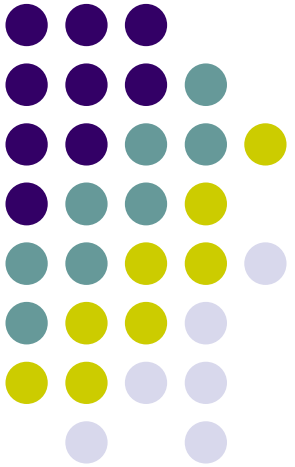
Árvores B

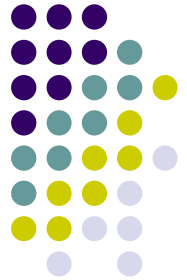
Prof. Leandro C. Fernandes

Estruturas de Dados

Adaptado de: *Leandro C. Cintra*
e *M.C.F. de Oliveira*



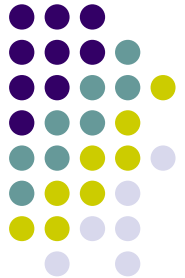
A invenção da árvore-B



- Bayer and McGreight, 1972, publicaram o artigo: "*Organization and Maintenance of Large Ordered Indexes*"

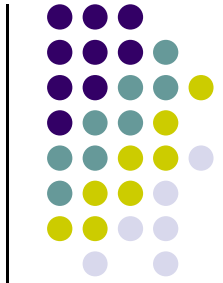
- Em 1979, o uso de árvores-B para manutenção de índices de bases de dados já era praticamente padrão em sistemas de arquivos de propósito geral

Problema



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

Problema (cont.)



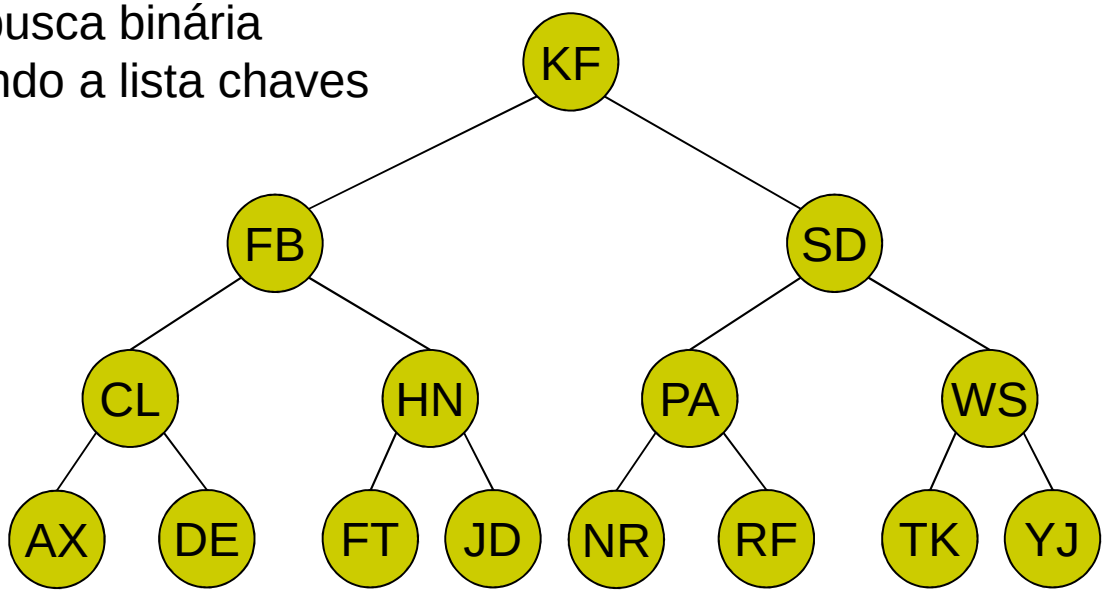
- O custo de manter em disco um índice ordenado de forma a permitir busca binária é proibitivo
- É necessário um método no qual a inserção e a eliminação de registros tenha apenas efeitos locais, isto é, não exija a reorganização total do índice

Solução:
árvores binárias de busca?

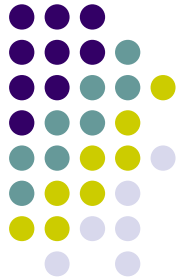
AX	CL	DE	FB	FT	HN	JD	KF	NR	PA	RF	SD	TK	WS	YJ

Lista ordenada de chaves

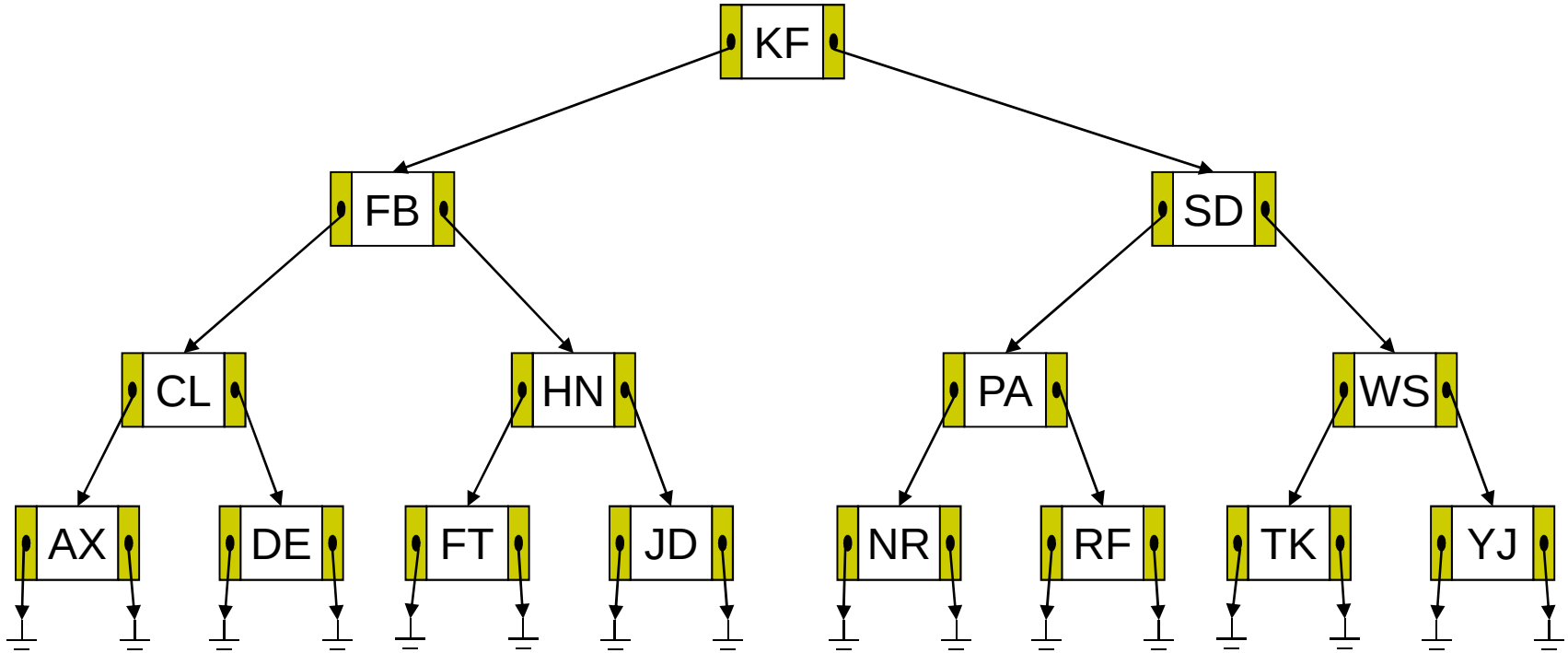
Árvore de busca binária
representando a lista chaves



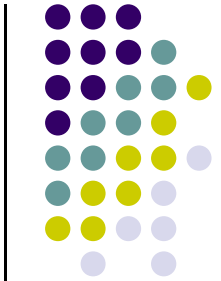
Solução:
árvores binárias de busca



Representação encadeada da
árvore de busca binária



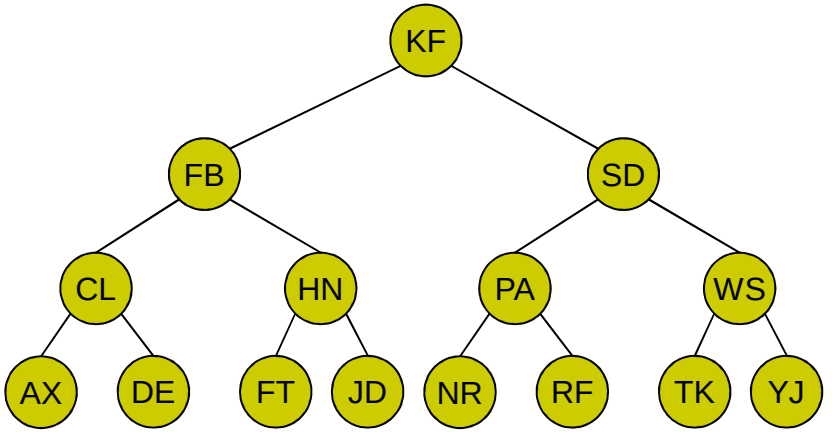
Representação da árvores em disco



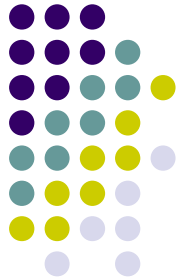
	key	*esq*	*dir*
0	FB	10	8
1	JD		
2	RF		
3	SD	6	13
4	AX		
5	YJ		
6	PA	11	2
7	FT		

	key	*esq*	*dir*
8	HN	7	1
9	KF	0	3
10	CL	4	12
11	NR		
12	DE		
13	WS	14	5
14	TK		

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

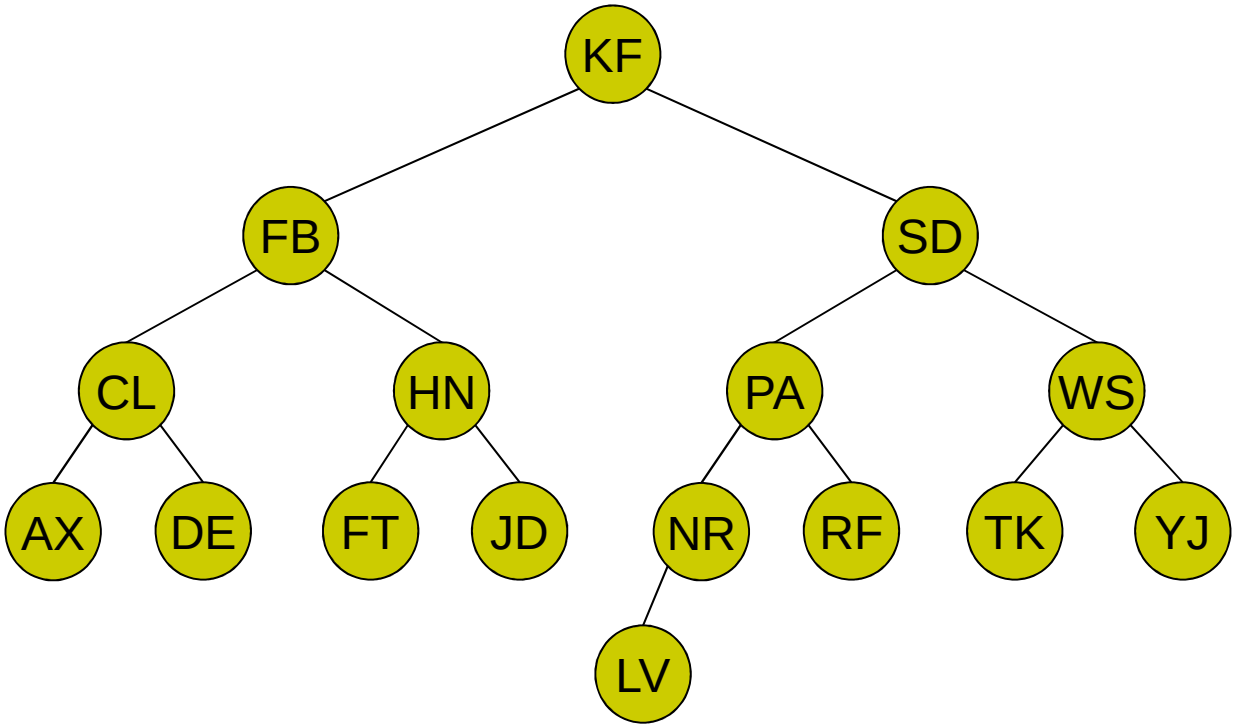


Vantagens



- 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

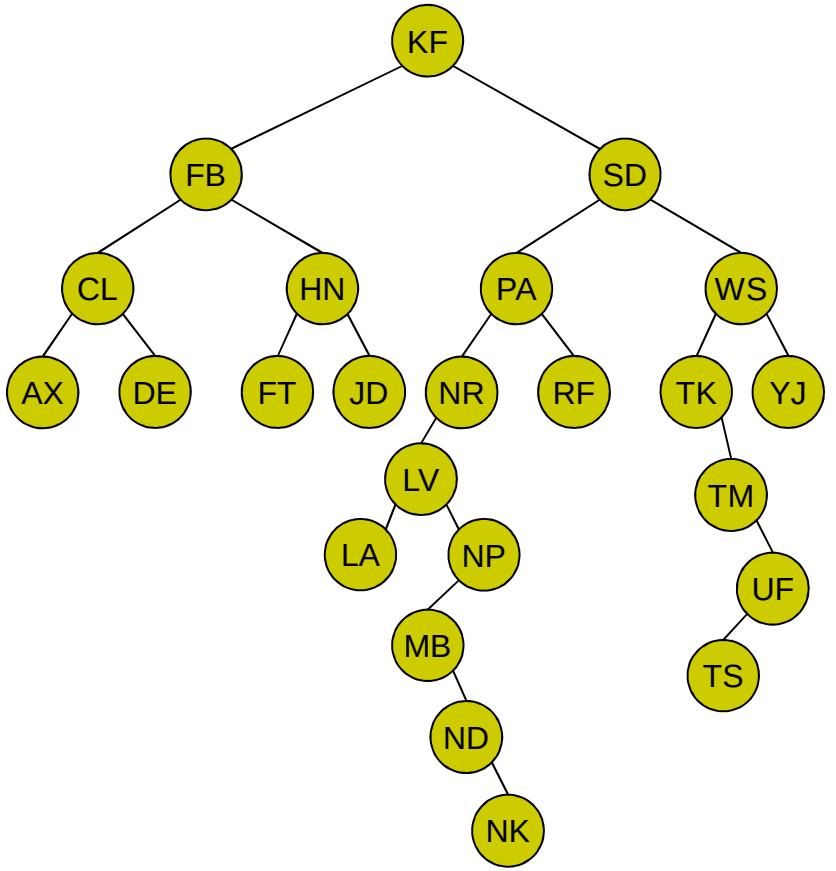
Inserção da chave LV



Problema: desbalanceamento

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

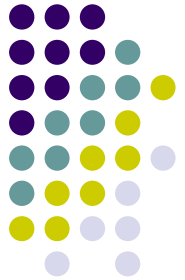
- Situação indesejável:
  Inserção em ordem alfabética
  - Desgeneração da árvore



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

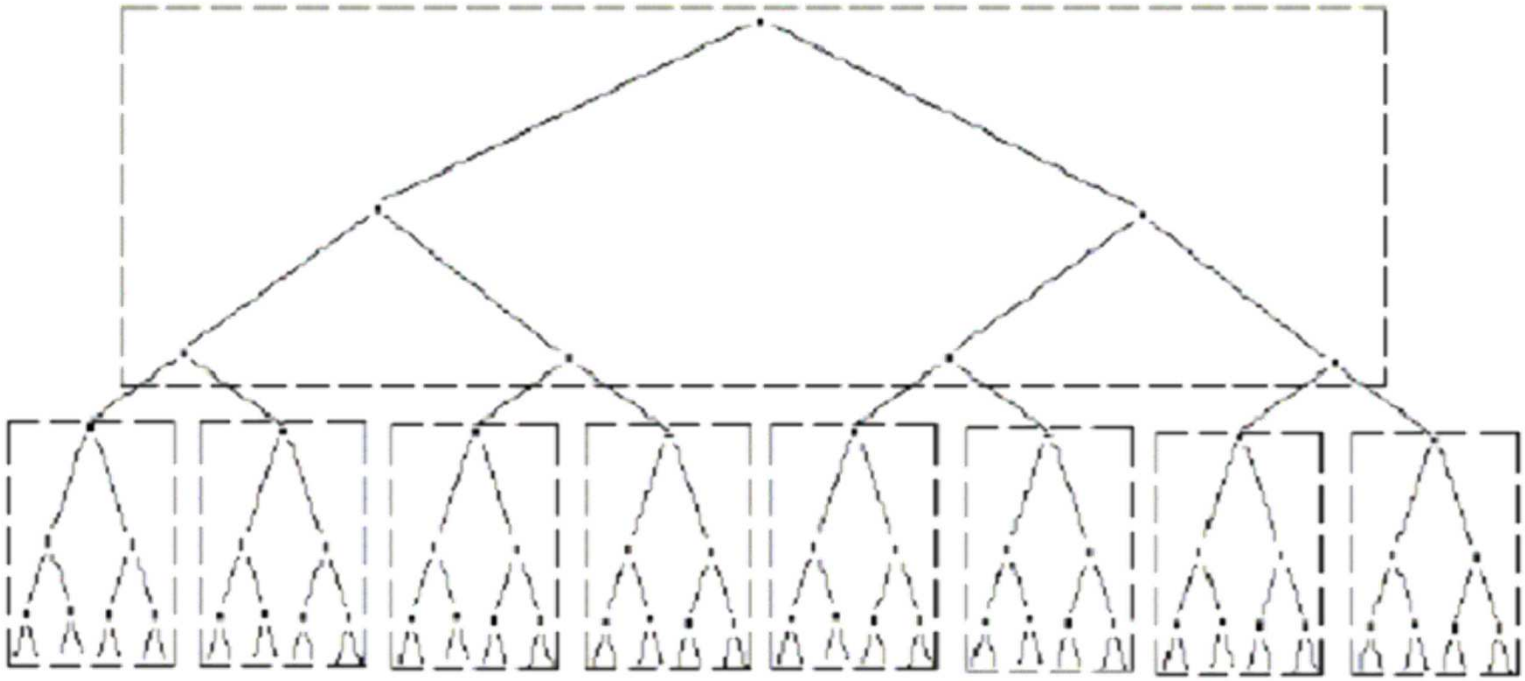


- Entretanto, se as chaves estão em memória secundária, qualquer procedimento que exija mais do que 5 ou 6 acessos para localizar uma chave é altamente indesejável
  - 20 ou 28 seeks são inaceitáveis
- Árvores balanceadas são uma boa alternativa para o problema da ordenação
  - não requerem a ordenação 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
- 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

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

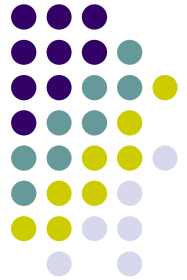


Árvore binária paginada

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*

Eficiência da árvore paginada

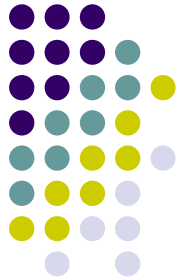


- Supondo que:
  - cada página dessa árvore ocupa 8KB e armazena 511 pares chave-referência
  - cada página contém uma árvore completa perfeitamente balanceada
- Então, a árvore pode armazenar 134.217.727 chaves
  - $512^0*511 + 512^1*511 + 512^2*511 = 134.217.727$
  - qualquer delas pode ser acessada em, no máximo, 3 acessos ao disco

Eficiência da árvore paginada

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

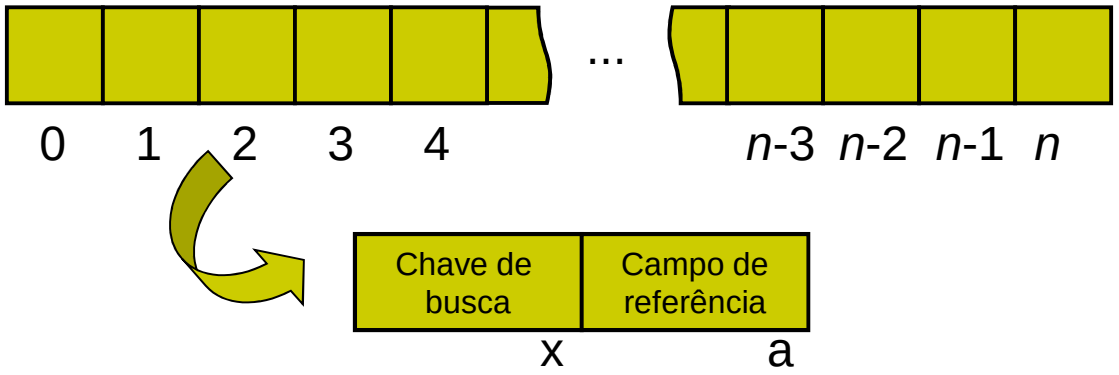
Preços a pagar



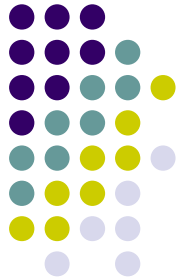
- Maior tempo na transmissão de grandes quantidades de dados, e, mais sério... necessidade manter a organização da árvore
- Árvores-B são uma generalização da idéia de ABB paginada
  - Não são binárias
  - Conteúdo de uma página não é mantido como uma árvore

Características Gerais

- Organizar e manter um índice para um arquivo de acesso aleatório altamente dinâmico
- Índice
  - *n* elementos (*x*,*a*) de tamanho fixo

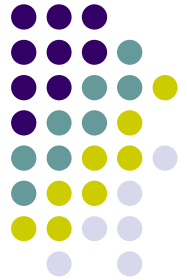


Características Gerais



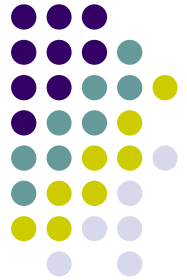
- Índice
  - extremamente volumoso
- Buffer-pool pequeno
  - apenas uma parcela do índice pode ser carregada em memória principal
  - operações baseadas em disco
- Desempenho
  - proporcional a $\log_k n$ ou melhor, onde
    - n tamanho do índice e k tamanho da página em disco

Construção *Top-Down* de árvores paginadas



- É simples construir uma árvore paginada se todo o conjunto de chaves é conhecido antes de iniciar a construção
  - Inicia-se pela chave do meio para obter uma árvore balanceada
- Porém, é complicado se as chaves são recebidas em uma seqüência aleatória

Construção *Top-Down* de árvores paginadas



- Ordem: 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

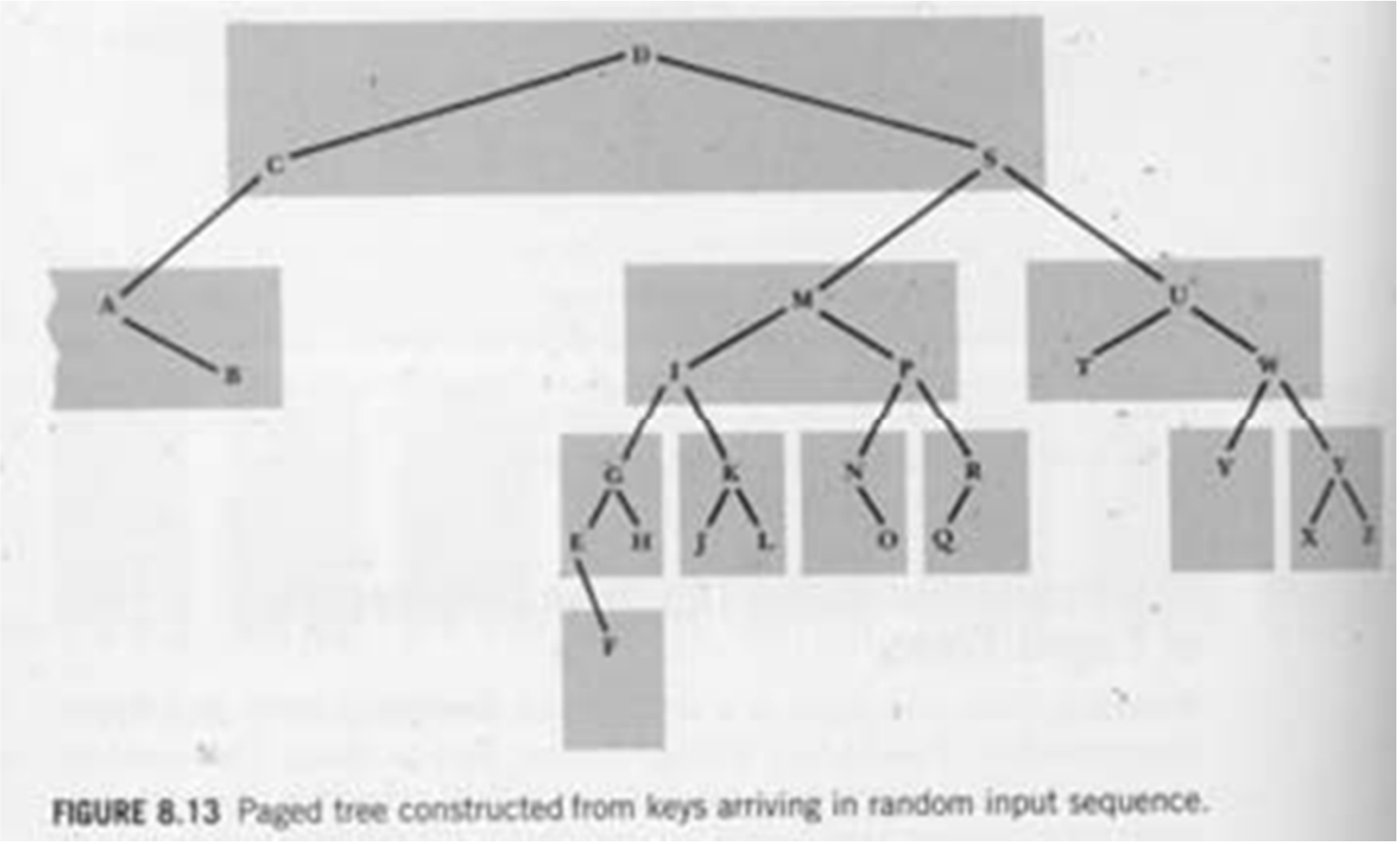
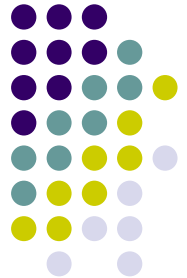


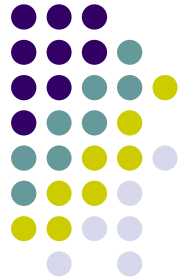
FIGURE 8.13 Paged tree constructed from keys arriving in random input sequence.

Construção *Top-Down* de árvores paginadas



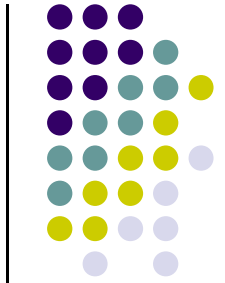
- Na figura anterior, a construção foi feita de cima para baixo, a partir da raiz.
- Sempre que uma chave é inserida a árvore dentro da página sofre uma rotação, sempre que necessário, para manter o balanceamento
- Construção a partir da raiz implica em que as chaves iniciais estarão necessariamente na raiz
  - C e D não deveriam estar no topo, pois acabam desbalanceando a árvore de forma definitiva
- Esta árvore não está tão ruim, mas o que aconteceria se as chaves fossem fornecidas em ordem alfabética?

Construção *Top-Down* de árvores paginadas



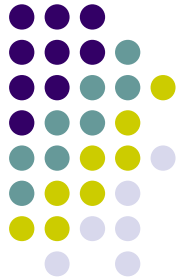
- Questões:
  - como garantir que as chaves na página raiz são boas separadoras, i.e., dividem o conjunto de chaves de maneira balanceada ?
  - como impedir o agrupamento de chaves que não deveriam estar na mesma página (como C, D e S, por exemplo)
  - como garantir que cada página contenha um número mínimo de chaves ?

Árvore B



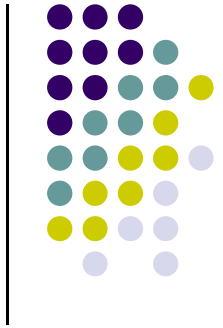
- Características:
  - balanceada
  - *bottom-up* para a criação (em disco)
    - nós folhas → nó raiz
- Inovação:
  - não é necessário construir a árvore a partir do nó raiz, como é feito para árvores em memória principal e para as árvores anteriores

Construção *Bottom-Up*



- Conseqüências
  - 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 usando algoritmos de reorganização da árvore
    - na árvore-B, as chaves na raiz da árvore emergem naturalmente

Características



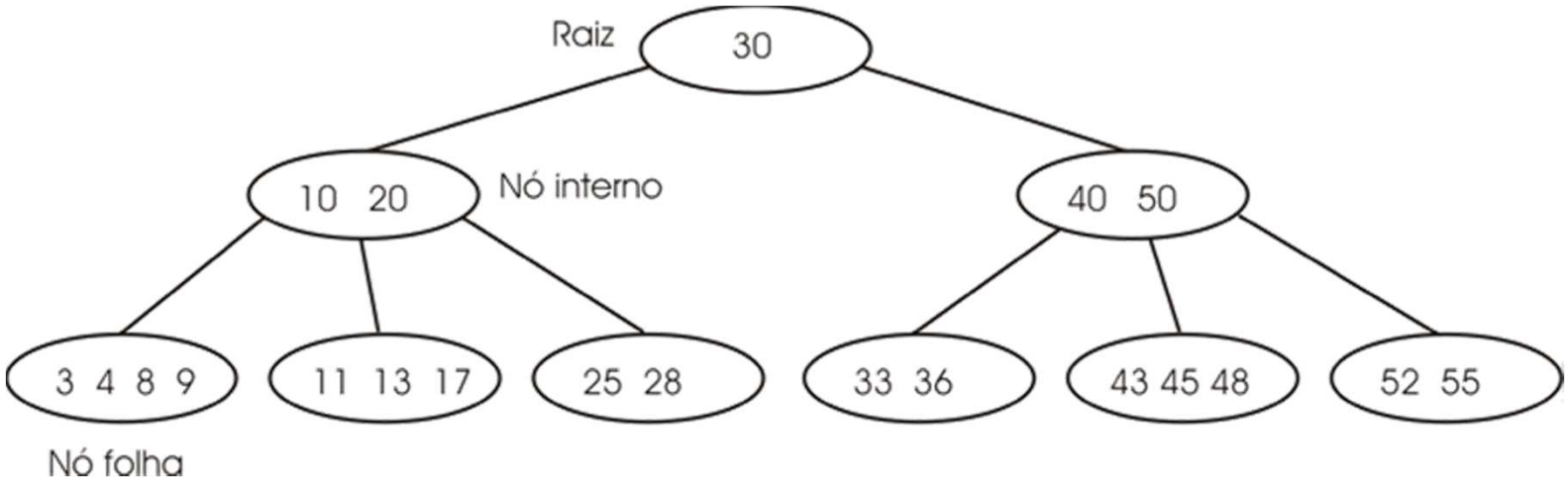
- Nó (= página de disco)
  - seqüência ordenada de chaves
  - conjunto de ponteiros
    - número de ponteiros = número de chaves + 1
  - não há uma árvore explícita dentro de uma página (ou nó da árvore)
- Observações
  - número máximo de ponteiros é igual ao número máximo de descendentes de um nó
  - nós folhas não possuem filhos, e seus ponteiros são nulos

Ordem e propriedades associadas

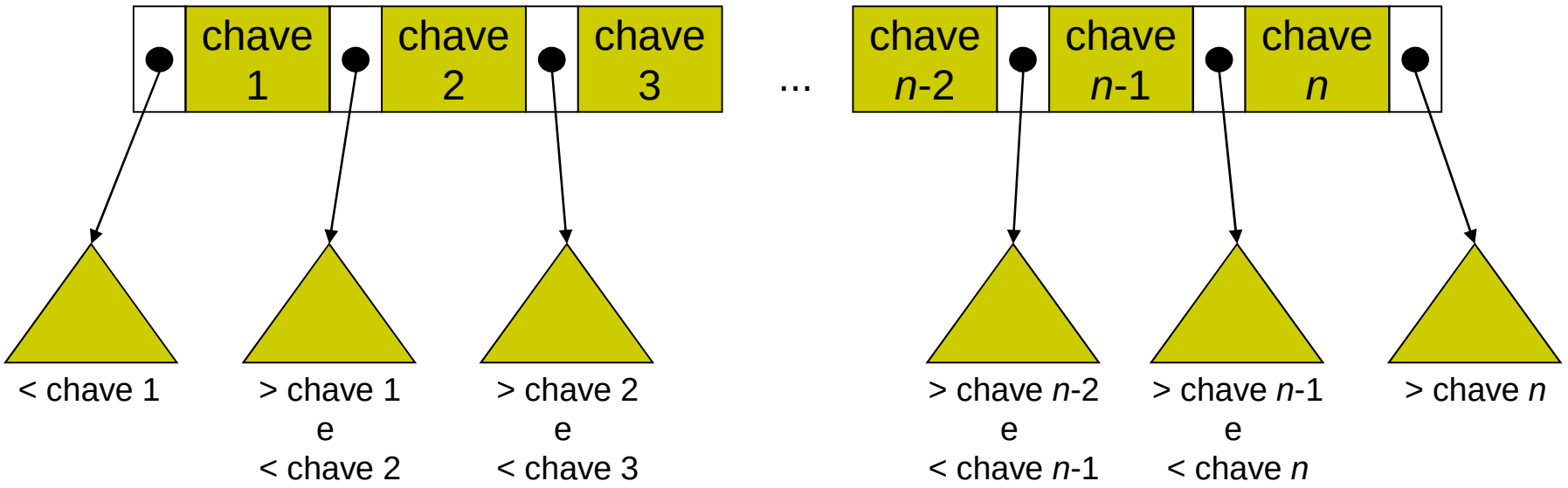
- Uma árvore ordem *d* possui entre *d* e *2d* chaves em seus nós
  - Os valores de *d* e *2d* representam os números mínimo e máximo, respectivamente, de chaves que um determinado nó da árvore pode armazenar.
- As propriedades de uma árvore B são:
  - Cada nó interno diferente da raiz possui no mínimo **d** registros e **d+1** filhos;
  - Cada nó interno diferente da raiz possui no máximo **2d** registros e **2d+1** filhos;
  - O nó raiz pode conter entre **1** e **2d** registros e indicação para nodos filhos entre **2** e **2d+1** apontadores; e
  - Todos os nós folhas estão no mesmo nível.

Exemplo

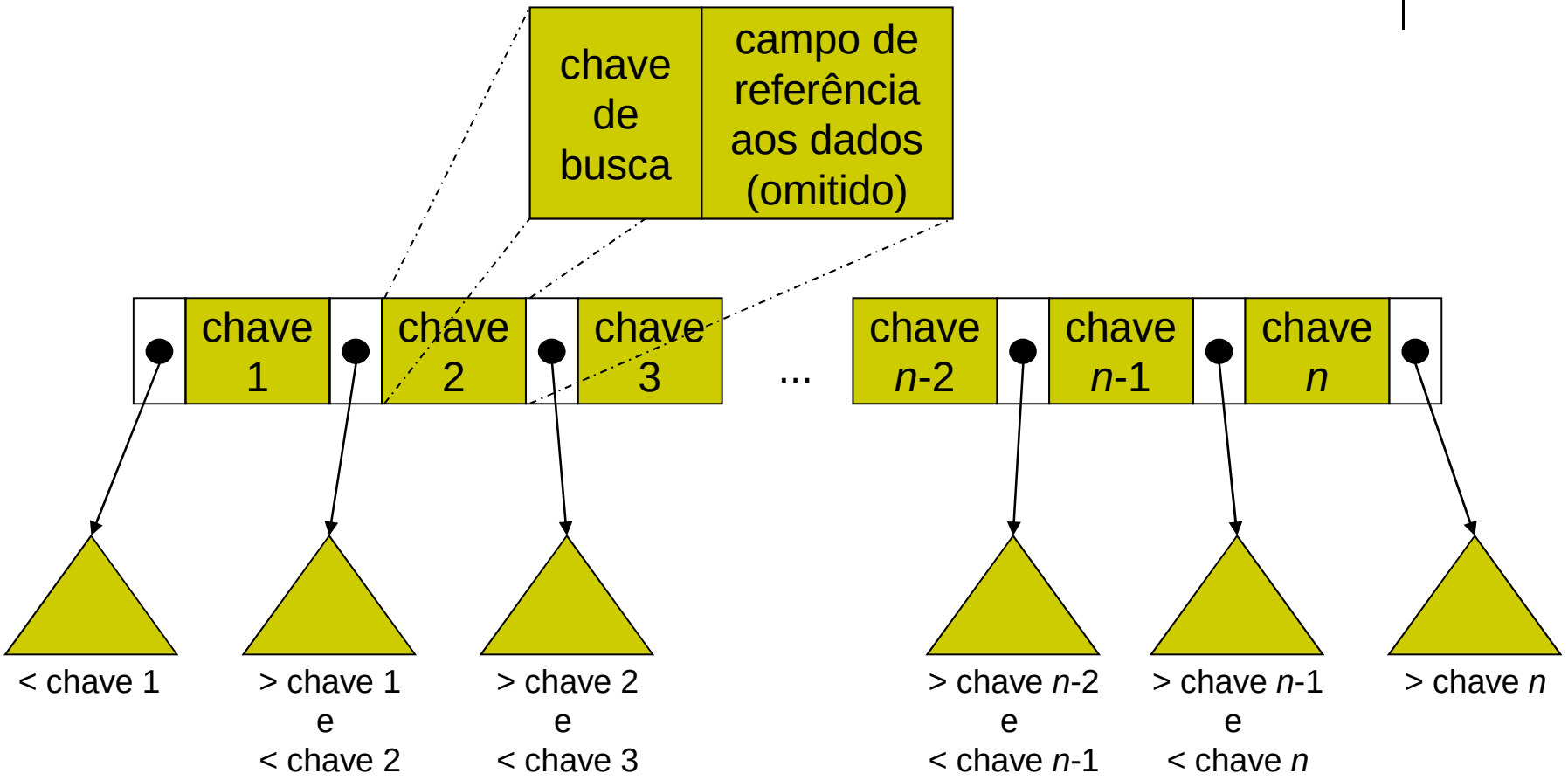
- A árvore B é de ordem 1 (Árvore 2-3)
  - O nó raiz pode conter no mínimo 1 chave e 2 apontadores e no máximo 4 chaves com 5 apontadores.
  - Os nós internos poderão conter no mínimo 2 chaves com 3 apontadores e no máximo 4 chaves com 5 apontadores.



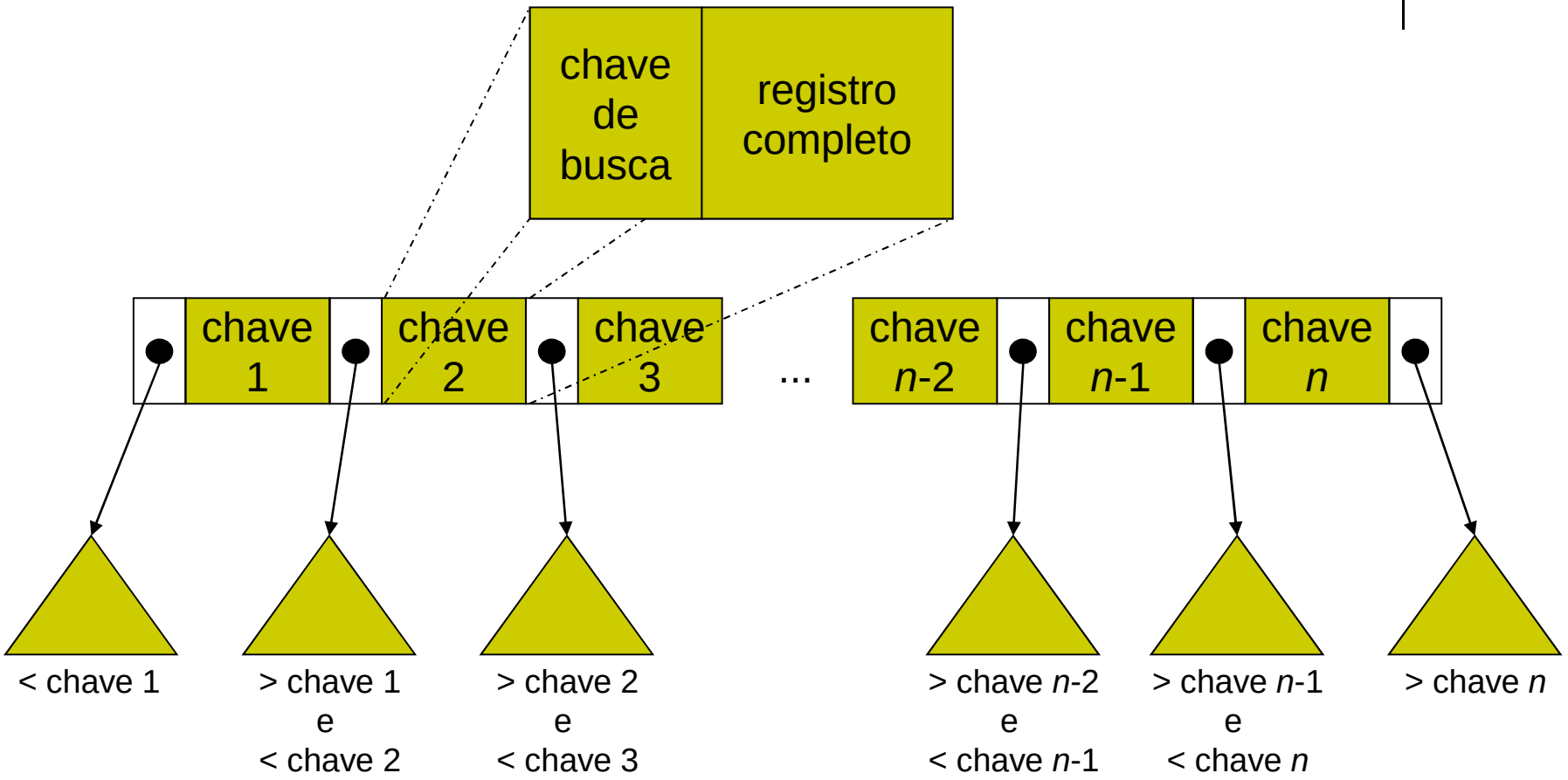
Estrutura Lógica de um Nó



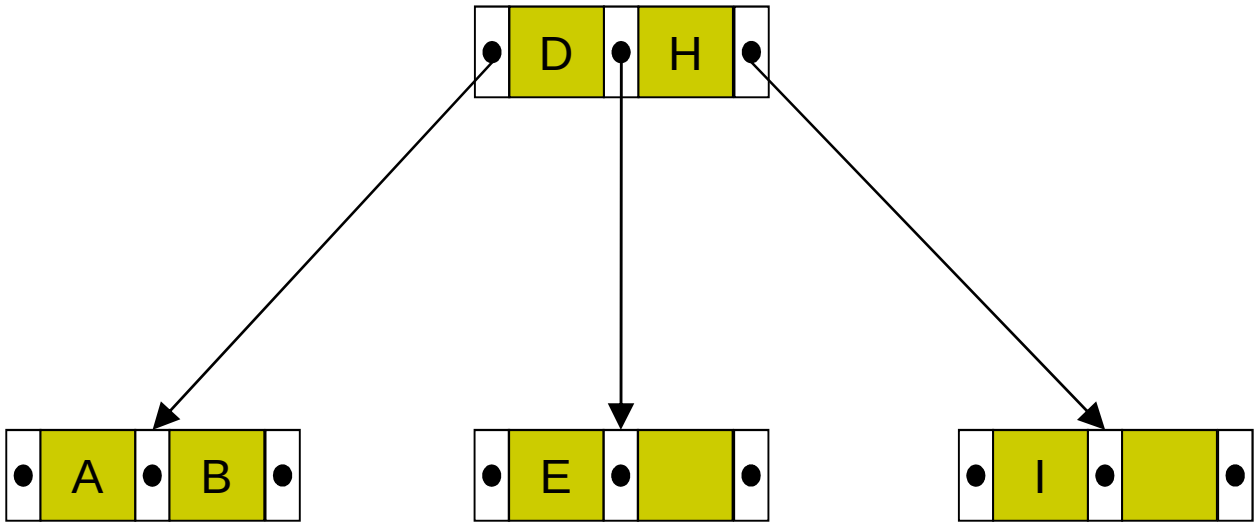
Estrutura Lógica de um Nó



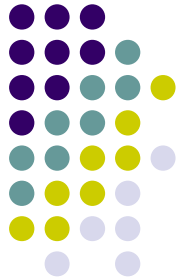
Estrutura Lógica de um Nó



Exemplo

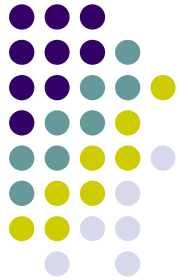


Inserção de Dados (Chave)



- Característica
  - sempre realizada nos nós folhas
- Situações a serem analisadas
  - árvore vazia
  - overflow no nó raiz
  - inserção nos nós folhas

Inserção: Situação Inicial



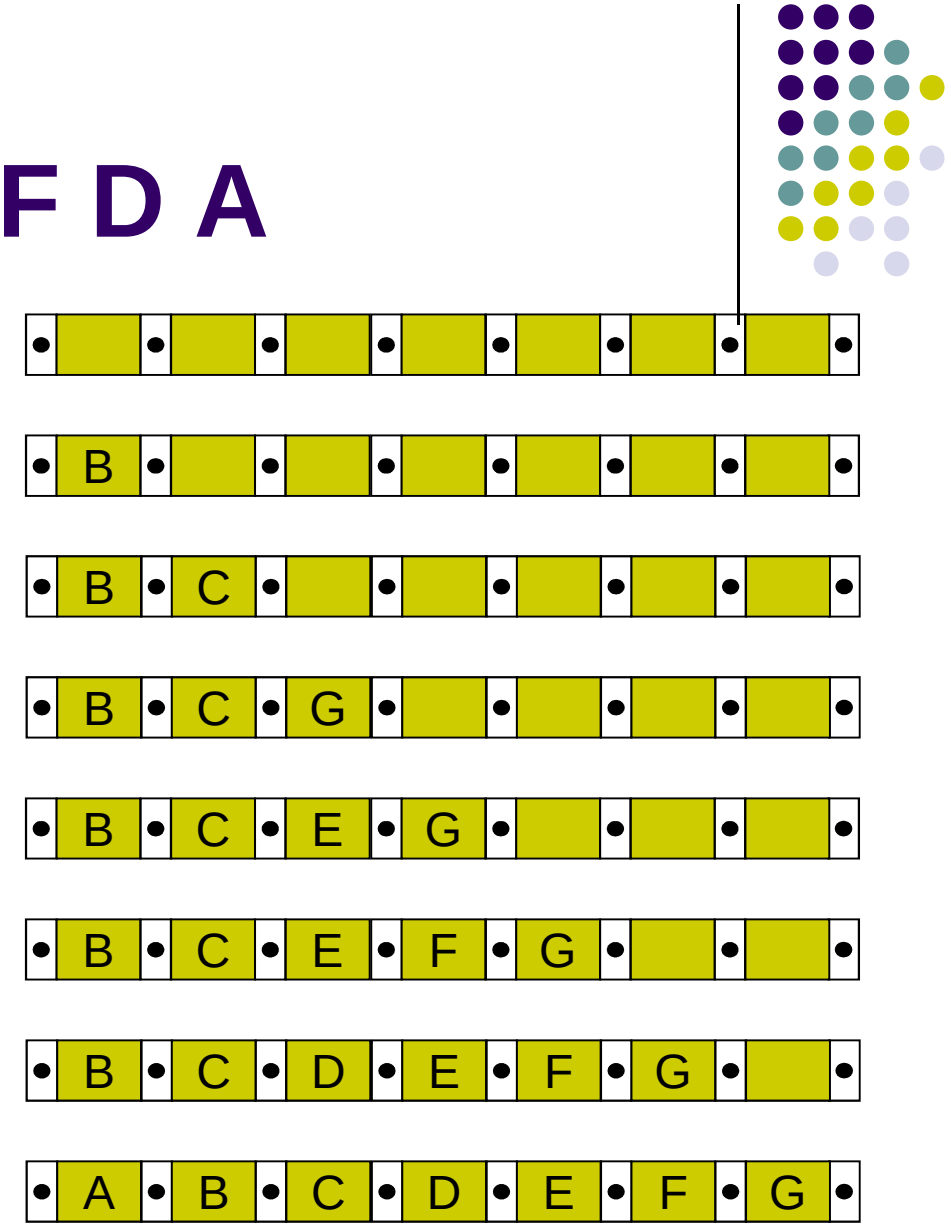
- Criação e preenchimento do nó
  - primeira chave: criação do nó raiz
  - demais chaves: inserção até a capacidade limite do nó
- Exemplo
  - nó com capacidade para 7 chaves
  - chaves: letras do alfabeto
  - situação inicial: árvore vazia

Inserção: Situação Inicial

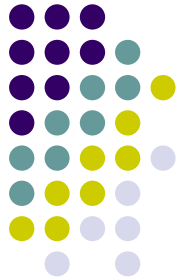
- Chaves B C G E F D A
  - inseridas desordenadamente
  - mantidas ordenadas no nó
- Ponteiros (*)
  - nós folhas: *NULL*
  - nós internos: RRN (*Relative Record Number*) ou *NULL*
- Árvore vazia, portanto: nó raiz = nó folha

Chaves B C G E F D A

- Cria-se o nó
- Procede-se a inserção até que seja preenchido todas as posições daquele nó
- Internamente os elementos são mantidos ordenados
- Como proceder a inserção de uma nova chave?

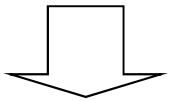


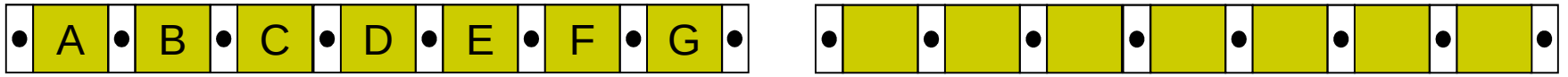
Inserção: *Overflow* Nó Raiz

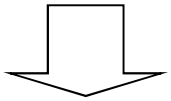


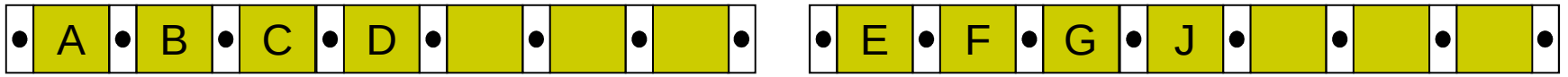
- Passo 1: particionamento do nó (split)
  - nó original → nó original + novo nó
    - split 1-to-2
  - as chaves são distribuídas uniformemente nos dois nós
    - chaves do nó original + nova chave
- Exemplo:
  - Situação anterior, agora com a inserção da chave J

Split



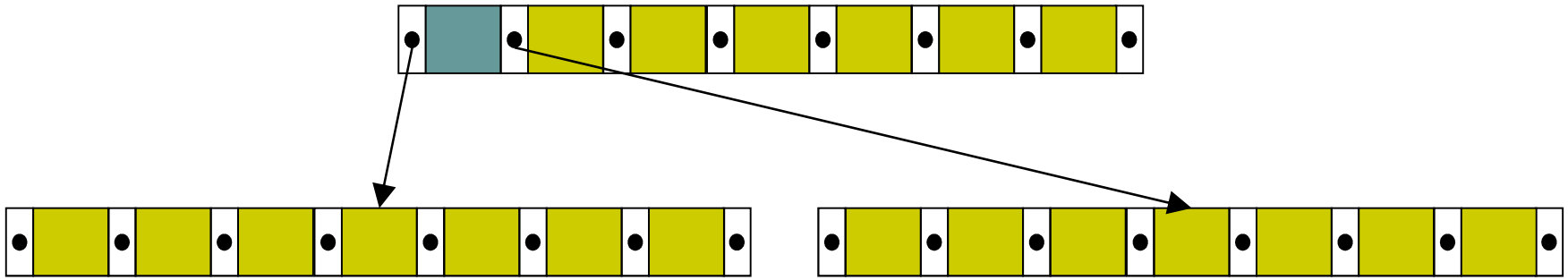




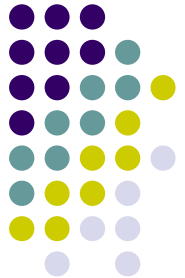


Inserção: *Overflow* Nó Raiz

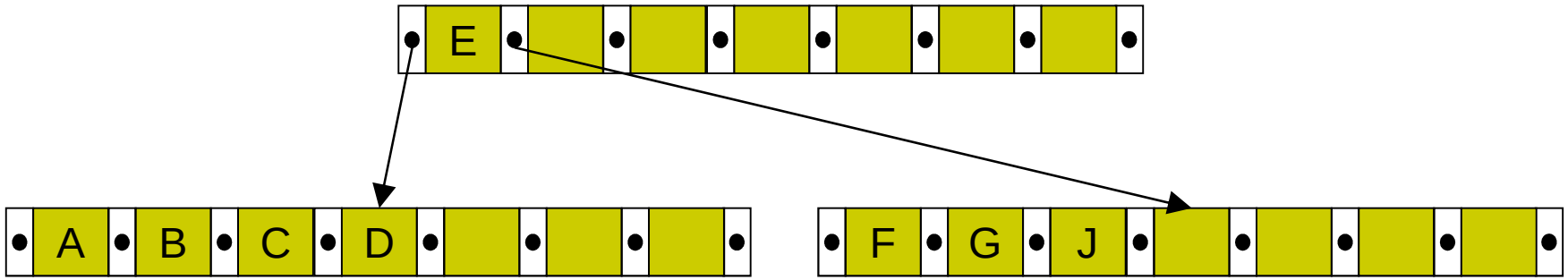
- Passo 2: criação de uma nova raiz
  - a existência de um nível mais alto na árvore permite a escolha das folhas durante a pesquisa
- Exemplo:



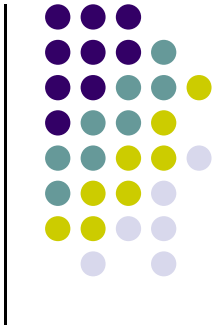
Inserção: *Overflow* Nó Raiz



- Passo 3: promoção de chave (promotion)
  - a primeira chave do novo nó resultante do particionamento é promovida para o nó raiz
- Exemplo:

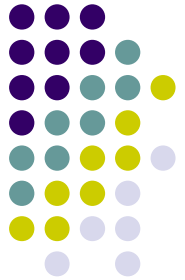


Inserção: Nós Folhas



- Passo 1: pesquisa
  - a árvore é percorrida até encontrar o nó folha no qual a nova chave será inserida
- Passo 2: inserção em nó com espaço
  - ordenação da chave após a inserção
  - alteração dos valores dos campos de referência
- Passo 2: inserção em nó cheio
  - particionamento
    - criação de um novo nó e distribuição uniforme das chaves nos nós
  - promoção
    - escolha da primeira chave do novo nó como chave separadora no nó pai
    - ajuste do nó pai para apontar para o novo nó
  - propagação de *overflow*

Exemplo

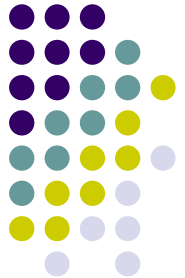


- Insira as seguintes chaves em um índice árvore-B

  03 19 04 20 01 13 16 09 02 22 14 07 21 18 11 05 08 15 12 10 24 17 25 06 23 26

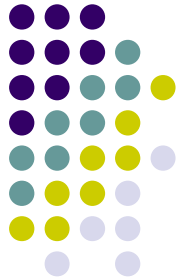
- Ordem da árvore-B: 2
  - em cada nó (página de disco)
    - número de chaves: 2
    - número de ponteiros: 3
- http://slady.net/java/bt/view.php?w=600&h=450

Remoção



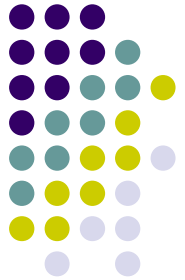
- É feita uma pesquisa até a folha da qual uma chave será removida. Existem 2 possibilidades:
  1. a chave a ser excluída está em um nó folha, logo ela é simplesmente excluída;
  2. a chave a ser excluída não está em um nó folha: uma chave adjacente é procurada e transferida para o local onde se encontra a chave a ser excluída.
     - Esta chave pode ser a primeira chave da folha mais à esquerda da sub-árvore à direita ou a última chave da folha mais à direita da sub-árvore à esquerda. Assim, a chave substituta é retirada de uma folha, o que leva à retirada de uma folha.

Remoção



- É importante verificar se a retirada "quebrou" a definição de árvore B:
  1. caso a folha, após a retirada, tenha ficado com menos de ***m*** chaves, verifica-se a ordem dos irmãos adjacentes.
  2. se existir um irmão adjacente com mais de ***m*** chaves, é feita uma redistribuição. A chave mais à esquerda do irmão adjacente é promovida a "chave pai" e a "chave pai“ anterior é inserida no nó em que a chave foi retirada (semelhante à inserção).
  3. Se o irmão adjacente tem ***m*** chaves, é feita uma concatenação. Este processo consiste em juntar em um único nó as chaves dos irmãos adjacentes mais a "chave pai", eliminado um nó folha e uma chave (a "chave pai") do nó pai. Neste caso, dependendo do número de chaves do nó pai, pode resultar em uma nova redistribuição ou em uma nova concatenação.

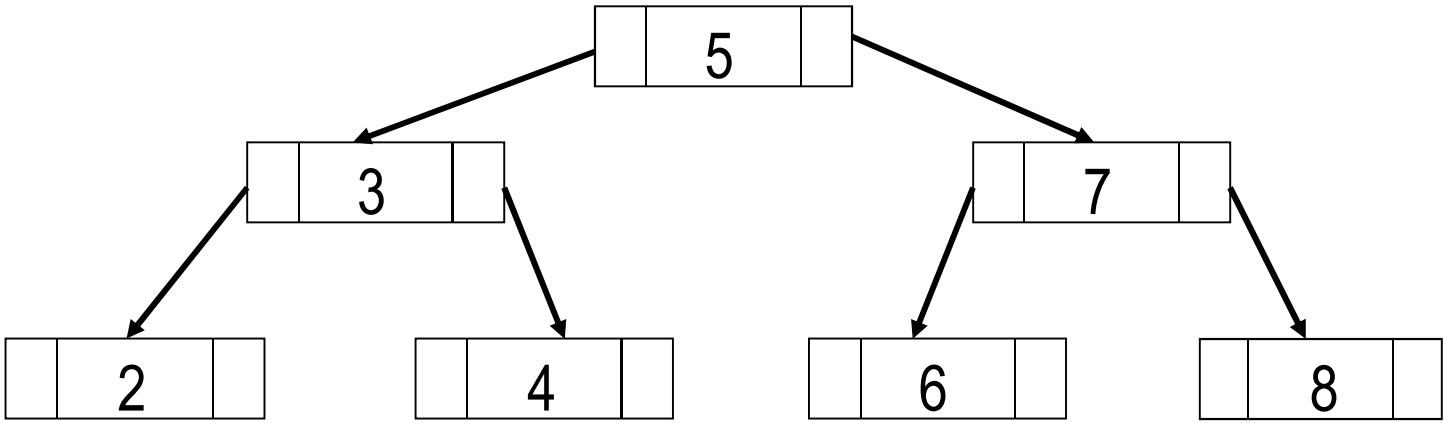
Concatenação



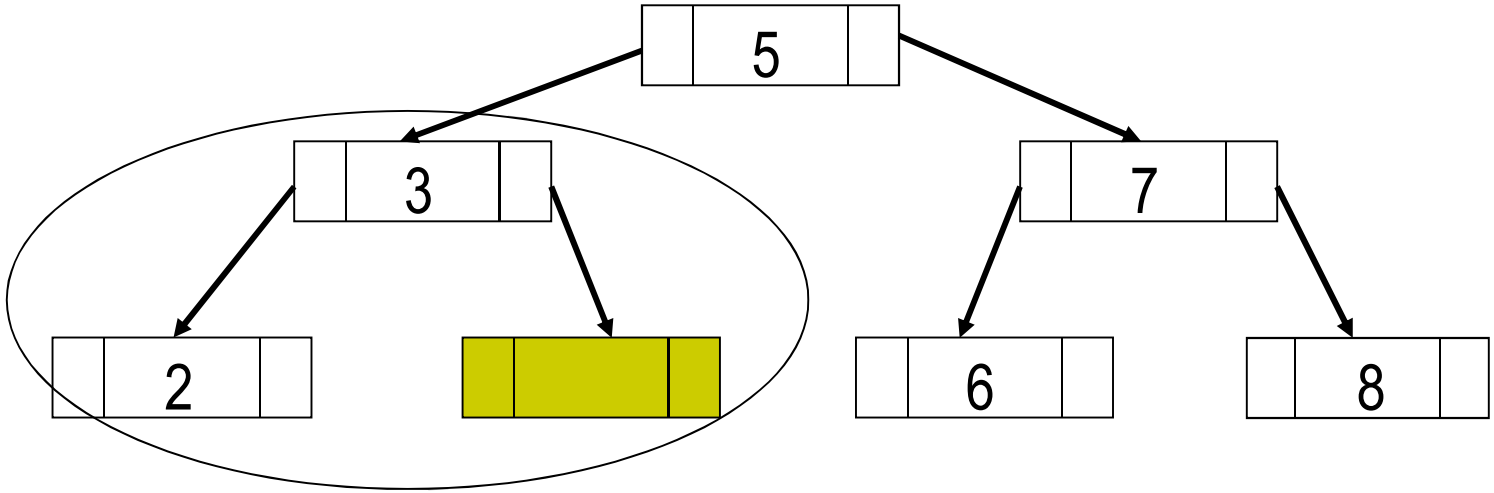
- Acontece quando, após a remoção, a página onde a chave foi removida e uma página adjacente possuem em conjunto menos de *2d* chaves.
- Concatena essa página com uma adjacente.
  - A chave do pai que estava entre elas fica na página que foi concatenada.
- Se esse procedimento resultar em uma página com menos de *d* chaves, realizar-se-á novamente o mesmo procedimento, podendo chegar até a raiz.

Concatenação

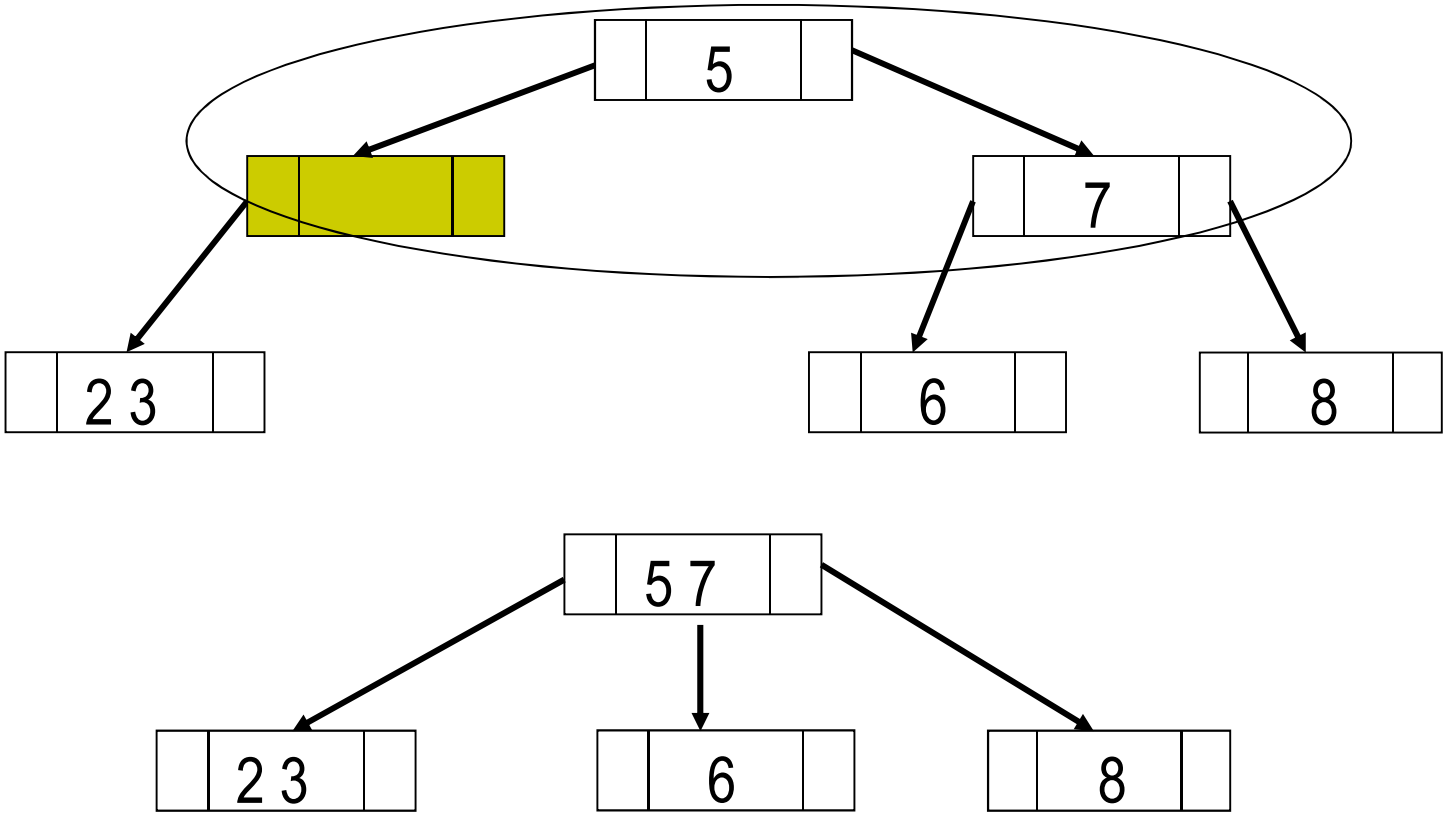
Remoção do elemento 4 em uma árvore com d = 1 (Árvore 2-3)



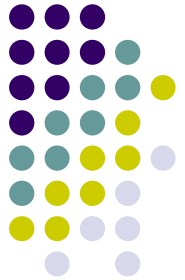
Concatenação



Concatenação

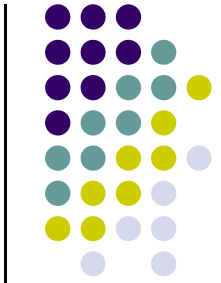


Redistrubição

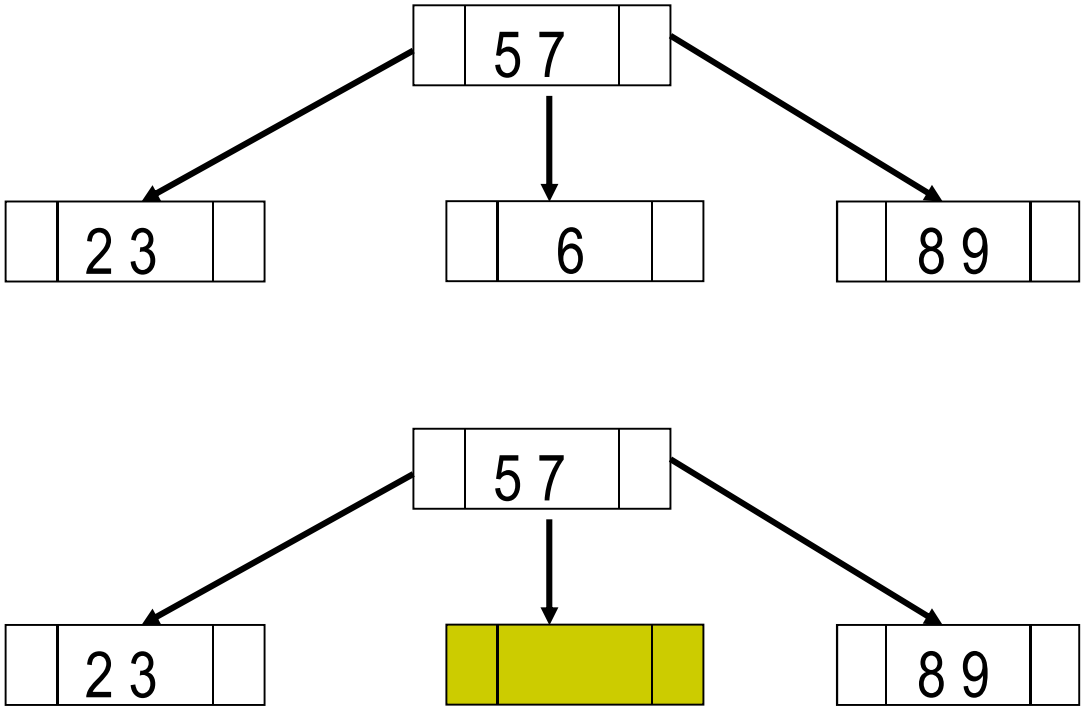


- Acontece quando, após a remoção, a página onde a chave foi removida e uma página adjacente possuem em conjunto *2d* chaves ou mais.
- Concatena essa página com uma adjacente. A chave do pai que estava entre elas fica na página que foi concatenada. Acontece uma cisão.
- Não é propagável, pois o número de chaves do pai não muda

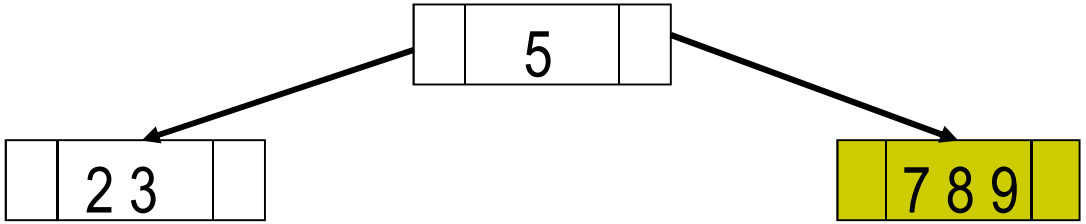
Redistribuição

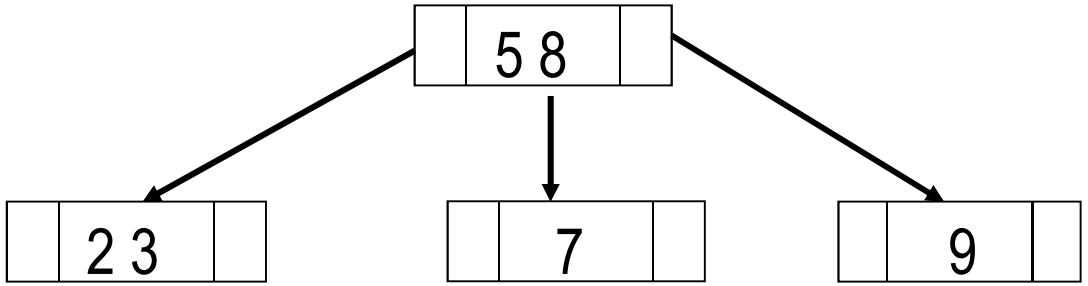


Remoção do elemento 6 em uma árvore com d = 1 (Árvore 2-3)

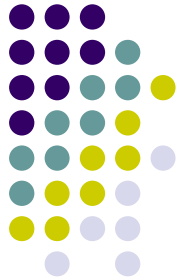


Redistribuição





Exercícios



- Insira as seguintes chaves em um índice árvore-B
  - 03 19 04 20 01 13 16 09 02 22 14 07 21 18 11 05 08 15 12 10 24 17 25 06 23 26 00
  - escolha o último elemento do primeiro nó para promoção durante o particionamento do nó.