Algoritmos e Estruturas de Dados II
Aula 23 – Árvores B

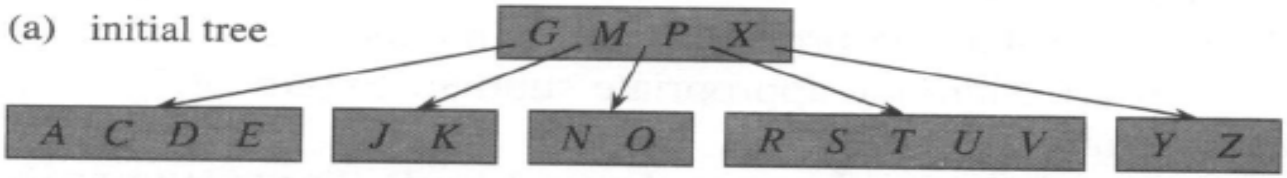
Prof. Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2025

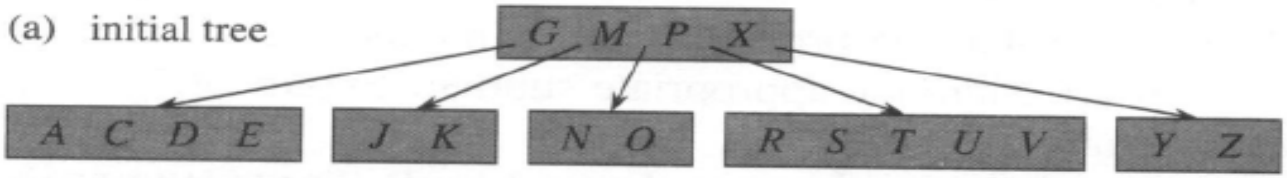
Alocação de Blocos na Memória Secundária

- Leitura, escrita, buscas, etc., são realizadas por blocos.
- Os arquivos não são estáticos, eles crescem e diminuem
- Estratégias de alocação de blocos no disco e organização de registros pelos blocos devem considerar esse fato:
  - Sequencial não ordenado (*heap files*)
  - Sequencial ordenado (*sorted files*)
  - Por listas ligadas
  - Indexado
  - Árvores B / B+
  - *Hashing*

Árvores B - Inserção - T=3

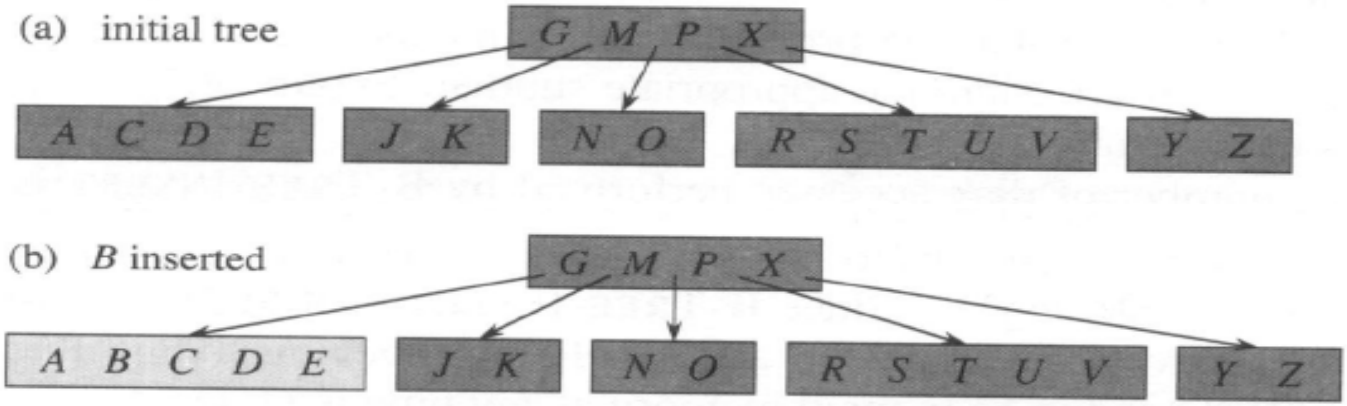


Árvores B - Inserção - T=3



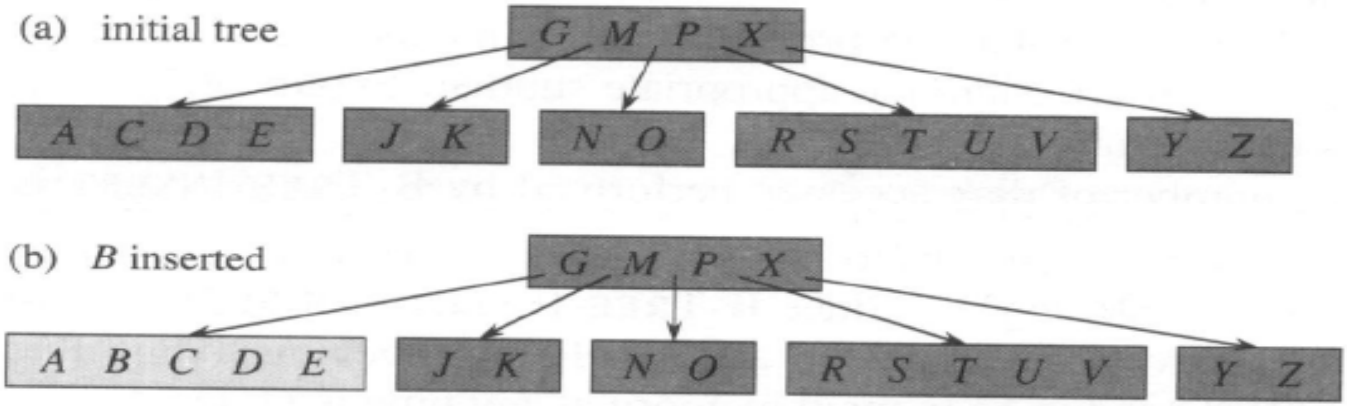
Queremos inserir B

Árvores B - Inserção - T=3



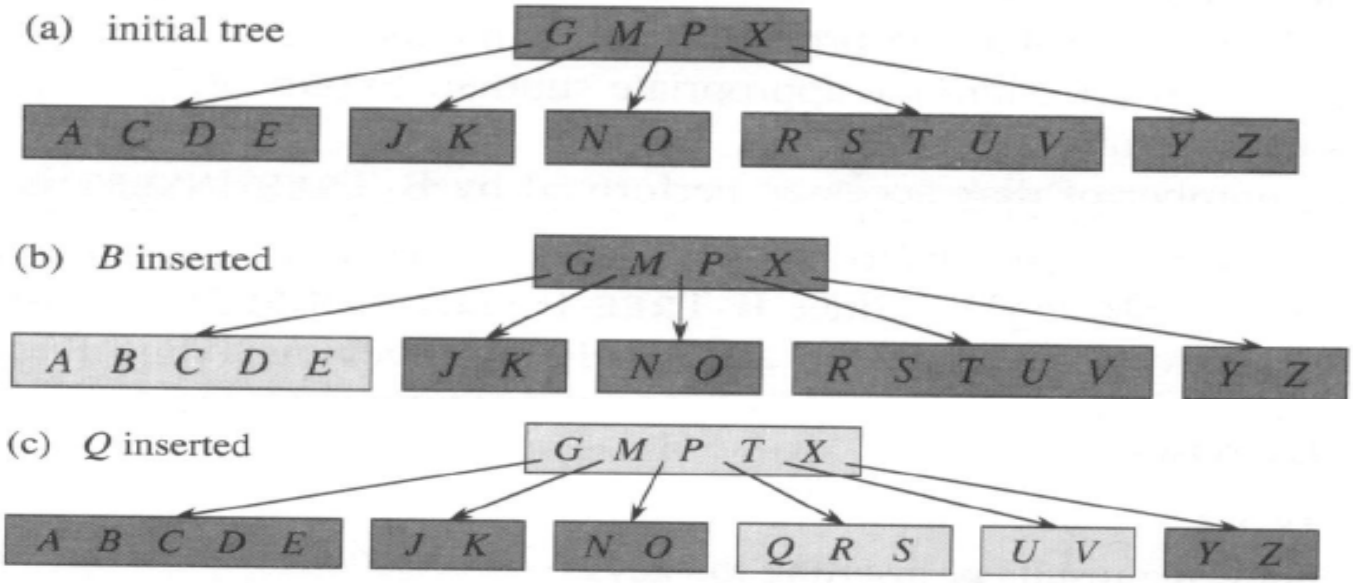
Queremos inserir B

Árvores B - Inserção - T=3



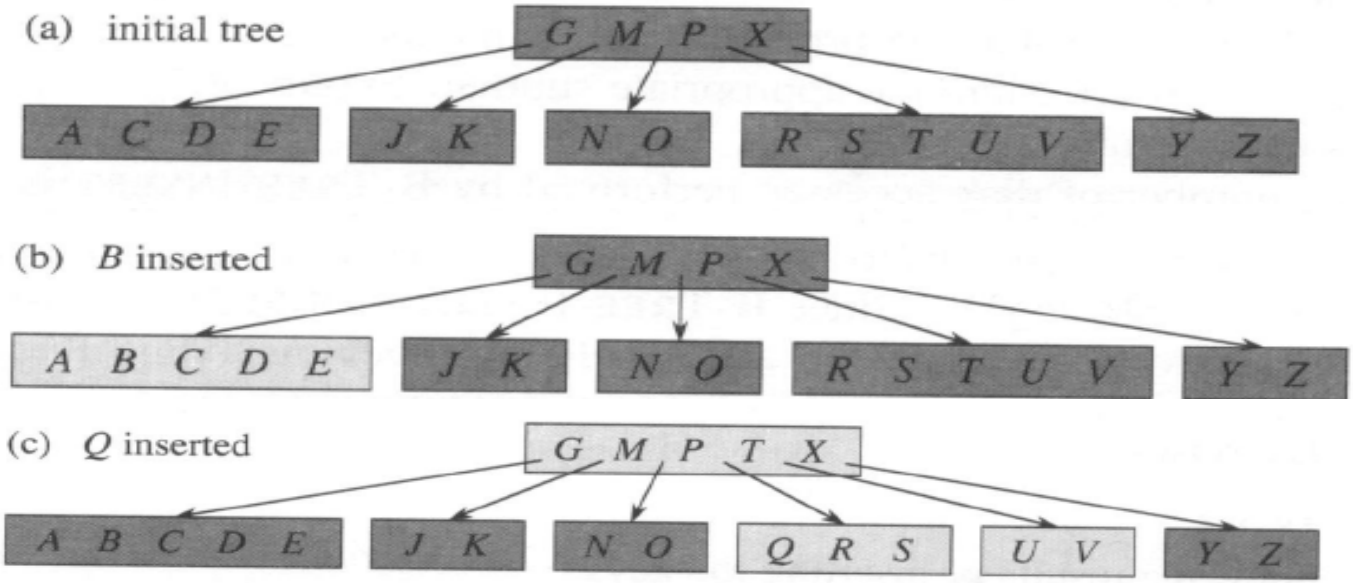
Queremos inserir Q

Árvores B - Inserção - T=3



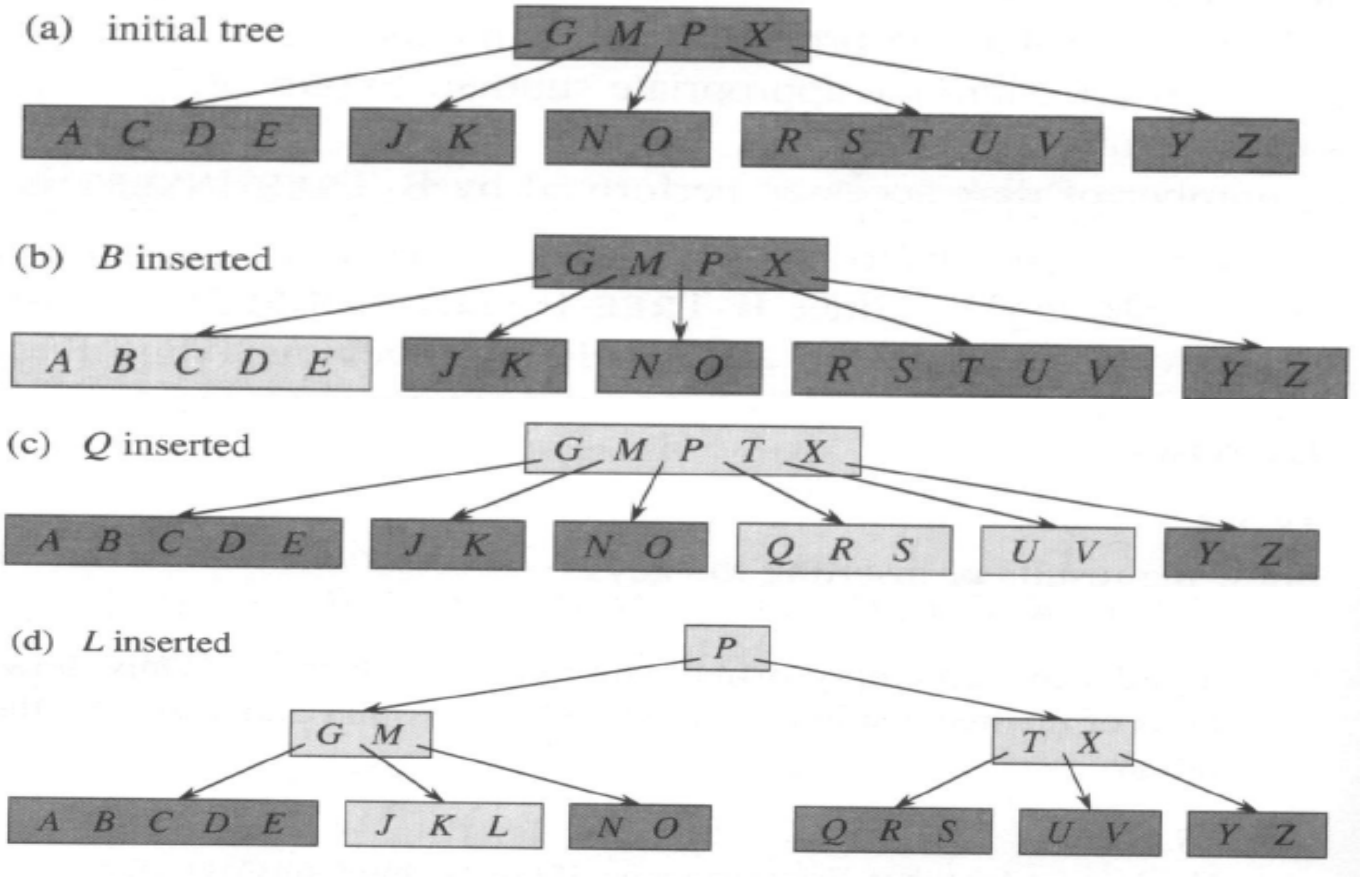
Queremos inserir Q

Árvores B - Inserção - T=3



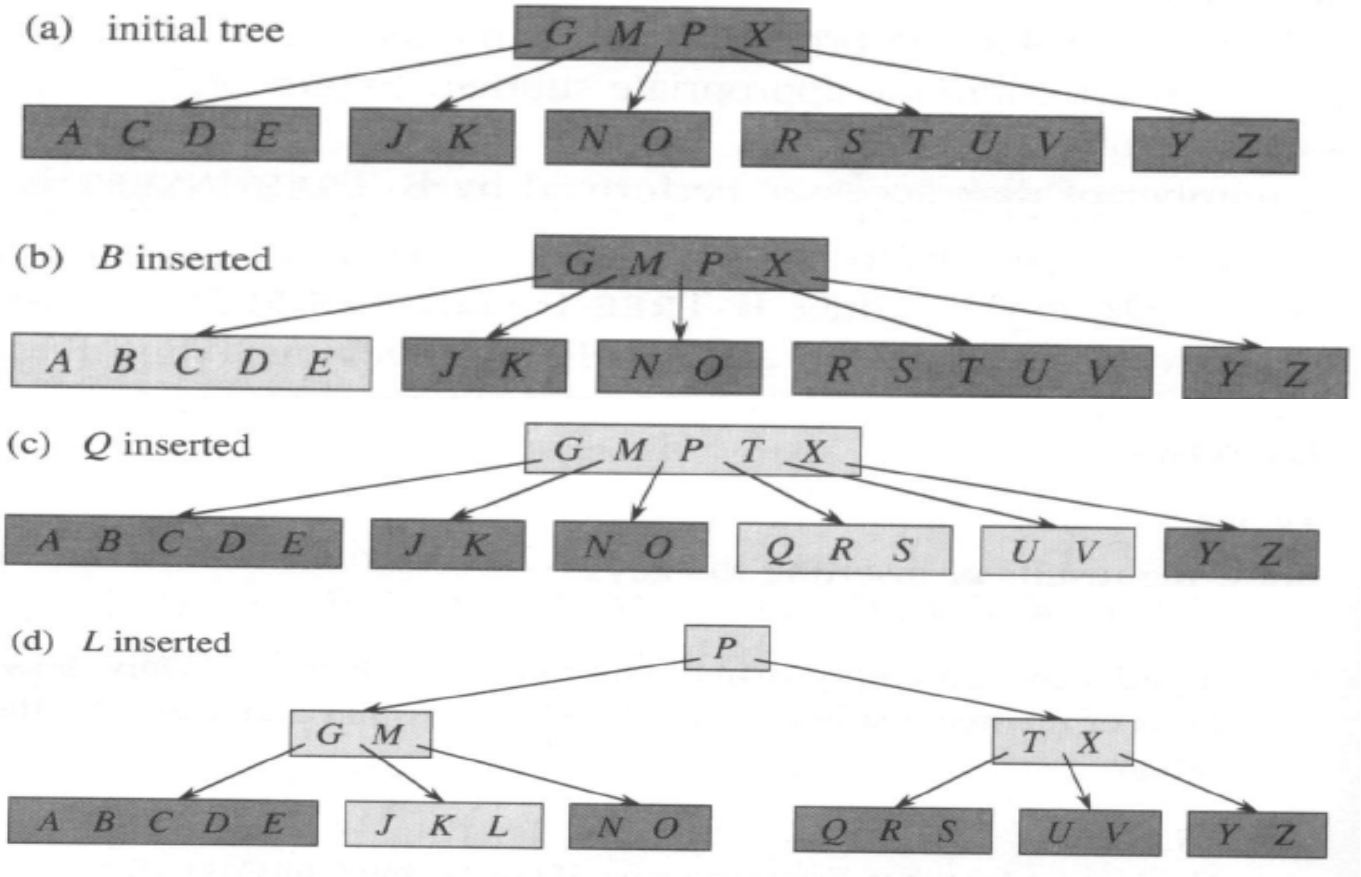
Queremos inserir L

Árvores B - Inserção - T=3



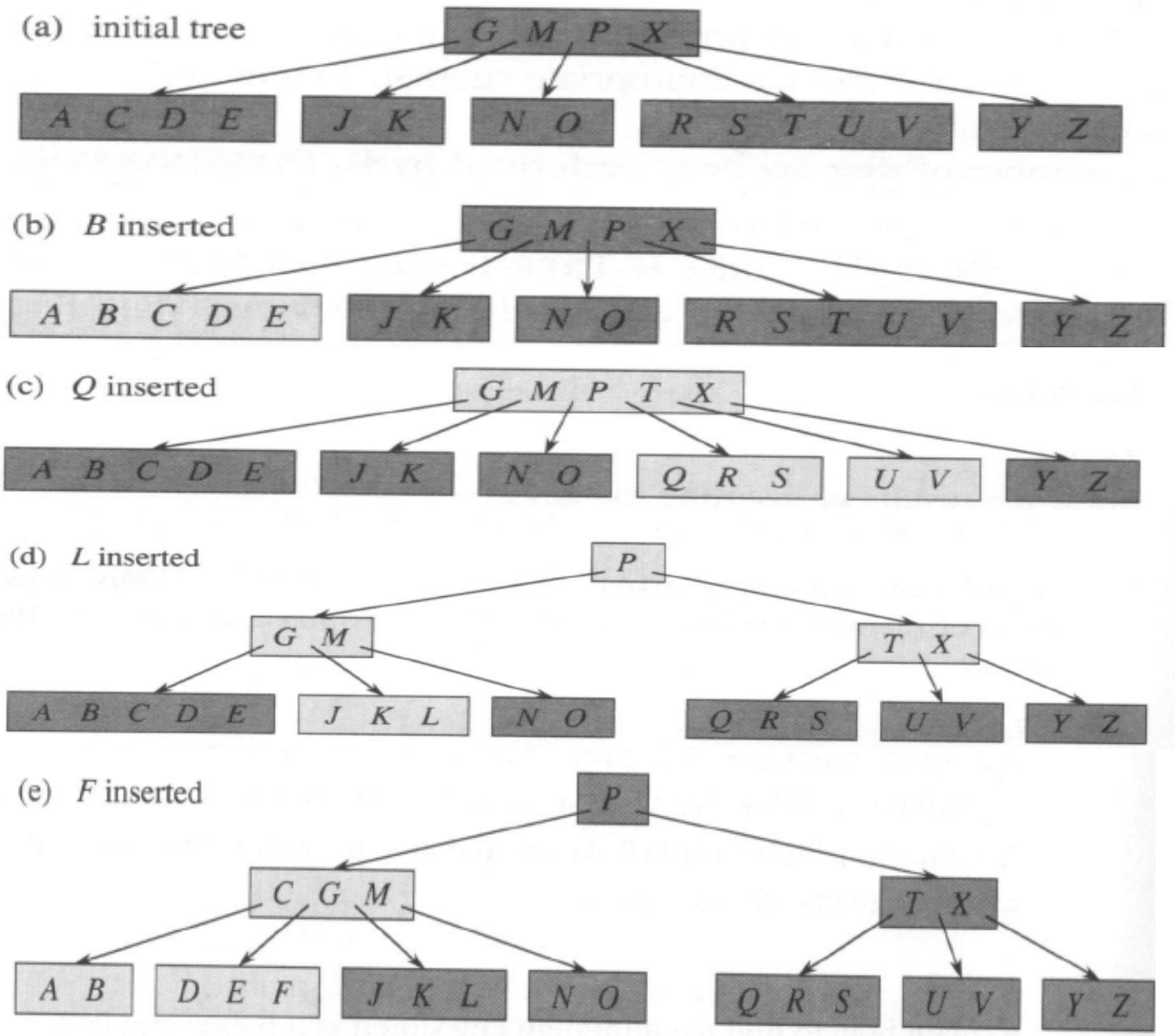
Queremos inserir L

Árvores B - Inserção - T=3



Queremos inserir F

Árvores B - Inserção - T=3



Queremos inserir F

Árvores B - Inserção - Func. Principal

Inserir(T, k):

- T: Árvore B em que a chave será inserido
- k: chave a ser inserida

Árvores B - Inserção - Func. Principal

Inserir(T, k):

- T: Árvore B em que a chave será inserido
- k: chave a ser inserida

1. Se houver espaço na raiz, chama função recursiva de inserção

Árvores B - Inserção - Func. Principal

Inserir(T, k):

- T: Árvore B em que a chave será inserido
- k: chave a ser inserida

1. Se houver espaço na raiz, chama função recursiva de inserção
2. Caso contrário, divide a raiz (criando uma raiz s) e chama a função recursiva de inserção a partir de s

Árvores B - Inserção - Func. Principal

Inserir(T, k):

- T: Árvore B em que a chave será inserido
- k: chave a ser inserida

1. Se houver espaço na raiz, chama função recursiva de inserção
2. Caso contrário, divide a raiz (criando uma raiz *s*) e chama a função recursiva de inserção a partir de *s*

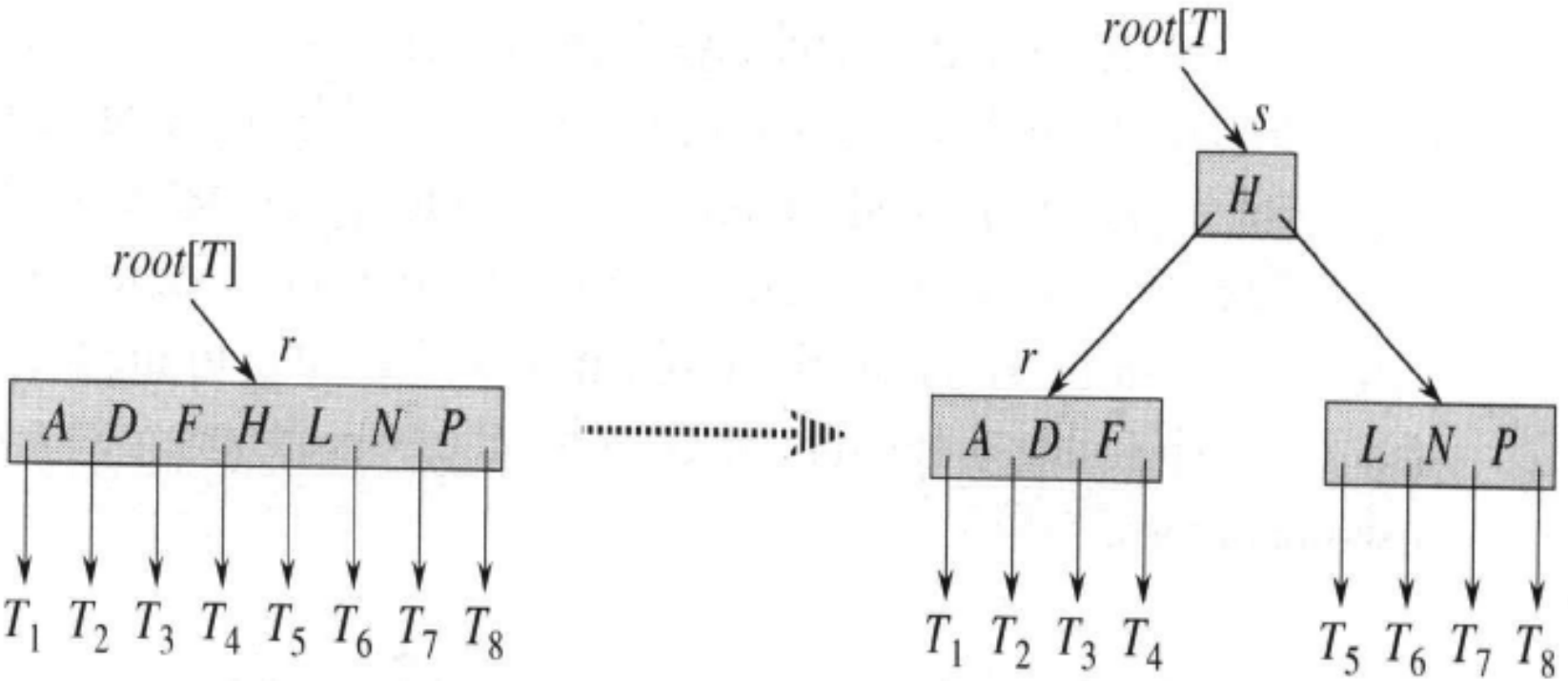
```
B-TREE-INSERT(T, k)
 1  r ← root[T]
 2  if n[r] = 2t − 1
 3     then s ← ALLOCATE-NODE()
 4          root[T] ← s
 5          leaf[s] ← FALSE
 6          n[s] ← 0
 7          c1[s] ← r
 8          B-TREE-SPLIT-CHILD(s, 1, r)
 9          B-TREE-INSERT-NONFULL(s, k)
10     else B-TREE-INSERT-NONFULL(r, k)
```

(CORMEN, LEISERSON & RIVEST, 2009)

Árvores B - Inserção - Raiz Cheia

Raiz cheia no início da inserção:

- Primeiro divide a raiz
- Depois reinicia a inserção a partir da nova raiz



(CORMEN, LEISERSON & RIVEST, 2009)

Árvores B - Inserção - Func. Auxiliar

InserirAux(x, k): x: nó atual; k: chave a ser inserida[i]

[i] Considera apenas nós não cheios. Qualquer nó cheio no "caminho" da inserção já terá sido dividido nas chamadas anteriores da função.

Árvores B - Inserção - Func. Auxiliar

InserirAux(x, k): *x*: nó atual; *k*: chave a ser inserida[i]

1. Se nó é folha:
   - Insere no local correto (lógica semelhante ao *insertion sort*)
   - Salva o nó / bloco no disco

[i] Considera apenas nós não cheios. Qualquer nó cheio no "caminho" da inserção já terá sido dividido nas chamadas anteriores da função.

Árvores B - Inserção - Func. Auxiliar

InserirAux(x, k): x: nó atual; k: chave a ser inserida[i]

1. Se nó é folha:
   - Insere no local correto (lógica semelhante ao *insertion sort*)
   - Salva o nó / bloco no disco
2. Caso contrário:
   - Encontra o filho para continuar recursivamente a inserção
   - Se o filho estiver lotado, divida-o
   - Continue recursivamente a inserção

[i] Considera apenas nós não cheios. Qualquer nó cheio no "caminho" da inserção já terá sido dividido nas chamadas anteriores da função.

Árvores B - Inserção - Func. Auxiliar

```
B-TREE-INSERT-NONFULL(x, k)
 1  i ← n[x]
 2  if leaf[x]
 3     then while i ≥ 1 and k < key_i[x]
 4                do key_{i+1}[x] ← key_i[x]
 5                   i ← i − 1
 6          key_{i+1}[x] ← k
 7          n[x] ← n[x] + 1
 8          DISK-WRITE(x)
 9     else while i ≥ 1 and k < key_i[x]
10                do i ← i − 1
11          i ← i + 1
12          DISK-READ(c_i[x])
13          if n[c_i[x]] = 2t − 1
14            then B-TREE-SPLIT-CHILD(x, i, c_i[x])
15                 if k > key_i[x]
16                   then i ← i + 1
17          B-TREE-INSERT-NONFULL(c_i[x], k)
```

(CORMEN, LEISERSON & RIVEST, 2009)

Árvores B - Inserção - Func. Auxiliar

```
B-TREE-INSERT-NONFULL(x, k)
 1  i ← n[x]
 2  if leaf[x]
 3     then while i ≥ 1 and k < key_i[x]
 4             do key_{i+1}[x] ← key_i[x]
 5                i ← i − 1
 6          key_{i+1}[x] ← k
 7          n[x] ← n[x] + 1
 8          DISK-WRITE(x)
 9     else while i ≥ 1 and k < key_i[x]
10             do i ← i − 1
11          i ← i + 1
12          DISK-READ(c_i[x])
13          if n[c_i[x]] = 2t − 1
14            then B-TREE-SPLIT-CHILD(x, i, c_i[x])
15                 if k > key_i[x]
16                   then i ← i + 1
17          B-TREE-INSERT-NONFULL(c_i[x], k)
```

Complexidade:

(CORMEN, LEISERSON & RIVEST, 2009)

Árvores B - Inserção - Func. Auxiliar

```
B-TREE-INSERT-NONFULL(x, k)
 1  i ← n[x]
 2  if leaf[x]
 3     then while i ≥ 1 and k < key_i[x]
 4             do key_{i+1}[x] ← key_i[x]
 5                i ← i − 1
 6          key_{i+1}[x] ← k
 7          n[x] ← n[x] + 1
 8          DISK-WRITE(x)
 9     else while i ≥ 1 and k < key_i[x]
10             do i ← i − 1
11          i ← i + 1
12          DISK-READ(c_i[x])
13          if n[c_i[x]] = 2t − 1
14            then B-TREE-SPLIT-CHILD(x, i, c_i[x])
15                 if k > key_i[x]
16                   then i ← i + 1
17          B-TREE-INSERT-NONFULL(c_i[x], k)
```

(CORMEN, LEISERSON & RIVEST, 2009)

Complexidade:

Acessos ao disco:
$O(h) = O(log_t(b))$

Árvores B - Inserção - Func. Auxiliar

```
B-TREE-INSERT-NONFULL(x, k)
 1  i ← n[x]
 2  if leaf[x]
 3     then while i ≥ 1 and k < key_i[x]
 4             do key_{i+1}[x] ← key_i[x]
 5                i ← i − 1
 6          key_{i+1}[x] ← k
 7          n[x] ← n[x] + 1
 8          DISK-WRITE(x)
 9     else while i ≥ 1 and k < key_i[x]
10             do i ← i − 1
11          i ← i + 1
12          DISK-READ(c_i[x])
13          if n[c_i[x]] = 2t − 1
14            then B-TREE-SPLIT-CHILD(x, i, c_i[x])
15                 if k > key_i[x]
16                   then i ← i + 1
17          B-TREE-INSERT-NONFULL(c_i[x], k)
```

(CORMEN, LEISERSON & RIVEST, 2009)

Complexidade:

Acessos ao disco:
$O(h) = O(log_t(b))$

CPU: $O(t * h) = O(t * log_t(b))$

Árvores B - Inserção - Func. Auxiliar

```
void insereNoNaoCheio(No* atual, Registro* reg){
  int i = atual->numChaves-1;
  if (atual->folha){
    while (i>=0 && reg->chave < atual->regs[i].chave){
      atual->regs[i+1] = atual->regs[i];
      i--;
    }
    atual->regs[i+1] = *reg;
    atual->numChaves++;
    salvarNoDisco(atual);
  } ...
```

Árvores B - Inserção - Func. Auxiliar

```
  }else{
    while (i>0 && reg->chave<atual->regs[i].chave) i--;
    if (reg->chave > atual->regs[i].chave) i++;
    lerDoDisco(atual->filhos[i]);
    if(atual->filhos[i]->numChaves==2*T-1){
      divideNoFilho(atual, i, atual->filhos[i]);
      if (reg->chave > atual->regs[i].chave) i++;
    }
    insereNoNaoCheio(atual->filhos[i], reg);
  }
}
```

Árvores B - Remoção - Func. Auxiliar

Há três casos principais:

- A chave está no nó atual, que é uma folha (com ao menos T elementos)
- A chave está no nó atual, que é um nó interno
- A chave não está no nó atual, que é um nó interno

Árvores B - Remoção - Func. Auxiliar (1)

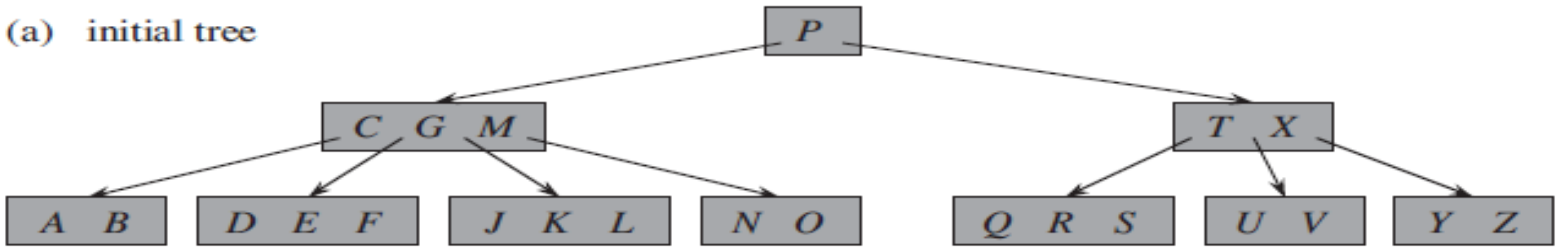
- B-Tree-Delete(x,k): remoção da chave k da subárvore com raiz x.

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

Árvores B - Remoção - Func. Auxiliar (1)

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

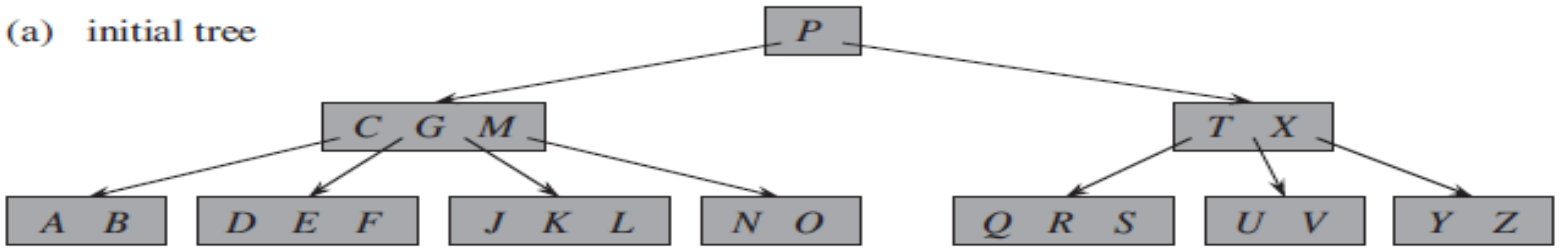
1. Se a chave k está no nó x e x é uma folha,



Árvores B - Remoção - Func. Auxiliar (1)

- 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,

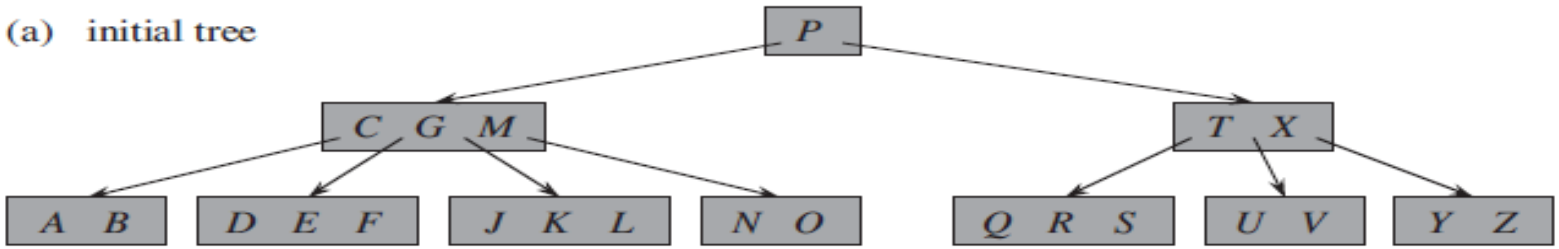


Queremos remover F

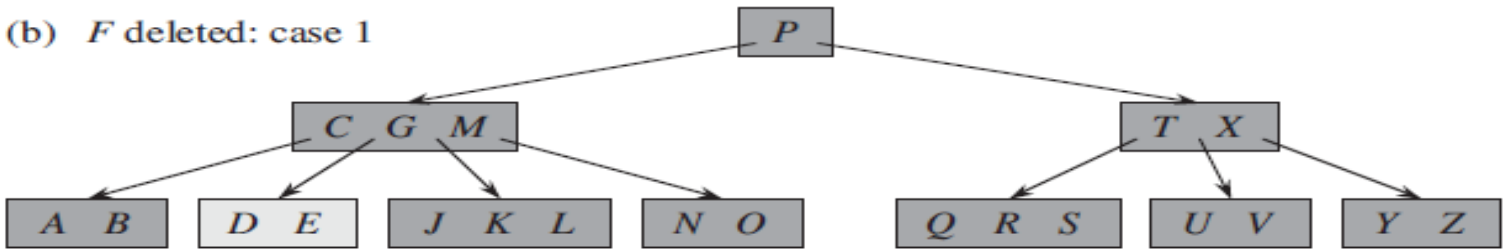
Árvores B - Remoção - Func. Auxiliar (1)

- 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,



Queremos remover F



Árvores B - Remoção - Func. Auxiliar (2)

- 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.

     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.

Árvores B - Remoção - Func. Auxiliar (2)

- 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.

     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.

O caso 2b é simétrico a 2a, por isso veremos dois exemplos de 2a. Mas existe ainda um terceiro caso, que será visto adiante.

Árvores B - Remoção - Func. Auxiliar (2a)

- 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:

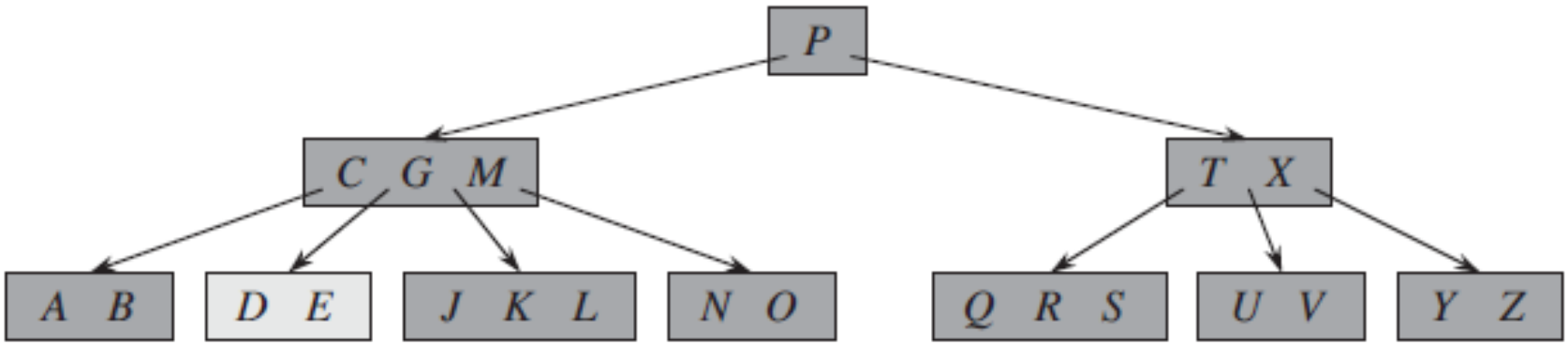
     a) Se o filho y que precede k no nó x tem pelo menos t chaves, então encontre o predecessor k' de k na subárvore com raiz y. Delete recursivamente k', e substitua k por k' em x.

Árvores B - Remoção - Func. Auxiliar (2a)

- 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.

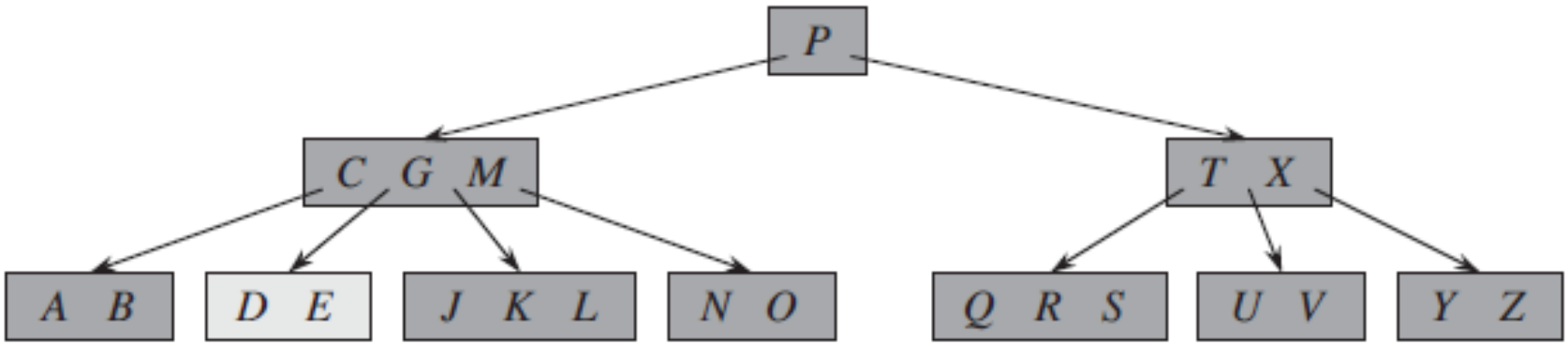


Árvores B - Remoção - Func. Auxiliar (2a)

- 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.



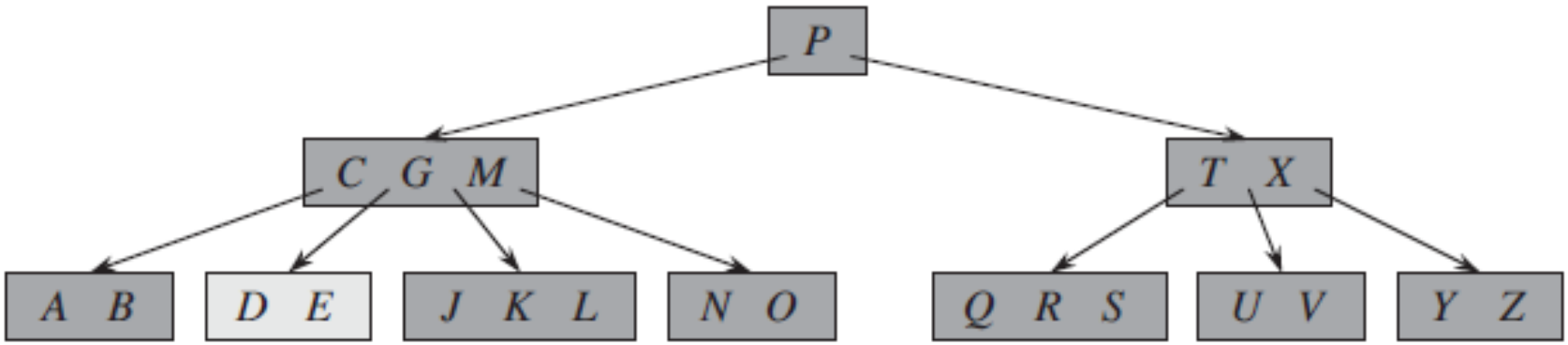
Queremos remover M

Árvores B - Remoção - Func. Auxiliar (2a)

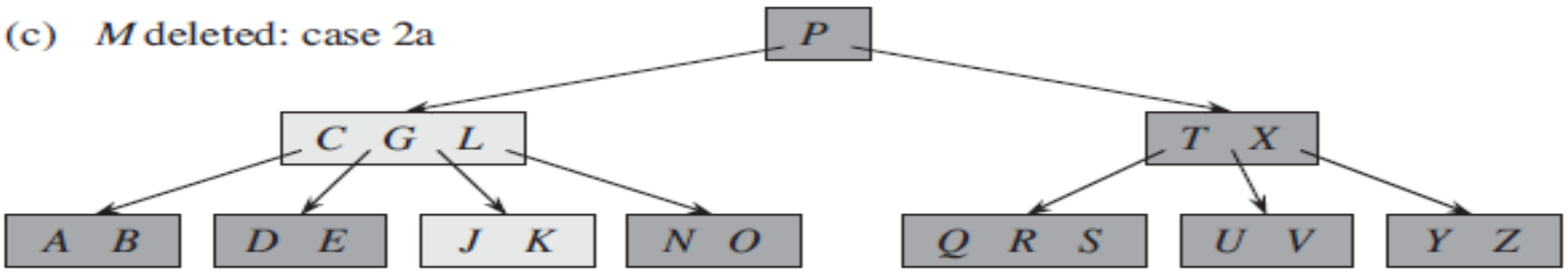
- B-Tree-Delete(x,k): remoção da chave k da subárvore com raiz x.

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

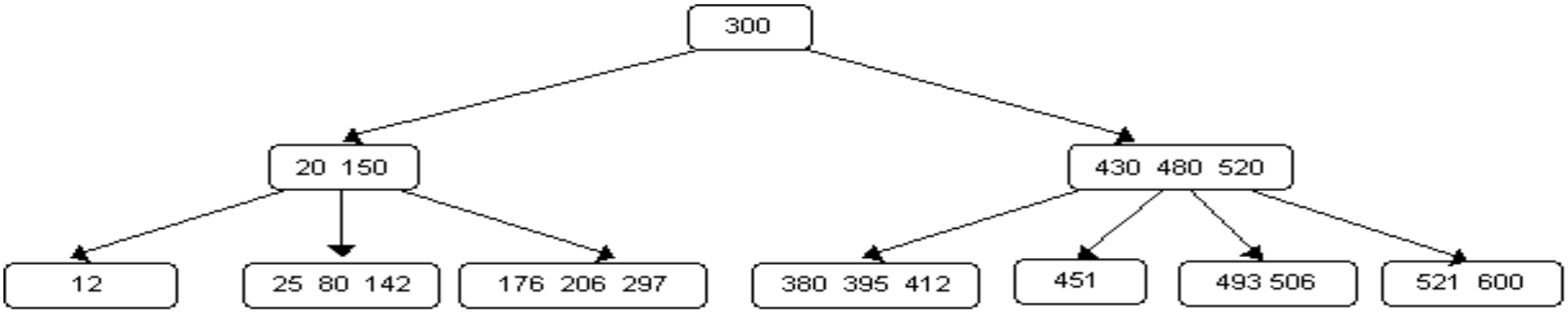
     a) Se o filho y que precede k no nó x tem pelo menos t chaves, então encontre o predecessor k' de k na subárvore com raiz y. Delete recursivamente k', e substitua k por k' em x.



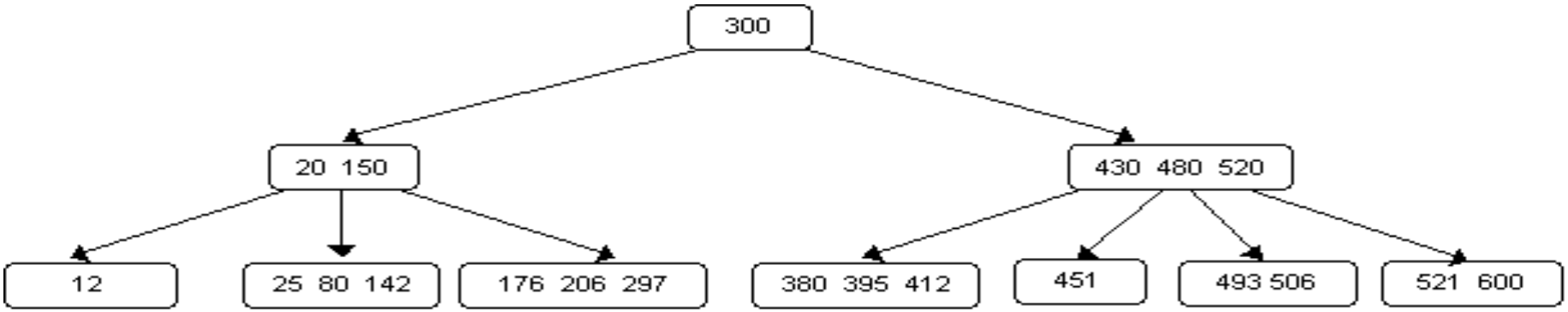
Queremos remover M



Árvores B - Remoção - Func. Auxiliar (2a)

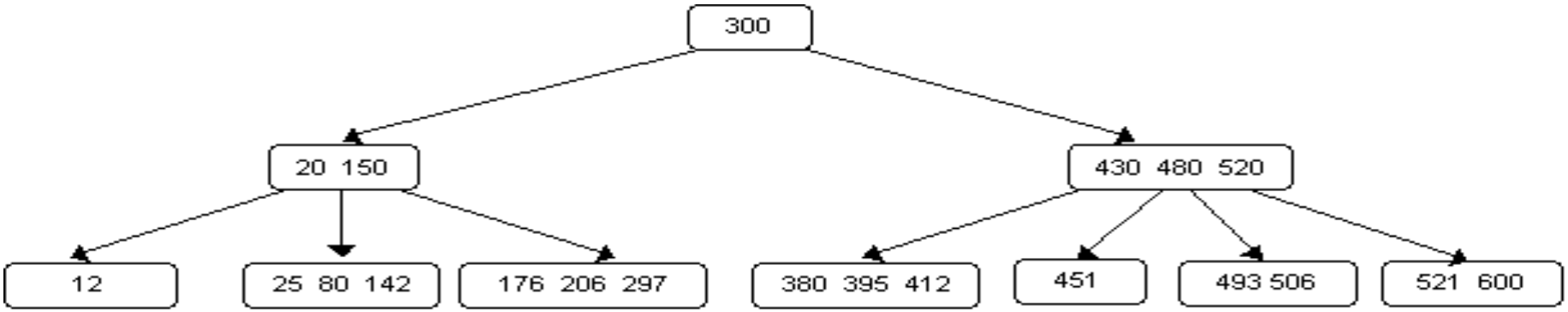


Árvores B - Remoção - Func. Auxiliar (2a)

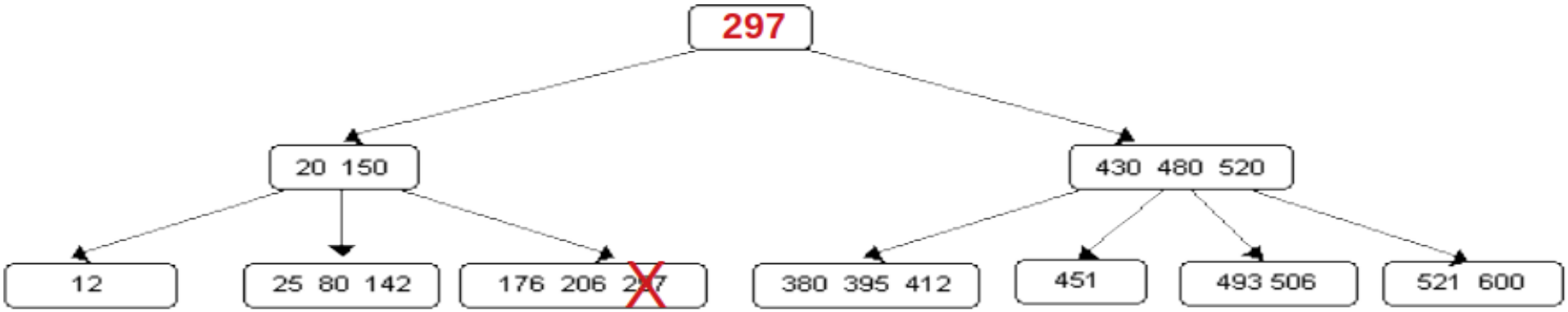


Queremos remover 300

Árvores B - Remoção - Func. Auxiliar (2a)



Queremos remover 300



Árvores B - Remoção - Func. Auxiliar (2c)

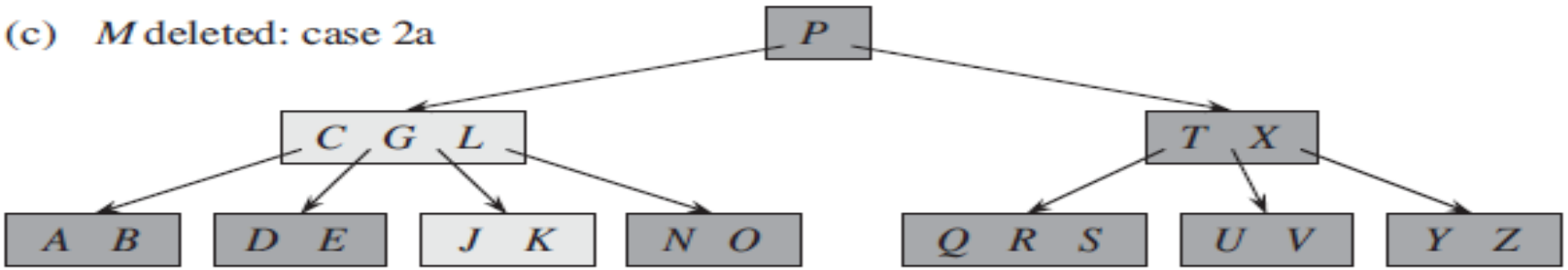
2. Se a chave k está no nó x e x é um nó interno, faça:

   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.

Árvores B - Remoção - Func. Auxiliar (2c)

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

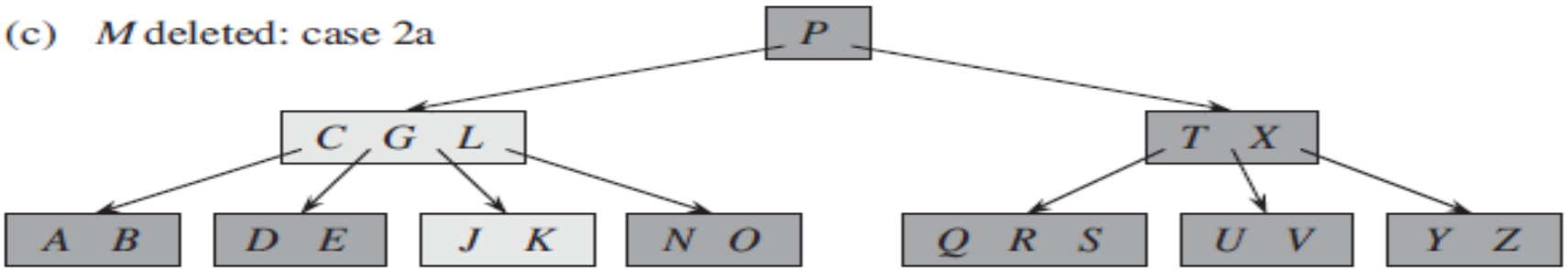
   c) Caso contrário, se ambos y e z possuem apenas $t - 1$ chaves, faça a junção de k e todas as chaves de z em y, de forma que x perde tanto a chave k como o ponteiro para z, e y agora contém $2t - 1$ chaves. Então, libere z e delete recursivamente k de y.



Árvores B - Remoção - Func. Auxiliar (2c)

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.

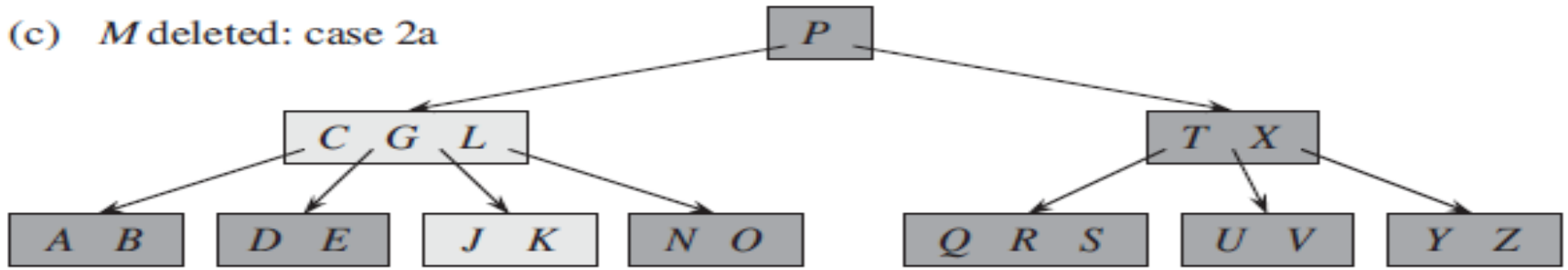


Queremos remover G

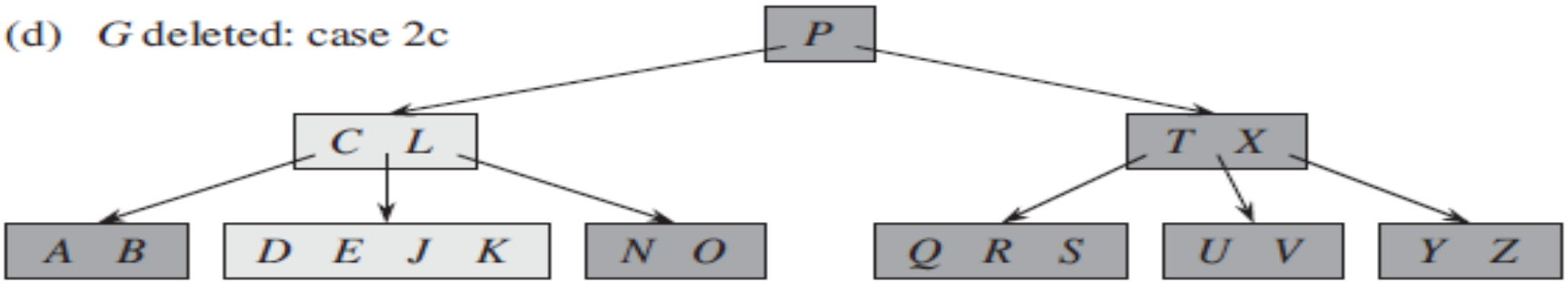
Árvores B - Remoção - Func. Auxiliar (2c)

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.



Queremos remover G



Árvores B - Remoção - Func. Auxiliar (3)

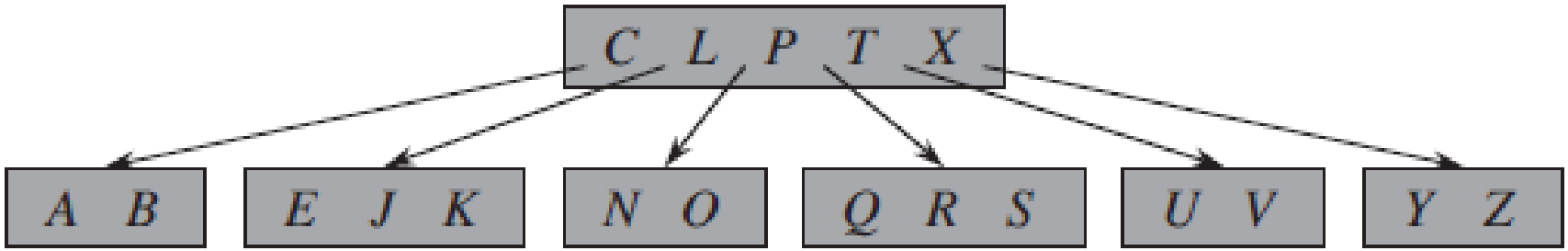
- B-Tree-Delete(x,k): remoção da chave k da subárvore com raiz x.
  3. Se a chave k não está presente no no interno x, determine a raiz $c_i[x]$ da subárvore apropriada que deve conter k (se k estiver presente na árvore).
     a) Se $c_i[x]$ contém apenas $t-1$ chaves mas tem um irmão imediato com pelo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

Árvores B - Remoção - Func. Auxiliar (3a)

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 $c_i[x]$.

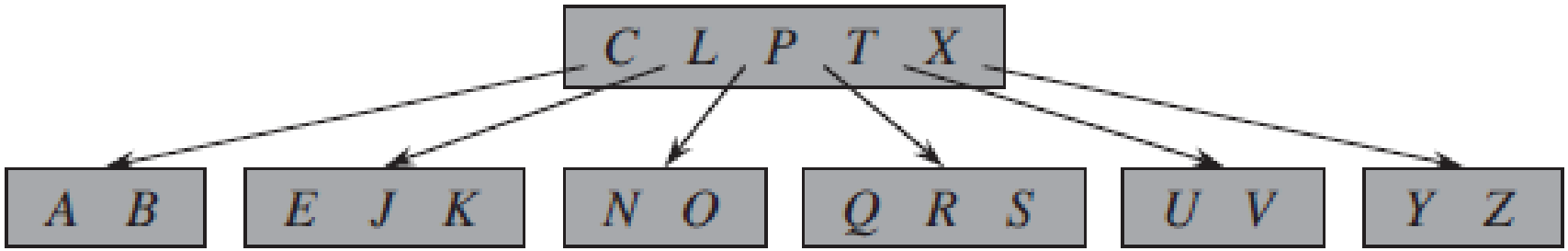
Árvores B - Remoção - Func. Auxiliar (3a)

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 $c_i[x]$.



Árvores B - Remoção - Func. Auxiliar (3a)

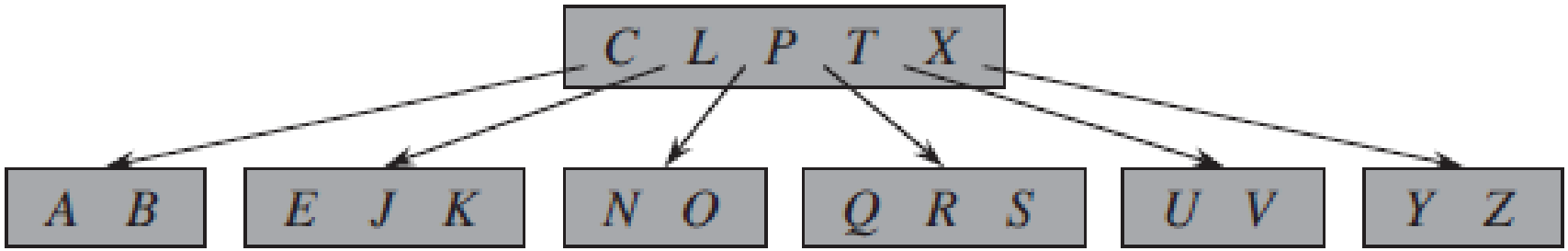
a) Se $c_i[x]$ contém apenas $t-1$ chaves mas tem um irmão imediato com pelo menos t chaves, dê para $c_i[x]$ uma chave extra movendo uma chave de x para $c_i[x]$, movendo uma chave do irmão imediato de $c_i[x]$ à esquerda ou à direita, e movendo o ponteiro do filho apropriado do irmão para o nó $c_i[x]$.



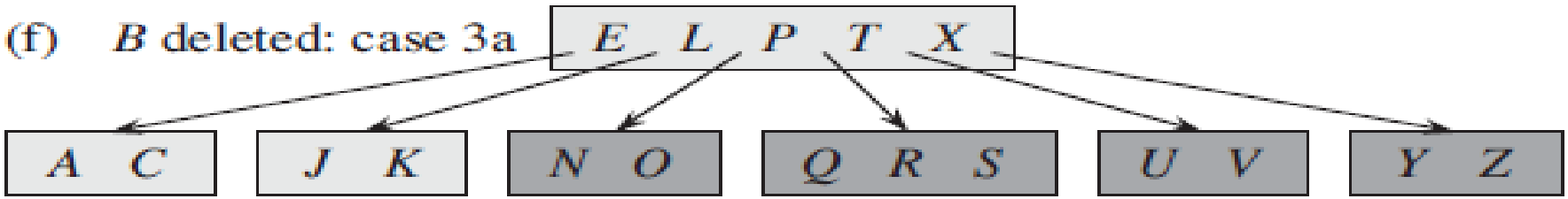
Queremos remover B

Árvores B - Remoção - Func. Auxiliar (3a)

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 $c_i[x]$.



Queremos remover B

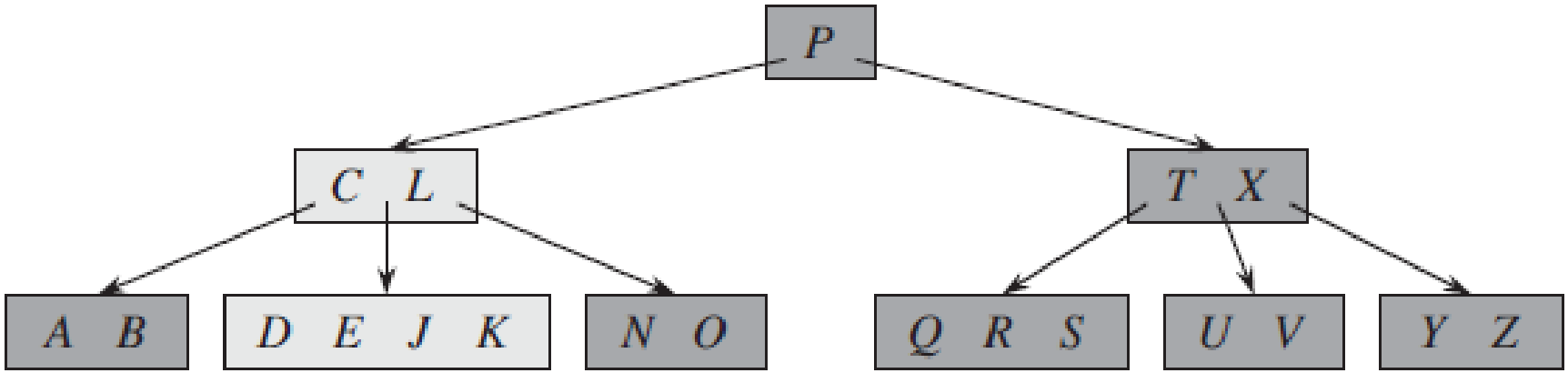


Árvores B - Remoção - Func. Auxiliar (3b)

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).

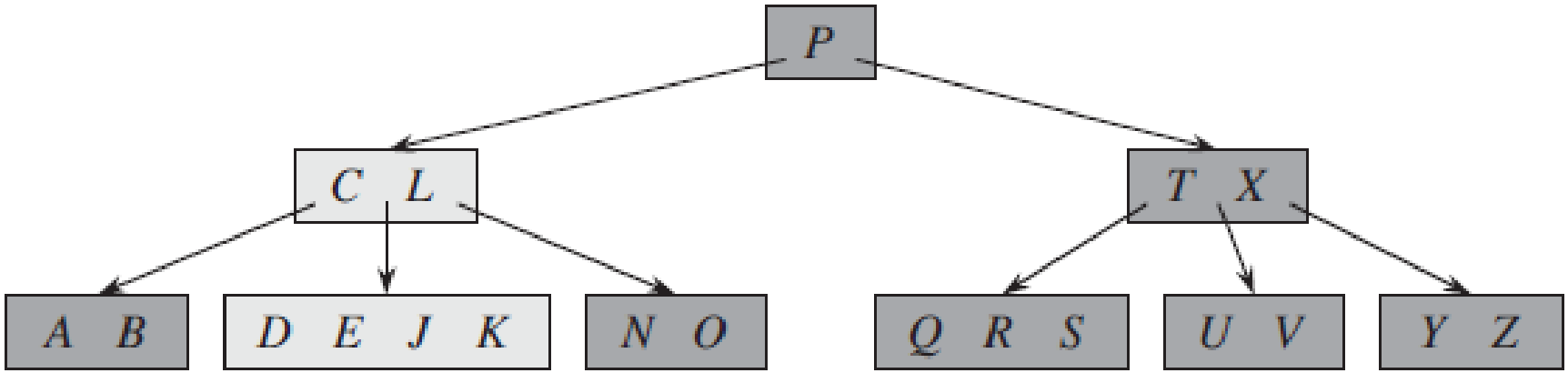
Árvores B - Remoção - Func. Auxiliar (3b)

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).



Árvores B - Remoção - Func. Auxiliar (3b)

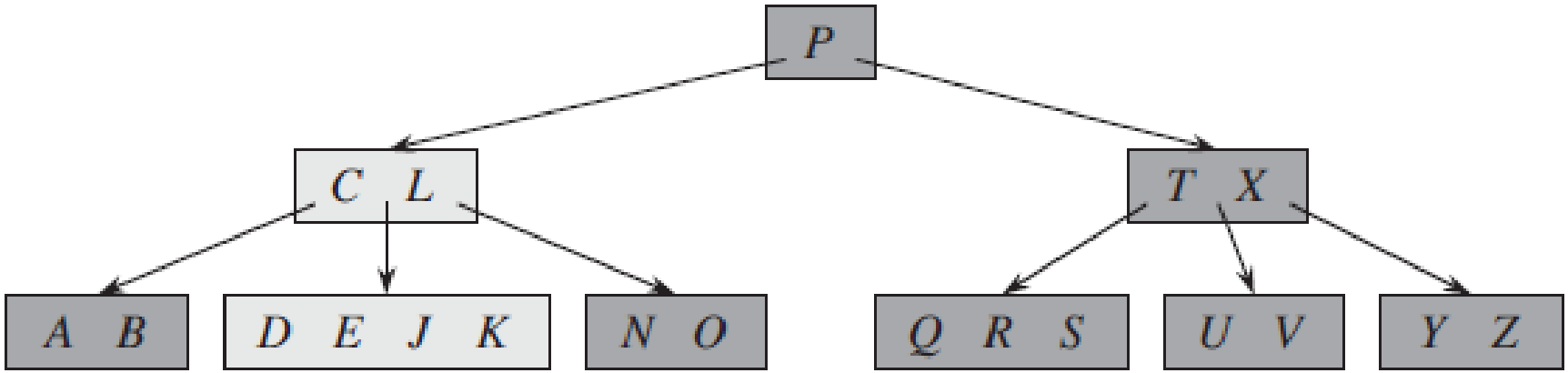
b) Se $c_i[x]$ e ambos os irmãos imediatos de $c_i[x]$ contêm $t - 1$ chaves, faça a junção de $c_i[x]$ com um de seus irmãos. Isso implicará em mover uma chave de x para o novo nó fundido (que se tornará a chave mediana para aquele nó).



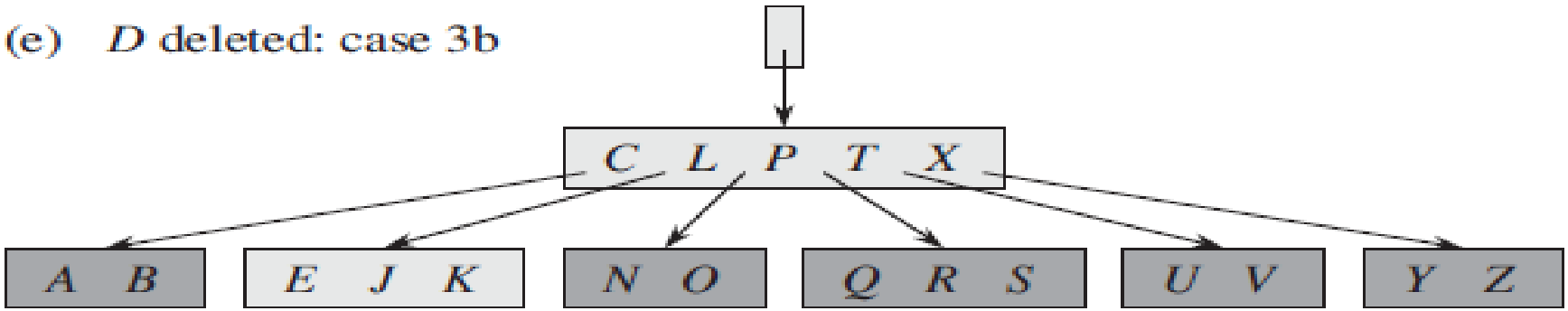
Queremos remover D

Árvores B - Remoção - Func. Auxiliar (3b)

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).



Queremos remover D



Árvores B - Remoção

Quando um dos nós for a raiz:

- A raiz 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 função inicial/principal antes da remoção recursiva

Árvores B - Remoção - Função *Principal*

```
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 (!raiz[r]){
    raiz[T] = c0[r];
    desaloca(r);
  }
}
```

Complexidade

	Sequencial		Ligada	Árvore B
Complex.	**Não Ordenado**	**Ordenado**	**Ordenado**	**Ordenado**
Busca	O(b)	O(log(b))	O(b)	$O(log_t(b))$
Inserção	O(1) se houver espaço, ou O(b)	O(b)	O(1)	$O(log_t(b))$
Remoção[ii]	O(b) (custo da busca), ou O(1)	$O(log(b))^2$	O(b) (custo da busca), ou O(1)	$O(log_t(b))$
Leitura Ordenada	$\omega(b)$[iii]	O(b), O(1) (leitura toda de uma vez)	O(b)	O(b)
Mínimo/ Máximo	O(b), ou O(1) se as chaves estiverem no cabeçalho	O(1)	O(1)	$O(log_t(b))$
Modificação	O(1), ou O(b) se precisar buscar	O(1), O(log(n)) (buscar), ou O(b) (modificação de chave)	O(1), ou O(b) se precisar buscar ou modificar a chave	$O(log_t(b))$

[ii] Considerando o uso de um bit de validade/exclusão

[iii] Depende do algoritmo de ordenação [externo]

Referência

- Slides baseados no material da profa. Ariane Machado Lima - ACH2024
- CORMEN, H. T.; LEISERSON, C.E.; RIVEST, R.L. Introduction to Algorithms, MIT Press, McGraw-Hill, 2009.
- DROZDEK, A. Estrutura de Dados e Algoritmos em C++, Cengage, 2017.

Algoritmos e Estruturas de Dados II
Aula 23 – Árvores B

Prof. Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2025