Árvores-B (Parte I)

Leandro C. Cintra

M.C.F. de Oliveira

Fonte: Folk & Zoelick, File Structures

Histórico

A invenção da B-*tree*

- Bayer and McGreight, 1972, publicaram o artigo: *"**Organization and Maintenance of Large Ordered Indexes**"*
- Em 1979, o uso de árvores-B para manutenção de índices de bases de dados já era praticamente padrão em sistemas de arquivos de propósito geral

Problema

- Acesso a disco é caro (lento)
- Até agora usamos **pesquisa binária** nos índices ordenados
- Mas se o índice é grande é não cabe em memória principal, então a pesquisa binária exige muitos acessos a disco
  - 15 itens podem requerer 4 acessos, 1.000 itens podem requerer até 11 acessos. São números muito altos

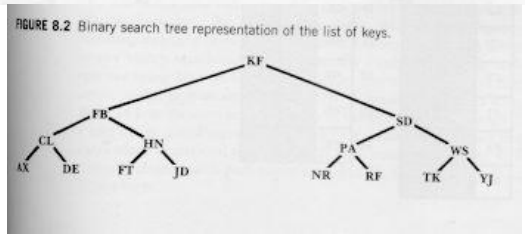
Problema

- O custo de manter em disco um índice ordenado de forma a permitir busca binária é proibitivo
- 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

Solução:árvores binárias de busca?

AX CL DE FB FT HN JD KF NR PA RF SD TK WS

FIGURE 8.1 Sorted list of keys.



Vetor ordenado e representação por árvore binária

Solução: árvores binárias de busca

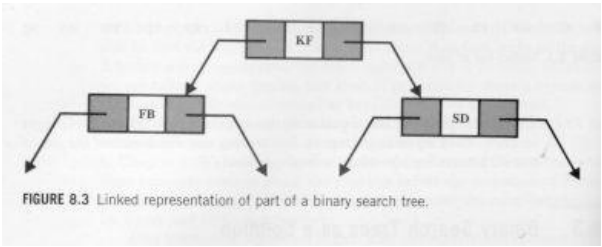
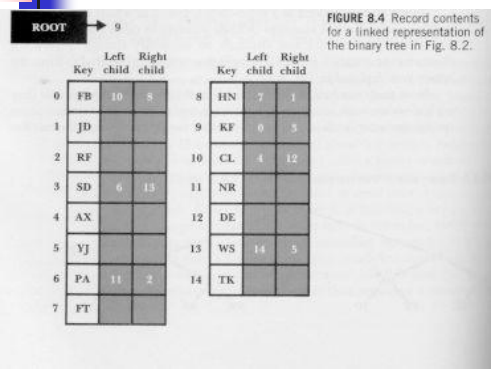


FIGURE 8.3 Linked representation of part of a binary search tree.

Representação da árvore no arquivo



os registros são mantidos em arquivo, e ponteiros (**esq** e **dir**) indicam aonde estão os registros filhos.

Vantagens

- A ordem lógica dos registros não está associada à ordem física no arquivo
- O arquivo físico do índice não precisa mais ser mantido ordenado: o que interessa é recuperar a estrutura lógica da árvore, o que é possível com os campos **esq** e **dir**
- Inserção de uma nova chave no arquivo
  - é necessário saber aonde inserir esta chave na árvore, de modo a mantê-la como ABB. A busca pelo registro é necessária, mas a reorganização do arquivo não é

9

Inserção da chave LV

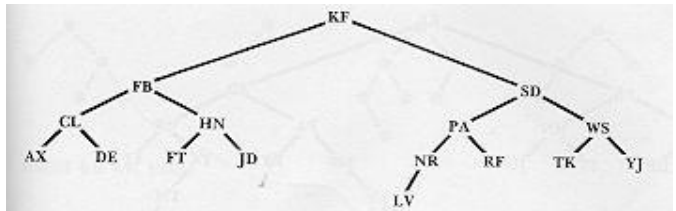


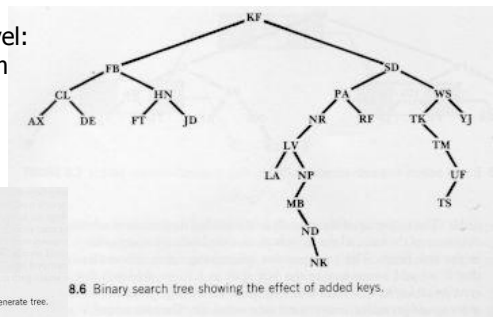
FIGURE 8.5 Binary search tree with LV added.

10

Problema: desbalanceamento

- Inserção das chaves NP MB TM LA UF ND TS NK

Situação indesejável: inserção em ordem alfabética



8.6 Binary search tree showing the effect of added keys.

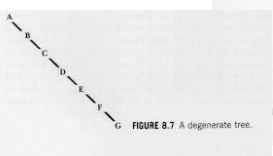


FIGURE 8.7 A degenerate tree.

11

Solução por árvores-AVL

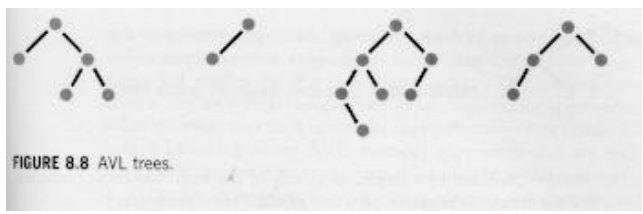


FIGURE 8.8 AVL trees.

12

Árvores binárias perfeitamente balanceadas e AVL

FIGURE 8.10 A completely balanced search tree.

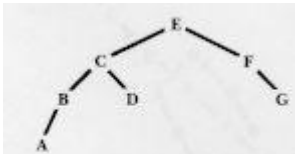


FIGURE 8.11 A search tree constructed using AVL procedures.

Solução por árvores-AVL

- A eficiência do uso de árvores binárias de busca exige que estas sejam mantidas balanceadas
- Isso implica no uso de árvores AVL
- Número máximo de comparações para localizar uma chave
  - Em uma árvore binária perfeitamente balanceada, é igual à altura da árvore, dada por $\log_2 (N+1)$
  - Para uma árvore AVL, esse número é $1.44*\log_2 (N+2)$
  - Para 1.000.000 de chaves
    - Na árvore completamente balanceada uma busca percorre até 20 níveis
    - Para uma árvore AVL, a busca poderia percorrer até 28 níveis

Solução por árvores-AVL

- Entretanto, se as chaves estão em memória secundária, qualquer procedimento que exija mais do que 5 ou 6 acessos para localizar uma chave é altamente indesejável
  - 20 ou 28 *seeks* são inaceitáveis
- Árvores balanceadas são uma boa alternativa para o problema da ordenação
  - não requerem a ordenação do índice a cada nova inserção
- As soluções até agora não resolvem o número excessivo de acessos a disco

Solução por Árvores Binárias Paginadas (*Paged Binary Trees*)

- a busca (seek) por uma posição específica do disco é muito lenta
- uma vez encontrada a posição, pode-se ler uma grande quantidade registros seqüencialmente a um custo relativamente pequeno

Solução por Árvores Binárias Paginadas (*Paged Binary Trees*)

- Esta combinação de busca (*seek*) lenta e transferência rápida sugere a noção de **página**
  - Em um sistema "paginado", uma vez realizado um *seek*, que consome um tempo considerável, todos os registros em uma mesma "página" do arquivo são lidos
  - Esta página pode conter um número grande de registros
    - se o próximo registro a ser recuperado estiver na mesma página já lida, evita-se um novo acesso ao disco

17

Solução por Árvores Binárias Paginadas (*Paged Binary Trees*)

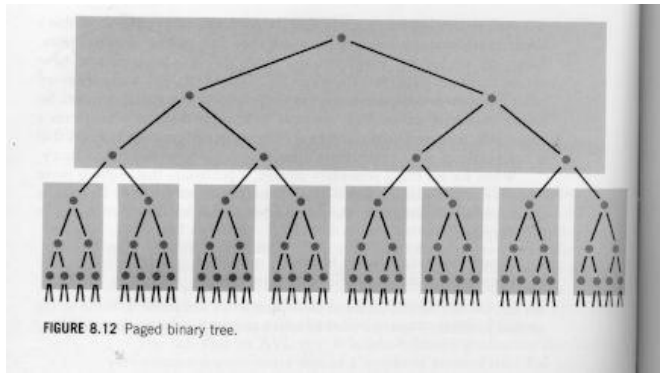


FIGURE 8.12 Paged binary tree.

Solução por Árvores Binárias Paginadas (*Paged Binary Trees*)

- Na árvore da figura anterior
  - qualquer um dos 63 registros pode ser acessado em, no máximo, 2 acessos
- Se a árvore é estendida com um nível de paginação adicional, adicionamos 64 novas páginas
  - podemos encontrar qualquer uma das 511 (64 x 7 + 63) chaves armazenadas fazendo apenas 3 *seeks*

19

Eficiência da árvore paginada

- Supondo que
  - cada página dessa árvore ocupa 8KB e armazena 511 pares chave-referência
  - cada página contém uma árvore completa perfeitamente balanceada
- Então, a árvore pode armazenar 134.217.727 chaves
  - $511 + 512*511 + 512^2*511 = 134.217.727$
  - qualquer delas pode ser acessada em, no máximo, 3 acessos ao disco

20

Eficiência da árvore

Pior caso para:

ABB completa, perfeitamente balanceada: $\log_2 (N+1)$

Versão paginada: $\log_{k+1} (N+1)$

onde **N** é o número total de chaves, e ***k*** é o número de chaves armazenadas em uma página

- ABB (pb): $\log_2 (134.217.727) = 27$ acessos
- Versão paginada : $\log_{511+1} (134.217.727) = 3$ acessos

Preços a pagar

- maior tempo na transmissão de grandes quantidades de dados, e, mais sério...
- necessário manter a organização da árvore
- Arvores-B são uma generalização da idéia de ABB paginada
  - Não são binárias
  - Conteúdo de uma página não é mantido como uma árvore