Capítulo

12 BTree

Professor Luiz Thadeu Grizendi
UniAcademia Estruturas de Dados

BTree

Introdução

A árvore B-tree é uma extensão da árvore binária de pesquisa e foi desenvolvida por Bayer e McGreight em 1972. Como a B-tree teve uma utilização muito grande no meio da computação e já em 1979 o uso de B-Trees era o padrão para a recuperação e armazenamentos de registros em memória secundária.

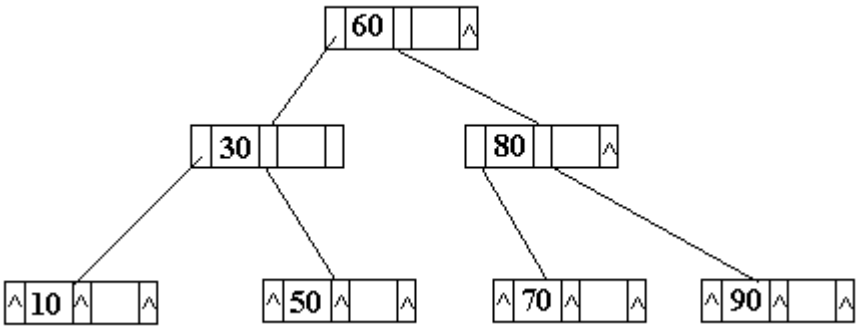
O problema fundamental com a manutenção de um índice em disco é que o acesso é muito lento. O melhor acesso a um índice ordenado, até agora, foi dado pela Pesquisa Binária, porém:

- Pesquisa binária requer muitos acessos. 15 registros podem requerer 4 acessos, 1000 registros requerem, em média, 9.5 acessos.
- Pode ser muito caro manter um índice ordenado. É necessário um método no qual a inserção e a eliminação de registros tenha apenas efeitos locais, isto é, não exija a reorganização total do índice.

Conceitos Importantes

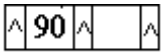
Ordem

A definição atual de B-Tree vincula a ordem de uma árvore B-Tree ao número de descontentes de um nó, isto é, o número de ponteiros. Deste modo, numa árvore B-Tree de ordem **m** o número máximo de chaves é **m**-1. Isto é, Uma árvore B de ordem 8 tem um máximo 7 chaves por página. A figura abaixo ilustra uma B-tree de ordem 2.



Página

A página corresponde ao nó da árvore. Por exemplo na figura abaixo corresponde a uma página com de ordem 3 com uma chave.



Número mínimo de chaves por página

Quando uma página é dividida na inserção, os nós são divididos igualmente entre as páginas velha e nova. Deste modo, o número mínimo de chaves em um nó é dado por **m/2** - 1 (exceto para a raiz). Por exemplo, uma árvore B-Tree de ordem 8 que tem um máximo de 7 chaves por página tem um mínimo de 3 chaves por página.

Nó folha

Os nós folhas são aqueles que não possuem filhos.

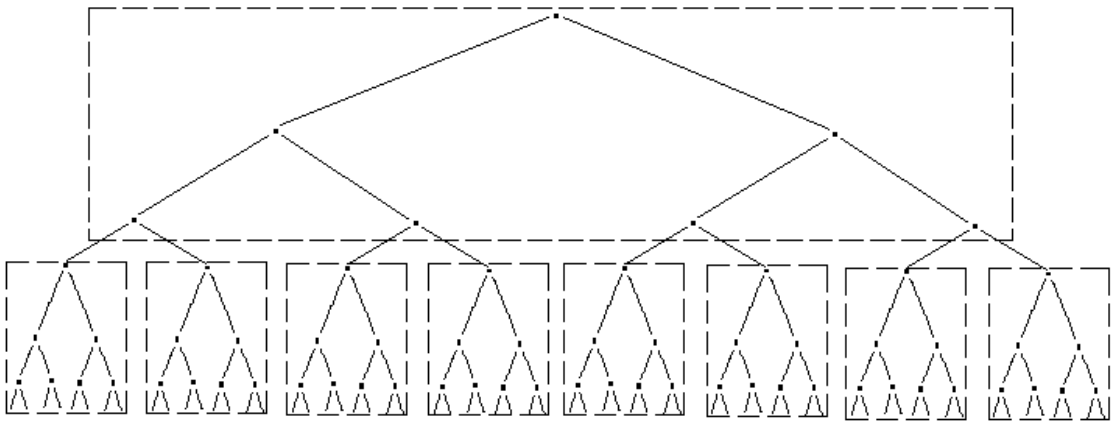
Definição formal das Propriedades de B-Trees

Para uma B-Tree de ordem **m**:

- cada página tem um máximo de **m** descendentes
- cada página, exceto a raiz e as folhas, tem um mínimo de **m/2** descendentes
- a raiz tem um mínimo dois descendentes, a menos que seja uma folha
- todas as folhas aparecem num mesmo nível
- uma página que não é folha e possui **k** descendentes contém **k-1** chaves
- uma página folha contém no mínimo **m/2 -1** e no máximo **m-1** chaves

Solução por Paged Binary Trees

O velho problema está de volta: a busca por uma posição é muito lenta. Novamente temos o caso de que, uma vez encontrada a posição, podemos ler uma grande quantidade registros sequencialmente. Isso nos leva à noção de página: uma vez localizada, todos os registros são lidos.



A divisão de uma árvore binária em páginas é ilustrada na figura acima. Nessa árvore de 9 páginas, quaisquer dos 63 registros podem ser acessados em no máximo 2 acessos. Se na figura, cada página, tiver 8kbytes e contiver 511 chaves; a árvore cheia e completamente balanceada, nas quais cada página é também uma árvore completamente balanceada: apresentará um total de 134.217.727 chaves que podem ser acessadas em no máximo 3 acessos ao disco.

Pior caso para árvore binária: $\log_2 (N+1)$ e versão em páginas: $\log_{k+1} (N+1)$

Onde k é o número de chaves de uma página; note que, para árvores binárias, k=2.

Veja o efeito da escala logarítmica quando k = 2 e k = 511:

- árvore binária: log2 (134.217.727) = 27 acessos
- versão em páginas: log511+1 (134.217.727) = 3 acessos

Preços a pagar:

- tempo na transmissão de grande quantidade de dados
- manutenção da ordem da árvore.

Construção Bottom-Up nas B-Trees

Na construção das Arvores Binárias de Pesquisa e AVLs a inserção é feita sempre nas folhas e no sentido top-Down, isto é, de cima para baixo. Bayer e McCreight propuseram que as árvores sejam construídas de baixo para cima. As chaves raiz da árvore emergem naturalmente.

Numa árvore B:

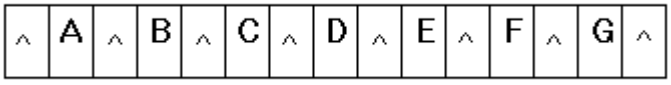
- Cada página é formada por uma sequência de chaves e conjunto de ponteiros.
- Não existe uma árvore explícita dentro de uma página (ou nó).
- O número de ponteiros em um nó excede o número de chaves em 1.
- O número máximo de ponteiros em um nó é a ordem da árvore.
- O número máximo de ponteiros é igual ao número máximo de descendentes de um nó.

Exemplo: uma árvore B-Tree de ordem 8 possui nós com no máximo 7 chaves e 8 filhos.

- Um nó folha não possui filhos, e seus ponteiros são nulos.

Splitting e Promoting

Seja a seguinte a página inicial de um B-Tree de ordem 8.



Esta folha está cheia. Como inserir uma nova chave, digamos J ?

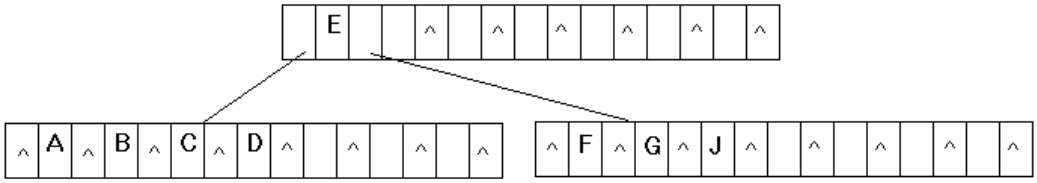
Split

Dividimos (split) o nó folha em dois nós folhas, distribuindo as chaves igualmente entre os nós. Sequência das chaves A, B, C, D, E, F, G, J. Incluir

as chaves A, B, C e D na célula da esquerda e as chaves F, G e J. Temos agora duas folhas: precisamos criar uma nova raiz. Fazemos isso promovendo uma das chaves que separam as folhas.

Promotion

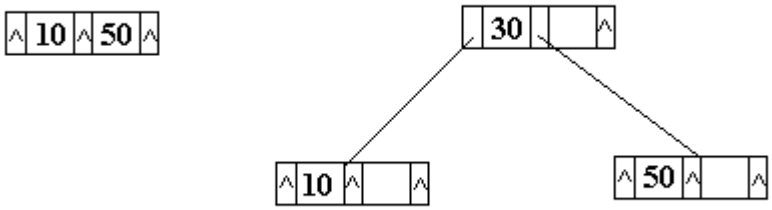
Nesse caso, promovemos a chave E para a raiz:



Criação de uma B-tree

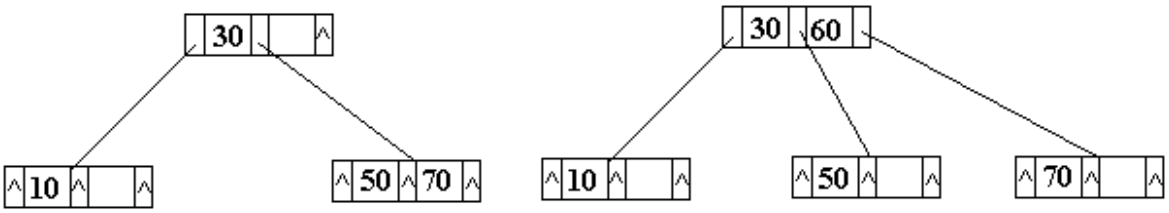
Criar uma B-tree de ordem 3 com as chaves 50, 10, 30, 70, 60, 80 e 90.

A figura seguinte ilustra a inserção das chaves 50, 10, 30. Primeiramente na inserção das chaves 50 e 10, cria-se a raiz com estes dois elementos. Pode-se observar que os ponteiros estão aterrados pois não existem filhos. Na inserção da chave 30, como não tem espaço disponível na célula para a inserção ocorre o denominado **spliting** e o **promotion**.

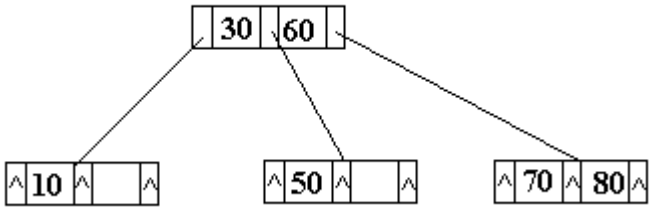


O **spliting** consiste na subdivisão de uma determinada célula em duas outras com a redistribuição equitativa dos elementos em cada célula, isto é, as células menores que a chave correspondentes à chave do meio são colocadas na célula à esquerda e as células maiores que a chave correspondente à chave do meio são colocadas na célula à direita. Na figura anterior as chaves são 10, 30 e 50. A chave do meio é 30, as chaves menores somente a chave 10 e as maiores somente a chave 50. Logo, a célula à esquerda recebe a chave 10, e a célula à direita recebe a chave 50 sendo que a chave do meio vai para o nível superior. A ida da chave do meio ao nível superior é denominada de **promotion**. No caso a chave 50 é levada ao nível superior. Como neste nível não existe célula, então é criada uma nova célula e a chave é incluída nesta célula e os ponteiros são atualizados.

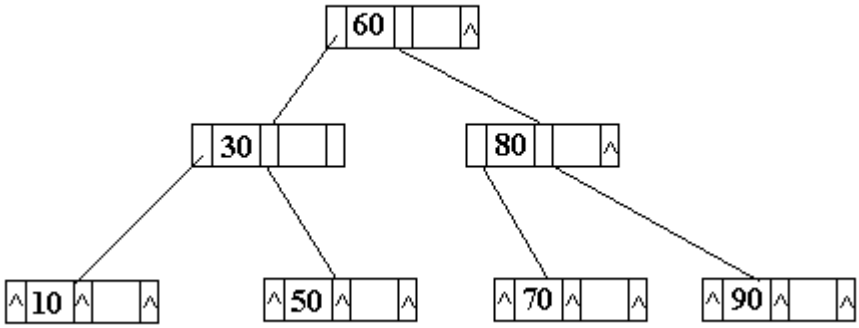
A figura seguinte ilustra a inserção das chaves 70 e 60. Primeiramente a inserção da chave 70. Pode-se observar neste caso que a inserção é feita na folha cuja chave é maior que 30. Na inserção da chave 60 vai ocorrer um novo **spliting** da célula correspondente. A sequencia de chaves é 50, 60 e 70. A chave 50 é incluída na célula à esquerda, a chave 70 na célula à direita e a chave 60 é promovida (**promotion**) ao nível superior. Como no nível superior existe vaga para esta célula, faz-se a inclusão normal.



A figura seguinte ilustra a inserção da chave 80. Pode-se observar que neste caso a inserção é feita na folha cuja chave é maior que 60 de modo normal sem ocorrer **spliting**.



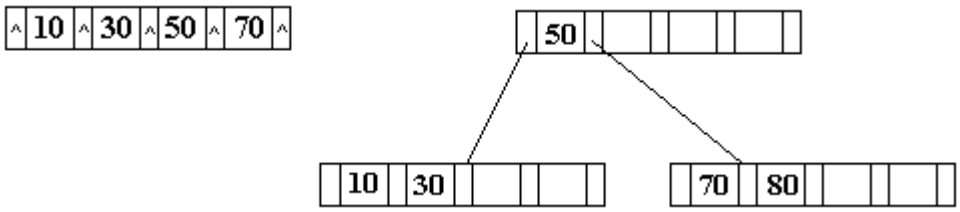
A figura seguinte ilustra a inserção da chave 90. Pode-se observar que neste caso a inserção é feita na folha cuja chave é maior que 60. Como a célula já está cheia, ocorre um novo **spliting**. A sequencia de chaves é 70, 80 e 90. A chave 70 é incluída na célula à esquerda, a chave 90 na célula à direita e a chave 80 é promovida (**promotion**) ao nível superior. Como no nível superior não existe vaga para esta célula, faz-se um novo **spliting**. A sequência de chaves é 30, 60 e 80. A chave 30 é incluída na célula à esquerda, a chave 80 na célula à direita e a chave 60 é promovida (**promotion**)ao nível superior.



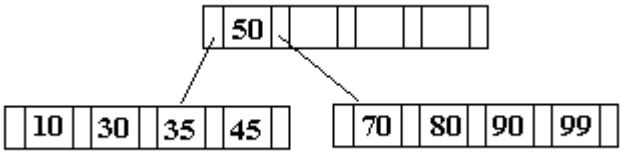
Outro Exemplo

Criar uma B-tree de ordem 5 com as chaves 10, 70, 30, 50, 80, 35, 45, 90, 99, 20 ,85,25, 26, 37, 46, 71, 75,28 e 74.

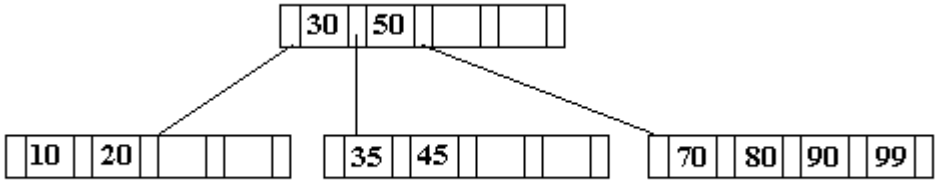
A figura seguinte ilustra a inserção das chaves 10, 70, 30, 50 e 80. Primeiramente na inserção das chaves 10, 70, 30 e 50, cria-se a raiz com este elementos. Pode-se observar que os ponteiros estão aterrados pois não existem filhos. Na inserção da chave 80, como não tem espaço disponível na célula para a inserção ocorre o denominado **spliting** e o **promotion**. A sequencia de chaves é 10, 30, 50,70 e 80. As chaves 10 e 30 são incluídas na célula à esquerda, as chaves 70 e 80 na célula à direita e a chave 50 é promovida (**promotion**) ao nível superior. Como neste nível não existe célula, então é criada uma nova célula e a chave é incluída nesta célula e os ponteiros são atualizados.



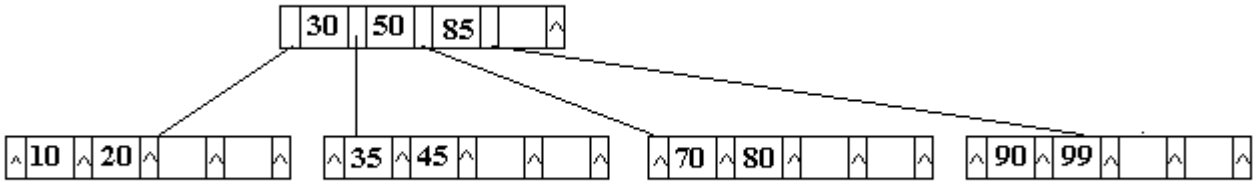
A figura seguinte ilustra a inserção das chaves 35, 45, 90 e 99. Estas chaves são incluídas à esquerda e à direita da raiz nos espaços disponíveis nas respectivas células de modo normal sem ocorrer **spliting**.



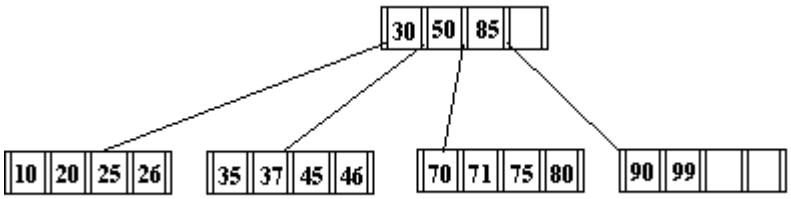
A figura seguinte ilustra a inserção da chave 20. Vai ocorrer um novo **spliting** da célula correspondente. A sequencia de chaves é 10, 20, 35 e 45. As chaves 10 e 20 são incluídas na célula à esquerda, as chaves 35 e 45 na célula à direita e a chave 30 é promovida (**promotion**) ao nível superior. Como no nível superior existe vaga para esta célula, faz-se a inclusão normal.



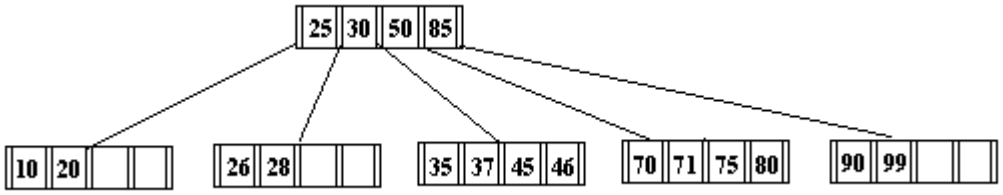
Na inserção da célula 85 vai ocorrer um novo **spliting** na célula à direita da raiz. A sequência de chaves é 70, 80, 85, 90 e 99. As chaves 70 e 80 são incluídas na célula à esquerda, as chaves 90 e 99 na célula à direita e a chave 85 é promovida (**promotion**)ao nível superior. Como no nível superior existe vaga para esta célula, faz-se a inclusão normal.



As próximas chaves a serem incluídas são 25, 26, 37, 46, 71 e 75. Estas chaves serão incluídas nas respectivas células sem ocorrer um **spliting**. A figura abaixo ilustra estas inclusões.

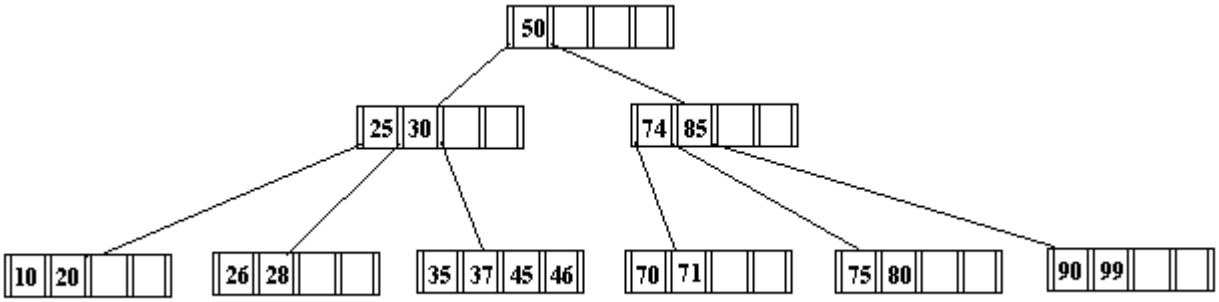


A figura seguinte ilustra a inserção da chave 28. Vai ocorrer um novo **spliting** da célula correspondente. A sequência de chaves é 10, 20, 25, 26 e 28. As chaves 10 e 20 são incluídas na célula à esquerda, as chaves 26 e 28 na célula à direita e a chave 25 é promovida (**promotion**) ao nível superior. Como no nível superior existe vaga para esta célula, faz-se a inclusão normal.



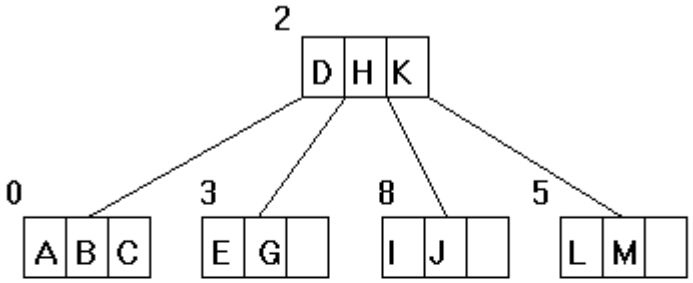
A figura seguinte ilustra a inserção da chave 74. Pode-se observar que neste caso a inserção é feita na folha cuja chave é maior que 50 e menor que 85. Como a célula já está cheia, ocorre um novo **spliting**. A sequencia de chaves é 70,

71, 74, 75 e 80. As chaves 70 e 71 são incluídas na célula à esquerda, as chaves 75 e 80 na célula à direita e a chave 74 é promovida (**promotion**) ao nível superior. Como no nível superior não existe vaga para esta célula, faz-se um novo **spliting**. A sequência de chaves é 25, 30, 50 e 74 e 85. As chaves 25 e 30 são incluídas na célula à esquerda, as chaves 74 e 85 na célula à direita e a chave 50 é promovida (**promotion**) ao nível superior. Como não existe célula neste nível, cria-se uma nova célula e depois é incluída a chave 50. O resultado final da operação é mostrada na figura abaixo.

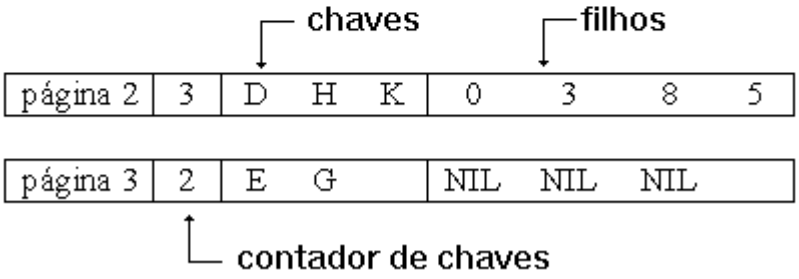


Algoritmos de Pesquisa e Inserção

Um exemplo da parte de uma B-tree com de ordem 4 é dado na figura abaixo. Nessa figura é mostrado um nó interno e 4 nós folha. É explicitado os RRN de cada página, o RRN 0 é um número válido de página, e os ponteiros das folhas apontam para nil (que pode ser -1).



Conteúdo das páginas 2 e 3:



Busca

Inserção e Divisão

Os algoritmos Insertion e Splitting começam a pesquisa no nó raiz e vão em direção à raiz depois de encontrar a posição de inserção ao nível das folhas, os processos de inserção, divisão e promoção trabalham de baixo para cima, do nível das folhas até a raiz.

O algoritmo insert insere KEY. A tentativa de inserção tem início na página CURRENT_RRN. Se essa página não é uma folha, a função se chama recursivamente até encontrar KEY ou chegar a uma folha. Se KEY for encontrada a função vai retornar erro e terminar (recursivamente). Senão a inserção vai começar na página folha.

Havendo espaço em PAGE a chave é inserida, caso contrário PAGE é dividida chamando-se a função split. Uma divisão faz com que PROMO_KEY seja a chave do meio, e PROMO_R_CHILD identifique a nova página criada: com esses dois valores, a inserção continua, recursivamente, no nível de cima da árvore.

A necessidade de nova inserção, no nível superior, é indicada com o retorno de PROMOTING.

A função split

Para tratar o overflow causado pela inserção de I_KEy e I_RRN na página PAGE, a função cria uma página NEWPAGE, distribui as chaves igualmente entre as chaves, e determina qual chave a ser promovida ao nível superior na árvore.

A chave a ser promovida é colocada em PROM_KEY, e o RRN da nova chave é retornado em PROMO_R_CHILD.

Profundidade de Busca no Pior Caso

É importante entender o relacionamento entre o tamanho de página, o número de chaves armazenados em uma página, e o número de níveis que uma B-tree pode ter.

Exemplo: "Você precisa armazenar 1.000.000 chaves e considera utilizar B-tree de ordem 512. Qual o número máximo de acessos para localizar uma chave ? "

Ou seja: quão profunda pode ser a árvore ?

O pior caso ocorre quando cada página tem apenas o número mínimo de descendentes, e possuem, portanto, altura máxima e largura mínima.

Em geral, para um nível d qualquer de um B-tree, o número mínimo de descendentes é dado por (2 * m/2) ^ (d-1).

Para uma árvore de N chaves, a sua profundidade no nível das páginas folhas, é dado por d onde:

d <= 1 + log(base 256)m/2((N+1)/2)

Então para a B-tree de ordem 512 com 1.000.000 de chaves

d <= 1 + log256(500.000,5) <= 3.37 Essa é a performance que procuramos !

Eliminação, Redistribuição e Concatenação

O processo de divisão (split) de páginas garante a manutenção das propriedades da B-tree durante a inserção. Essas propriedades precisam ser mantidas, também, durante a eliminação de chaves.

Caso 1: eliminação mantém número mínimo de chaves na página.
Solução: Chave é retirada e registros internos à página reorganizados.

Caso 2: eliminação de chave que não está numa folha.

Solução: Sempre eliminamos da folha. Se uma chave deve ser eliminada de uma página que não é folha, trocamos a chave com sua sucessora imediata, a qual, com certeza está numa folha, e então eliminamos a chave da folha.

Caso 3: eliminação causa underflow na página.
O número mínimo de chaves por página nessa árvore é n/2 -1 onde n é a ordem da B-Tree.

Solução: Redistribuição. Procura-se uma página irmã (mesmo pai) que contenha mais chaves que o mínimo: se existir redistribui-se as chaves entre essas páginas. A redistribuição causa a colocação de uma nova chave de separação no nó pai.

Caso 4: ocorre underflow e a redistribuição não pode ser aplicada.
Quando não existirem chaves suficientes para dividir entre as duas páginas irmãs.

Solução: Deve-se utilizar concatenação: combina-se o conteúdo das duas páginas e a chave da página pai para formar uma única página. Concatenação é o inverso do Splitting. Como consequência, pode causar o underflow da página pai.

Caso 5: underflow da página pai

Solução: No exemplo, a concatenação das páginas 3 e 4 retira D a página 1, causando underflow na página 1. Redistribuição não pode ser aplicada (por quê?). Deve-se utilizar concatenação novamente.

Caso 6: Diminuição da altura da árvore.
Consequência da concatenação dos filhos do nó.

Solução: A concatenação das páginas 1 e 2 absorve a única chave da raiz.

Este caso mostra o que ocorre quando a concatenação é propagada até a raiz. Note que esse nem sempre é o caso: se a página 2 (Q e W) tivesse mais uma chave, aplicar-se-ia redistribuição em vez de concatenação.

Eliminação de chave em árvore B

1. Se a chave não estiver numa folha, troque-a com seu sucessor imediato.
2. Elimine a chave da folha.
3. Se a folha continuar com o número mínimo de chaves, fim.
4. A folha tem uma chave a menos que o mínimo. Verifique as páginas irmãs da esquerda e direita:

4.1.se uma delas tiver mais que o número mínimo de chaves, aplique redistribuição.

4.2.senão concatene a página com uma das irmãs e a chave pai.

5. Se ocorreu concatenação, aplique passos de 3 a 6 para a página pai.
6. Se a última chave da raiz for removida, a altura da árvore diminui.

Redistribuição durante Inserção

Diferentemente da divisão e da concatenação, o efeito da redistribuição é local. Não existe propagação.

Outra diferença: não existe regra fixa para o rearranjo das chaves e a eliminação de uma chave pode causar o underflow de apenas uma chave na página, e a redistribuição pode mover apenas uma chave para página com problema.

Exemplo:

Dada uma B-tree de ordem 101, o número mínimo e máximo de chaves é, respectivamente, 50 e 100. Se ocorre o underflow numa página, e a página irmã tem 100 chaves, qualquer número de chaves entre 1 e 50 pode ser transferido. Normalmente transfere-se 25, e deixa-se as páginas equilibradas.

Redistribuição é uma ideia nova, a qual não foi explorada no algoritmo de inserção. E é uma opção desejável. Em vez de dividir uma página cheia em duas meia-página, pode-se optar por colocar a chave que sobra (ou mais que uma!) em outra página. Essa estratégia deve resultar numa melhor utilização do espaço.

Redistribuição X Divisão
Depois da divisão de uma nó, cada página está 50% vazia. Portanto a utilização do espaço, no pior caso, em uma árvore B que utiliza splitting é de cerca de 50%. Em média, para árvores grandes, prova-se que o índice é de 69%.

Estudos empíricos indicam que a utilização de redistribuição pode elevar o índice de 67% para 85%. Esses resultados sugerem que qualquer aplicação séria de árvore B utilize, de fato, redistribuição durante a inserção.

http://slady.net/java/bt/view.php

Bibliografia

1. HOROWITZ, E. ; SAHNI, Sartaj. Fundamental of Data Structures in Cl. New York, Computer Science Press, 1994.

2. ZIVIANI, N. *Projeto de Algoritmos com Implementação em Pascal e C*. 4ª ed. São Paulo: Pioneira, 1999.

3. Folk, Michael J. & Zoelik, Bill, *File Structures*, Addison-Wesley , 1992 (2ª Edition);

4. Tharp, Alan j., *File organization and Processing*, John Wiley & Sons, Inc., 1988;

5. AHO, A. V. ; HOPCROFT, John E. Data Structures and Algorithms. Massachusetts: Addison Wesley, 1983;