ACH2024

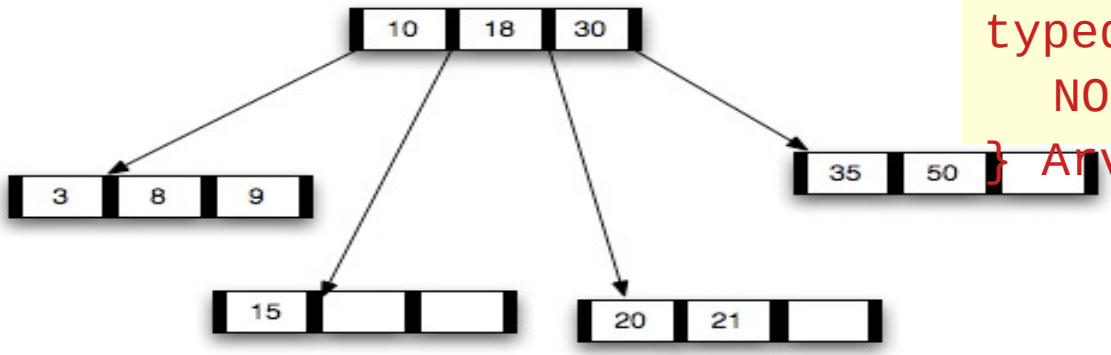
Aula 19

Organização de arquivos
Árvores B – parte 3 (remoção)
e Árvores B+

Profa. Ariane Machado Lima

Aulas passadas

Estrutura de uma árvore B




```
#define t 2  /* grau min da árvore*/
typedef  int  TipoChave;
typedef struct str_no {
    TipoChave chave[2*t-1];
    struct str_no* filhos[2*t];
    int numChaves;
    bool folha;
} NO;
typedef struct {
   NO* raiz;
} ArvB;
```

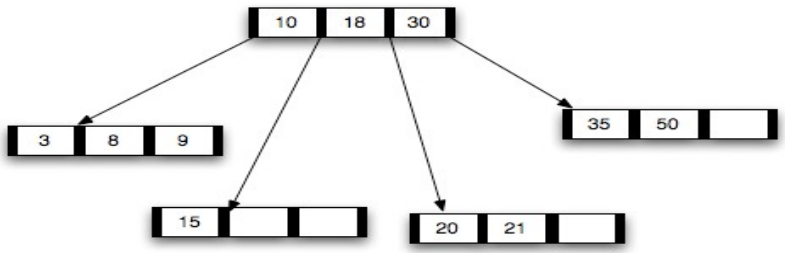
Note que nesses pseudocódigos assume-se que as chaves e filhos começam na posição 1 !!!

Busca na árvore B

- B-Tree-Search(x, k): tem como parâmetros um ponteiro para o nó x raiz de uma subárvore e uma chave k a ser procurada na subárvore. Se k está na subárvore, retorna o par ordenado (y, i) composto pelo ponteiro do nó y e o índice i tal que $key_i[y] = k$. Caso contrário, retorna NIL.
- Chamada inicial: B-Tree-Search$(root[T], k)$.

Ex: B-Tree-Search(T→raiz, 22)

```
B-TREE-SEARCH(x, k)
1  i ← 1
2  while i ≤ n[x] and k > key_i[x]
3        do i ← i + 1
4  if i ≤ n[x] and k = key_i[x]
5     then return (x, i)
6  if leaf[x]
7     then return NIL
8     else DISK-READ(c_i[x])
9          return B-TREE-SEARCH(c_i[x], k)
```



Complexidade: O(h) seeks (O($\log_t$ b))

- Inserção
  - Sempre nas folhas
  - Faz split quando nó cheio
  - $O(\log_t b)$: b = nr de blocos do arquivo (ie, nr de nós da árvore)

- Início da remoção

Remoção em árvores B

B-Tree-Delete(x,k): remoção da chave k da subárvore com raiz x.

Que propriedades gostaríamos que nossa remoção tivesse?

1. Ela precisa manter as propriedades da árvore B
2. Já que vou remover, será que consigo diminuir a altura da árvore? Para isso, a cada remoção quero liberar espaço em uma folha

3 casos a serem tratados (com seus subcasos):

- x é nó interno e a chave k está lá
- x é nó interno mas a chave k não está lá
- x é folha (e a chave k está ou não lá)

Remoção em árvores B

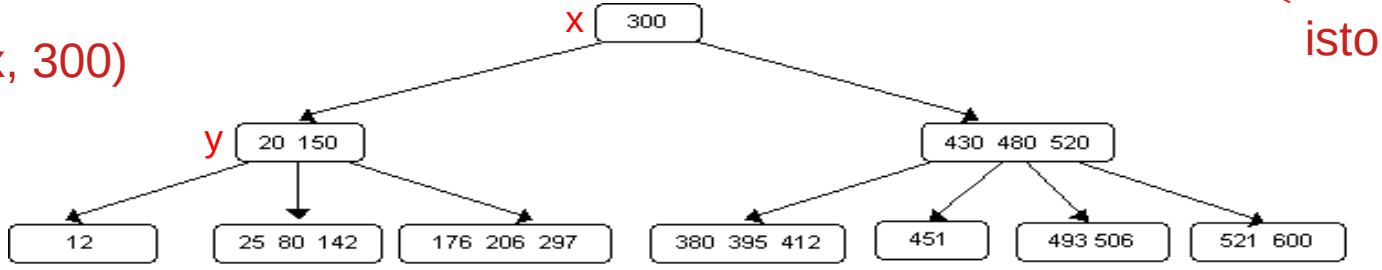
- B-Tree-Delete(x,k): remoção da chave k da subárvore com raiz x.

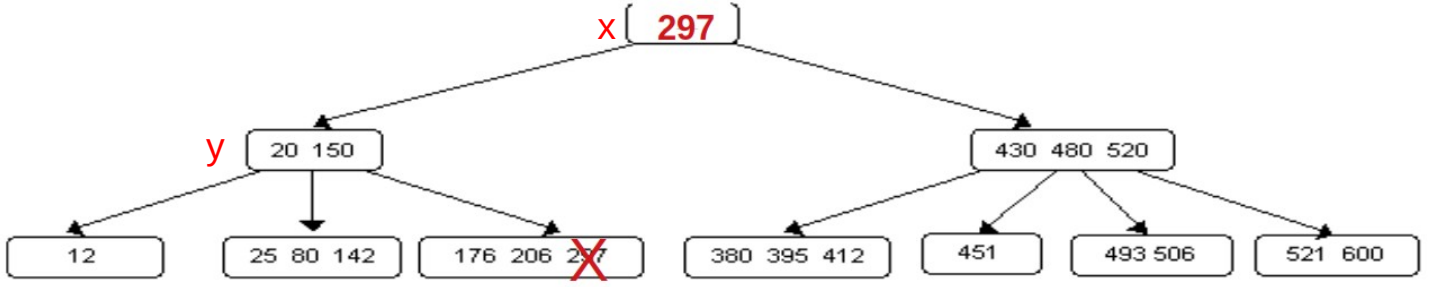
  2. Se a chave k está no nó x e x é um nó interno, faça:

     a) Se o filho y que precede k no nó x tem pelo menos t chaves, então encontre o predecessor k' de k na subárvore com raiz y. Delete recursivamente k', e substitua k por k' em x.

isto é, max(y)

B-Tree-Delete(x, 300)
(t = 2)





Remoção em árvores B

- B-Tree-Delete(x,k): remoção da chave k da subárvore com raiz x.

  2. Se a chave k está no nó x e x é um nó interno, faça:

     a) Se o filho y que precede k no nó x tem pelo menos t chaves, então encontre o predecessor k' de k na subárvore com raiz y. Delete recursivamente k', e substitua k por k' em x.

     OU

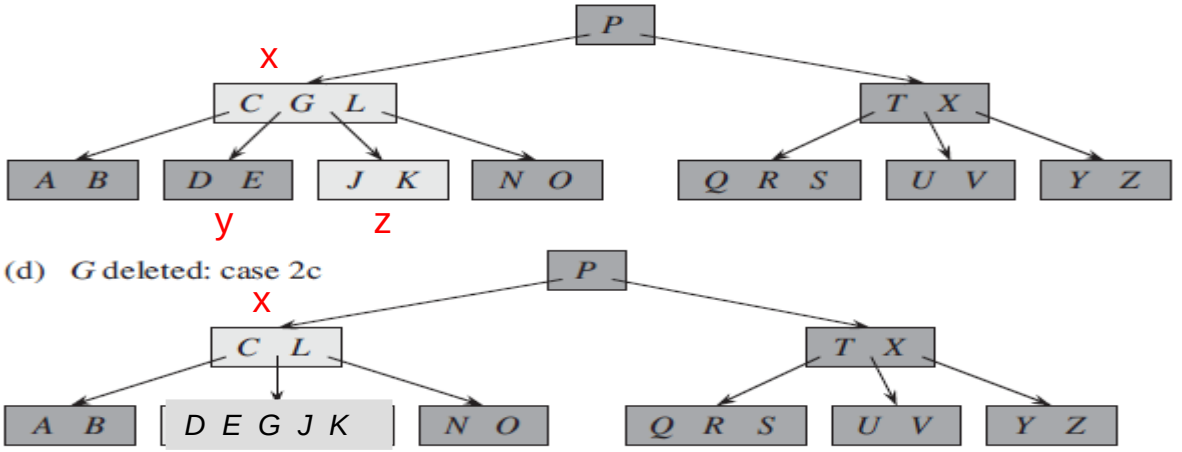
     b) Simetricamente, se o filho z imediatamente após k no nó x tem pelo menos t chaves, então encontre o sucessor k' de k na subárvore com raiz z. Delete recursivamente k', e substitua k por k' em x.

Remoção em árvores B

- B-Tree-Delete(x,k): remoção da chave k da subárvore com raiz x.

  2. Se a chave k está no nó x e x é um nó interno, faça:

     c) Caso contrário, se ambos y e z possuem apenas $t - 1$ chaves, faça a junção de k e todas as chaves de z em y, de forma que x perde tanto a chave k como o ponteiro para z, e y agora contém $2t - 1$ chaves. Então, libere z e delete recursivamente k de y.

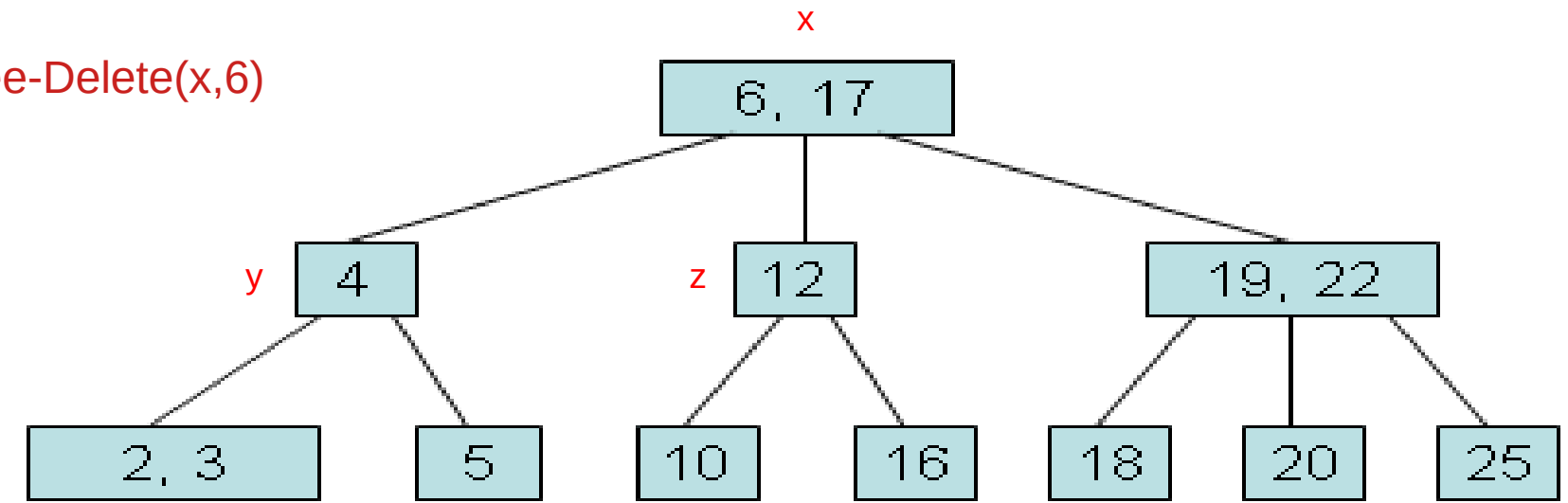


B-Tree-Delete(x,G)

Outro exemplo deste último caso: remoção do 6 (t = 2) : exercício

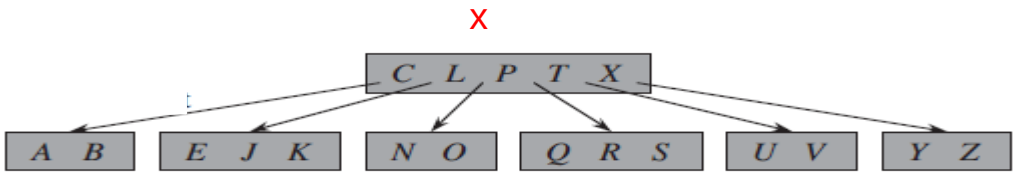
c) Caso contrário, se ambos y e z possuem apenas $t - 1$ chaves, faça a junção de k e todas as chaves de z em y, de forma que x perde tanto a chave k como o ponteiro para z, e y agora contém $2t - 1$ chaves. Então, libere z e delete recursivamente k de y.

B-Tree-Delete(x,6)



Remoção em árvores B

- B-Tree-Delete(x,k): remoção da chave k da subárvore com raiz x.
  3. Se a chave k não está presente no no interno x,



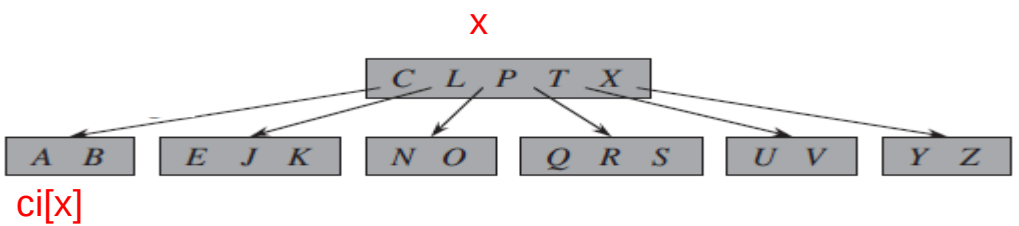
Deletar B:

B-Tree-Delete(x,B)

Remoção em árvores B

- B-Tree-Delete(x,k): remoção da chave k da subárvore com raiz x.
  3. Se a chave k não está presente no no interno x, determine a raiz $c_i[x]$ da subárvore apropriada que deve conter k (se k estiver presente na árvore).

Algum problema?



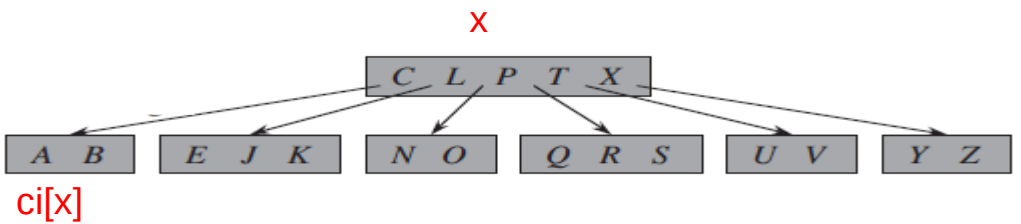
Deletar B:

B-Tree-Delete(x,B)

Remoção em árvores B

- B-Tree-Delete(x,k): remoção da chave k da subárvore com raiz x.
  3. Se a chave k não está presente no no interno x, determine a raiz $c_i[x]$ da subárvore apropriada que deve conter k (se k estiver presente na árvore).

Algum problema?
Se ci[x] tem o nr mínimo de chaves,
se tiver que deletar desse nó vai dar problema...

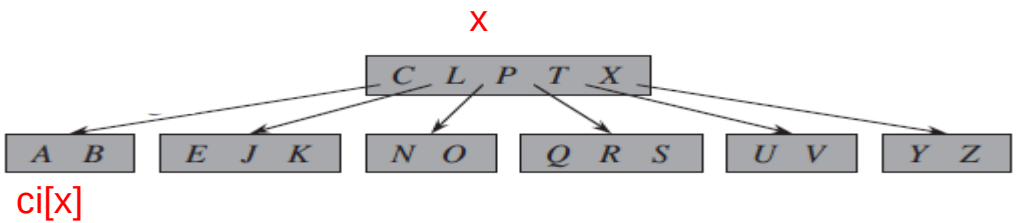


Deletar B:

B-Tree-Delete(x,B)

Remoção em árvores B

- B-Tree-Delete(x,k): remoção da chave k da subárvore com raiz x.
  3. Se a chave k não está presente no no interno x, determine a raiz $c_i[x]$ da subárvore apropriada que deve conter k (se k estiver presente na árvore). Se $c_i[x]$ tem apenas $t - 1$ chaves,

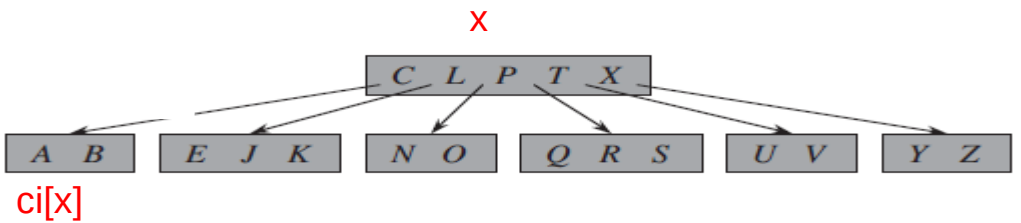


Deletar B:

B-Tree-Delete(x,B)

Remoção em árvores B

- B-Tree-Delete(x,k): remoção da chave k da subárvore com raiz x.
  3. Se a chave k não está presente no no interno x, determine a raiz $c_i[x]$ da subárvore apropriada que deve conter k (se k estiver presente na árvore). Se $c_i[x]$ tem apenas $t - 1$ chaves, execute o passo 3a ou 3b conforme necessário para garantir que o algoritmo desça para um nó contendo pelo menos t chaves. Então, continue no filho apropriado de x.

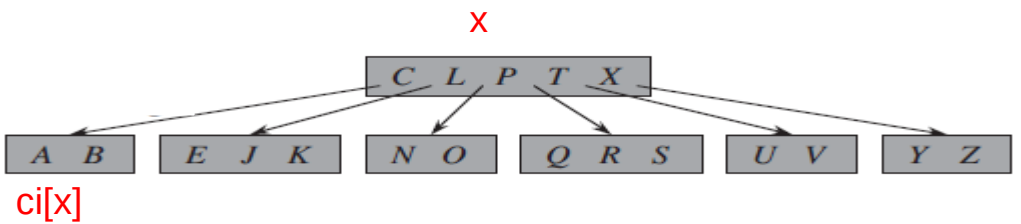


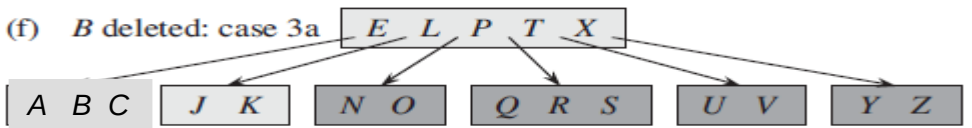
Deletar B:

B-Tree-Delete(x,B)

a) Se $c_i[x]$ contém apenas $t-1$ chaves mas tem um irmão imediato com pelo menos t chaves, dê para $c_i[x]$ uma chave extra movendo uma chave de x para $c_i[x]$, movendo uma chave do irmão imediato de $c_i[x]$ à esquerda ou à direita, e movendo o ponteiro do filho apropriado do irmão para o nó $c_i[x]$.

b)

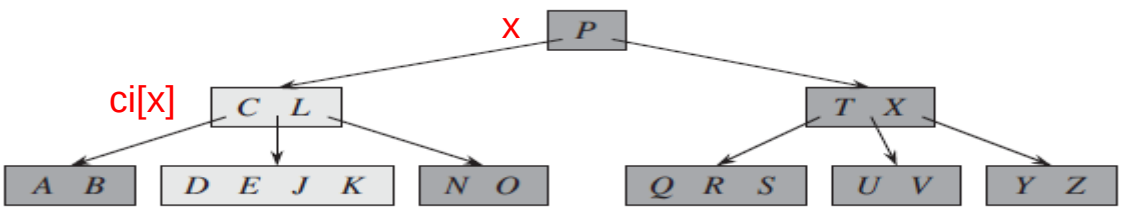




B-Tree-Delete(x,B)

a)

b) Se $c_i[x]$ e ambos os irmãos imediatos de $c_i[x]$ contêm $t - 1$ chaves,

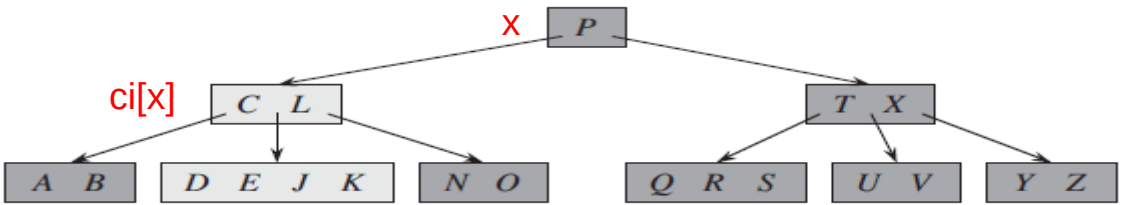


(e) *D* deleted: case 3b

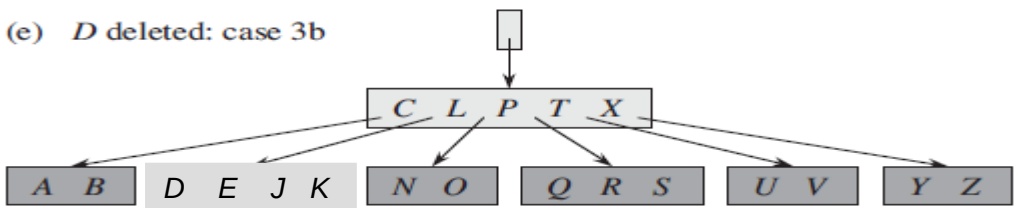
B-Tree-Delete(x,D)

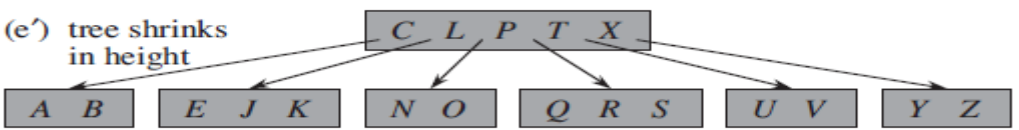
a)

b) Se $c_i[x]$ e ambos os irmãos imediatos de $c_i[x]$ contêm $t - 1$ chaves, faça a junção de $c_i[x]$ com um de seus irmãos. Isso implicará em mover uma chave de x para o novo nó fundido (que se tornará a chave mediana para aquele nó).



B-Tree-Delete(x,D)

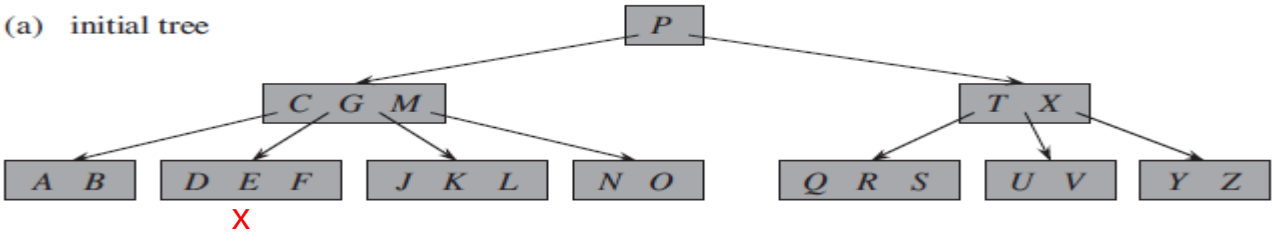




Remoção em árvores B

- B-Tree-Delete(x,k): remoção da chave k da subárvore com raiz x.

1. Se a chave k está no nó x e x é uma folha,



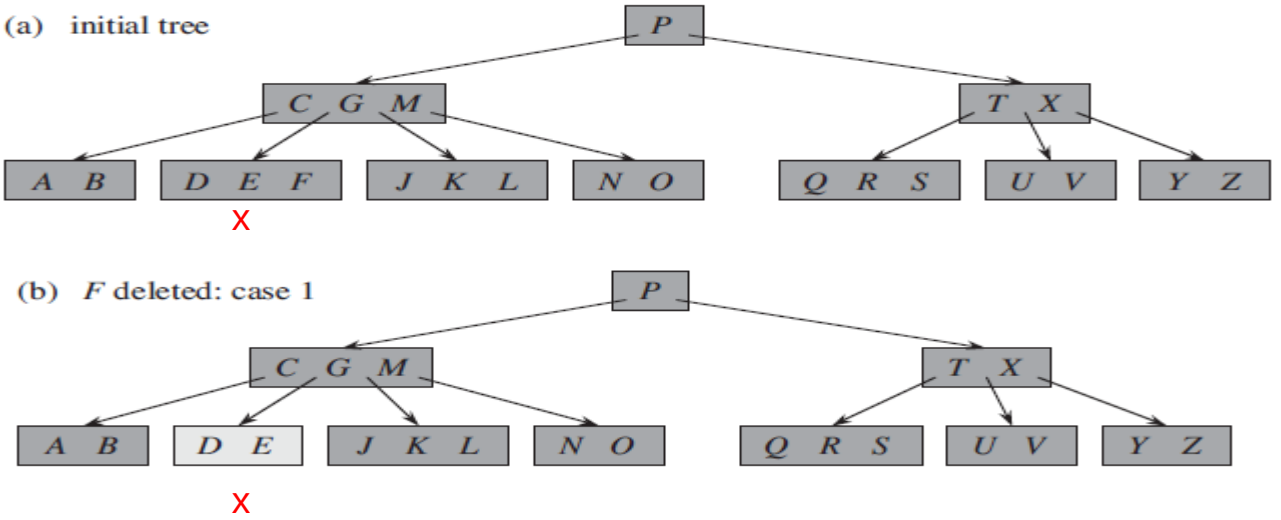
B-Tree-Delete(x,F)

Remoção em árvores B

- B-Tree-Delete(x,k): remoção da chave k da subárvore com raiz x.

1. Se a chave k está no nó x e x é uma folha, exclua a chave k de x.

(pelos procedimentos anteriores, já sei que o nó x tem pelo menos t chaves)



B-Tree-Delete(x,F)

Remoção em árvores B

- Atenção quando um dos nós for a raiz:
  - Ela pode ter menos do que t-1 chaves
  - Se ficar com zero chaves precisa desalocar o bloco e atualizar quem é a nova raiz.
  - Precisa de uma camada extra sobre a chamada da deleção:

Remoção em árvores B

```
B-Tree-Delete-From-Root(T, k){
        r ← raiz[T];
        se (n[r] = 0) retorna;
        senão B-Tree-Delete(r,k);
        se (n[r] = 0  e  (! leaf[r]){
                raiz[T] ← c1[r];
                desaloca(r);
        }
}
```

Remoção em árvores B

```
B-Tree-Delete-From-Root(T, k){
        r ← raiz[T];
        se (n[r] = 0) retorna;
        senão B-Tree-Delete(r,k);
        se (n[r] = 0  e  (! leaf[r]){
                raiz[T] ← c1[r];
                desaloca(r);
        }
}
```

Complexidade total: ?

Remoção em árvores B

```
B-Tree-Delete-From-Root(T, k){
        r ← raiz[T];
        se (n[r] = 0) retorna;
        senão B-Tree-Delete(r,k);
        se (n[r] = 0  e  (! leaf[r]){
                raiz[T] ← c1[r];
                desaloca(r);
        }
}
```

Complexidade total: $O(h) = O(\log_t b)$

Remoção em árvores B

Exercício: IMPLEMENTEM EM CASA!!!!

Cuidado quando os nós internos se tornam folhas!

Comparação de complexidades

	Sequencial		Ligada	Indexada	Indexada Multinível	Árvores B
	Não-Ordenado	**Ordenado**	**Ordenado**	**Ordenado**	**Ordenado**	**Ordenado**
Busca	O(b)	O(lg b)	O(b), O(lg b) só se usar FAT (discos pequenos)	O(log bi)	O(log $_{fbi}$ r^{1})	O(h = lg$_t$ b)
Inserção**	O(1) se tiver espaço no final, O(b) c.c.	O(1) se tiver espaço no bloco, O(b) c.c.	O(1)	O(b+bi) = O(b)	O(b+bi) = O(b)	O(h = lg$_t$ b)
Remoção* , **	O(1)	O(1)	O(1)	O(1)	O(1)	O(h = lg$_t$ b)
Leitura ordenada	ω(b) (depende do alg de ord. externa)	O(b), O(1) se fizer a leitura toda de uma vez	O(b)	O(b+bi) = O(b)	O(b+bi) = O(b)	
Mínimo/máximo	O(b)	O(1)	O(1)	O(1)	O(1)	
Modificação**	O(1)	O(b) se no campo chave, O(1) c.c.	O(1)	O(b) se no campo chave, O(1) c.c.	O(b) se no campo chave, O(1) c.c.	
* considerando uso de bit de validade						
** considerando que já se sabe a localização do registro (busca já realizada)						

Comparação de complexidades

	Sequencial		Ligada	Indexada	Indexada Multinível	Árvores B
	Não-Ordenado	**Ordenado**	**Ordenado**	**Ordenado**	**Ordenado**	**Ordenado**
Busca	O(b)	O(lg b)	O(b), O(lg b) só se usar FAT (discos pequenos)	O(log bi)	O(log $_{fbi}$ r[1])	O(h = lg$_t$ b)
Inserção**	O(1) se tiver espaço no final, O(b) c.c.	O(1) se tiver espaço no bloco, O(b) c.c.	O(1)	O(b+bi) = O(b)	O(b+bi) = O(b)	O(h = lg$_t$ b)
Remoção* , **	O(1)	O(1)	O(1)	O(1)	O(1)	O(h = lg$_t$ b)
Leitura ordenada	ω(b) (depende do alg de ord. externa)	O(b), O(1) se fizer a leitura toda de uma vez	O(b)	O(b+bi) = O(b)	O(b+bi) = O(b)	O(n), n = nr de chaves
Mínimo/máximo	O(b)	O(1)	O(1)	O(1)	O(1)	
Modificação**	O(1)	O(b) se no campo chave, O(1) c.c.	O(1)	O(b) se no campo chave, O(1) c.c.	O(b) se no campo chave, O(1) c.c.	
* considerando uso de bit de validade						
** considerando que já se sabe a localização do registro (busca já realizada)						

Comparação de complexidades

	Sequencial		Ligada	Indexada	Indexada Multinível	Árvores B
	Não-Ordenado	Ordenado	Ordenado	Ordenado	Ordenado	Ordenado
Busca	O(b)	O(lg b)	O(b), O(lg b) só se usar FAT (discos pequenos)	O(log bi)	O($\log_{fbi} r^1$)	O(h = lg_t b)
Inserção**	O(1) se tiver espaço no final, O(b) c.c.	O(1) se tiver espaço no bloco, O(b) c.c.	O(1)	O(b+bi) = O(b)	O(b+bi) = O(b)	O(h = lg_t b)
Remoção* , **	O(1)	O(1)	O(1)	O(1)	O(1)	O(h = lg_t b)
Leitura ordenada	ω(b) (depende do alg de ord. externa)	O(b), O(1) se fizer a leitura toda de uma vez	O(b)	O(b+bi) = O(b)	O(b+bi) = O(b)	O(n), n = nr de chaves
Mínimo/máximo	O(b)	O(1)	O(1)	O(1)	O(1)	O(h = lg_t b)
Modificação**	O(1)	O(b) se no campo chave, O(1) c.c.	O(1)	O(b) se no campo chave, O(1) c.c.	O(b) se no campo chave, O(1) c.c.	

* considerando uso de bit de validade

** considerando que já se sabe a localização do registro (busca já realizada)

Como seria a função?

Comparação de complexidades

	Sequencial		Ligada	Indexada	Indexada Multinível	Árvores B
	Não-Ordenado	**Ordenado**	**Ordenado**	**Ordenado**	**Ordenado**	**Ordenado**
Busca	O(b)	O(lg b)	O(b), O(lg b) só se usar FAT (discos pequenos)	O(log bi)	O($\log_{fbi} r^1$)	O(h = lg_t b)
Inserção**	O(1) se tiver espaço no final, O(b) c.c.	O(1) se tiver espaço no bloco, O(b) c.c.	O(1)	O(b+bi) = O(b)	O(b+bi) = O(b)	O(h = lg_t b)
Remoção* , **	O(1)	O(1)	O(1)	O(1)	O(1)	O(h = lg_t b)
Leitura ordenada	ω(b) (depende do alg de ord. externa)	O(b), O(1) se fizer a leitura toda de uma vez	O(b)	O(b+bi) = O(b)	O(b+bi) = O(b)	O(n), n = nr de chaves
Mínimo/máximo	O(b)	O(1)	O(1)	O(1)	O(1)	O(h = lg_t b)
Modificação**	O(1)	O(b) se no campo chave, O(1) c.c.	O(1)	O(b) se no campo chave, O(1) c.c.	O(b) se no campo chave, O(1) c.c.	O(h = lg_t b)
* considerando uso de bit de validade						
** considerando que já se sabe a localização do registro (busca já realizada)						

Se no campo chave, envolve uma remoção e inserção

Comparação de complexidades

	Sequencial		Ligada	Indexada	Indexada Multinível	Árvores B
	Não-Ordenado	**Ordenado**	**Ordenado**	**Ordenado**	**Ordenado**	**Ordenado**
Busca	O(b)	O(lg b)	O(b), O(lg b) só se usar FAT (discos pequenos)	O(log bi)	O($\log_{fbi} r^1$)	O(h = lg_t b)
Inserção**	O(1) se tiver espaço no final, O(b) c.c.	O(1) se tiver espaço no bloco, O(b) c.c.	O(1)	O(b+bi) = O(b)	O(b+bi) = O(b)	O(h = lg_t b)
Remoção* , **	O(1)	O(1)	O(1)	O(1)	O(1)	O(h = lg_t b)
Leitura ordenada	ω(b) (depende do alg de ord. externa)	O(b), O(1) se fizer a leitura toda de uma vez	O(b)	O(b+bi) = O(b)	O(b+bi) = O(b)	O(n), n = nr de chaves
Mínimo/máximo	O(b)	O(1)	O(1)	O(1)	O(1)	O(h = lg_t b)
Modificação**	O(1)	O(b) se no campo chave, O(1) c.c.	O(1)	O(b) se no campo chave, O(1) c.c.	O(b) se no campo chave, O(1) c.c.	O(h = lg_t b)
* considerando uso de bit de validade						
** considerando que já se sabe a localização do registro (busca já realizada)						

MELHOROU MUITO!

Mas será que podemos melhorar a leitura ordenada?

Árvores B+

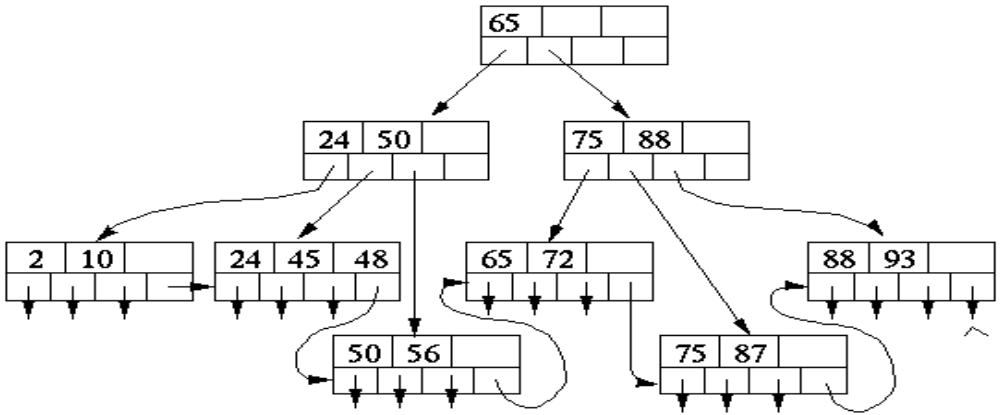
(tema do EP 2 !!!)

ÁRVORES B+

Variação da árvore B, na qual:

- os nós internos armazenam apenas os índices (ponteiros de filhos e chaves)
- as folhas armazenam os registros de dados (ou ponteiros para os registros) e são conectadas da esquerda para a direita, permitindo acesso sequencial ordenado mais eficiente

- Blocagem menor (cabem mais registros nos nós internos → altura menor)

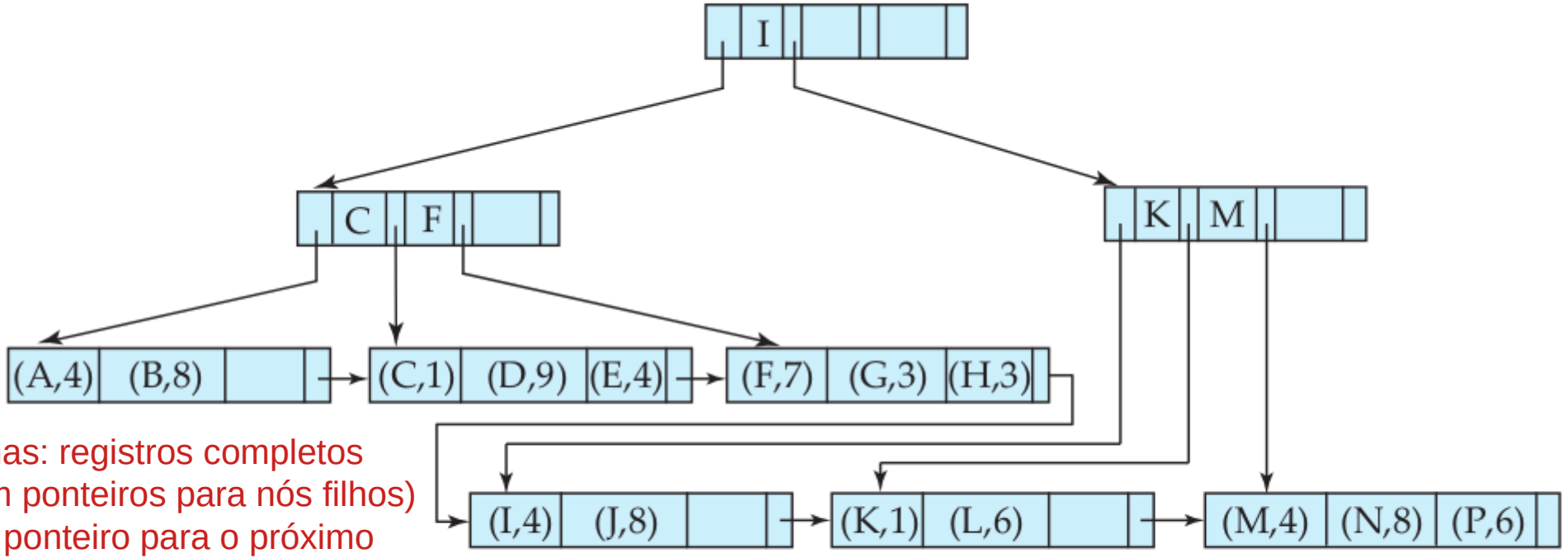


Nós internos (de índices)

Note que há repetição entre os dois conjuntos

Nós folhas (de dados)

Árvore B+ como forma de organização de arquivo



Folhas: registros completos (sem ponteiros para nós filhos) + 1 ponteiro para o próximo nó folha

=> struct diferente para nós internos e folhas

Figure 11.20 B$^+$-tree file organization.

(Silberschatz)

(Silberschatz)

Árvore B+ como índice secundário

Folhas: devem

Cada folha: de

+ t

+ 1

=> struct “pare

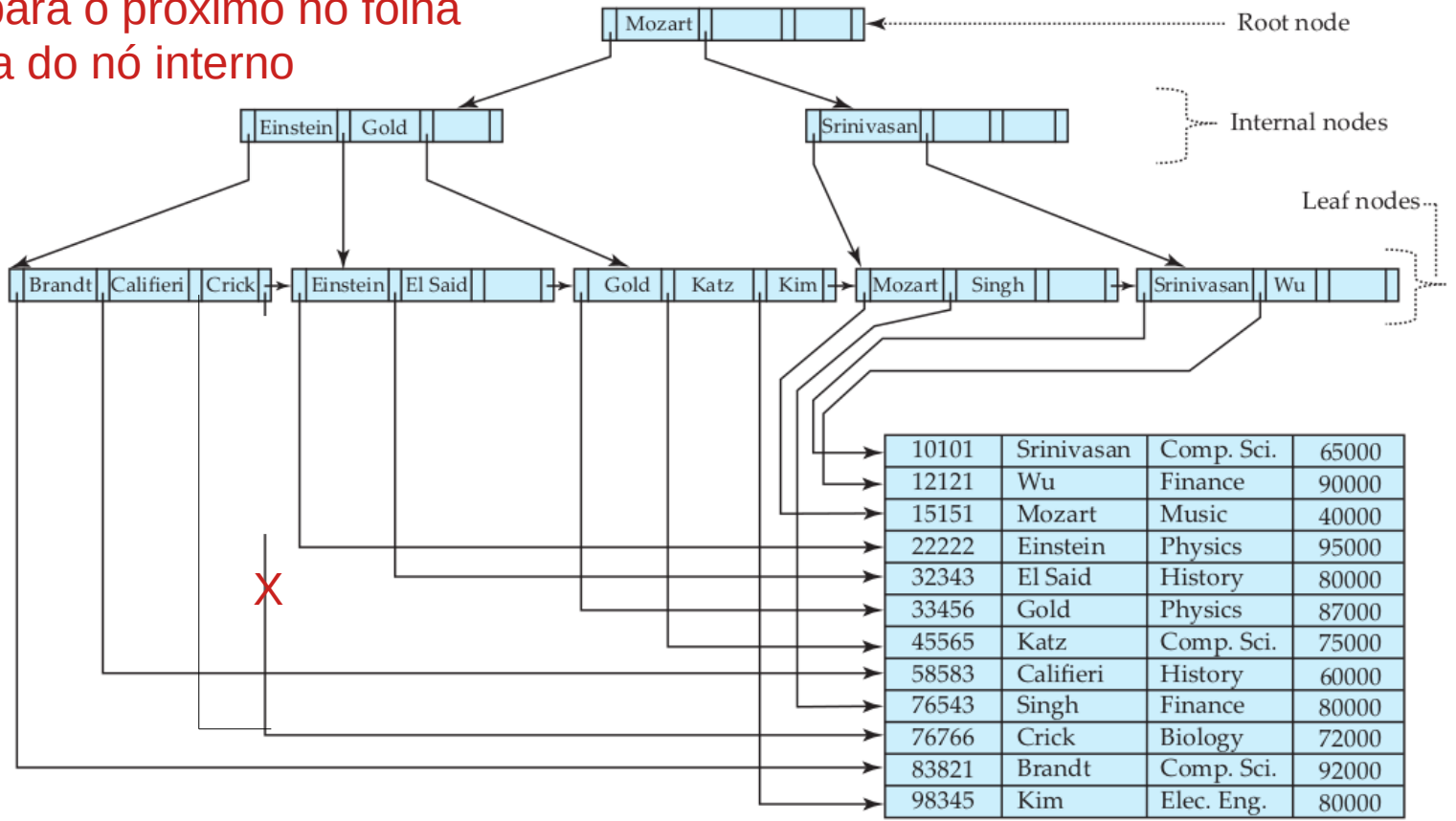
Não confundir com organização indexada!!!

(Silberschatz)

Árvore B+ como índice secundário

Folhas: devem conter TODAS as chaves do arquivo (ordenadas)
Cada folha: de t-1 a 2t-1 chaves
+ t-1 a 2t-1 ponteiros para os respectivos registros de cada chave
+ 1 ponteiro para o próximo nó folha
=> struct “parecida” com a do nó interno

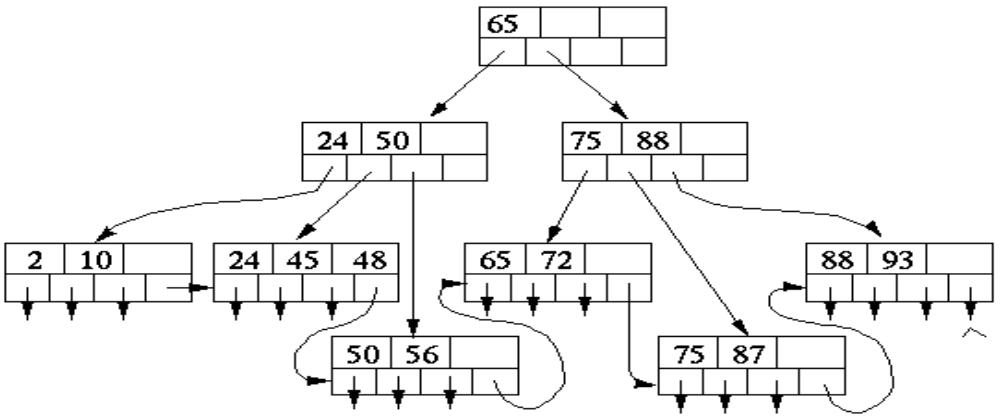
EP 2: Para simplificar, usaremos essa versão mas desconsiderando os blocos de registros => folhas possuem os 2t-1 ponteiros = NULL e o último filho deve apontar para o próximo nó irmão



Comentários sobre árvores B+

Adaptações dos algoritmos:

- Busca:



Nós internos (de índices)

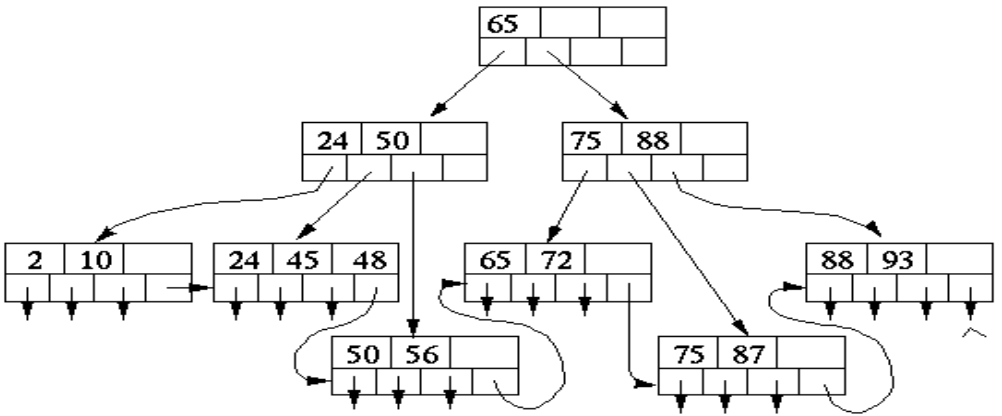
Note que há repetição entre os dois conjuntos

Nós folhas (de dados)

Comentários sobre árvores B+

Adaptações dos algoritmos:

- Busca: tem sempre que descer às folhas



Nós internos (de índices)

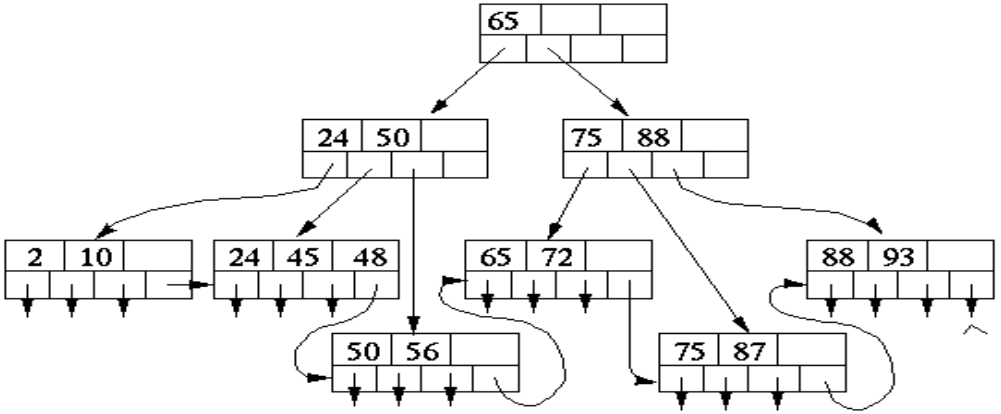
Note que há repetição entre os dois conjuntos

Nós folhas (de dados)

Comentários sobre árvores B+

Adaptações dos algoritmos:

- Inserção:
  - Split :
    - Chave mediana de uma **folha** é COPIADA para o nó pai
    - Chave mediana de um **nó interno** é MOVIDA para o nó pai



Nós internos (de índices)

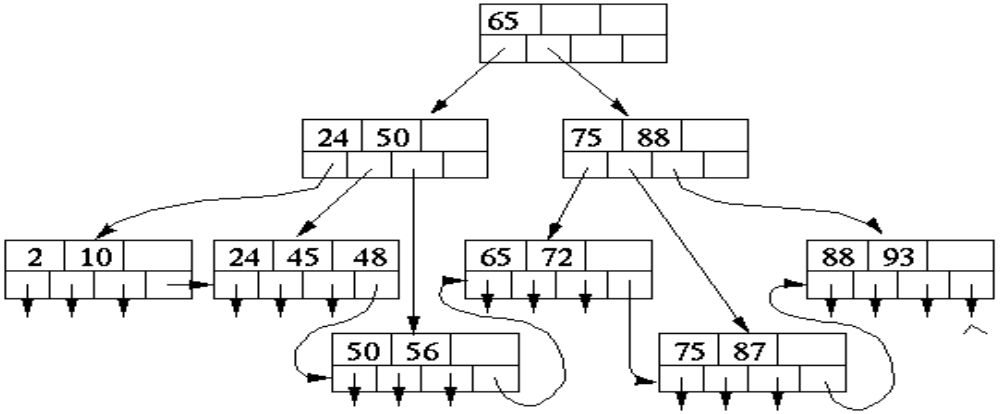
Note que há repetição entre os dois conjuntos

Nós folhas (de dados)

Comentários sobre árvores B+

Adaptações dos algoritmos:

- Remoção: nas folhas
  - Se k (chave a ser removida) ocorrer em um nó interno, o valor deve ser substituído pela chave do valor predecessor nas folhas (além da remoção daquele registro no nó folha)



Nós internos (de índices)

Note que há repetição entre os dois conjuntos

Nós folhas (de dados)

Comparação de complexidades

	Sequencial		Ligada	Indexada	Indexada Multinível	Árvores B	Árvores B+
	Não-Ordenado	Ordenado	Ordenado	Ordenado	Ordenado	Ordenado	Ordenado
Busca	O(b)	O(lg b)	O(b), O(lg b) só se usar FAT (discos pequenos)	O(log bi)	O(log $_{fbi}$ r^{1})	O(h = lg_t b)	O(h = lg_t b)
Inserção**	O(1) se tiver espaço no final, O(b) c.c.	O(1) se tiver espaço no bloco, O(b) c.c.	O(1)	O(b+bi) = O(b)	O(b+bi) = O(b)	O(h = lg_t b)	O(h = lg_t b)
Remoção* , **	O(1)	O(1)	O(1)	O(1)	O(1)	O(h = lg_t b)	O(h = lg_t b)
Leitura ordenada	ω(b) (depende do alg de ord. externa)	O(b), O(1) se fizer a leitura toda de uma vez	O(b)	O(b+bi) = O(b)	O(b+bi) = O(b)	O(n), n = nr de chaves	O(b_d), b_d = nr de blocos de dados (folhas)
Mínimo/máximo	O(b)	O(1)	O(1)	O(1)	O(1)	O(h = lg_t b)	O(h = lg_t b)
Modificação**	O(1)	O(b) se no campo chave, O(1) c.c.	O(1)	O(b) se no campo chave, O(1) c.c.	O(b) se no campo chave, O(1) c.c.	O(h = lg_t b)	O(h = lg_t b)
* considerando uso de bit de validade							
** considerando que já se sabe a localização do registro (busca já realizada)							

Outras variações

- Árvores B*:
  - Propostas por Knuth em 1973
  - Preenchimento mínimo (*fill factor*) de 2/3 (mais precisamente (2*t-1)/3)
  - Split postergado até que dois nós irmãos (imediatos) estejam cheios → split desses 2 nós em 3 nós
- Muitos sistemas gerenciadores de banco de dados usam B+ permitindo definir valores de *fill factor* entre 0.5 e 1.0:
  - Um para nós internos
  - Um para nós folhas

Referências

Livro do Cormen: (3ª ed.) cap 18

Livro do Drozdek (4ª ed) cap 7

Sobre árvores B+:

SILBERSCHATZ A. et al. Database System Concepts. 6th ed. Ed. McGrawHill. Seção 11.3