Árvore B (Parte I)

Estruturas de Dados

Engenharia de Computação

Baseado nos slides da Prof. Dra. Graça Nunes, do ICMC-USP

Problema

- Cenário até então
  - Acesso a disco é caro (lento)
  - Pesquisa binária é útil em índices ordenados...
  - mas com índice grande que não cabe em memória principal, pesquisa binária exige muitos acessos a disco
  - Exemplo: 15 itens podem requerer 4 acessos, enquanto 1.000 itens podem requerer até 11 acessos

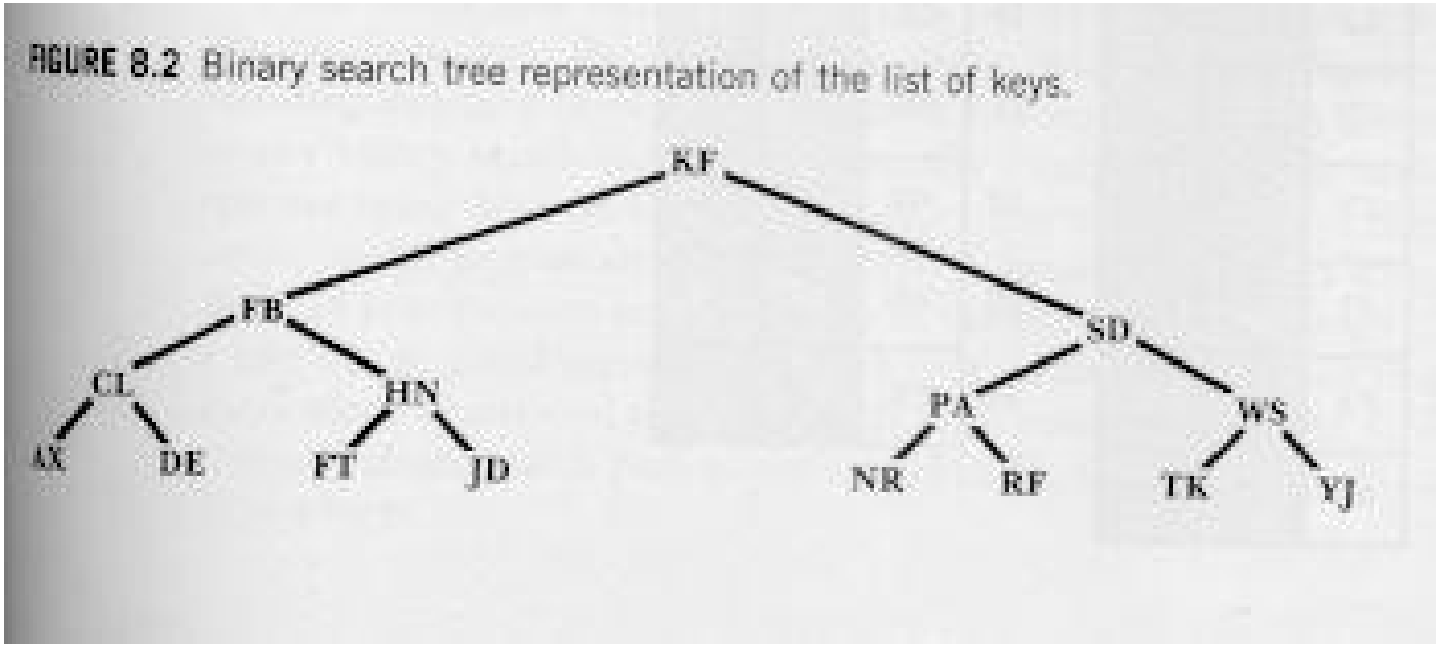
Problema

- Cenário
  - Manter em disco um índice ordenado para busca binária tem custo proibitivo
    - Inserir ou eliminar, mantendo o arquivo ordenado custa muito caro.
  - Necessidade de método com inserção e eliminação com apenas efeitos locais, isto é, que 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.



Solução: árvores binárias de busca?

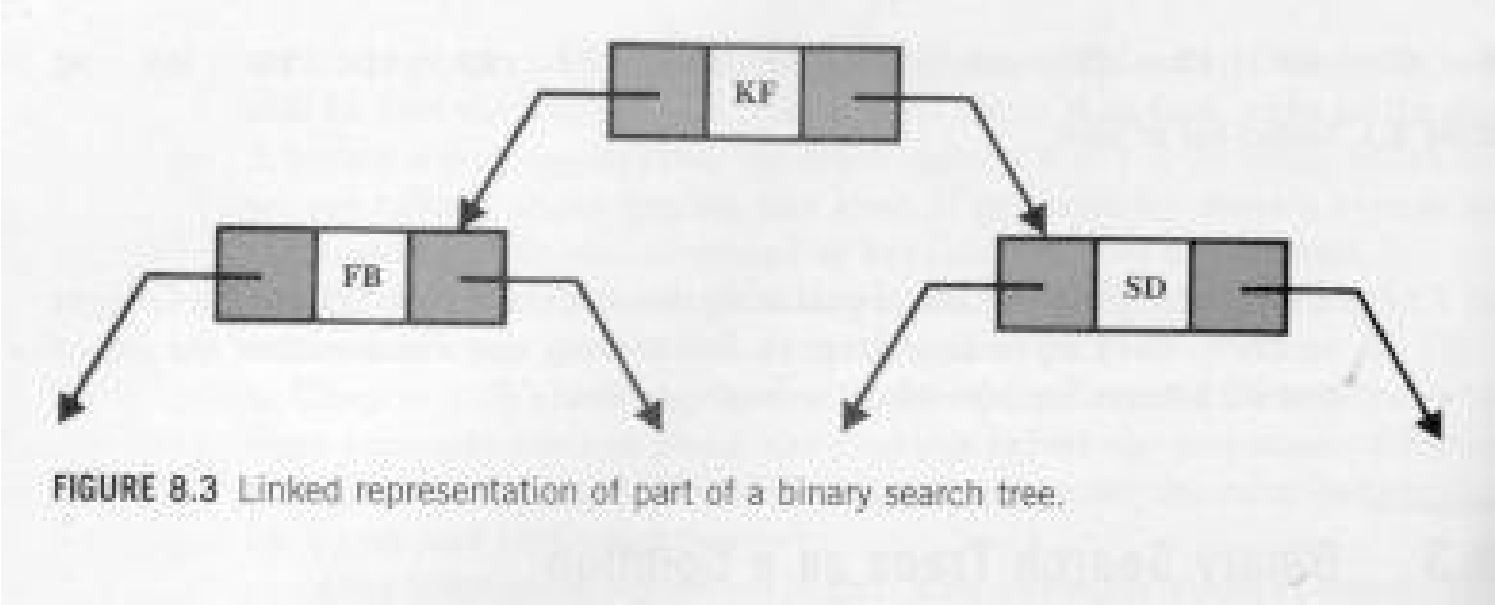


FIGURE 8.3 Linked representation of part of a binary search tree.

Representação da árvore no arquivo

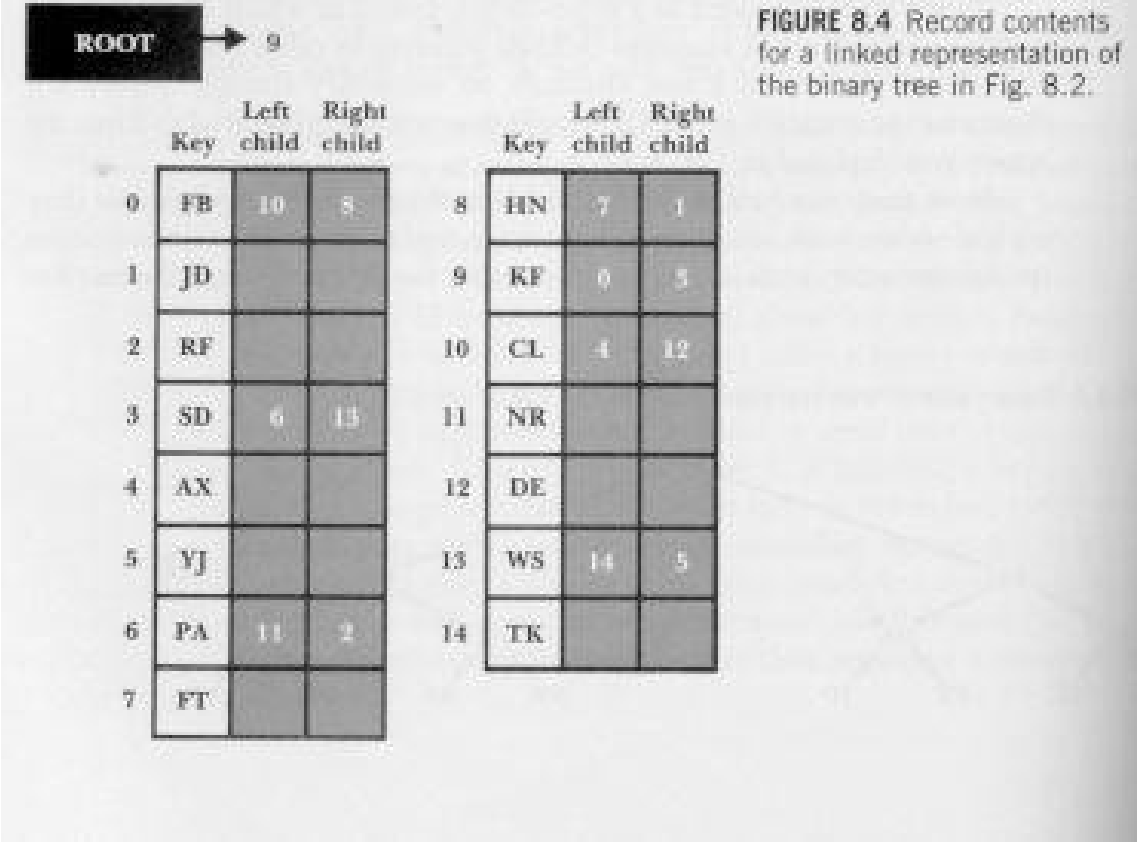


FIGURE 8.4 Record contents for a linked representation of the binary tree in Fig. 8.2.

- Registros (tam. fixo) são mantidos em arquivo, e ponteiros (**esq** e **dir**) indicam onde estão os registros filhos.

Quais as vantagens de se utilizar ABB's?

- Ordem lógica dos registros ≠ ordem física no arquivo
  - Ordem lógica: dada por ponteiros **esq** e **dir**
  - Registros não precisam estar fisicamente ordenados
- Inserção de uma nova chave no arquivo
  - É necessário saber onde inserir
  - Busca pelo registro é necessária, mas reorganização do arquivo não

Inserção da chave LV

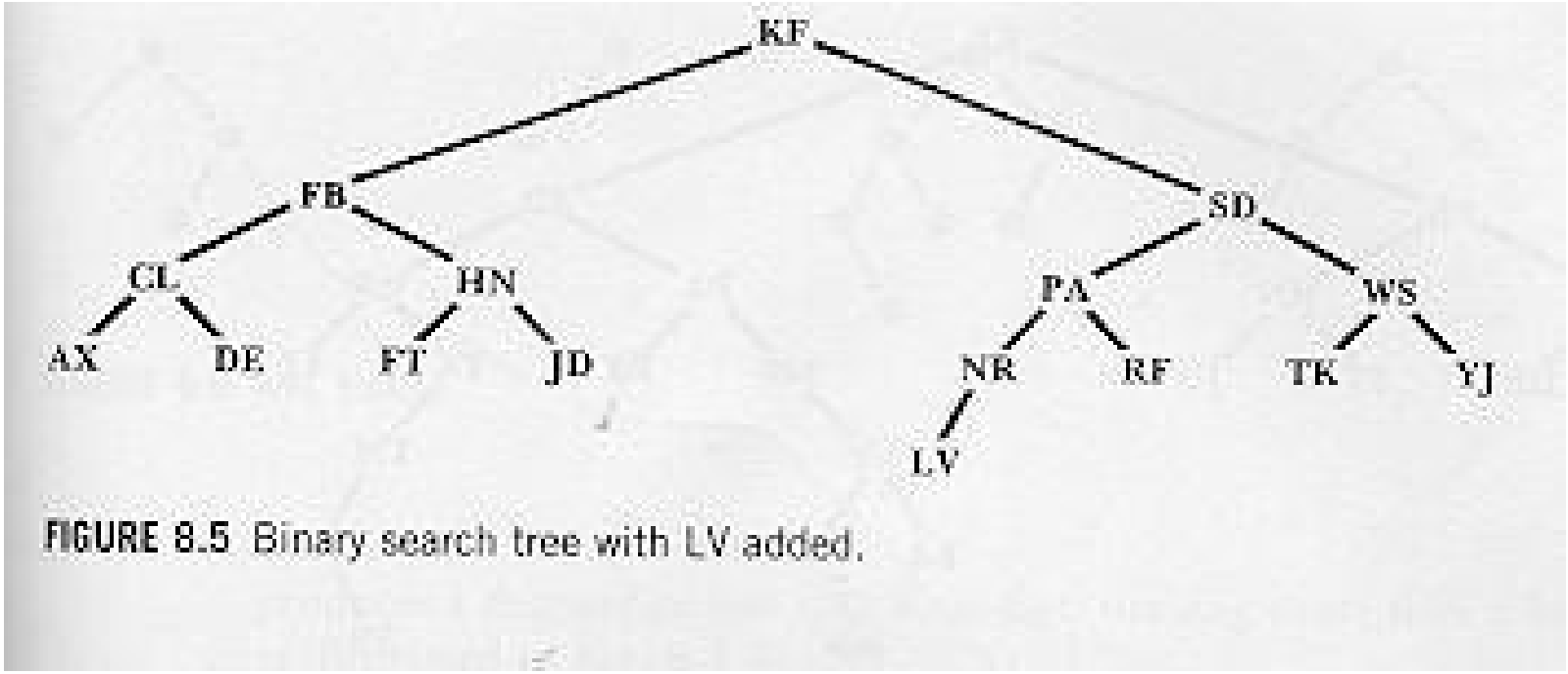
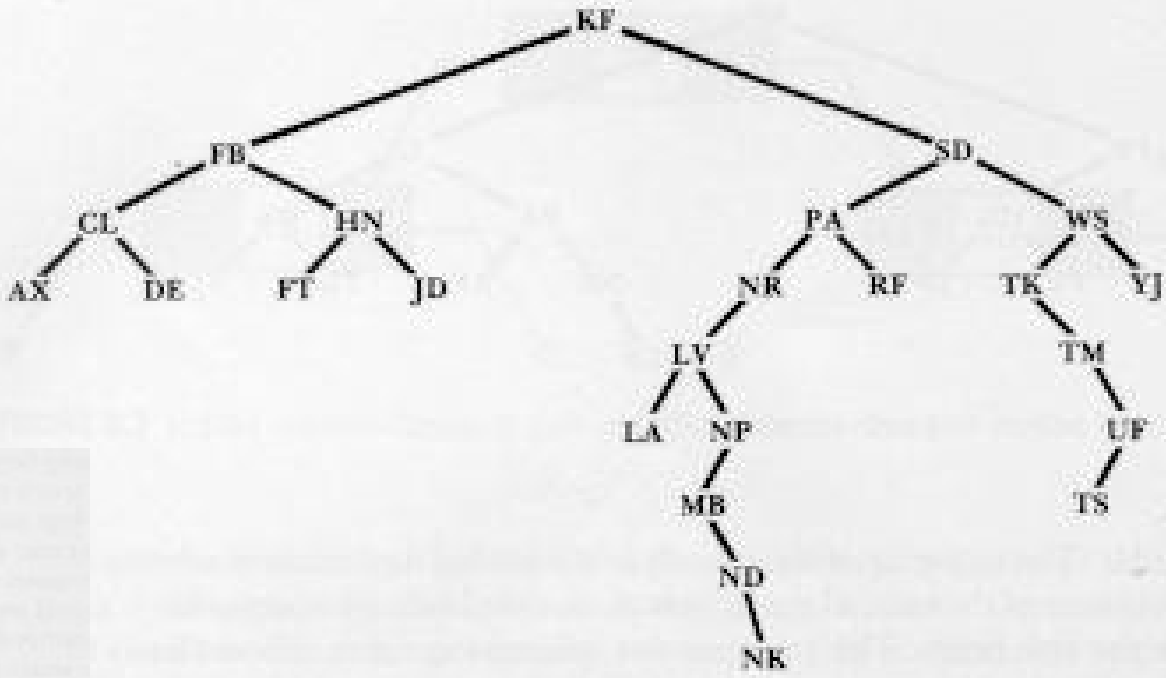


FIGURE 8.5 Binary search tree with LV added.

Problema: desbalanceamento

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

Situação indesejável: inserção em ordem alfabética



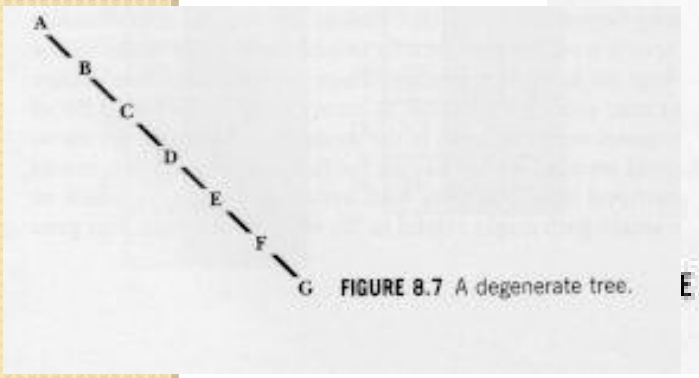
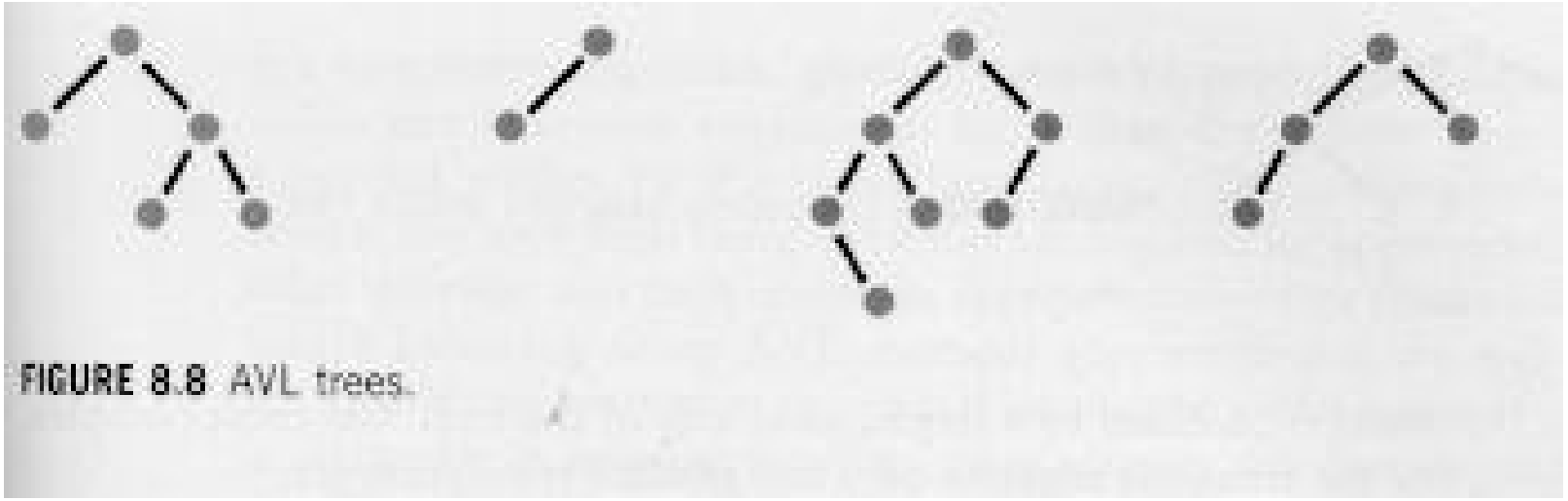


FIGURE 8.7 A degenerate tree.

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

Solução por árvores AVL



- Para todo nó: as alturas de suas duas sub-árvores diferem de, no máximo, 1.

Solução por árvores AVL

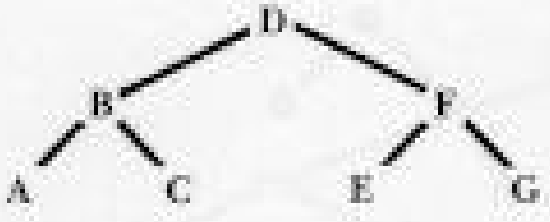


FIGURE 8.10 A completely balanced search tree.

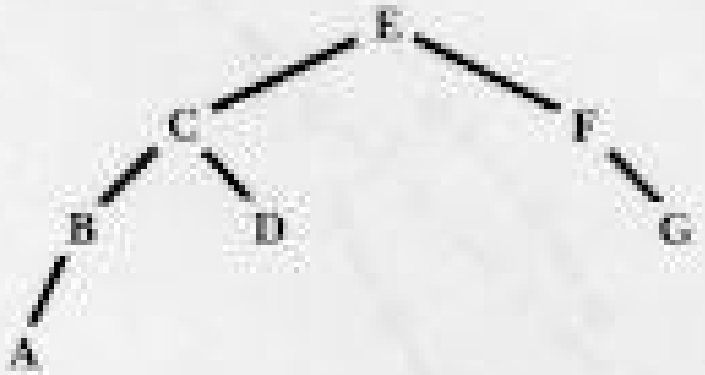


FIGURE 8.11 A search tree constructed using AVL procedures.

Solução por árvores AVL

- Árvores binárias de busca balanceadas garantem eficiência
  - AVLs
  - Busca no pior caso
  - Arvore binária perfeitamente balanceada: altura da árvore, ou seja, $\log_2(N+1)$
- Exemplo: com 1.000.000 chaves
  - Árvore binária perfeitamente balanceada: busca em até 20 níveis

Solução por árvores AVL

- Problema
  - .. Se chaves em memória secundária, ainda há muitos acessos!
    - .. 20 são inaceitáveis!
- .. Até agora...
  - .. Árvores binárias de busca dispensam ordenação dos registros 
  - .. Mas número excessivo de acessos .. 

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

- Solução
  - Paginação
    - A busca (*seek*) por uma posição específica do disco é muito lenta
    - Mas, uma vez na posição, pode se ler uma grande quantidade de registros sequencialmente a um custo relativamente pequeno

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

- Noção de **página** em sistemas paginados
  - Feito o *seek*, todos os registros de 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

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

8 páginas filhas

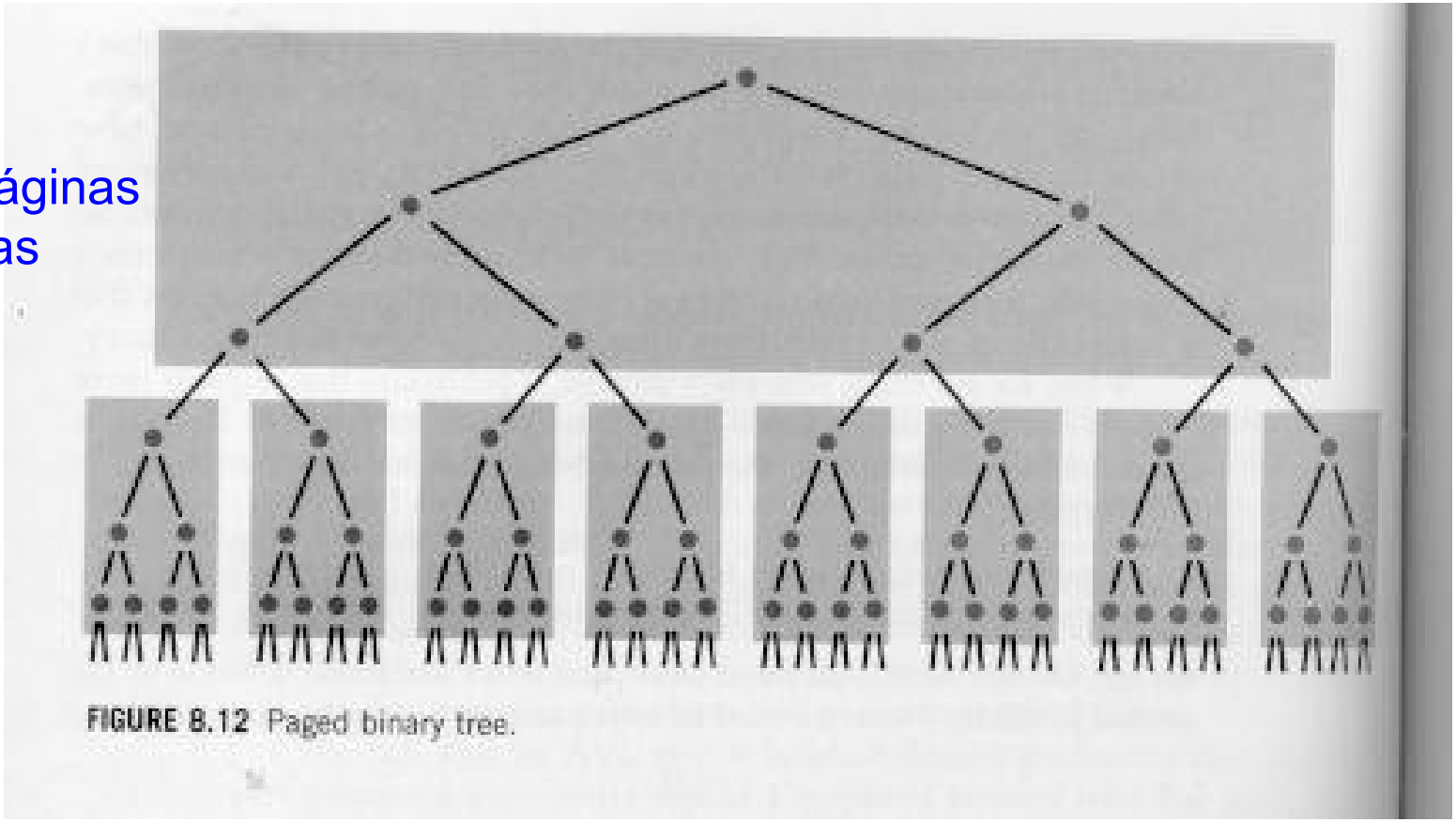


FIGURE 8.12 Paged binary tree.

7 registros por página (por seek);; Nível da árvore 2 e ordem 8 = (8+1)*7 registros, se completa

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

Qualquer um dos 63 registros pode ser acessado em, no máximo, 2 acessos!

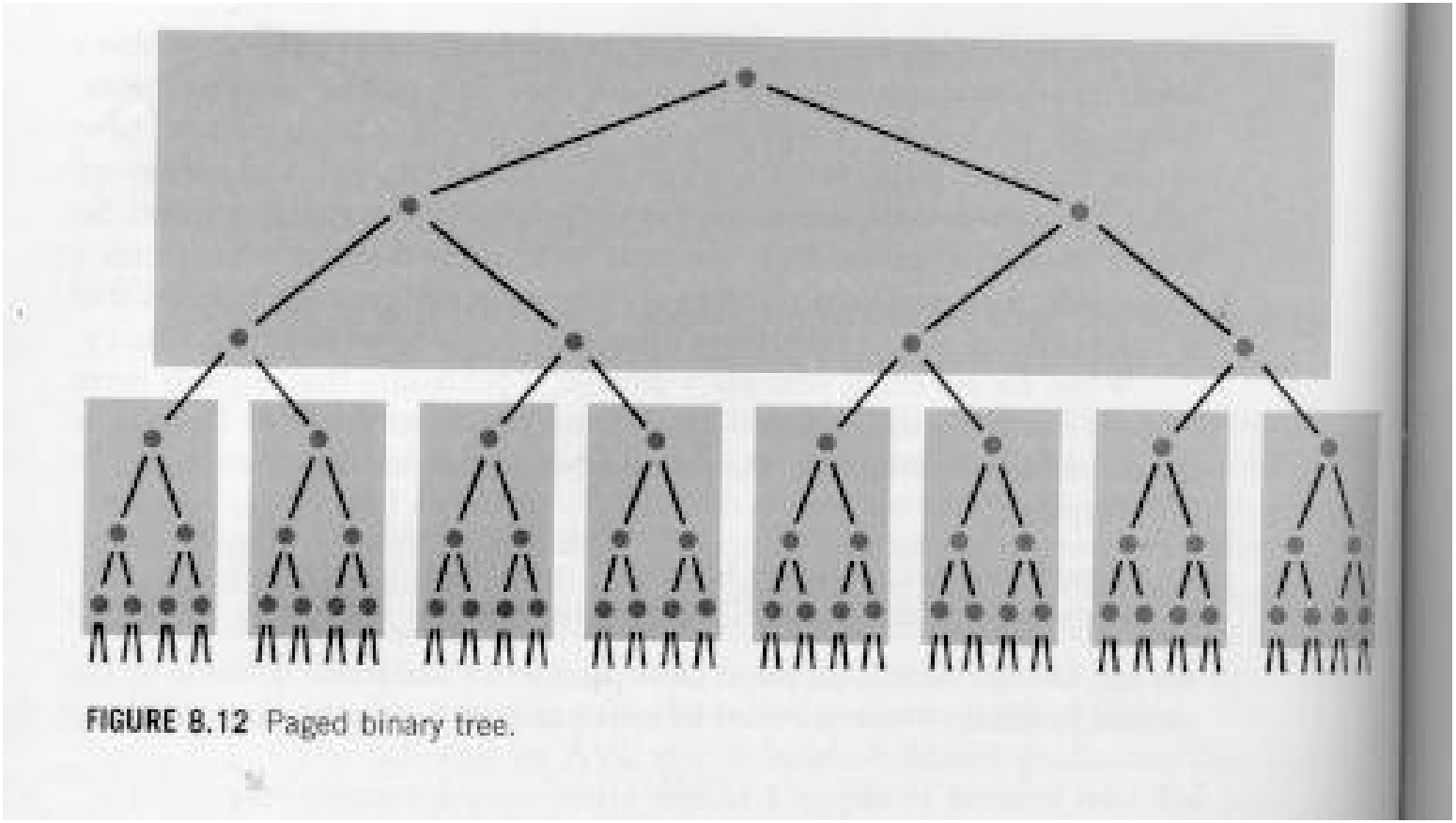


FIGURE 8.12 Paged binary tree.

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

- 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 fazendo apenas 3 *seeks*

8 páginas filhas

8^2 páginas filhas

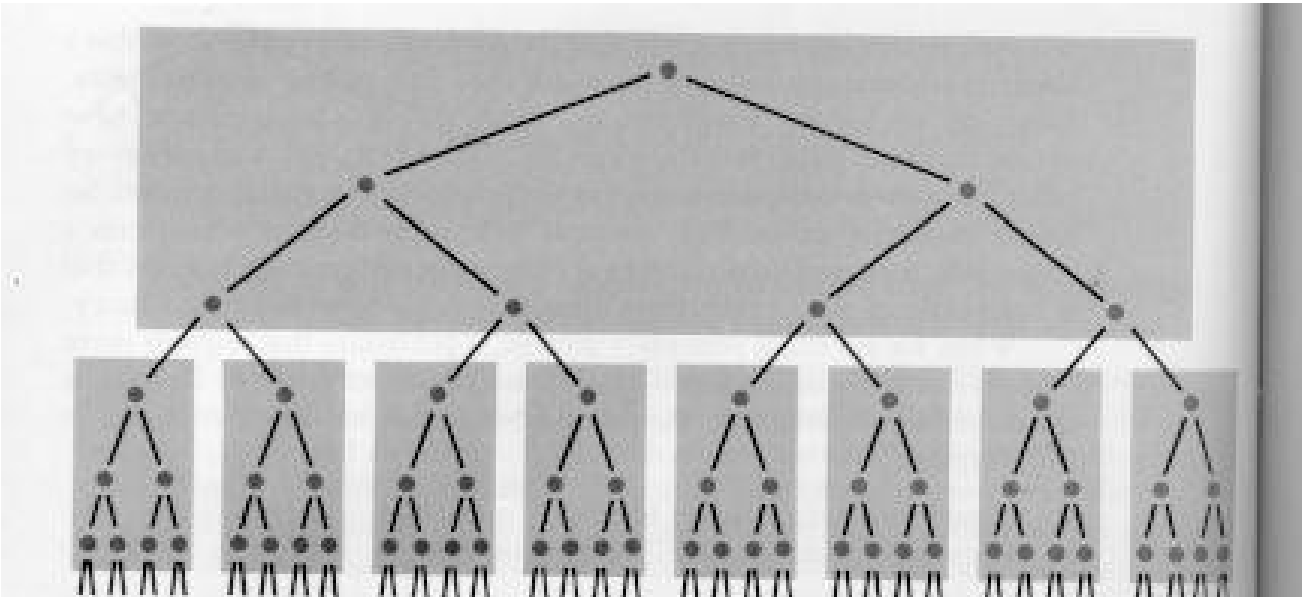


FIGURE 8.12 Paged binary tree.

Eficiência da árvore paginada

- Mais realisticamente....supondo que
  - Cada página de uma árvore ocupa 8KB e armazena 511 chaves
  - Cada página contém uma árvore completa perfeitamente balanceada com 9 níveis (=$\log_2$ 512)
  - A árvore de 3 níveis pode armazenar 134.217.727 chaves, ou seja, (511+(512*511)+(512*512*511))

Eficiência da árvore paginada

- Eficiência na busca
  - ABB completa, perfeitamente balanceada: $\log_2 (N+1)$
  - Versão paginada: $\log_{k+1} (N+1)$
  - em que **N** é o número total de chaves, e **k** é o número de chaves armazenadas em uma página
    - ABB (perfeitamente balanceada): $\log_2(134.217.727) = 27$ acessos
    - Versão paginada : $\log_{511+1}(134.217.727) = 3$ acessos

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

- **Preços a pagar**
  - Menos seeks, mas maior tempo na transmissão de dados
  - Necessário manter a organização da árvore
  - Pense como seria a construção da árvore

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

- **Preços a pagar**
  - Menos seeks, mas maior tempo na transmissão de dados
  - Necessário manter a organização da árvore
  - Pense como seria a construção da árvore

Construção *Top-Down* de ABB Paginadas

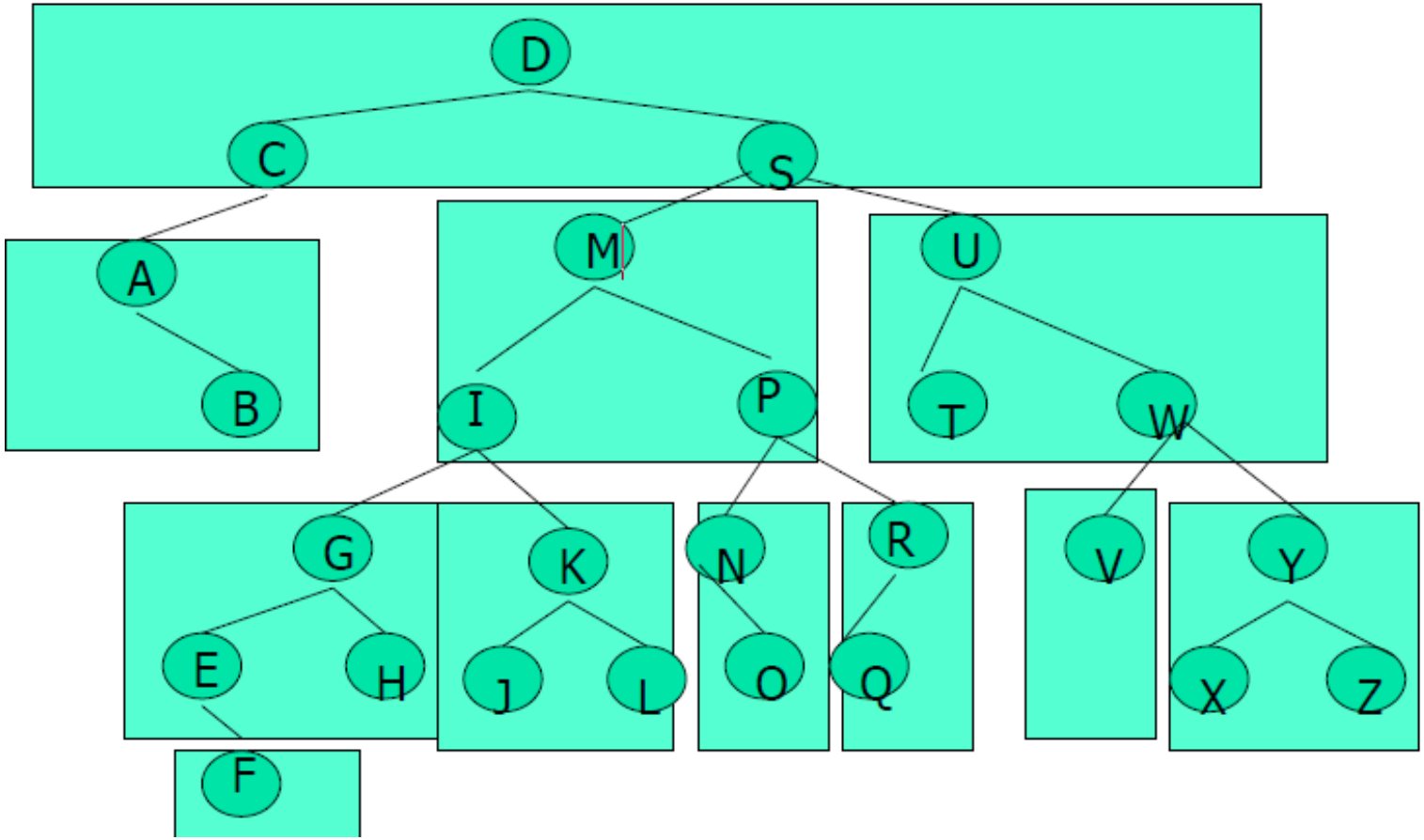
- **Se tivermos todas as chaves a priori:**
  - **Ordenar as chaves e inserir de forma a manter balanceada (mediana será a raiz, etc.)**
- **Se a construção for dinâmica .. chaves são inseridas em ordem aleatória**
  - **Conforme inserimos, podemos manter a ABB de cada página balanceada (ou seja, uma AVL), rotacionando-a conforme necessário**

Exemplo

- Assuma:
  - uma ABB paginada com páginas de até 3 chaves por página
  - a seguinte sequência de chaves a inserir:

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

Exemplo



Exemplo

- Observe:
  - A construção *top-down* faz com que as primeiras chaves fiquem na página-raiz.
  - No exemplo, C e D (consecutivas e no início do alfabeto) não são boas escolhas para a raiz, pois fatalmente fazem a árvore tender à direita.
- E se rotacionássemos as páginas (como fazemos com os nós)?
  - Tente formular um algoritmo para isso
    - Muito complexo!

Problemas a resolver

1. Como garantir que as chaves da raiz sejam boas separadoras, tal que dividam o conjunto mais ou menos ao meio?
2. Como evitar agrupamento de certas chaves (como C, D e S) na página raiz?
3. Como garantir que cada página contenha um certo número mínimo de chaves?

A invenção das árvores-B

- Arvores-B são uma generalização da ideia de ABB paginada
  - Não são binárias
  - Conteúdo de uma página não é mantido como uma árvore
  - A construção é *bottom-up*
- Um pouco de história
  - 1972: Bayer and McGreight publicam o artigo *Organization and Maintenance of Large Ordered Indexes*
  - 1979: árvores-B viram praticamente padrão em sistemas de arquivos de propósito geral

Árvore-B

- Características
  - Completamente balanceadas
  - criação bottom-up (em disco)
  - nós folhas $\rightarrow$ nó raiz
- Inovação
  - Não é necessário construir a árvore a partir do nó raiz, como é feito para árvores em memória principal e para as árvores anteriores

Construção *Bottom-up*

- Consequências
  - Chaves ‘’erradas’’ não são mais alocadas no nó raiz
    - Elimina as questões em aberto de chaves separadoras e de chaves extremas
- Não é necessário tratar o problema de desbalanceamento