Algoritmos e Estruturas de Dados II

Profa. Debora Medeiros

Árvores B – Parte I

Introdução

Adaptado dos Originais de:

Ricardo J. G. B. Campello
Leandro C. Cintra
Maria Cristina F. de Oliveira

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 custoso (lento)
- Se o índice é grande e não cabe em memória principal, **busca binária** exige muitos acessos
  - 15 itens podem requerer $\log_2(15) \cong$ **4 acessos**
  - 1.000 itens podem requerer $\log_2(1000) \cong$ **10 acessos**
    - São números muito altos para relativamente poucos itens
- Manter ordenado?

Problema

- Um problema ainda mais crítico é o custo de **manter o índice ordenado** em disco
  - BB demanda reorganização de índices dinâmicos
  - Inserção ou remoção de item pode afetar todo o índice
- É necessária uma ED na qual a inserção e a remoção de registros tenha apenas efeitos locais
  - ED que não exija a reorganização total do índice
- **Solução** ?

Árvores Binárias de Busca

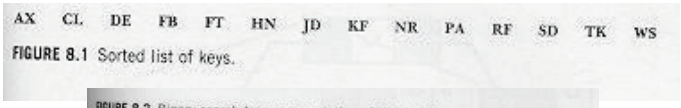


FIGURE 8.1 Sorted list of keys.

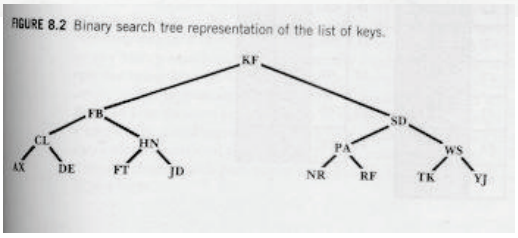


FIGURE 8.2 Binary search tree representation of the list of keys.

Conjunto de Chaves e Representação por ABB

Árvores Binárias de Busca

Representação de Parte da ABB

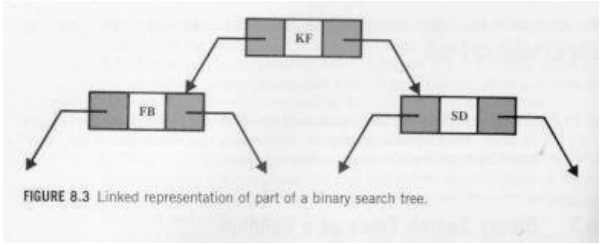


FIGURE 8.3 Linked representation of part of a binary search tree.

Registros de Índice com 3 campos: Chave e 2 "Ponteiros"

(na prática tem-se um "ponteiro" adicional para o registro no arq. principal)

Árvores Binárias de Busca

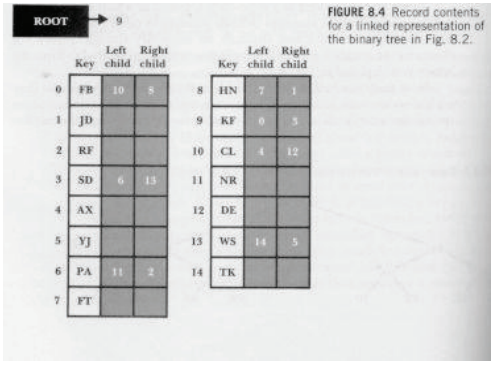


FIGURE 8.4 Record contents for a linked representation of the binary tree in Fig. 8.2.

Representação lógica do arquivo índice

ABBs: Vantagens

- Ordem lógica da ED não está associada à ordem lógica ou física dos registros no arquivo de índice
- Índice não precisa mais ser mantido ordenado:
  - O que interessa é recuperar a estrutura lógica da árvore, o que é possível com os "ponteiros" RRN
- Inserção de uma nova chave no arquivo
  - É necessário saber onde inserir esta chave na árvore
    - Busca?

ABBs: Vantagens

- Ordem lógica da ED não está associada à ordem lógica ou física dos registros no arquivo de índice
- Índice não precisa mais ser mantido ordenado:
  - O que interessa é recuperar a estrutura lógica da árvore, o que é possível com os "ponteiros" RRN
- Inserção de uma nova chave no arquivo
  - É necessário saber onde inserir esta chave na árvore
    - Busca é necessária, mas reorganização do arquivo não é

Exemplo: Inserção de LV

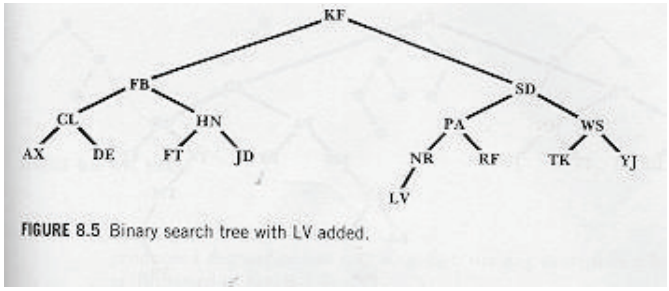
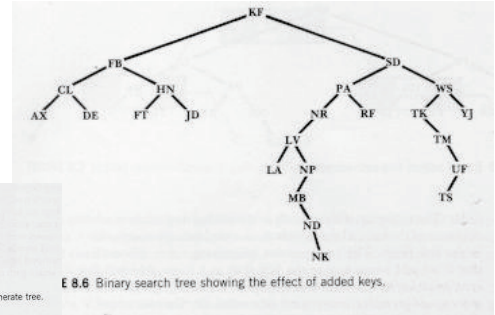


FIGURE 8.5 Binary search tree with LV added.

Como fica a representação lógica do arquivo de índice mostrada anteriormente?

Problema: Desbalanceamento

Inserção das chaves NP MB TM LA UF ND TS NK



E 8.6 Binary search tree showing the effect of added keys.

Pior Caso
(inserção em ordem alfabética)

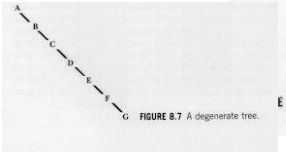


FIGURE 8.7 A degenerate tree.

Árvores Completamente Balanceadas e AVL

FIGURE 8.10 A completely balanced search tree.

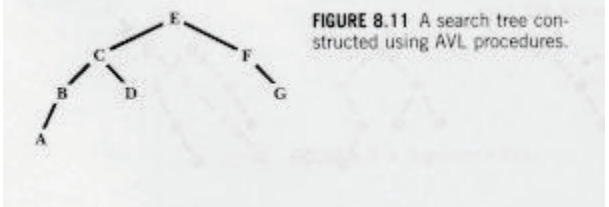


FIGURE 8.11 A search tree constructed using AVL procedures.

Árvores Completamente Balanceadas e 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
  - $O(\log_2(N))$ acessos são inaceitáveis
- Em resumo, no caso de índices dinâmicos:
  - AVLs resolvem apenas um dos dois principais problemas dos índices lineares ordenados:
    - nunca requerem a re-organização global do índice em uma modificação
    - no entanto, reduzem mas não resolvem o no. excessivo de acessos requerido em uma busca e, portanto, em qualquer outra operação

Árvores Binárias Paginadas

- A busca (seek) por uma posição específica do disco é lenta
- Mas uma vez encontrada a posição, pode-se ler uma grande quantidade registros seqüencialmente a um custo relativamente baixo
- Esta combinação de busca (*seek*) lenta e transferência rápida sugere a noção de **página**

Árvores Binárias Paginadas

- **Página:**
  - Um conjunto de registros fisicamente contíguos passíveis de serem recuperados em um único acesso
    - p. ex. registros ocupando um setor ou cluster
    - se o próximo registro a ser recuperado estiver na mesma página do anterior, evita-se um novo acesso ao disco
  - Pode conter um número grande de registros
- Refere-se aos arquivos organizados em páginas como **arquivos paginados** (paged files)

Árvores Binárias Paginadas

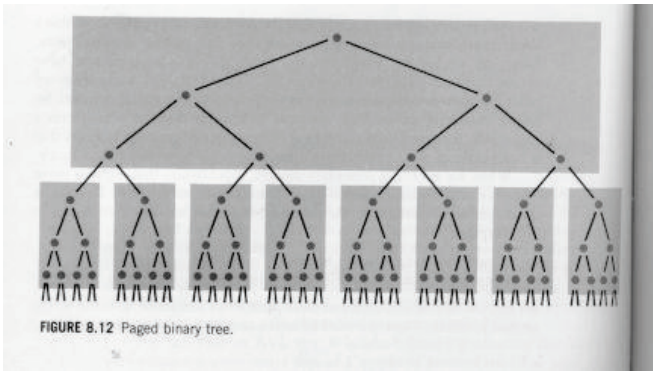


FIGURE 8.12 Paged binary tree.

Árvores Binárias Paginadas

- Na ABB da figura anterior:
  - qq. dos 63 itens é recuperado em, no máximo, **2 acessos** !
- Com um nível adicional:
  - ter-se-ão 64 novas páginas
  - o que representa 64 x 7 = 448 itens adicionais
  - qq. dos 511 itens é recuperado em, no máximo, **3 acessos** !
- Com outro nível adicional:
  - qualquer dos 4095 itens pode ser recuperado em no máx. **4 acessos** !
  - busca binária de 4095 itens pode demandar até **12 acessos**

Árvores Binárias Paginadas

- Outro Exemplo:
  - páginas com 511 itens
  - cada página contém uma árvore completamente balanceada
    - 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 + 256 = 511 itens
    - OBS: $511 = 2^9 - 1 = 2^0 + 2^1 + \ldots + 2^8$
- ABB com 3 níveis pode armazenar 134.217.727 itens
  - $511 + 512*511 + 512^2*511 = 134.217.727$
  - qq. item é recuperado em, no máximo, **3 acessos** !!!

Árvores Binárias Paginadas

- Desempenho de Pior caso:
  - ABB completamente balanceada: $\log_2 (N+1)$ acessos
  - Versão paginada: $\log_{k+1} (N+1)$ acessos
    - k = no. de itens em uma página (k=1 para a versão não paginada)
    - N = no. de itens
  - Exemplo anterior:
    - ABB (k=1): $\log_2 (134.217.727)$ = **27 acessos**
    - Versão paginada (k = 511): $\log_{511+1} (134.217.727)$ = **3 acessos**

Árvores Binárias Paginadas

- É simples construir uma árvore paginada se todo o conjunto de chaves é conhecido antes de iniciar a construção
  - Toma-se a chave mediana para obter uma divisão balanceada, de forma recursiva
- Porém, é complicado se as chaves são recebidas em uma seqüência aleatória

Árvores Binárias Paginadas

- **Exemplo:** C S D T A M P I B W N G U R K E H O L J Y Q Z F X V

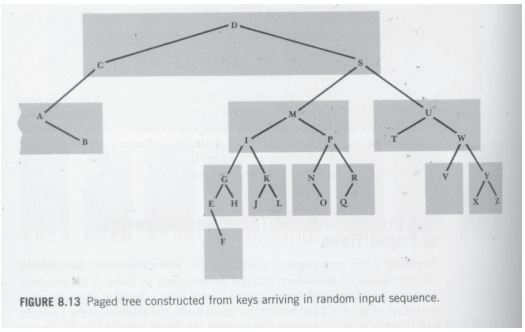


FIGURE 8.13 Paged tree constructed from keys arriving in random input sequence.

Árvores Binárias Paginadas

- No exemplo anterior:
  - Construção *top-down,* a partir da raiz
  - Sempre que uma chave é inserida, a árvore dentro da página é reajustada para manter o balanceamento interno
- **Problema**:
  - Chaves pequenas no topo, como C e D, podem acabar desbalanceando a árvore de forma definitiva
    - A árvore do exemplo não está tão ruim, mas o que aconteceria se as chaves fossem fornecidas em ordem alfabética?

Árvores Binárias Paginadas

- **Problema**:
  - É necessário manter o balanceamento após inserções e remoções
    - com a restrição dos itens estarem agrupados em páginas...
  - Questões :
    - como garantir que as páginas formem uma árvore balanceada ?
      - p. ex., como impedir o agrupamento de chaves como C, D e S no exemplo anterior ?
    - como garantir que cada página contenha um no. mínimo de chaves ?

Árvores Binárias Paginadas

- Em resumo, como garantir:
  - que as páginas formem uma árvore balanceada ?
  - que cada página contenha um no. mínimo de chaves ?
- Rotações AVL não satisfazem as necessidades acima
  - não se pode rotar páginas inteiras como se fossem itens individuais
    - páginas contêm múltiplas chaves e múltiplos descendentes
  - rotações tradicionais implicariam a modificação de várias páginas
    - não possuem apenas efeitos locais
- **Solução**: **Árvores B !**

Bibliografia

- **M. J. Folk and B. Zoellick, *File Structures: A Conceptual Toolkit*, Addison Wesley, 1987.**