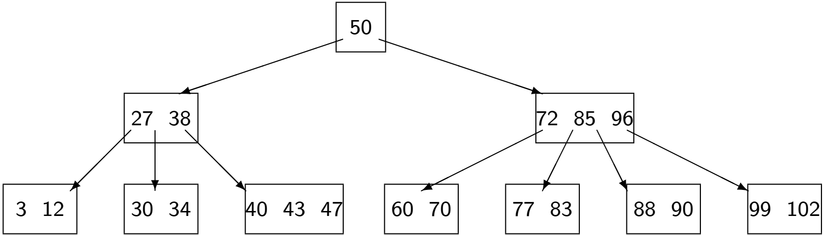
Aula 19: Árvores B

David Déharbe
Programa de Pós-graduação em Sistemas e Computação
Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada

Plano da aula



Introdução

Propriedades

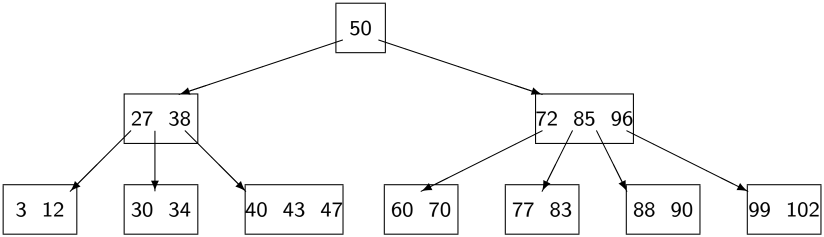
Operações

- Em 1971, Rudolf Bayer e Ed McCreight projetaram a estrutura de dados árvores B (*B-trees*).
- Complexidade
  - busca em $O(\lg n)$,
  - remoção em $O(\lg n)$ e
  - inserção em $O(\lg n)$
- Motivação: hierarquia de memória
- Árvores de busca, balanceadas
  - aumentar grau de ramificação
  - diminuir altura
- Aplicação: banco de dados, sistemas de arquivos

1. raiz r:
2. atributos do nó x:
   2.1 $x.n$: número de chaves de x;
   2.2 $x.keys$: vetor ordenado das chaves
   $$\forall i | 1 \leq i < x.n \cdot x.keys[i] < x.keys[i+1];$$
   2.3 $x.regs$: vetor de registros
   $$x.regs.len = x.keys.len \wedge \forall i \mid 1 \leq i \leq x.n \cdot x.regs[i].key = x.keys[i]$$
   2.4 $x.leaf$: booleano indicando se x é uma folha;
   2.5 se $\neg x.leaf$: $x.sub$ vetor de sub-árvores;
3. $x.leaf \vee len(x.sub) = 1 + x.n$
4. $k_i \in keys(x.sub[i]) \wedge k_{i+1} \in keys(x.sub[i+1]) \Rightarrow k_i < x.keys[i] < k_{i+1}$;
5. $\forall x \cdot x.leaf \Rightarrow level(x) = \alpha(r)$;
6. $g > 1$: grau de ramificação *mínimo* da árvore:
   6.1 $x \neq r \Rightarrow g - 1 \leq x.n \leq 2g - 1$;
   6.2 $1 \leq r.n \leq 2g - 1$,

Ilustração

$g = 3$



Grau de ramificação

- N: nó
- K: chave
- A_n: endereço de um nó
- A_r: endereço de um registro de dados
- $g_{max} \times A_n + (g_{max} - 1) \times (K + A_r) \leq N$
- Supondo: $N = 4096, K = A_n = A_r = 4$
- $g_{max} = 342$
-

altura	capacidade	
1	1×341	$= 341$
2	$(1 + 342) \times 341$	$= 116.963$
3	$(1 + 342 + 342^2) \times 341$	$= 40.001.674$

Grau de ramificação

- N: nó
- K: chave
- A_n: endereço de um nó
- A_r: endereço de um registro de dados
- $g_{max} \times A_n + (g_{max} - 1) \times (K + A_r) \leq N$
- Supondo: $N = 4096, K = A_n = A_r = 4$
- $g_{max} = 342$
-

altura	capacidade	
1	1×341	$= 341$
2	$(1 + 342) \times 341$	$= 116.963$
3	$(1 + 342 + 342^2) \times 341$	$= 40.001.674$

$g = 2$: árvore 2-3-4.

Altura de árvore B

Teorema

A altura a de uma árvore B com n registros e de grau mínimo g é tal que

$$a \leq \log_g \frac{n-1}{2} + 1.$$

PPgSC

Altura de árvore B

Demonstração.

Pior caso:

- raiz: uma chave e dois filhos,
- outros nós internos: $g - 1$ chaves e g filhos
- folhas: $g - 1$ chaves.

nível	quantidade de nós
1	1
2	2
3	$2g$
4	$2g^2$
a	$2g^{a-2}$

Demonstração.

$$
\begin{aligned}
n &\geq 1 + (g-1) \times 2 \times \Sigma_{i=0}^{a-2} g^i \\
n &\geq 1 + (g-1) \times 2 \times \frac{1 - g^{a-1}}{1 - g} \\
n &\geq 1 + 2 \times (g^{a-1} - 1) \\
\frac{n+1}{2} &\geq g^{a-1} \\
a &\leq \log_g \frac{n-1}{2} + 1
\end{aligned}
$$

Implementação

Escrita/leitura de nós

- WRITE-NODE
- READ-NODE

Busca

```
SEARCH-NODE(T, x, k)
i = 0
while i < x.n and x.keys[i] < k
    i = i + 1
if i ≤ x.n and x.keys[i] == k
    return x.refs[i]
if x.leaf
    return NIL
READ-NODE(T, x.sub[i])
return SEARCH-NODE(T, x.sub[i], k)
```

Busca

```
SEARCH-NODE(T, x, k)
i = 0
while i < x.n and x.keys[i] < k
    i = i + 1
if i ≤ x.n and x.keys[i] == k
    return x.refs[i]
if x.leaf
    return NIL
READ-NODE(T, x.sub[i])
return SEARCH-NODE(T, x.sub[i], k)
```

- uma chamada: $O(1)$ acesso disco, $O(g)$ busca sequencial;
- total: $O(a) = O(\log_g n)$ acessos disco, e $O(ag) = O(g \log_g n)$ operações processador.

- todas as folhas devem ter o mesmo nível antes e depois da inserção
- na descida recursiva na árvore B, garante-se que há espaço para armazenar o novo registro
- se um nó a visitar estiver cheio, ele é dividido em dois

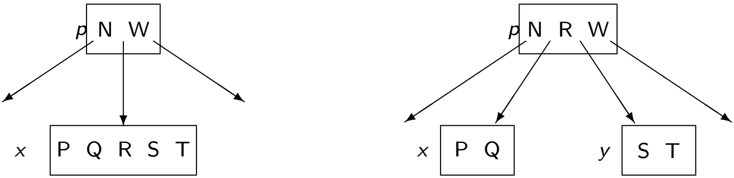
Divisão

- operação auxiliar para a inserção
- entrada: nó cheio x com $2g - 1$ chaves, tal que
  - x é a raiz ou
  - o nó pai de x não é cheio
- seja k é a chave mediana de x
- efeito:
  - se $x = r$, um novo nó raiz é criado, com a chave k
  - se $x \neq r$:
    - a chave k é inserida no nó pai
    - um novo nó y é criado, e recebe $g - 1$ chaves de x
    - x permanece com $g - 1$ chaves

PPgSC

Divisão
Exemplo

$g = 3$



(antes) (depois)

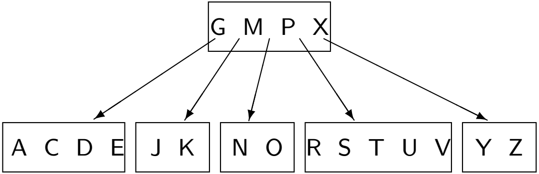
Divisão

Algoritmo

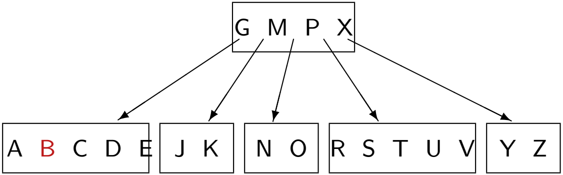
```
DIVIDE-NODE(T, p, i, x)
y = MAKE-NODE(T)
x.n = y.n = T.degree − 1
for j = 1 to y.n
    y.keys[j] = x.keys[j + y.n]
if not x.leaf
    for j = 1 to y.n + 1
        y.sub[j] = x.sub[j + y.n]
for j = p.n downto i
    p.sub[j + 1] = p.sub[j]
p.sub[i] = y
for j = p.n downto i − 1
    p.keys[j + 1] = p.keys[j]
p.keys[i − 1] = x.keys[x.n]
p.n = p.n + 1
WRITE-NODE(T, x)
WRITE-NODE(T, y)
WRITE-NODE(T, p)
```

Inserção

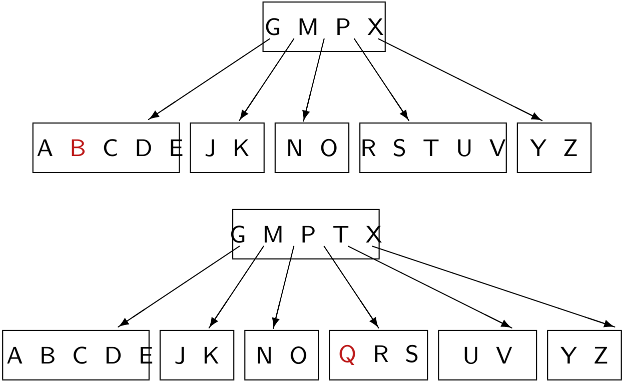
Exemplo



Estado inicial



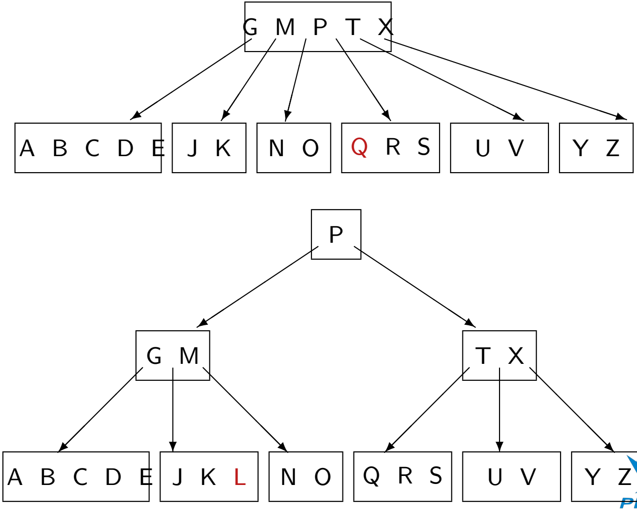
Inserção de B



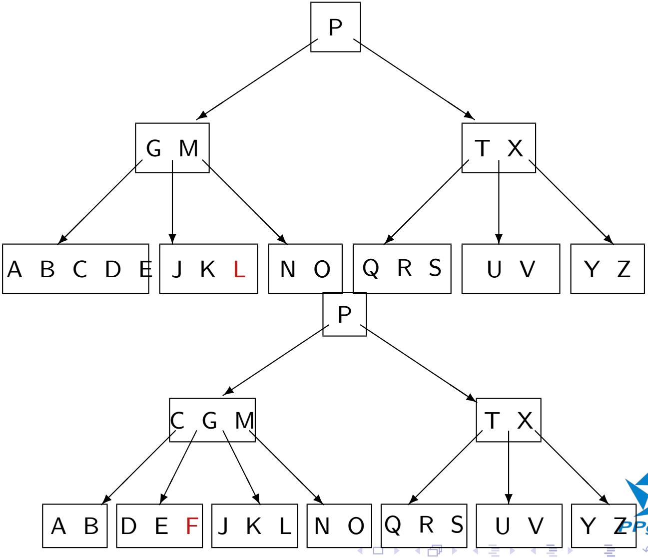
Inserção de Q

Inserção

Exemplo



Inserção de L



Inserção de F

Inserção

Algoritmo

```
INSERT(T, r)
if T.root == NIL
    y = MAKE-NODE(T)
    y.n = 1
    y.leaf = TRUE
    y.keys[0] = r.key
    y.refs[0] = r
elseif T.root.n == 2 × T.degree − 1
    y = MAKE-NODE(T)
    y.n = 1
    y.leaf = FALSE
    y.sub[0] = T.root
    T.root = y
    WRITE-NODE(T, y)
    DIVIDE-NODE(T, y, 0, y.sub[0])
NODE-INSERT(T, T.root, r)
```

Inserção

NODE-INSERT(T, x, r)

if $x.leaf$ == TRUE
    LEAF-INSERT(T, x, r)
else INTERIOR-INSERT(T, x, r)

LEAF-INSERT(T, x, r)

```
i = x.n − 1
while i ≥ 0 and r.key < x.keys[i]
        x.keys[i + 1] = x.keys[i]
        i = i − 1
x.keys[i + 1] = r.key
x.refs[i + 1] = r
NODE-WRITE(T, x)
```

Inserção

Sub-rotina auxiliar para inserção em nó interno

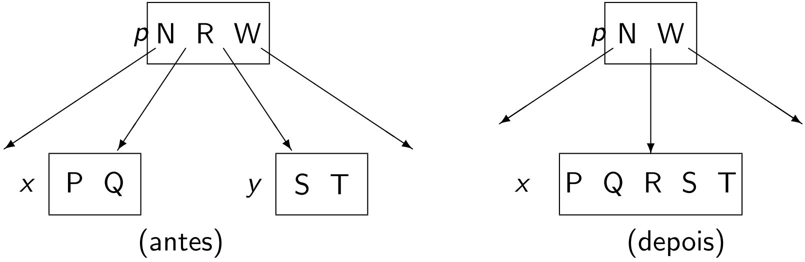
```
INTERIOR-INSERT(T, x, r)
i = x.n − 1
while i ≥ 0 and r.key < x.keys[i]
    x.keys[i + 1] = x.keys[i]
    i = i − 1
i = i + 1
y = READ-NODE(T, x.sub[i])
if y.n == 2 × T.degree − 1
    NODE-DIVIDE(T, y)
    if x.keys[i] < c
        i = i + 1
NODE-INSERT(T, y, r)
```

- todas as folhas devem ter o mesmo nível antes e depois da remoção
- na descida recursiva na árvore B, garante-se que pode se remover um registro sem quebrar o invariante da sub-árvore
- se um nó a visitar estiver com $g - 1$ chaves, opera-se uma operação de fusão com um nó vizinho

- dois nós filhos vizinhos x e y de um nó p
- $g - 1$ chaves cada
- seja k a chave “entre” x e y
- as chaves de x, k, e as chaves de y formam o novo nó

Fusão

Exemplo, $g = 3$



- se $x = T.root$ fica sem chaves
  - se tem filhos, só tem um, e estetorna-se a nova raiz
  - caso contrário, a árvore torna-se vazia

- se $k \in x.keys$, e $x.leaf$, k é eliminado de x.

- se $k \in x.keys$, e $\neg x.leaf$:
  - se o filho y que precede k em x tem pelo menos g chaves
    - achar o predecessor k' de k na subárvore y
    - remover recursivamente k',
    - substituir k por k' em x.

- se $k \in x.keys$, e $\neg x.leaf$:
  - senão, se o filho z que sucede k no nó x tem pelo menos g chaves:
    - achar o sucessor k' de k na subárvore z
    - remover recursivamente k',
    - substituir k por k' em x.

- se $k \in x.keys$, e $\neg x.leaf$:
  - senão, $y.n = z.n = g - 1$:
    - y e z são fusionados em y, com chave mediana k
    - k e z são eliminados de x
    - y tem $2g - 1$ chaves
    - liberar o nó z e
    - remover recursivamente k de y.

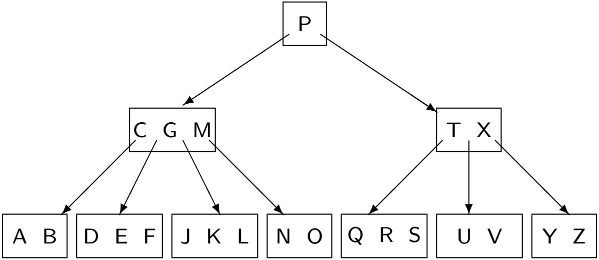
- se $k \notin x.keys$
  - determinar a sub-árvore y que deve conter k
  - se $y.n \geq n$, eliminar k de y

- se $k \notin x.keys$
  - determinar a sub-árvore y que deve conter k
  - se $y.n = g - 1$ e tem um vizinho z com $z.n \geq g$
    - deslocar uma chave e um filho de x em y
    - deslocar uma chave e um filho de z para x
    - liminar recursivamente k de y.

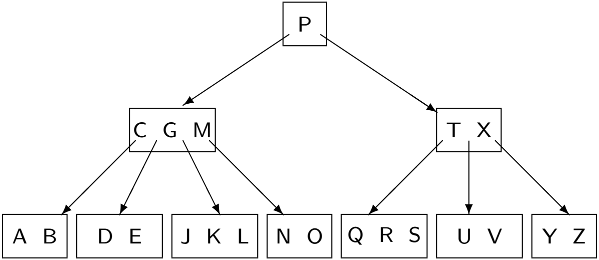
- se $k \notin x.keys$
  - determinar a sub-árvore y que deve conter k
  - se os vizinhos de y tem $g-1$ chaves,
    - fusionar y com um dos irmãos
    - liminar recursivamente k de y.

Remoção

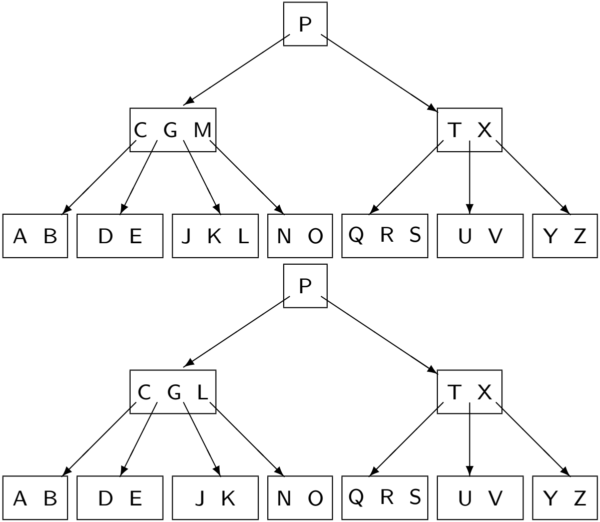
Exemplos



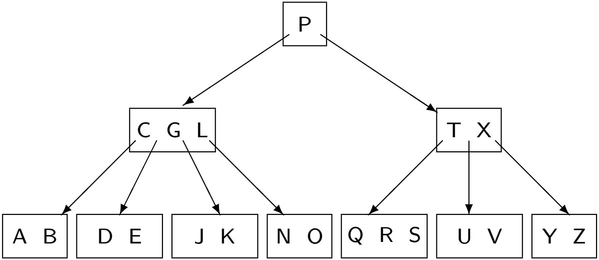
Estado inicial

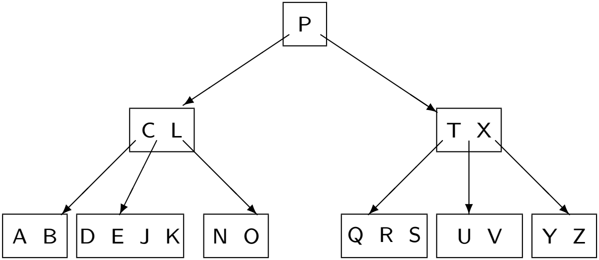


Remoção de F

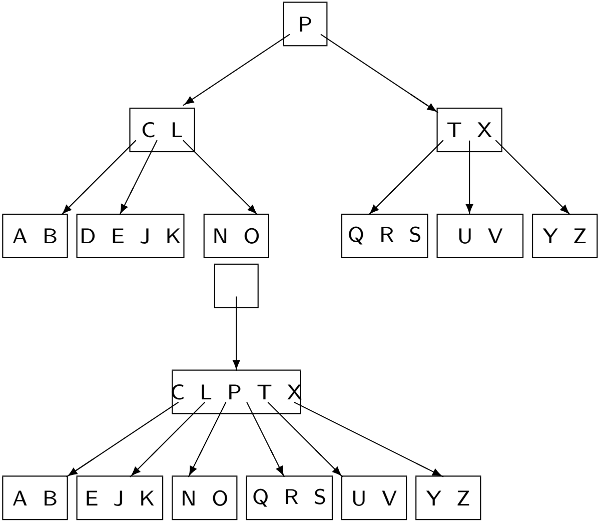


Remoção de M



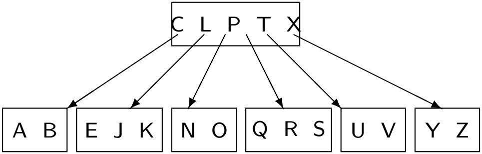


Remoção de G

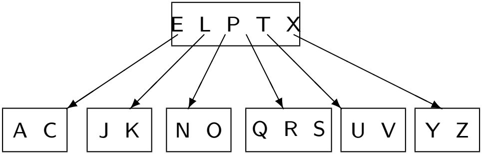


Remoção de D

A altura diminui



B é removido



Variantes

- Árvore B+: referências são armazenadas nas folhas apenas
  - um número de chaves maior pode ser armazenado nos nós internos
  - a altura é menor
  - o tempo de acesso é uniforme
- Árvores B*: nós internos tem entre $2g/3$ e g sub-árvores
  - melhora ocupação do espaço alocado

PPgSC

Exercícios

1. Projetar o algoritmo de remoção em árvores B.
2. S é uma coleção de registros S. O objetivo é criar uma árvore B contendo exatamente todos os registros de S.
   - A árvore B pode ser construída por inserção sucessiva. Qual seria o custo?
   - Projetar uma solução mais eficiente.
3. Em uma árvore B+, K, R e N são respectivamente o tamanho da representação de uma chave, de um registro, e da referência para um nó; g é o grau máximo; h é a altura da árvore.
   Expressar o número máximo de referências que esta árvore pode conter.