

Árvores B – Parte I

Introdução

Adaptado e Estendido dos Originais de:

Leandro C. Cintra

Maria Cristina F. de Oliveira

A Invenção da B-Tree

- Bayer and McCreight, 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

2

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

3

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 ?**

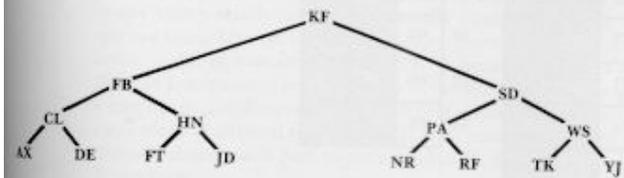
4

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.

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

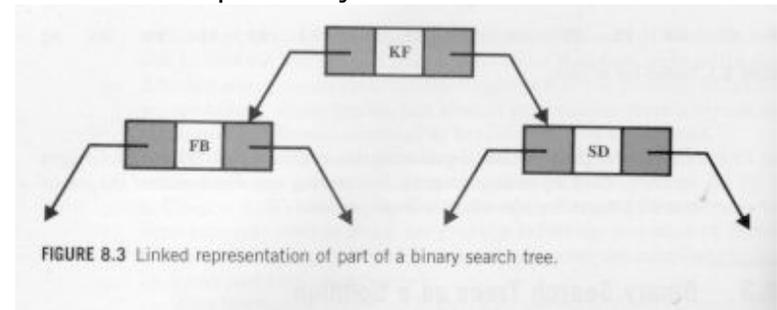


FIGURE 8.3 Linked representation of part of a binary search tree.

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.

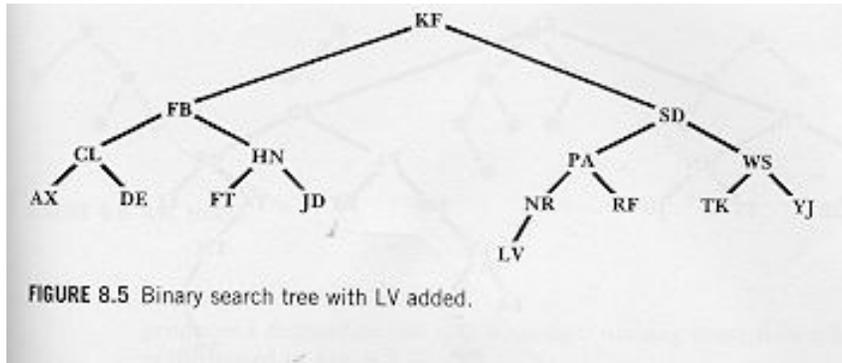
Key	Left child	Right child
0 FB	10	8
1 JD		
2 RF		
3 SD	6	13
4 AX		
5 YJ		
6 PA	11	2
7 FT		
8 HN	7	1
9 KF	0	3
10 CL	4	12
11 NR		
12 DE		
13 WS	14	5
14 TK		

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 é necessária, mas reorganização do arquivo não é

Exemplo: Inserção de LV

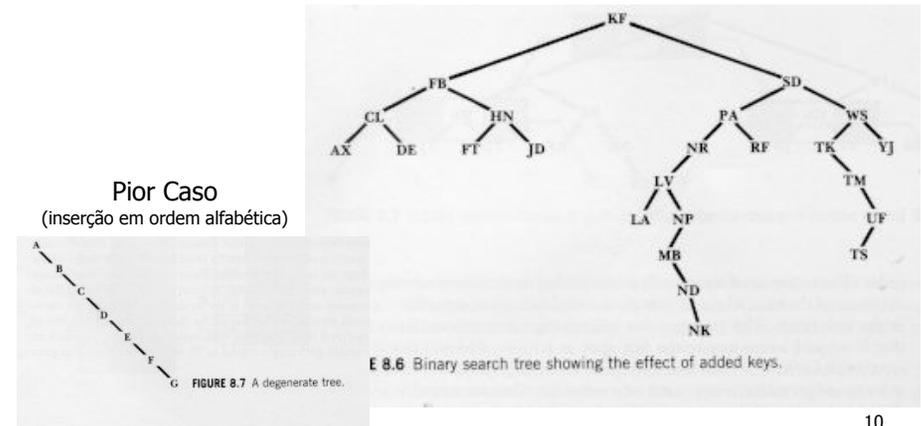


Como fica a representação lógica do arquivo de índice mostrada anteriormente?

9

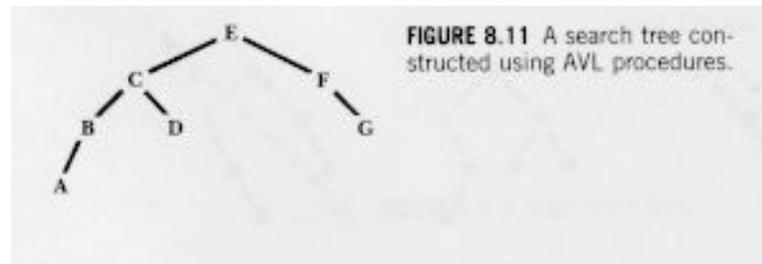
Problema: Desbalanceamento

Inserção das chaves NP MB TM LA UF ND TS NK



10

Árvores Completamente Balanceadas e AVL



11

Árvores Completamente Balanceadas e AVL

- A eficiência de ABBs exige mantê-las balanceadas
- Árvores completamente balanceadas:
 - Busca demanda $\approx \log_2(N+1)$ comparações de chaves no pior caso
 - Corresponde à altura da árvore
 - Para $N = 1.000.000$ chaves, busca percorre até **20 níveis**
 - Mas só é viável computacionalmente se árvore for estática
- No caso dinâmico, balanceamento requer árvores AVL
 - Busca demanda $\approx 1.44 * \log_2(N+2)$ comparações no pior caso
 - Para $N = 1.000.000$ chaves, busca percorre até **28 níveis**

12

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
 - 20 ou 28 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

13

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**

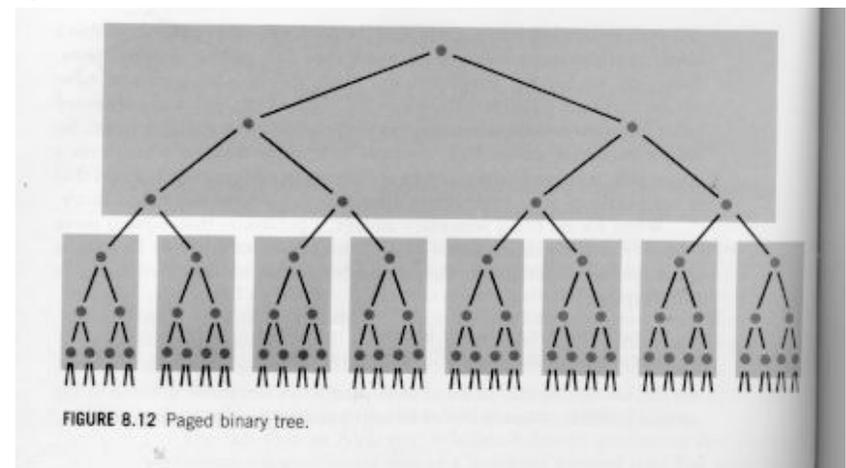
14

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)

15

Árvores Binárias Paginadas



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 $64 \times 7 = 448$ 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**

17

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: $511 = 2^9 - 1 = 2^0 + 2^1 + \dots + 2^8$
- ABB com 3 níveis pode armazenar 134.217.727 itens
 - $511 + 512 \times 511 + 512^2 \times 511 = 134.217.727$
 - qq. item é recuperado em, no máximo, **3 acessos** !!!

18

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

19

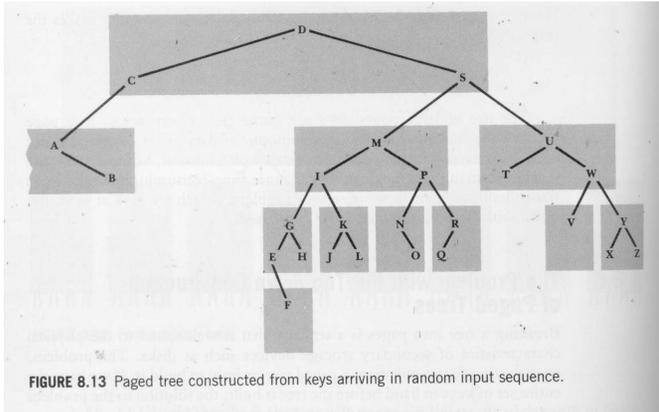
Árvores Binárias Paginadas

- 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

20

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



21

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?

22

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 ?

23

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 !**

24



Exercícios

1. Ilustre graficamente uma ABB completamente balanceada com 15 chaves numéricas inteiras agrupadas em páginas de $k=3$ chaves internamente também balanceadas.
2. Repita o exercício anterior assumindo que as chaves são inseridas de forma aleatória.
3. Estime a quantidade de acessos no pior caso de busca por um item em um arquivo com 16.777.215 itens organizado como uma ABB completamente balanceada não paginada
4. Repita o exercício 3 para uma ABB do tipo AVL.
5. Repita o exercício 3 assumindo uma paginação de $k=63$

25



Bibliografia

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

26