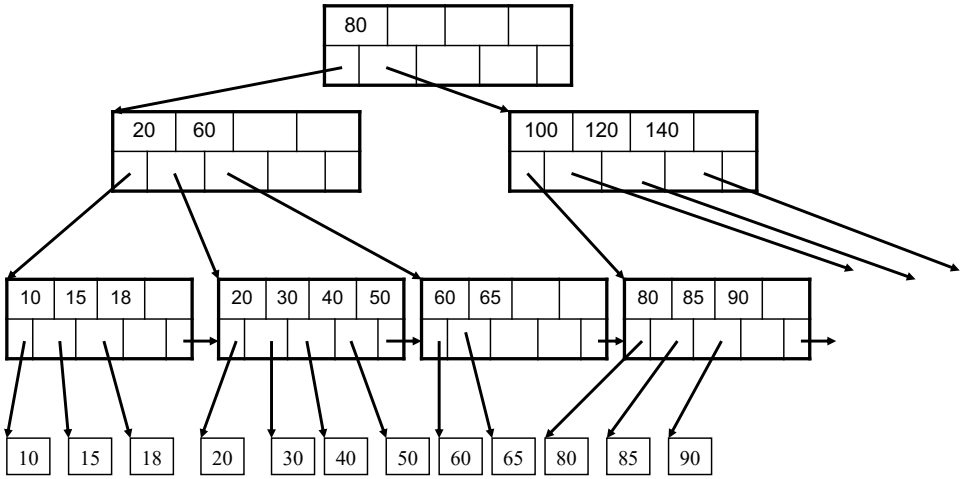


Árvores B+
Inserção (3)



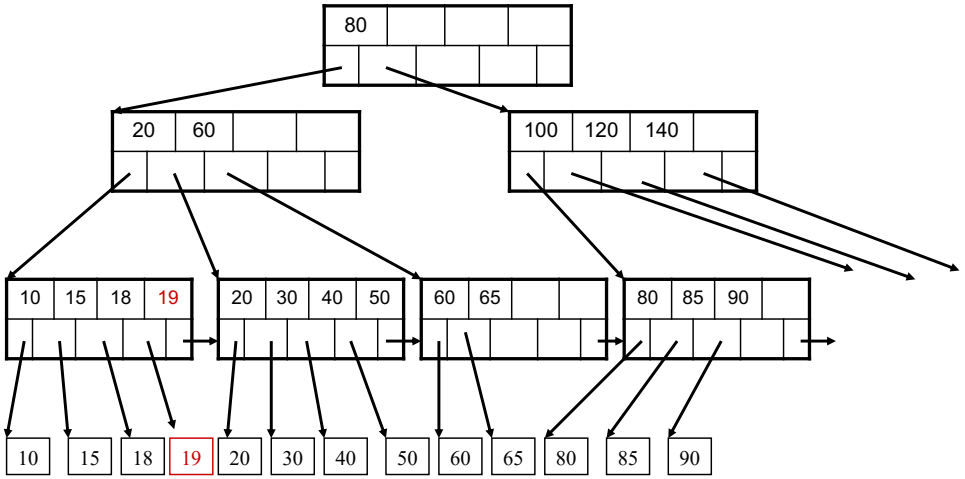
Árvores B+
Inserção – Exemplo 1

Inserir K=19



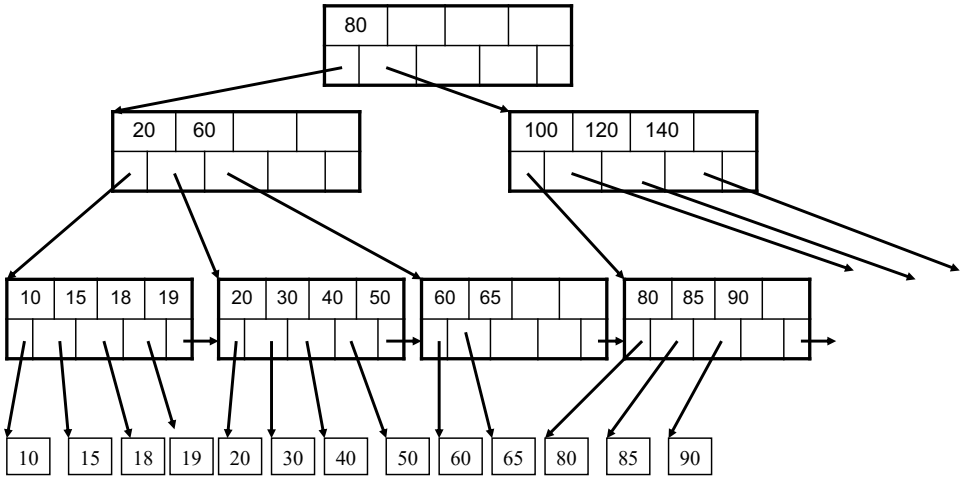
Árvores B+
Inserção – Exemplo 1 (2)

Depois da Inserção



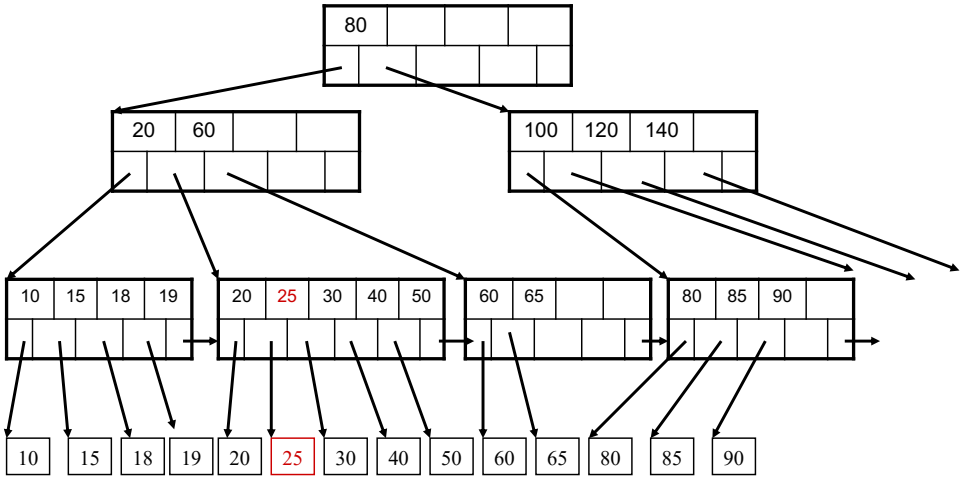
Árvores B+
Inserção – Exemplo 2

Inserir K=25



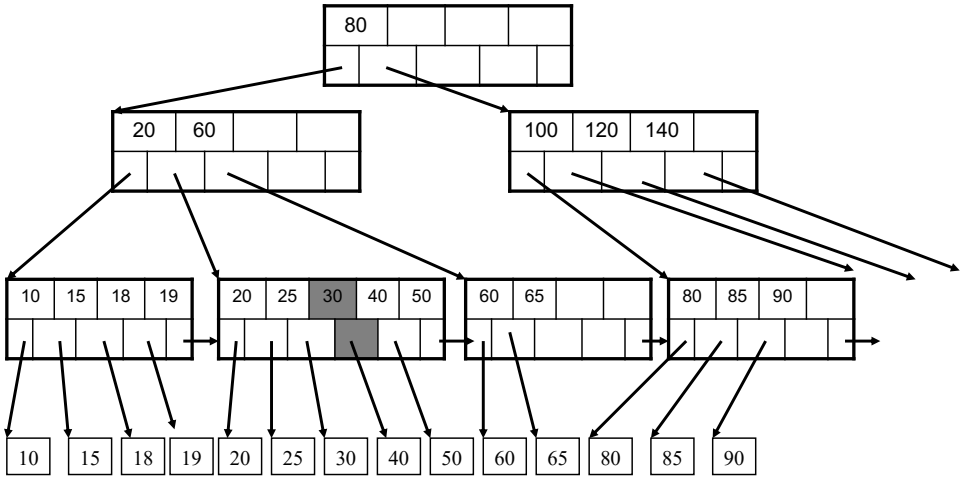
Árvores B+
Inserção – Exemplo 2 (2)

Depois da inserção



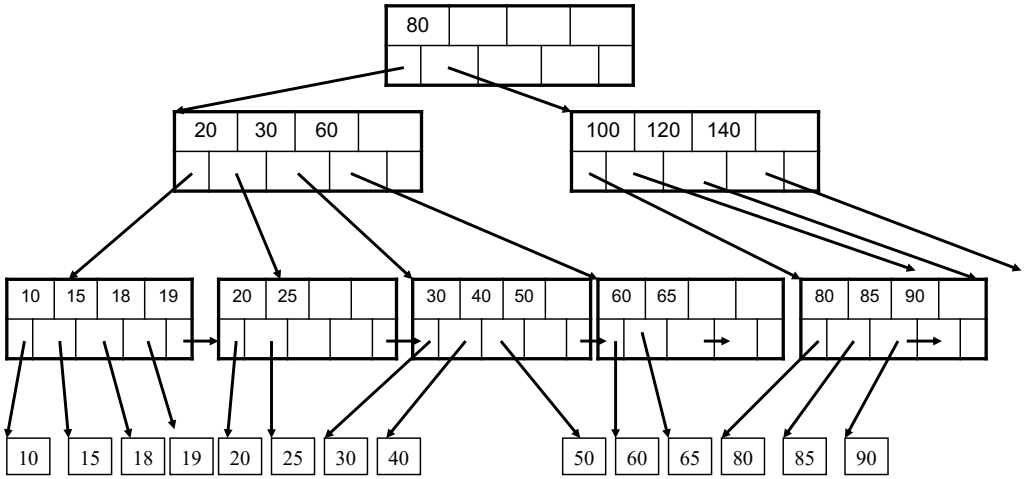
Árvores B+
Inserção – Exemplo 2 (3)

Split



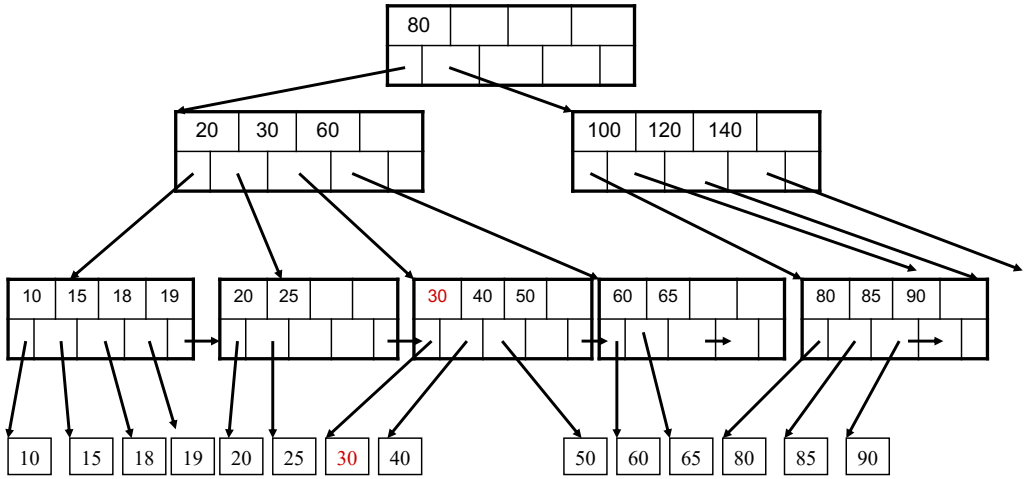
Árvores B+
Inserção – Exemplo 2 (4)

Depois do Split



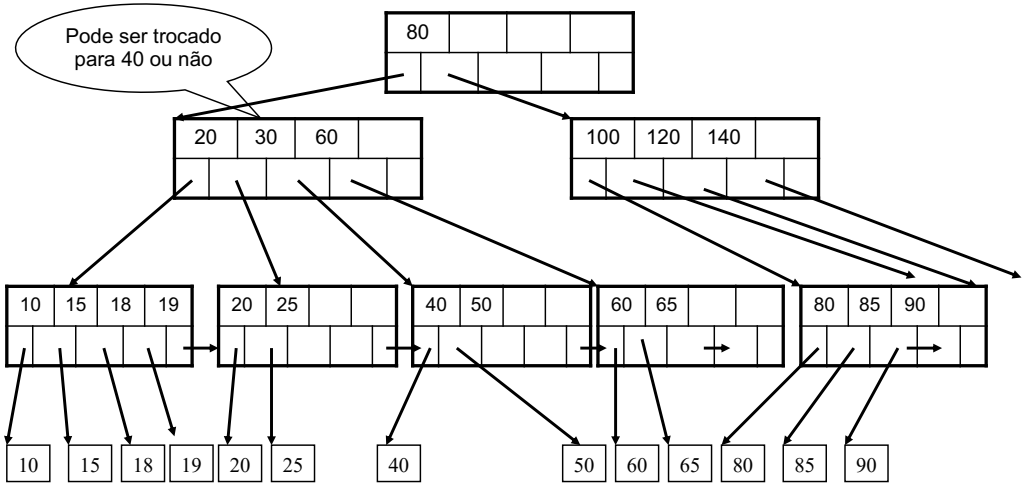
Árvores B+
Remoção – Exemplo 1

Remover chave 30



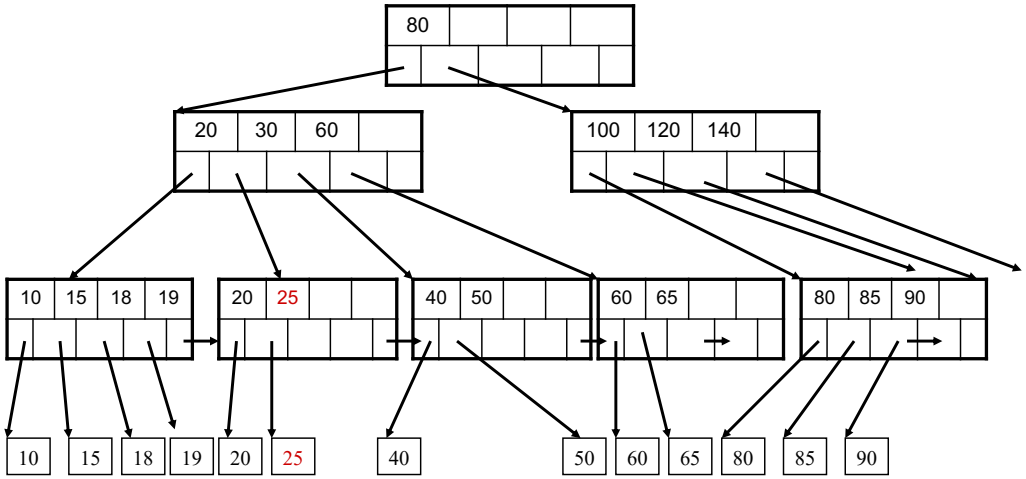
Árvores B+
Remoção – Exemplo 1 (2)

Depois da Remoção



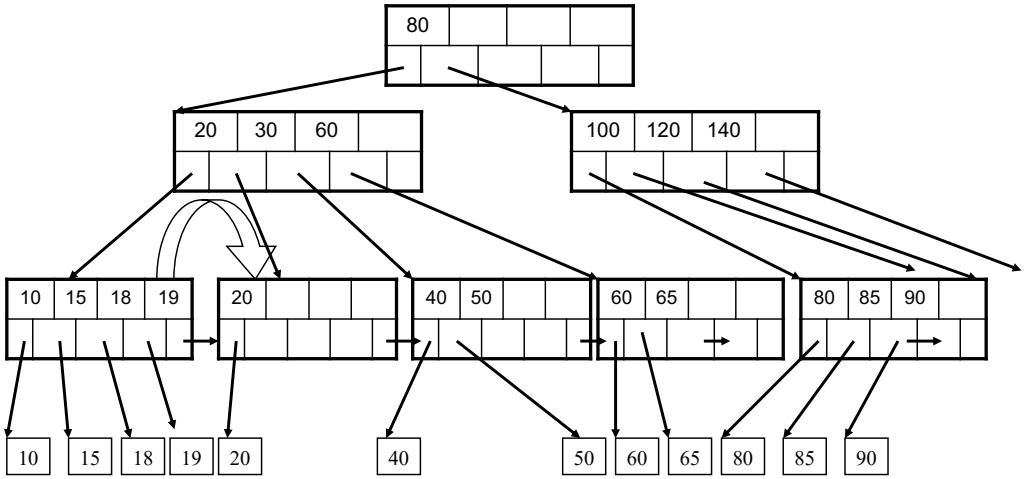
Árvores B+
Remoção – Exemplo 2

Remoção da chave 25



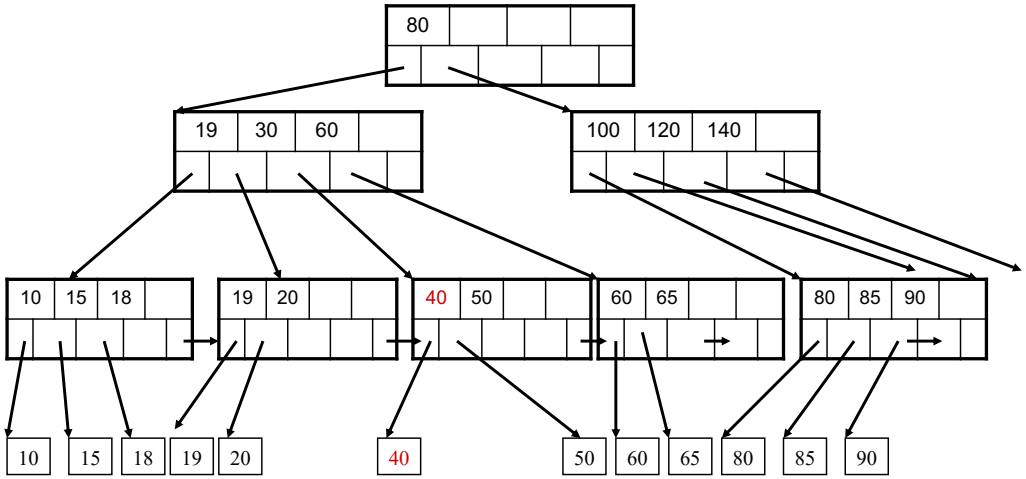
Árvores B+
Remoção – Exemplo 2 (2)

Depois da remoção da chave 25, rebalacear por rotação



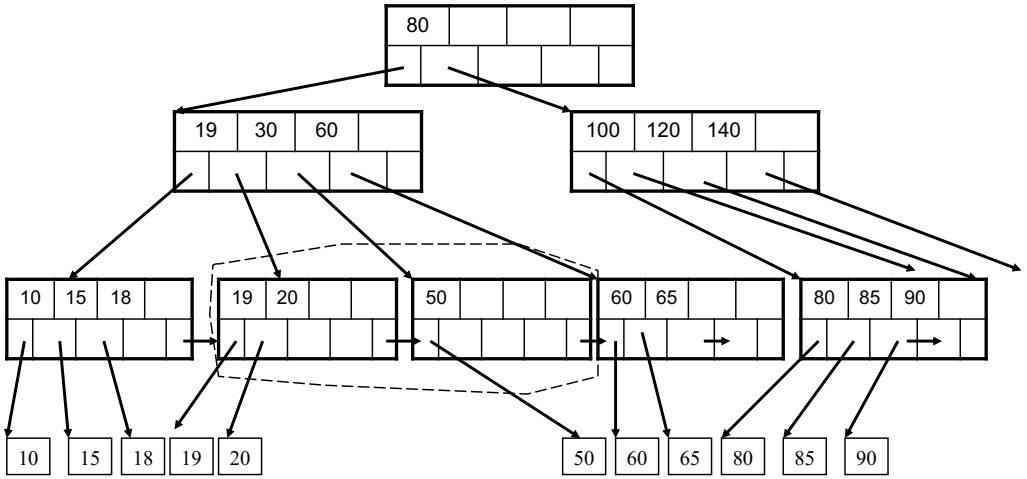
Árvores B+
Remoção – Exemplo 3

Remoção da chave 40



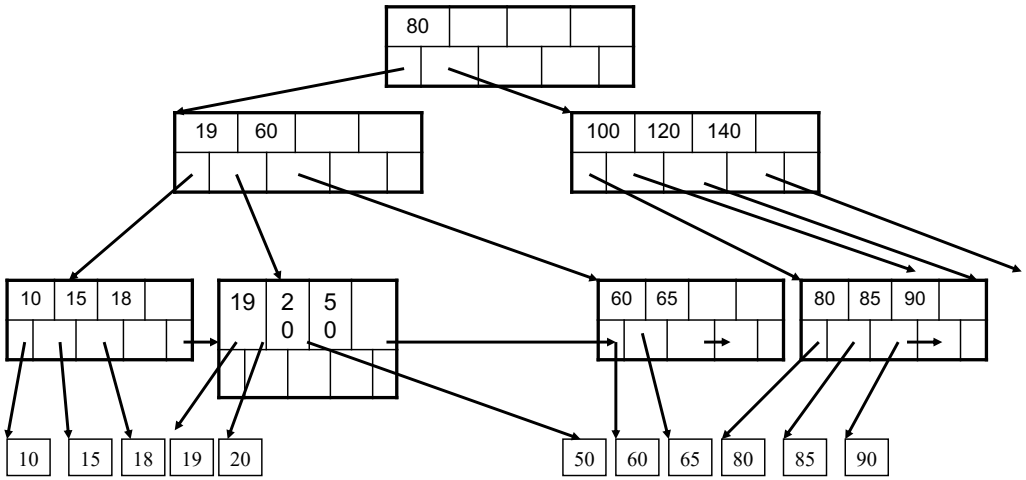
Árvores B+
Remoção – Exemplo 3 (2)

Depois da remoção, balancear por concatenação



Árvores B+
Remoção – Exemplo 3 (3)

Árvore final



Árvores B+
Otimizações

- A melhoria da performance nas operações é conseguida diminuindo a altura da árvore.
- Uma maneira de obter essa redução é aumentando o fan-out das folhas representando as chaves de forma comprimida
- Isso é particularmente interessante quando as chaves são strings longs
- A compressão pode complicar os algoritmos de inserção, remoção e busca

Compressão de Sufixo

- Considere uma página contendo as seguintes chaves
  - Manino, Manna, Mannari, Mannarino, Mannella, Mannelli
  - “Man” é um prefixo comum
  - As chaves podem ser armazendas como
    - (3, ino): “Man” + ino = Manino
    - (3 na): “Man” + na = Manna
    - (5 ri): “Manna” + ri = Mannari
    - (7 no): “Mannari” + no = Mannarino
    - (4 ella): “Mann” + ella = Mannella
    - (7 i): “Mannell” + l = Mannelli

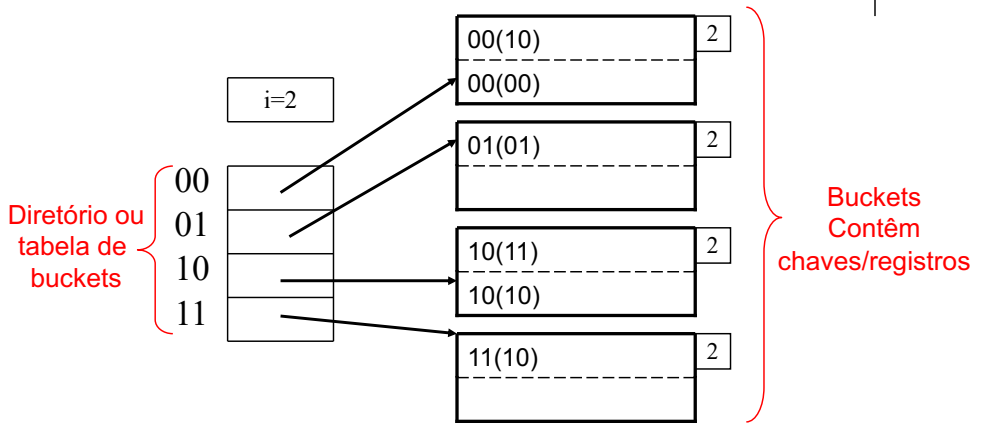
Hashing na indexação

- Estruturas de hashing podem ser usadas também para indexação
- A performance para busca é em geral melhor do que as Árvores B+ : O(1)
- Muito ineficiente para buscas por intervalo (range queries)
- Desperdício de espaço
  - Soluções:
    - Hashing Extensível
    - Hashing Linear

Hash Extensível

- Técnica para permitir que a tabela hash cressa para evitar degradação de performance
- Assume que a função de hashing retorna números no intervalo $\{0, \ldots, 2^k - 1\}$
- Começa com $n = 2^i << 2^k$
  - n e i vão crescendo quando necessário
- Usa apenas os i bits mais significantes da função de hashing para endereçar os registros

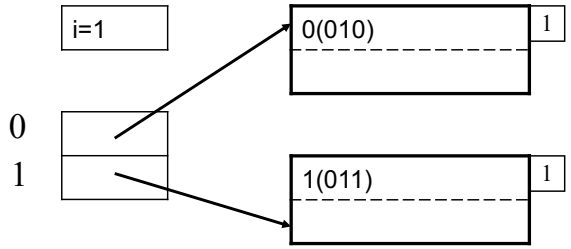
Hash Extensível (2)



- Somente os 2 primeiros bits são usados para determinar o bucket onde o registro/chave
- Idealmente, o diretório cabe na memória principal

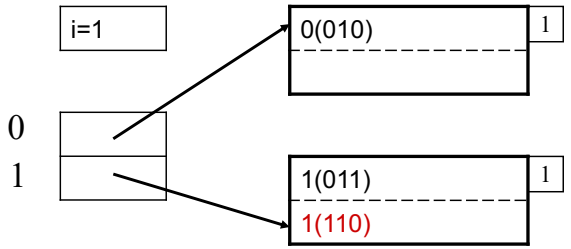
Hash Extensível (3)

- Exemplo $i=1$, $n=2^i=2$, $k=4$



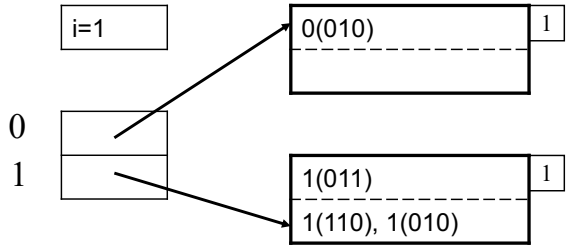
Hash Extensível – Inserção

- Inserir 1110



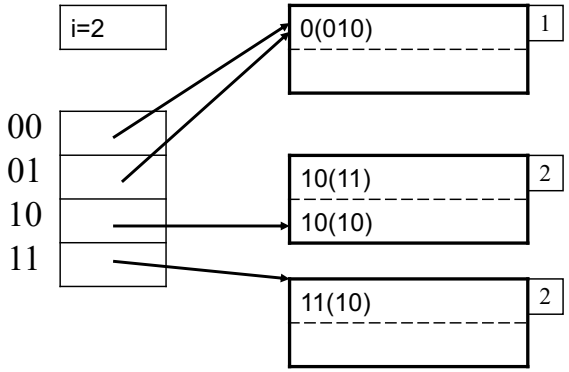
Hash Extensível – Inserção (2)

- Inserir 1010
  - Tabela é estendida pela divisão de blocos
  - i passa a ser 2



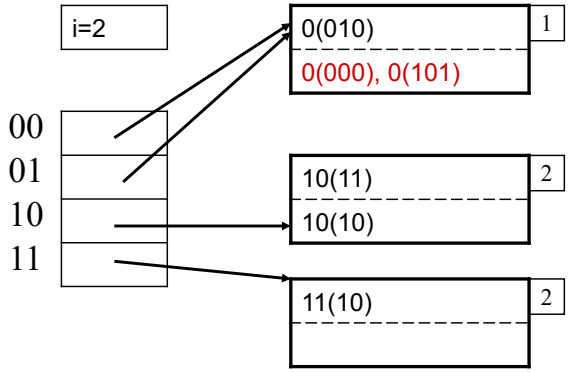
Hash Extensível – Inserção (3)

- Depois da inserção de 1010

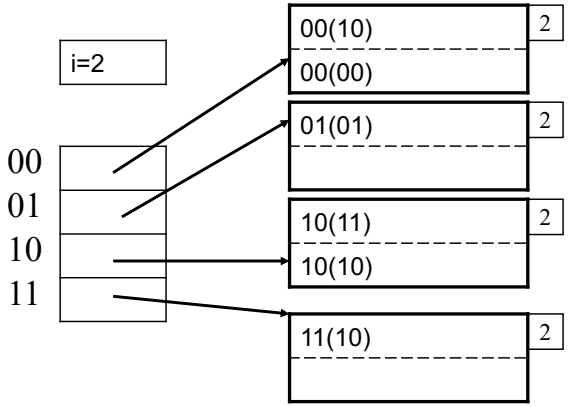


Hash Extensível – Inserção (4)

- Inserir 0000 e depois 0101



Hash Extensível – Inserção (5)



Hash Extensível – Sumário

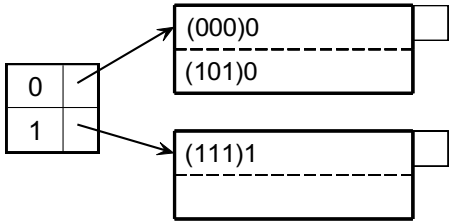
- ☺ Permite expansão dinâmica de arquivos de dados ou de indíce implementados com hashing
  - Menos espaço disperdiçado
  - Não requer reorganização total dos arquivos
- ☺ Não requer blocos de overflow
  - Acesso toma sempre uma leitura de bloco
- ☹ Toda extensão requer dobrar o número de blocos
  - Para valores altos de i, algumas inserções tomam tempo desproporcionalmente alto
- ☹ Depois de várias extensões o diretório pode não caber mais na memória

Hashing Linear

- Idéia: Estender 1 bucket de cada vez, ao invés de dobrar o número de bucktes
  - Problema: n não será mais uma potência de 2
- Seja i tal que $2^i <= n < 2^{i+1}$
- Depois de computar $h(k)$, usar os <u>últimos</u> i bits para endereçar:
- Se os últimos i bits representam um número $> n$, troque o bit mais significativo de 1 para 0.Isso transformar em um número $<= n$

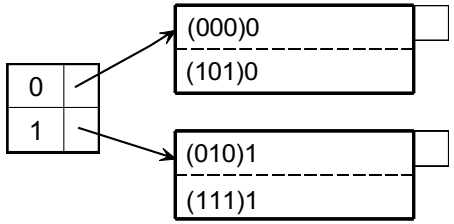
Hashing Linear – Exemplo

- Nível: i=1
- Buckets: n=2
- Tam.Chave: k=4
- Registros: r=3
- Taxa de ocupação:
  - r/n = 3/2 = 1,5 = 75%
- Taxa de ocupação desejada:
  - <80%



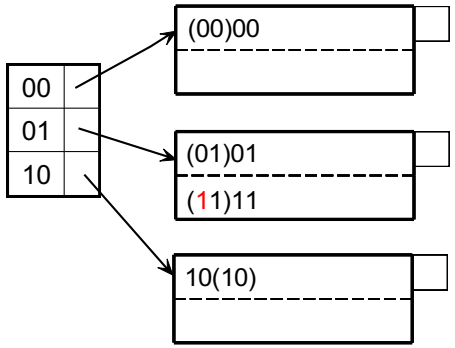
Hashing Linear – Exemplo (2)

- Inserir chave 0101
  - Ordenação desejável, mas não essencial
  - Não causa extensão
- i=1,n=2,k=4 e r=4
  - r/n = 4/2 = 2 = 100%



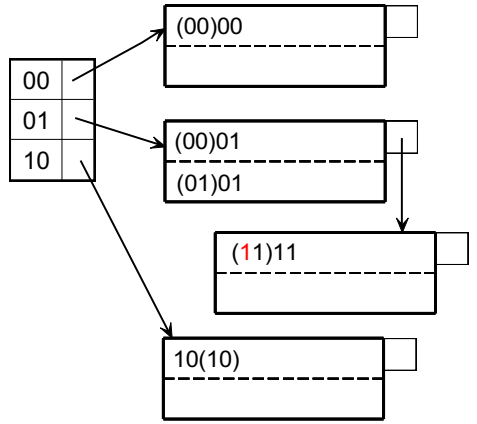
Hashing Linear – Exemplo (3)

- Inserção da chave 0101 “enche” a estrutura
- Crescer um bloco para dar “folga”
- Diretório ganha mais uma entrada
- i=2,n=3,k=4 e r=4
  - r/n = 4/3 = 1,33 = 60%
- Chave 1111 “flipada” para armazenar no bucket 01



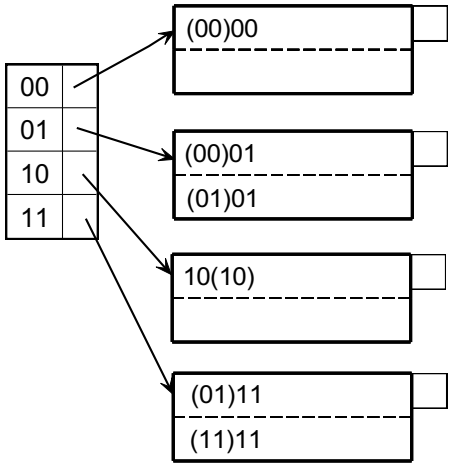
Hashing Linear – Exemplo (4)

- Inserção da chave ”0001”
- Criado um bucket de overflow para a entrada 01
- i=2,n=3,k=4 e r=5
  - r/n = 5/3 = 1,66 = 60%



Hashing Linear – Exemplo (5)

- Inserção da chave ”0111”
- Criado novo bucket e nova entrada no diretório
- i=2,n=4,k=4 e r=6
  - r/n = 6/4 = 1,5 = 75%

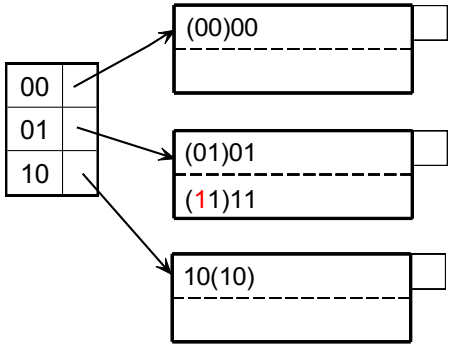


Hashing Linear – Busca

- Seja m=h(K)[i]= $a_0\, a_1 \ldots a_{i-1}$
  - Últimos i bits de h(K)
- Seja n o número de buckets
- Se m<n
  - Buscar no bucket $a_0\, a_1 \ldots a_{i-1}$
  - a_0 é necessariamente 1
- Se m≥n (n = número de buckets)
  - Transformar a_0 em 0
  - Buscar no bucket $0a_1 \ldots a_{i-1}$

Hashing Linear – Busca (2)

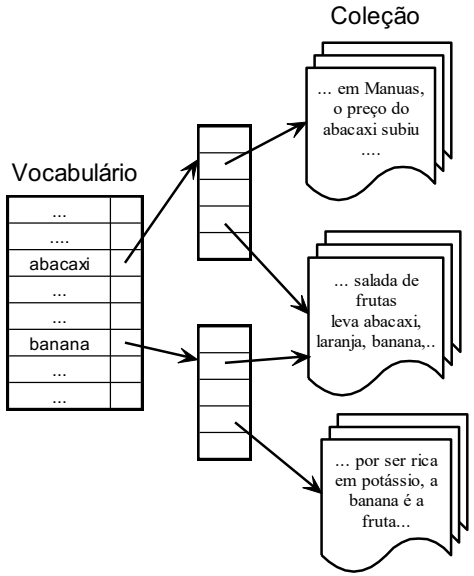
- i=2, n=3
- Buscar 1010
  - $m=10_2=2_{10} < n$
  - Buscar no bucket 10
- Buscar 1011
  - $m=11_2=4_{10} > n$
  - Buscar no bucket 01



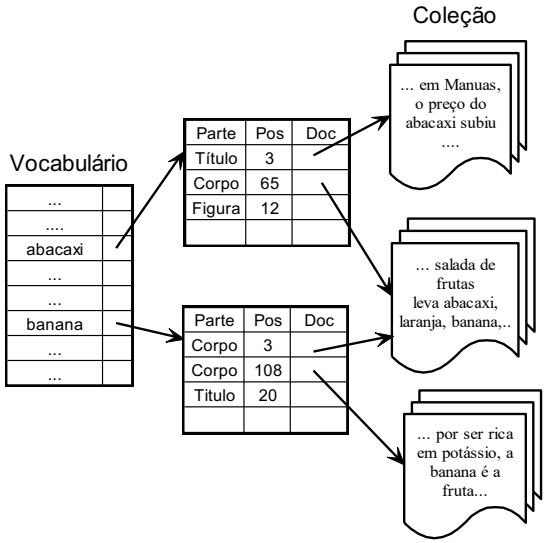
Índices para documentos

- Motivação:
  - Que documentos falam sobre “abacaxi” ?
- Documentos vistos como conjuntos de palavras ou *termos*
- Coleção = conjunto de todos os documentos a indexar
- Vocabulário = conjunto de todos os termos de todos os documentos da coleção

Índices Invertidos



Índices Invertidos (2)



Consultas típicas em RI

- Consultas simples
  - Encontre documentos com abacaxi e banana
  - Encontre documentos com abacaxi ou banana
  - Encontre documentos com abacaxi e sem banana
- Consultas estruturadas
  - Encontre documentos com abacaxi no título
  - Encontre documentos com abacaxi seguido de banana