MC202 – Estruturas de Dados

Árvores B

Aula 24

Emilio Francesquini
francesquini@ic.unicamp.br

Disclaimer

- Esses slides foram preparados pelo Prof. Orlando Lee para o curso de Estrutura de Dados ministrado na UNICAMP.
- Este material pode ser usado livremente desde que sejam mantido os créditos dos autores e da instituição.
- Muitos dos exemplos apresentados aqui foram retirados de uma apostila ou slides preparados pelo Prof. Tomasz Kowaltowski da UNICAMP. Outros exemplos foram retirados do livro "Algoritmos em Linguagem C" de Paulo Feofiloff, Editora Campus.

Referências


T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein (2001), "Introduction to Algorithms", 2nd edition, The MIT Press, pp. 434–454.

Agradecimentos ao Prof. Fábio Usberti por ter emprestado seus slides sobre árvores B.

Memória principal × memória secundária

- Vimos várias estruturas de dados adequadas para gerenciar informações armazenadas na memória principal (primária), onde o tempo de acesso é extremamente rápido.
- Por exemplo, árvores AVL tem desempenho $O(\log n)$ (proporcional à altura da árvore) para realizar busca, inserção e remoção, onde n é o número de nós.

Memória principal × memória secundária

- Dispositivos de memória secundária (fita ou disco magnético) permitem armazenar um volume imenso de dados, **muito maior** que a disponível na memória principal.
  - Memória principal: ≈ 8 Gb
  - Memória secundária: ≈ 1 Tb $= 1024$ Gb

Memória principal × memória secundária

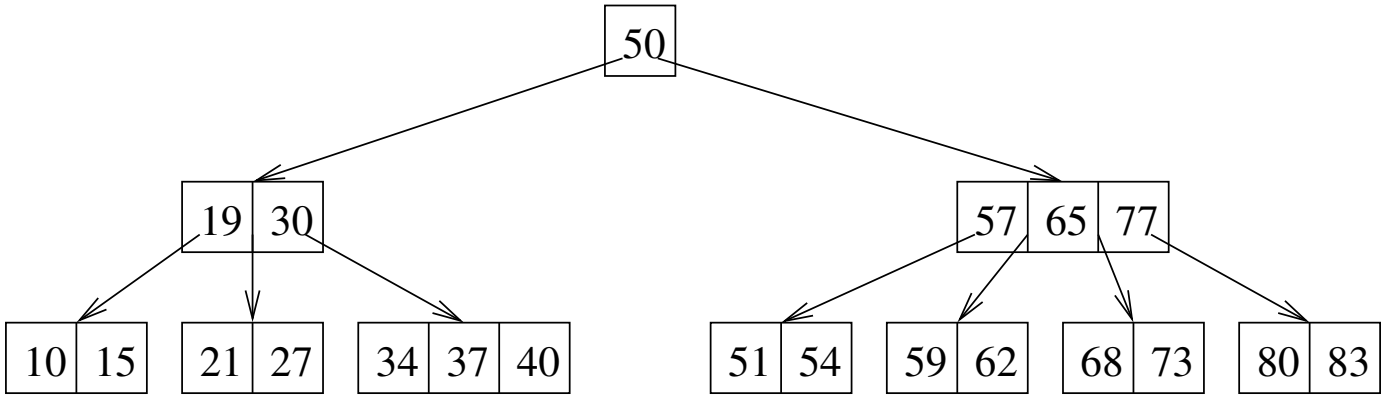
- Dispositivos de memória secundária (fita ou disco magnético) permitem armazenar um volume imenso de dados, **muito maior** que a disponível na memória principal.
  - Memória principal: ≈ 8 Gb
  - Memória secundária: ≈ 1 Tb $= 1024$ Gb
- Por outro lado, o tempo de acesso em memória secundária é **muito maior** do que na memória principal.
  - Memória principal: ≈ 100 ns
  - Memória secundária: ≈ 1 ms $= 10^6$ ns

Árvores AVL tem um desempenho muito pobre quando implementados na memória secundária.

Árvores B

- Em memória secundária, os tempos de execução dos algoritmos são regidos pelo número de leituras e escritas (acesso a disco) em disco.
- Árvore B é uma estrutura de dados que
  - reduz o número de acessos a disco e
  - reduz o tempo gasto em operações (busca, inserção e remoção).

Exemplo de árvore B



- Cada nó contém uma sequência ordenada de chaves e apontadores.
- As chaves são usadas para separar as chaves dos nós filhos.

Árvore B

- Um disco (memória secundária) é dividido em páginas de mesmo tamanho. Tipicamente, o tamanho de uma página é de 2^{11} a 2^{14} bytes.
- Quando a cabeça de leitura do disco está na posição correta, é possível ler rapidamente grandes quantidades de dados do disco.
- Em aplicações que usam árvores B, a quantidade total de dados **não cabe** na memória principal.
- Idealmente, cada nó da árvore B tem o tamanho em bytes de uma página do disco.

Leitura e escrita em disco

- Suponha que x é um apontador para um objeto (nó).
- Um nó x pode estar na memória principal ou na memória secundária.
- LEITURA(x) - lê o nó x do disco copiando-o para a memória principal.
- ESCRITA(x) - escreve/grava o nó x na memória secundária.

Leitura e escrita em disco

Uma padrão típico para acessar ou modificar um objeto x é o seguinte:

- LEITURA(x)
- acesse/modifique o nó x
- ESCRITA(x)

Leitura e escrita em disco

Uma padrão típico para acessar ou modificar um objeto x é o seguinte:

- LEITURA(x)
- acesse/modifique o nó x
- ESCRITA(x)

Convencionamos que

- se x está na memória principal então LEITURA(x) não faz nada, e
- se x não foi modificado então ESCRITA(x) não faz nada.

Representação de árvore B

```
typedef struct No {
    int n;
    int folha;
    int chave[MAX];
    struct No *filho[MAX+1];
} NoArvB, *ArvoreB;
```

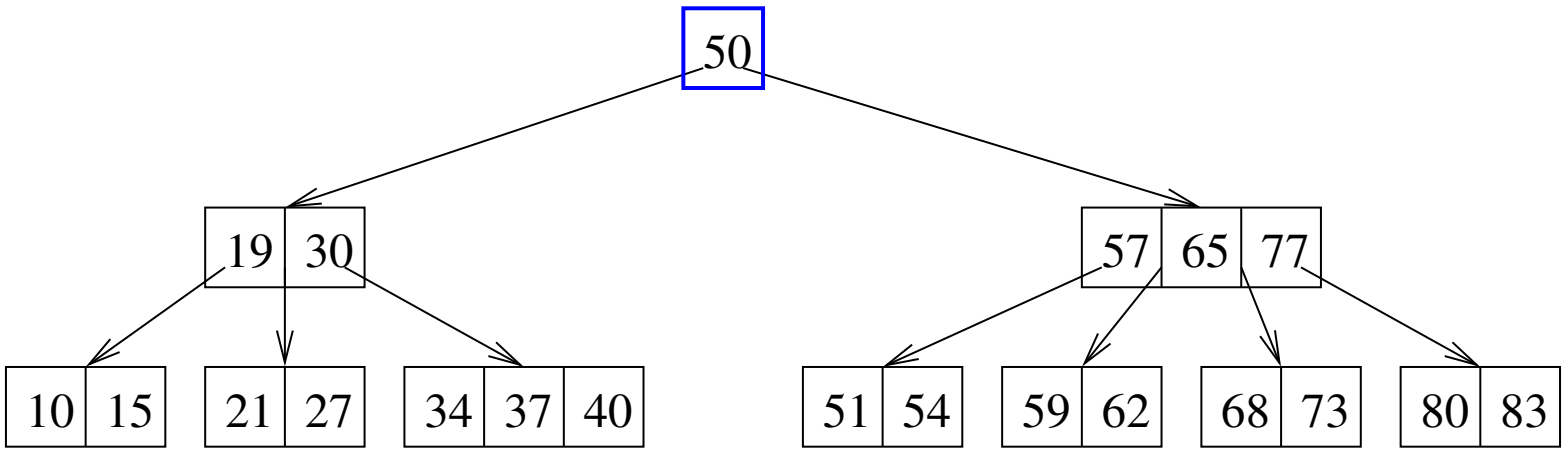
Para facilitar a descrição, vamos supor que os vetores são indexados a partir de 1. Assim, um nó x tem $x->n$ chaves, as chaves ficam em $x->chave[1..n]$ e os apontadores para os filhos em $x->filho[1..n+1]$. O campo $x->folha$ diz se x é uma folha.

Usamos $raiz[T]$ para denotar a raiz de uma árvore T.

Propriedades

1. Um nó x contém um conjunto de n $(= x{->}n)$ chaves ordenadas de modo não-decrescente.

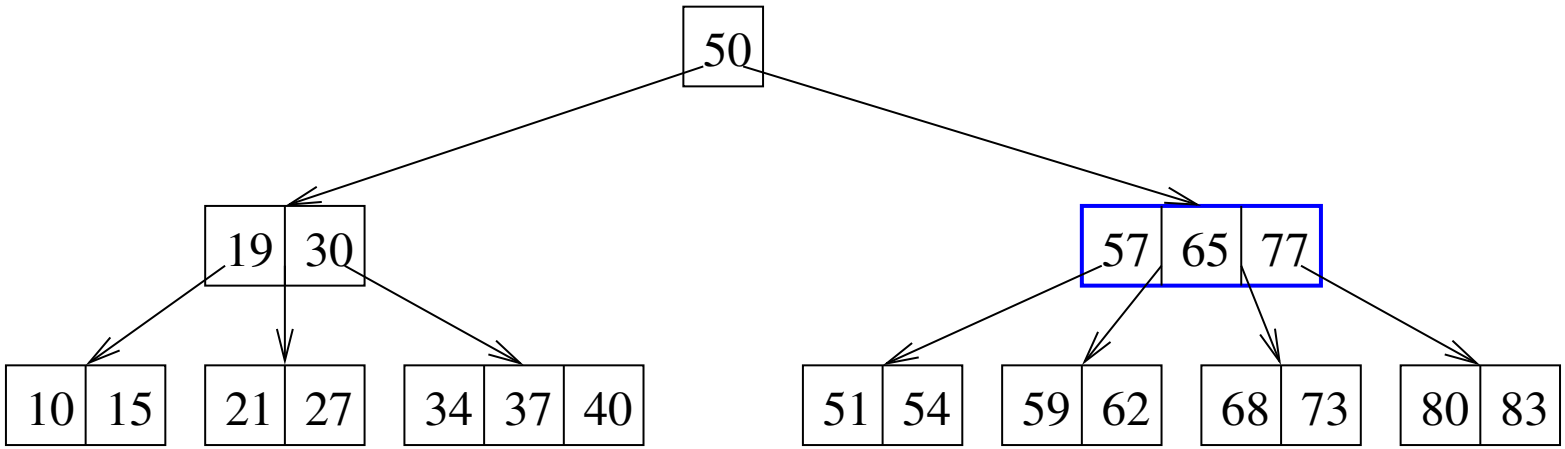
$$x{->}chave[1] \leqslant x{->}chave[2] \leqslant \cdots \leqslant x{->}chave[n]$$



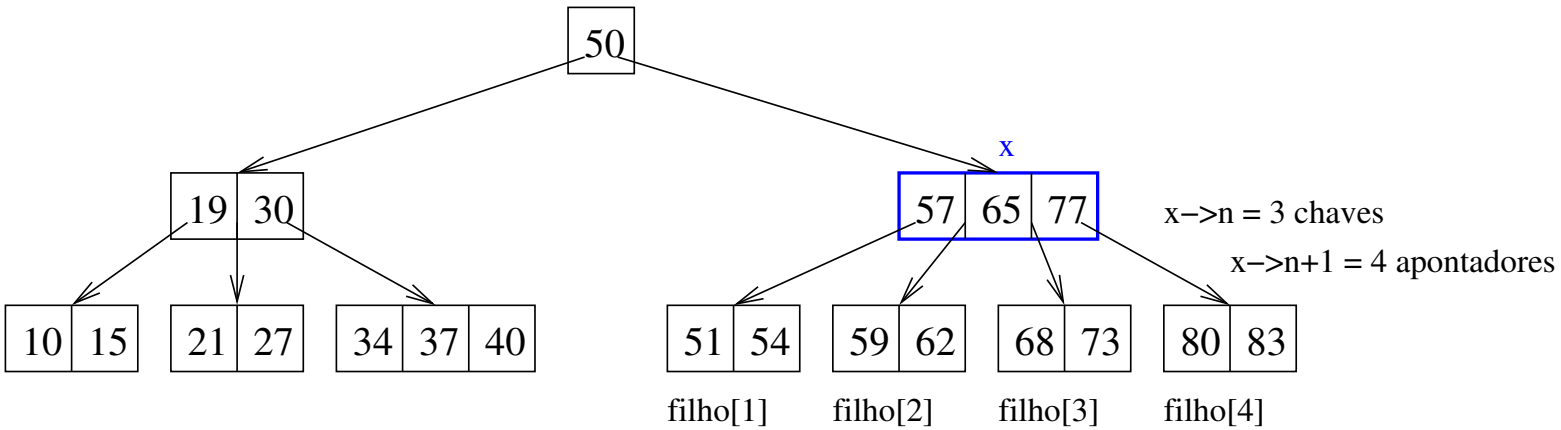
Propriedades

1. Um nó x contém um conjunto de n $(= x{->}n)$ chaves ordenadas de modo não-decrescente.

$$x{->}chave[1] \leqslant x{->}chave[2] \leqslant \cdots \leqslant x{->}chave[n]$$

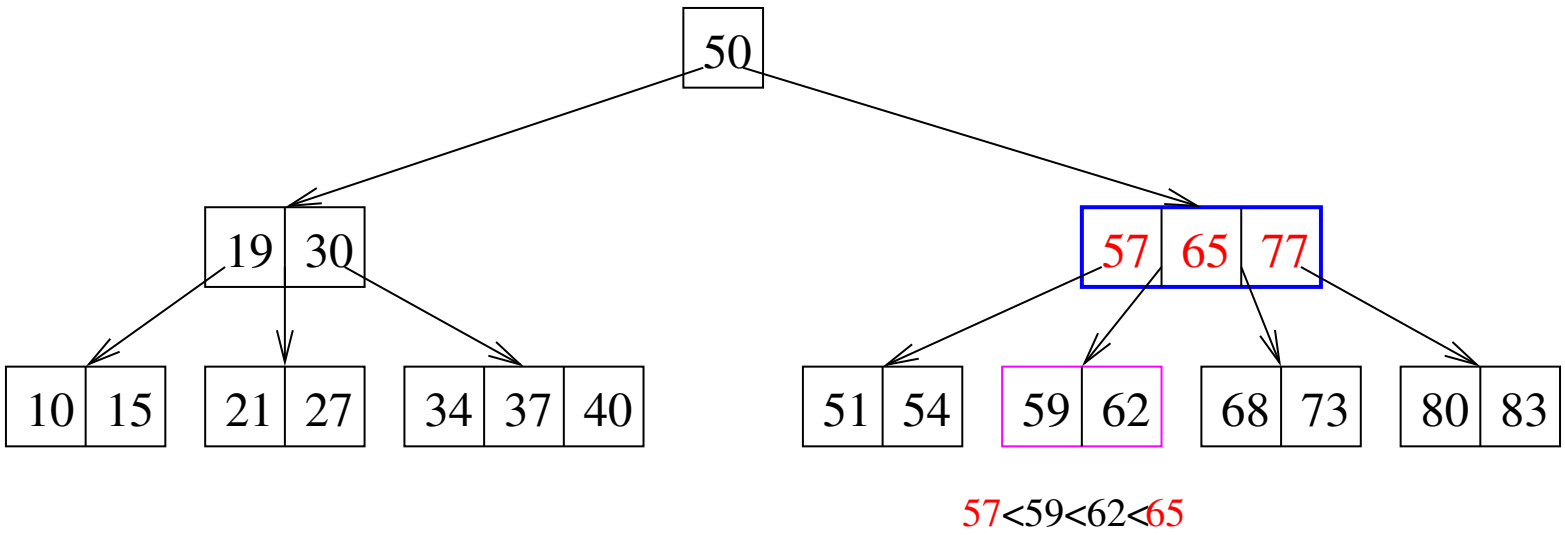


2. Um nó interno x ($x{->}folha == 0$) contém $n+1$ apontadores $x{->}filho[1], x{->}filho[2], \ldots, x{->}filho[n+1]$ para seus filhos.

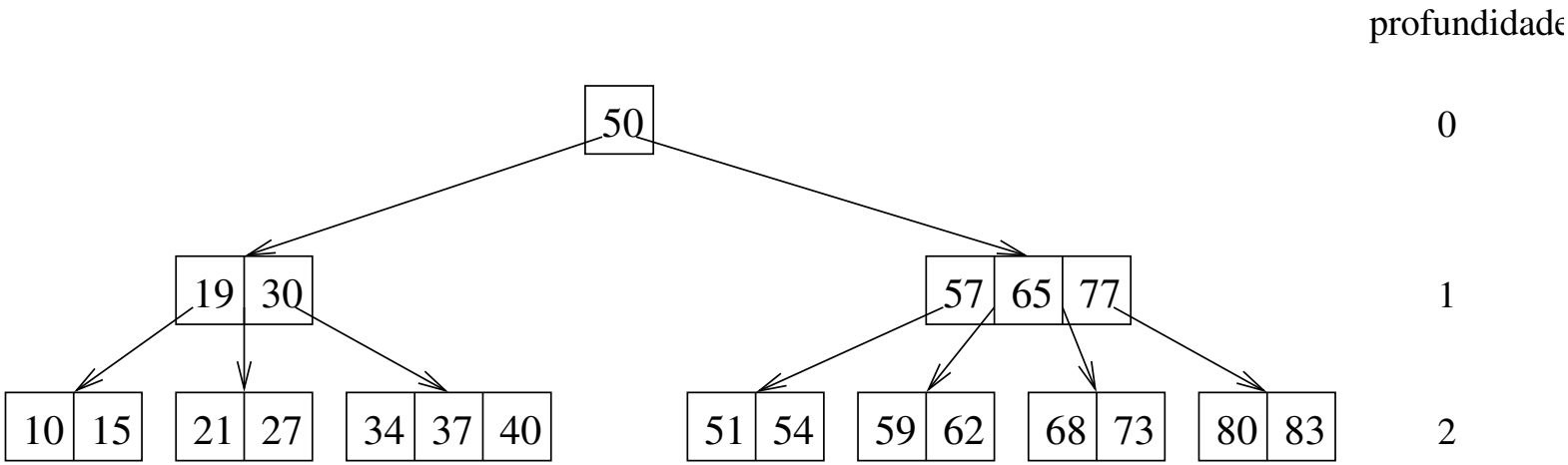


3. Uma chave $x->chave[i]$ delimita os intervalos de valores das chaves armazenadas nas subárvores vizinhas. Se c_i é uma chave qualquer armazenada na subárvore enraizada em $x->filho[i]$ então:

$$c_1 \leqslant x->chave[1] \leqslant c_2 \leqslant \cdots \leqslant c_n \leqslant x->chave[n] \leqslant c_{n+1}$$



4. Todas as folhas se encontram na mesma profundidade correspondente à altura da árvore h. Supõe-se que a raiz da árvore se encontra na profundidade 0.

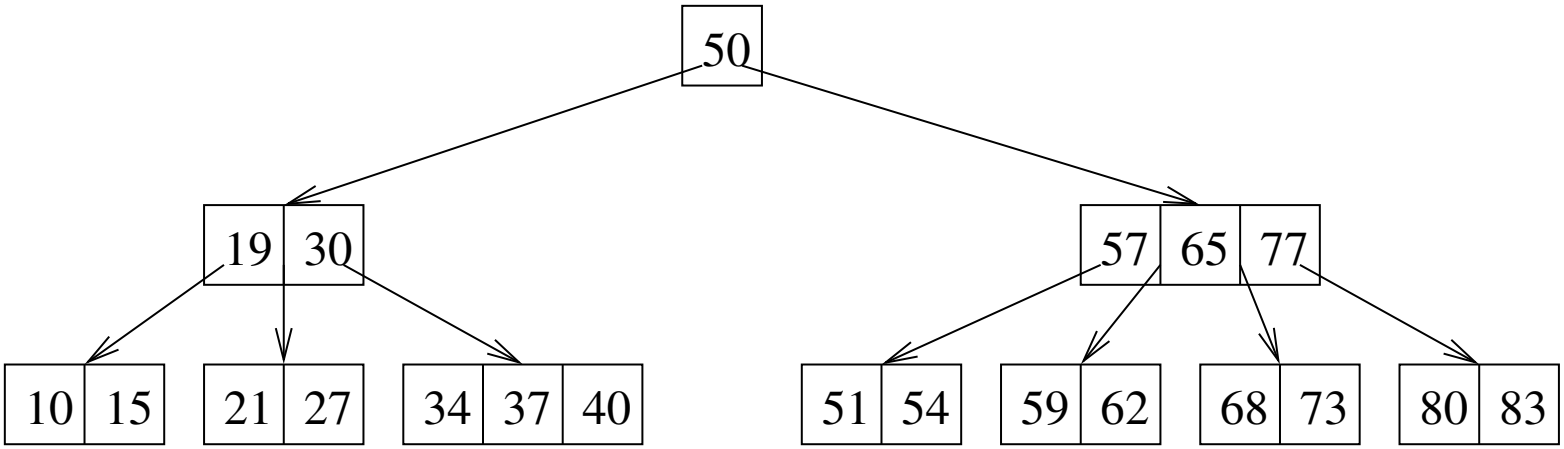


Propriedades

5. Há um valor inteiro $t \geqslant 2$ que fornece limitantes para o número de chaves em um nó.

- Todos os nós, exceto $raiz[T]$, possuem pelo menos $t - 1$ chaves.
- $raiz[T]$ tem pelo menos uma chave.
- Todos os nós possuem no máximo $2t - 1$ chaves. Isso implica que o número máximo de filhos por nó é $2t$.
- Uma árvore é de **ordem b** se $b = 2t - 1$. Em outros textos, a ordem é definida de modo diferente — não há consenso na literatura.

Propriedades



t = 2

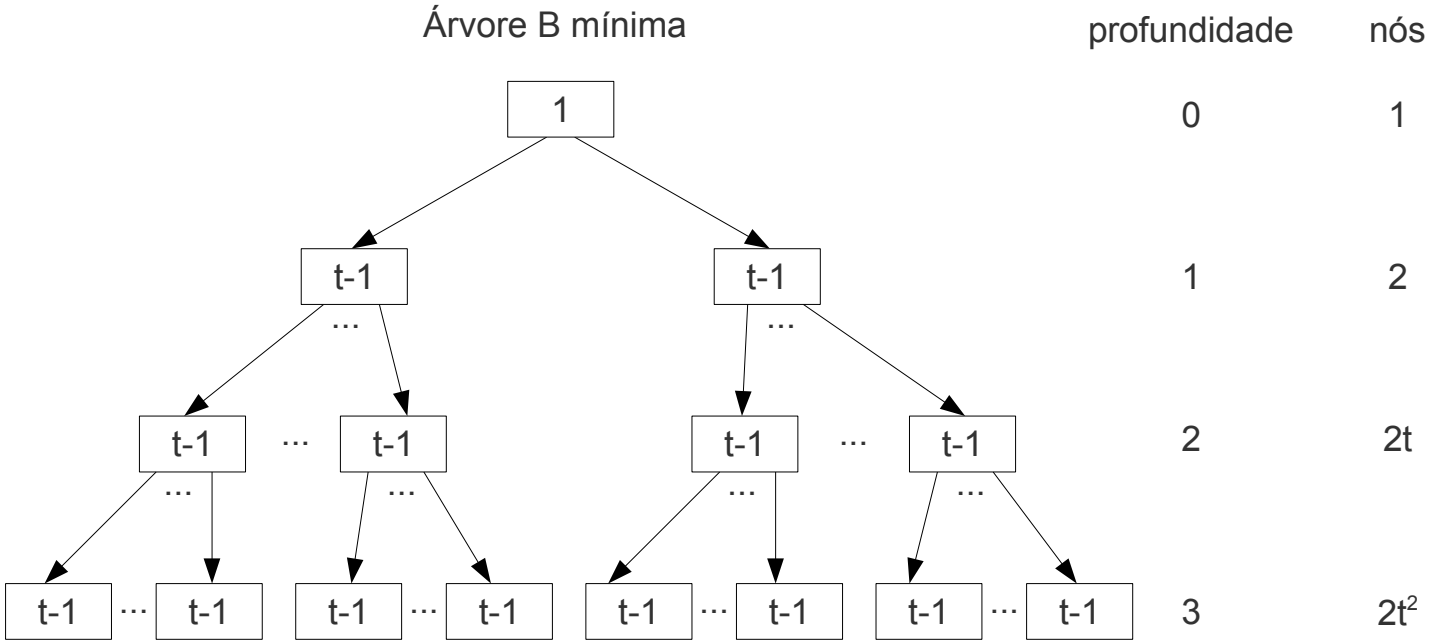
>= t−1 chaves (exceto raiz)

<= 2t−1 chaves

ordem 3

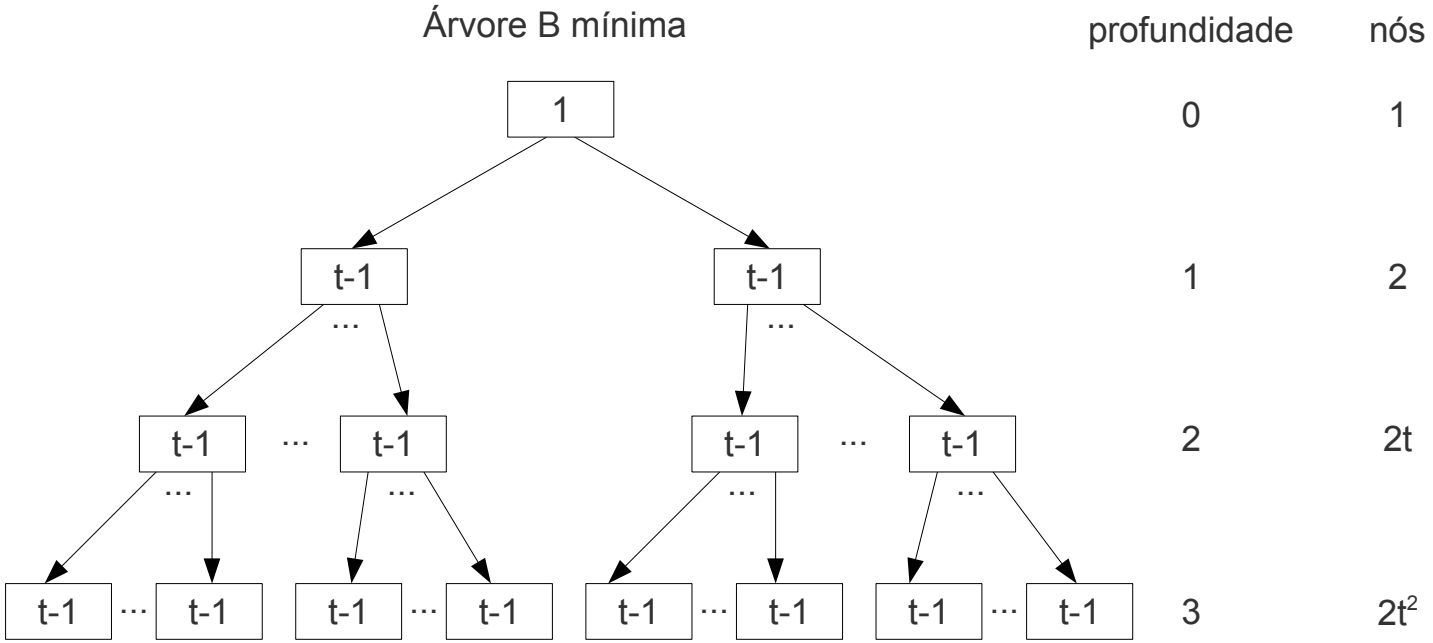
Altura de uma árvore B

Dada uma árvore de ordem $b = 2t - 1$ com n chaves, qual a altura máxima h dessa árvore?



Altura de uma árvore B

Dada uma árvore de ordem $b = 2t - 1$ com n chaves, qual a altura máxima h dessa árvore?



Resposta: $h \leq \log_t \frac{n+1}{2}$.

Comparação com outras árvores balanceadas

- Suponha uma base de dados com 1 milhão de registros ($n = 10^6$). Para esse exemplo temos as seguintes alturas máximas de árvore:
- AVL: $h_{AVL} = \lceil 1.44 \log(n+2) - 0.328 \rceil = 29$
- rubro-negra: $h_{RN} = \lceil 2 \log(n+1) \rceil = 40$
- B ($t = 100$): $h_B = \lceil \log_t(\frac{n+1}{2}) \rceil = 3$

Como veremos, as operações de busca, inserção e remoção em uma árvore B consomem tempo $O(h_B)$ (proporcional à altura da árvore B).

Descreveremos a seguir as operações

- BuscaArvoreB
- InsereArvoreB
- RemoveArvoreB

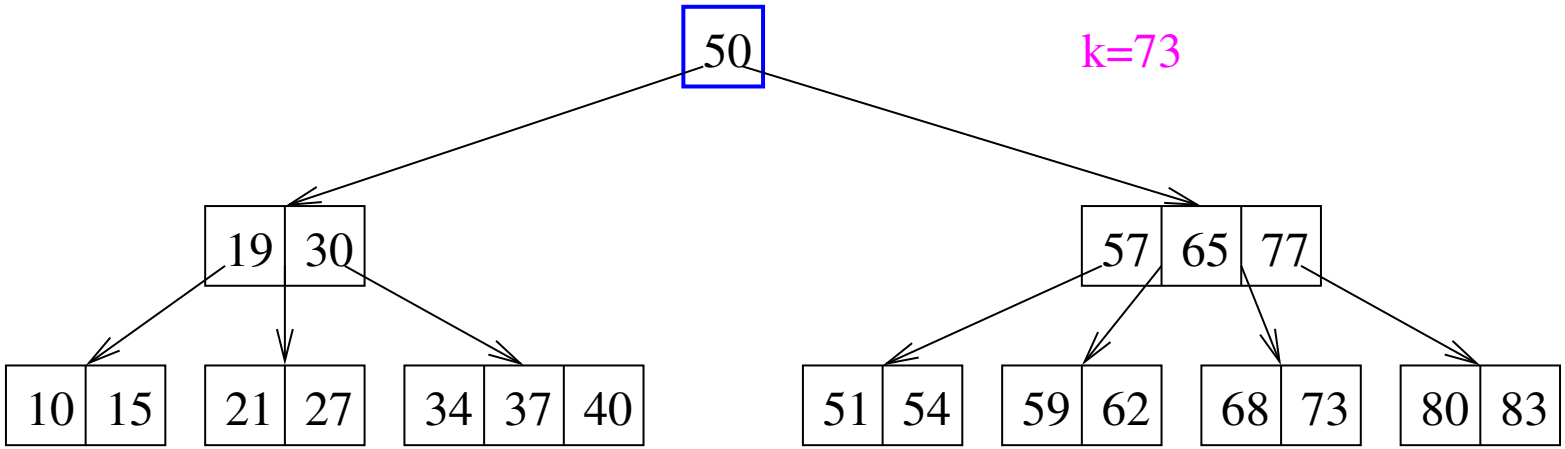
Descreveremos a seguir as operações

- BuscaArvoreB
- InsereArvoreB
- RemoveArvoreB

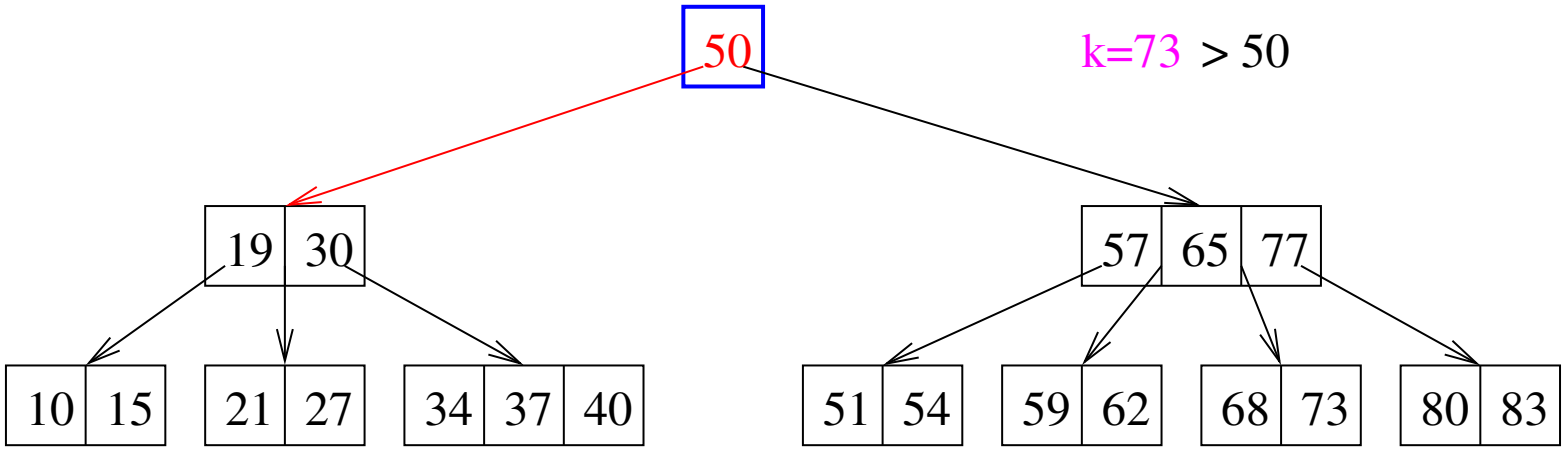
No que segue supomos que

- A raiz da árvore B está na memória principal,
- todo nó passado como parâmetro está na memória principal, ou seja, é preciso executar Leitura previamente.

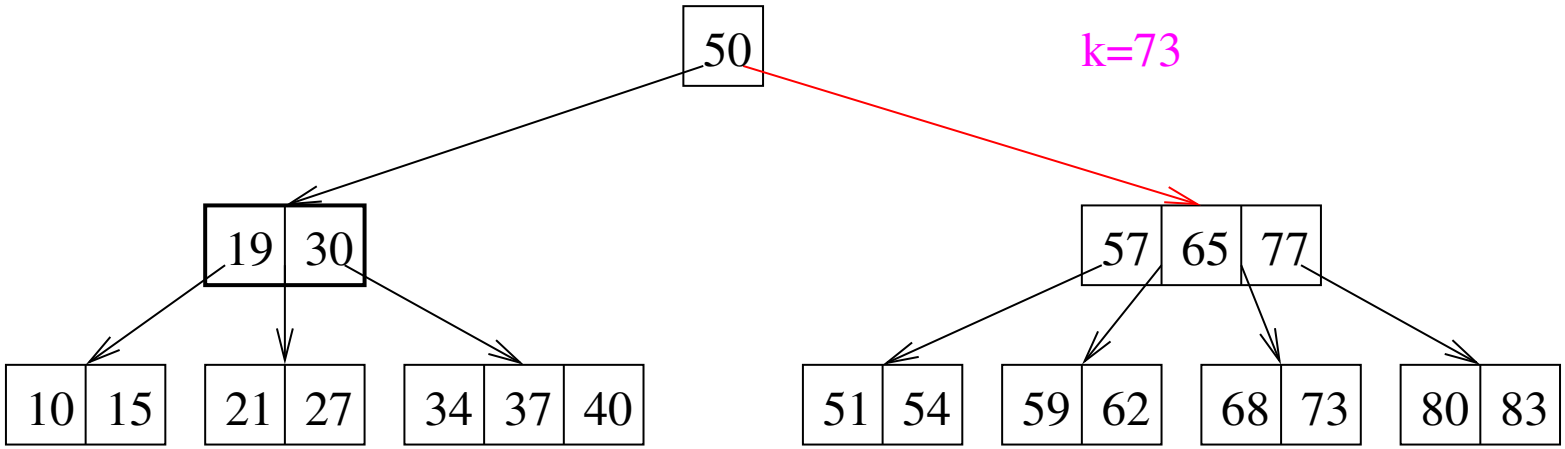
Exemplo de busca em árvore B



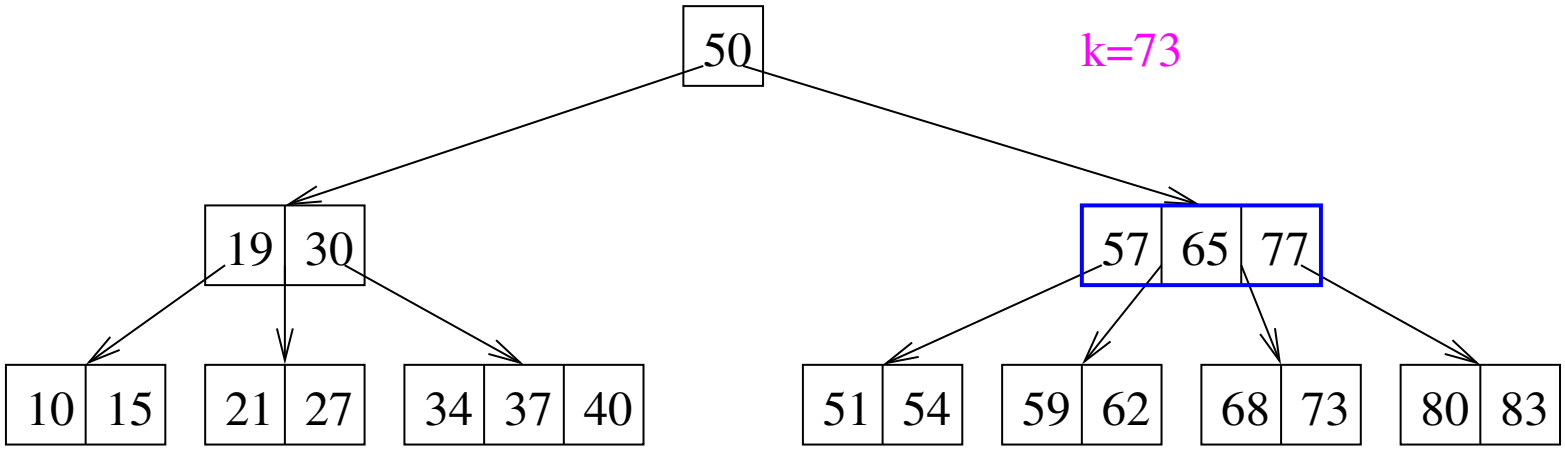
Exemplo de busca em árvore B



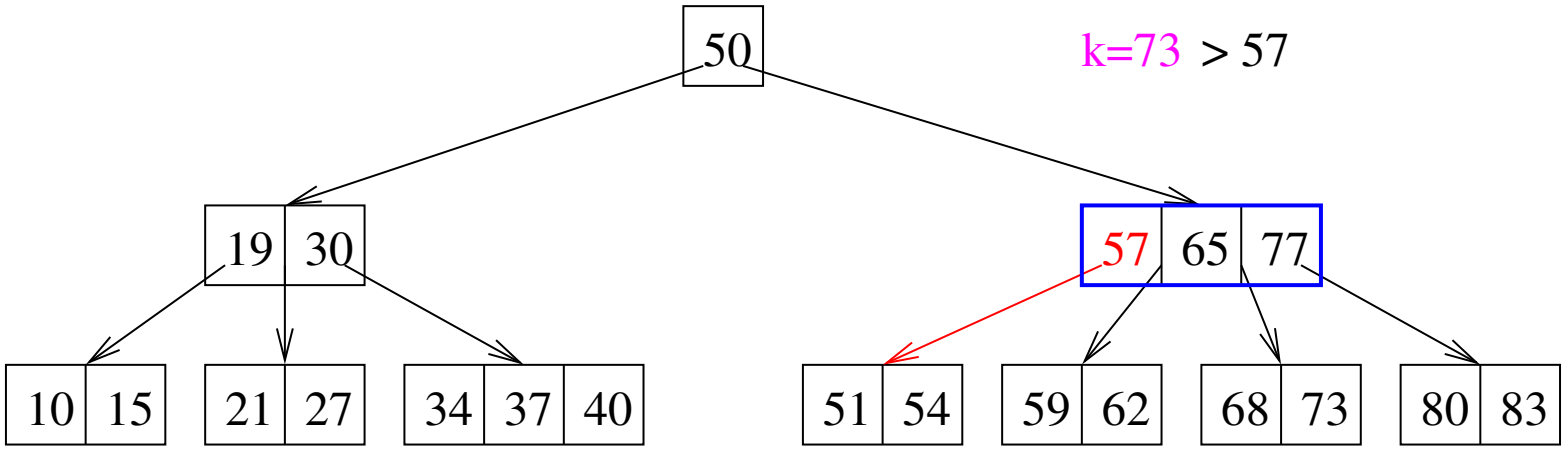
Exemplo de busca em árvore B



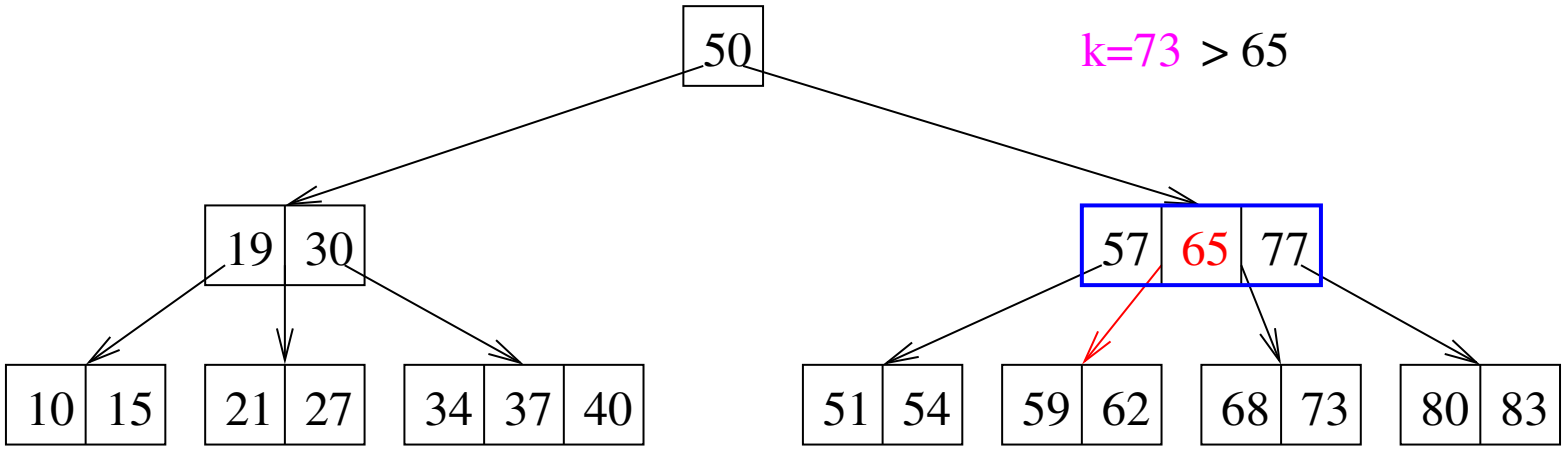
Exemplo de busca em árvore B



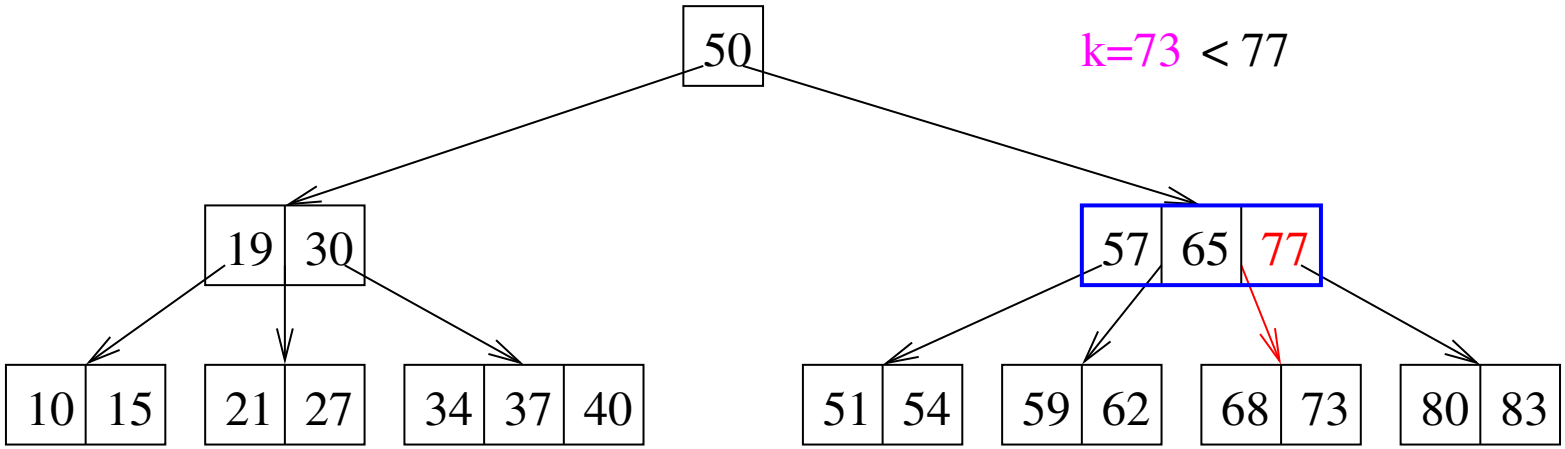
Exemplo de busca em árvore B



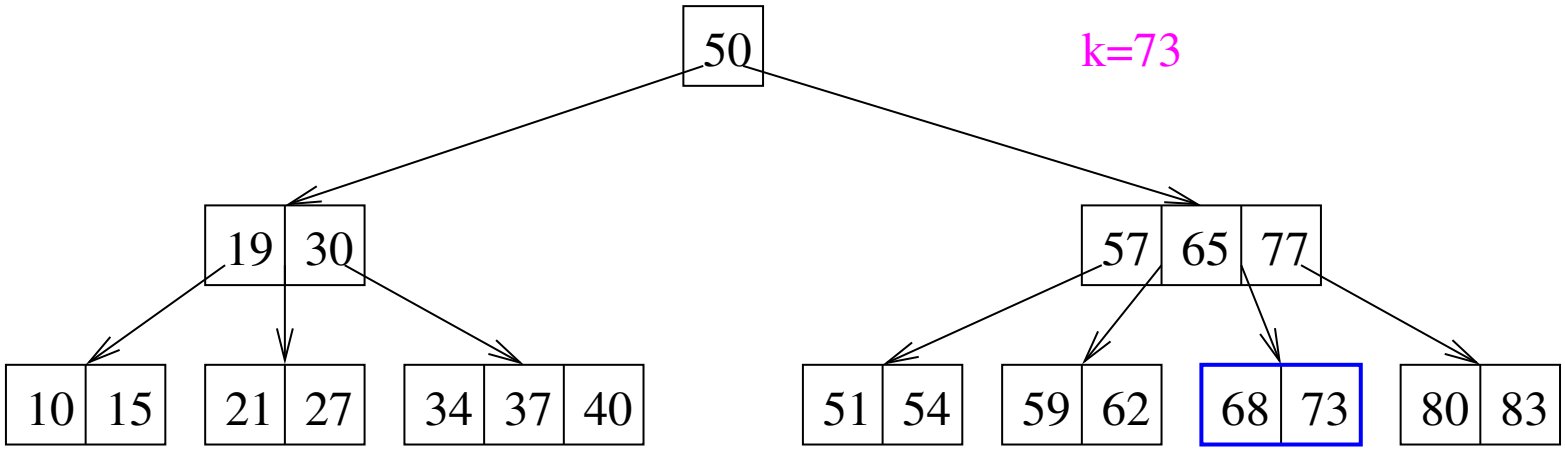
Exemplo de busca em árvore B



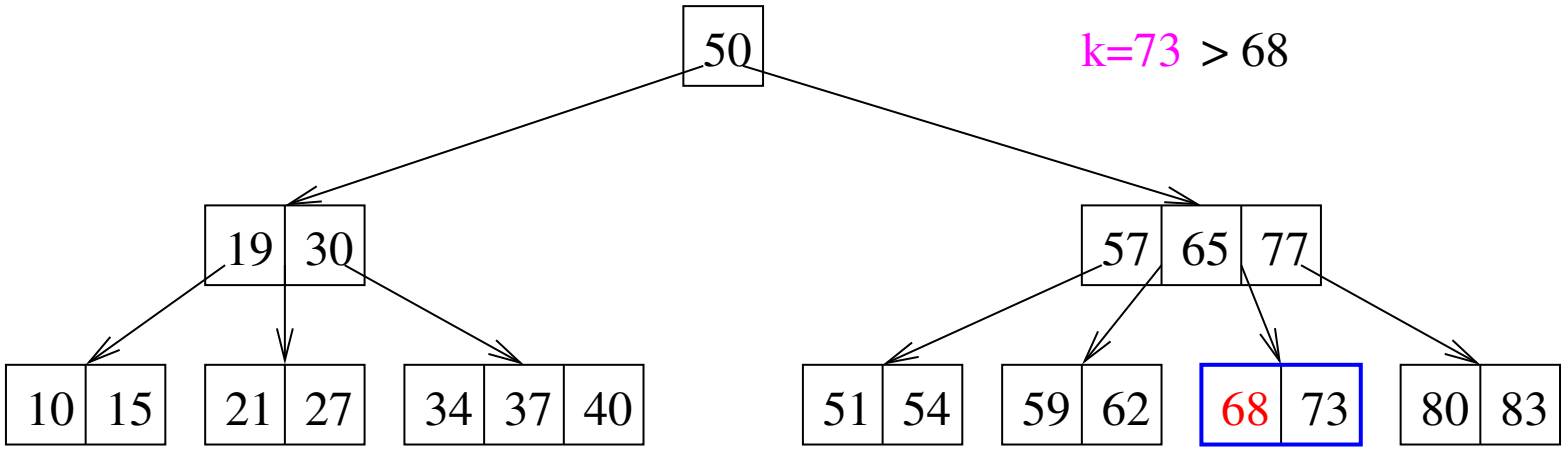
Exemplo de busca em árvore B



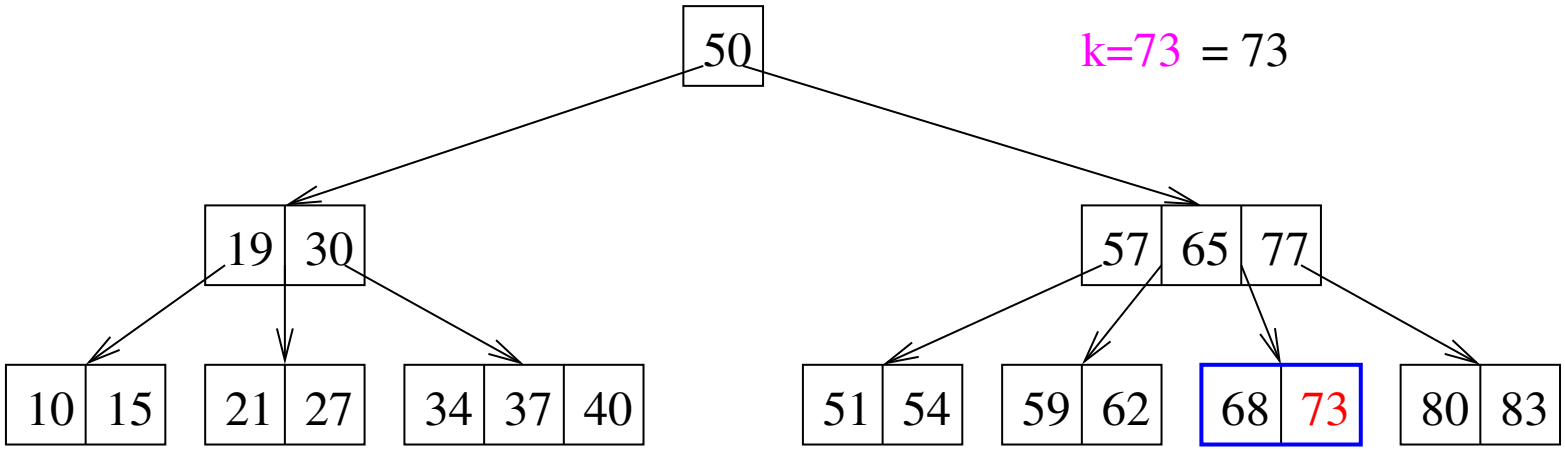
Exemplo de busca em árvore B



Exemplo de busca em árvore B



Exemplo de busca em árvore B



Busca em Árvore B – pseudocódigo

Entrada: a raiz r de uma árvore B e uma chave k
Saída: o nó (e sua posição nele) em que x está na árvore, ou NULL, se tal nó não existir.

```
BuscaArvoreB(r, k)
 1: i ← 1
 2: while i ⩽ r->n and k > r->chave[i] do
 3:     i ← i + 1
 4: if i ⩽ r->n and k = r->chave[i] then
 5:     return (x, i)
 6: if r->folha then
 7:     return NULL
 8: else
 9:     Leitura(r->filho[i])
10:     return BuscaArvoreB(r->filho[i], k)
```

Exercício. Escreva uma versão de BuscaArvoreB que usa busca binária no vetor de chaves em vez de busca sequencial.

Inserção em árvore B

- Suponha que queiramos inserir uma chave k em uma árvore B.
- Fazemos uma busca na árvore até encontrar um nó (folha) x com n chaves na qual queremos inserir k.

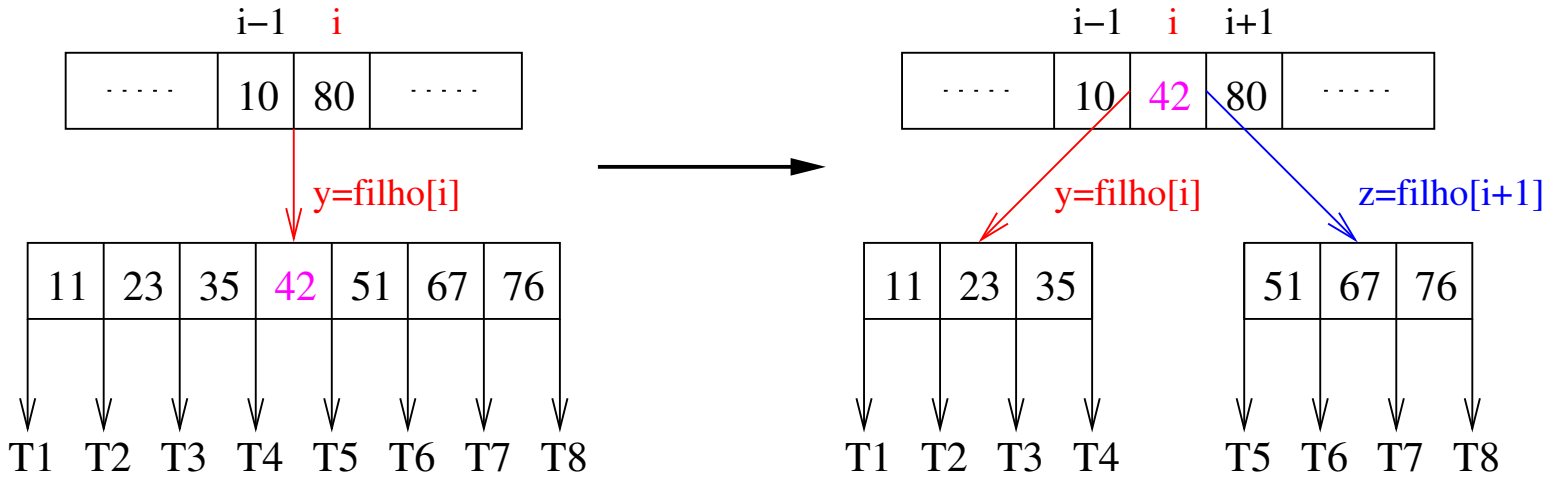
Inserção em árvore B

- Suponha que queiramos inserir uma chave k em uma árvore B.
- Fazemos uma busca na árvore até encontrar um nó (folha) x com n chaves na qual queremos inserir k.
- Se houver espaço em x ($n < 2t - 1$) então insera k na posição adequada do nó e o problema está resolvido.

Inserção em árvore B

- Suponha que queiramos inserir uma chave k em uma árvore B.
- Fazemos uma busca na árvore até encontrar um nó (folha) x com n chaves na qual queremos inserir k.
- Se houver espaço em x ($n < 2t - 1$) então insera k na posição adequada do nó e o problema está resolvido.
- O problema ocorre se x estiver cheio ($n = 2t - 1$). Neste caso é necessário fazer uma operação de split (divisão) do nó x.

Split ou divisão

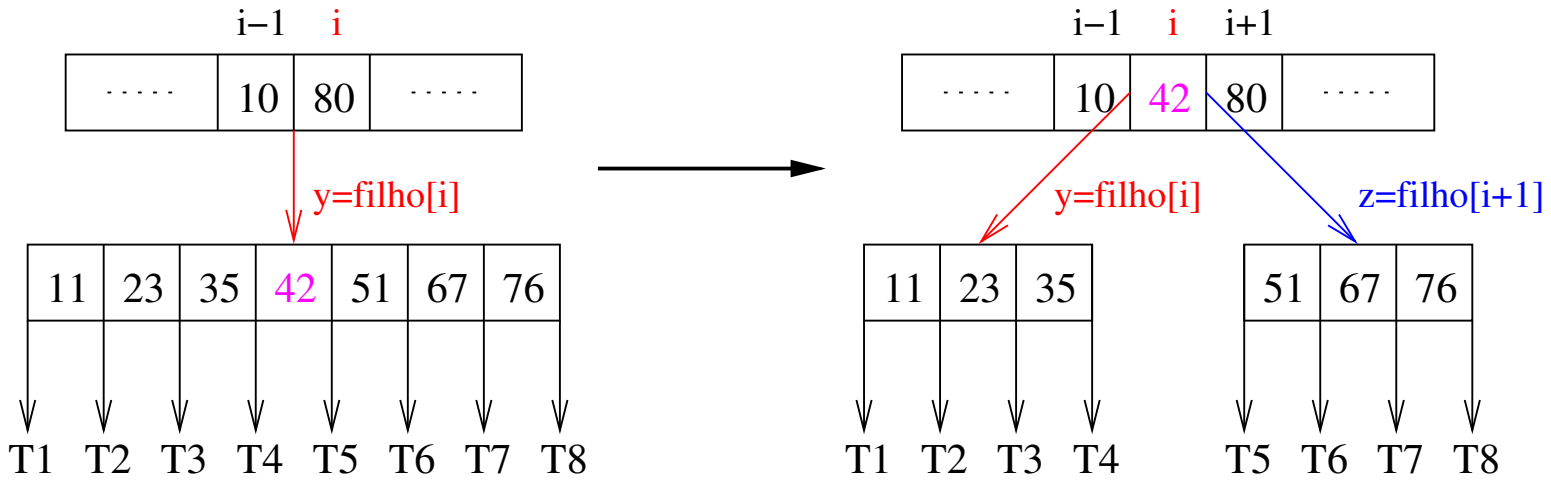


Ideia: seja x um nó cheio. Crie dois novos nós:

- um com as chaves $chave[1], \ldots, chave[t-1]$ e
- outro com as chaves $chave[t+1], \ldots, chave[t]$.

A mediana $chave[t]$ sobe para o nó pai.

Split ou divisão

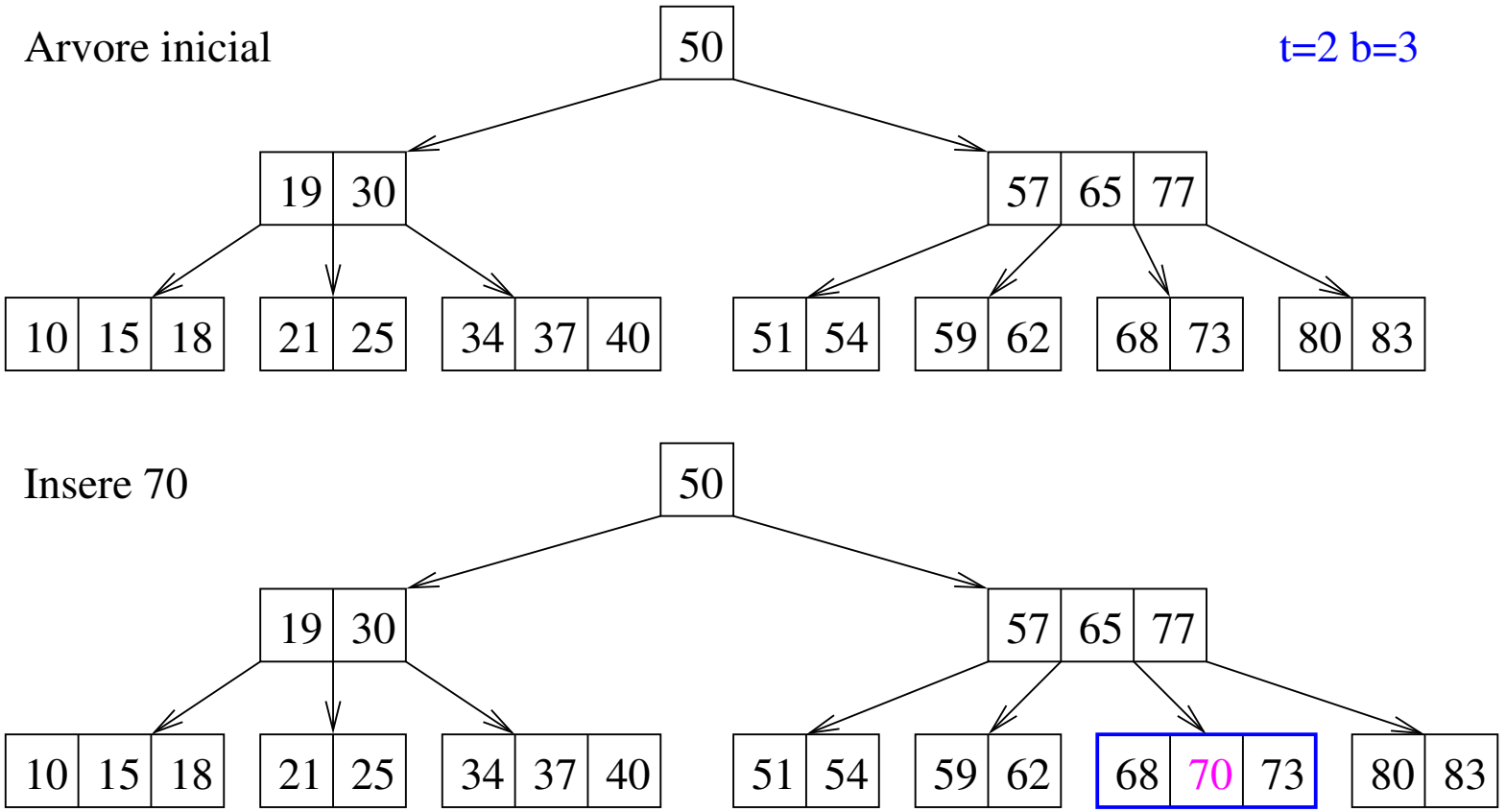


Note que se o nó pai estiver **cheio** será necessário um novo split. Pode ser necessário fazer uma sequência de splits até encontrar um nó não-cheio ou criar uma nova raiz.

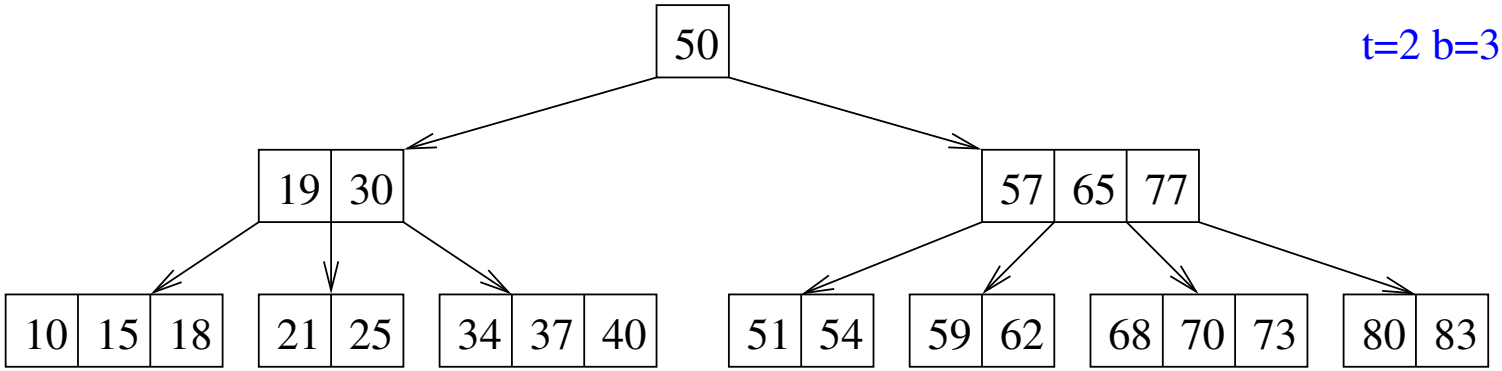
Método tradicional de inserção

- encontre a folha x onde deve ser inserido a nova chave k;
- se x não está cheio ($n < 2t - 1$), então insira k no vetor ordenado $chave[1 \,..\, n]$;
- se x estiver cheio ($n = 2t - 1$), então insira k no vetor ordenado $chave[1 \,..\, 2t - 1]$ (isto causa um overflow) e faça o split de x.
- propague os splits até encontrar um nó não-cheio ou criar uma nova raiz.

Método tradicional de inserção



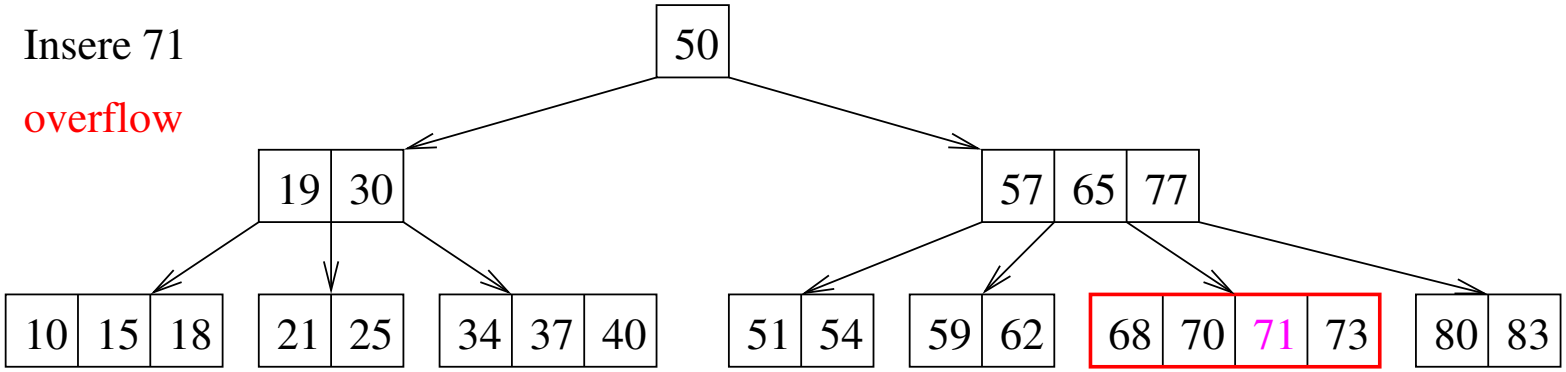
Método tradicional de inserção



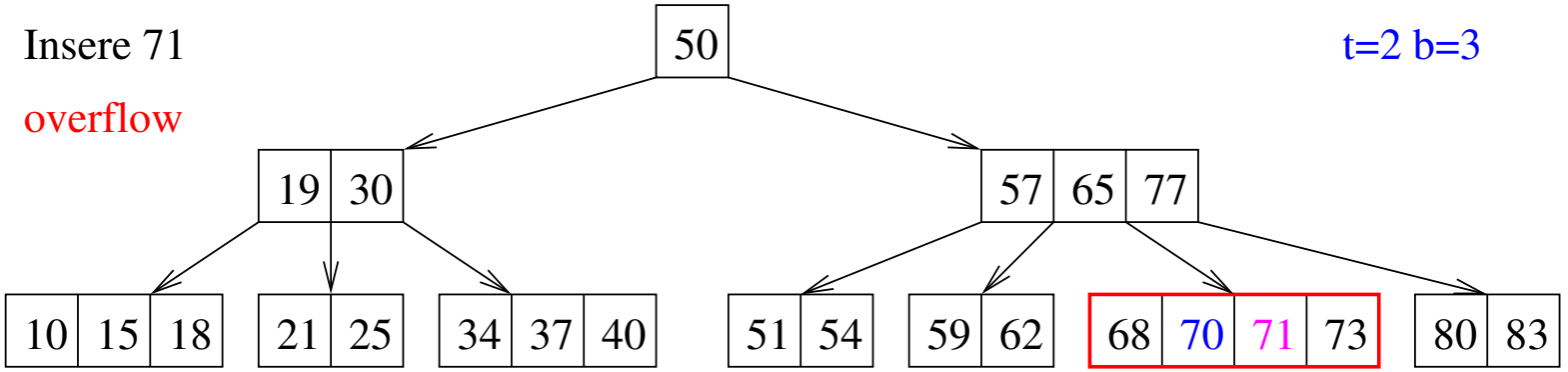
t=2 b=3

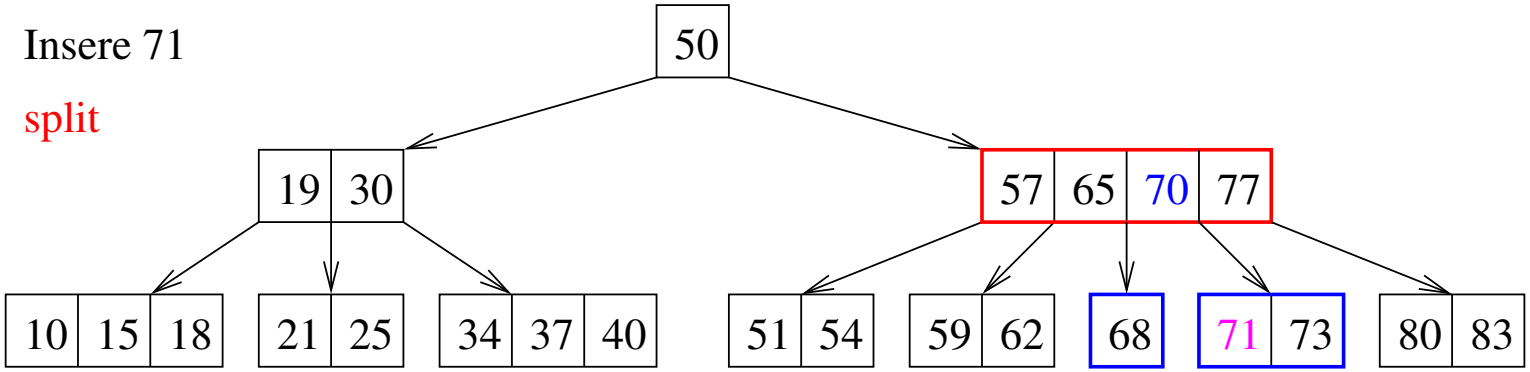
Insere 71

overflow

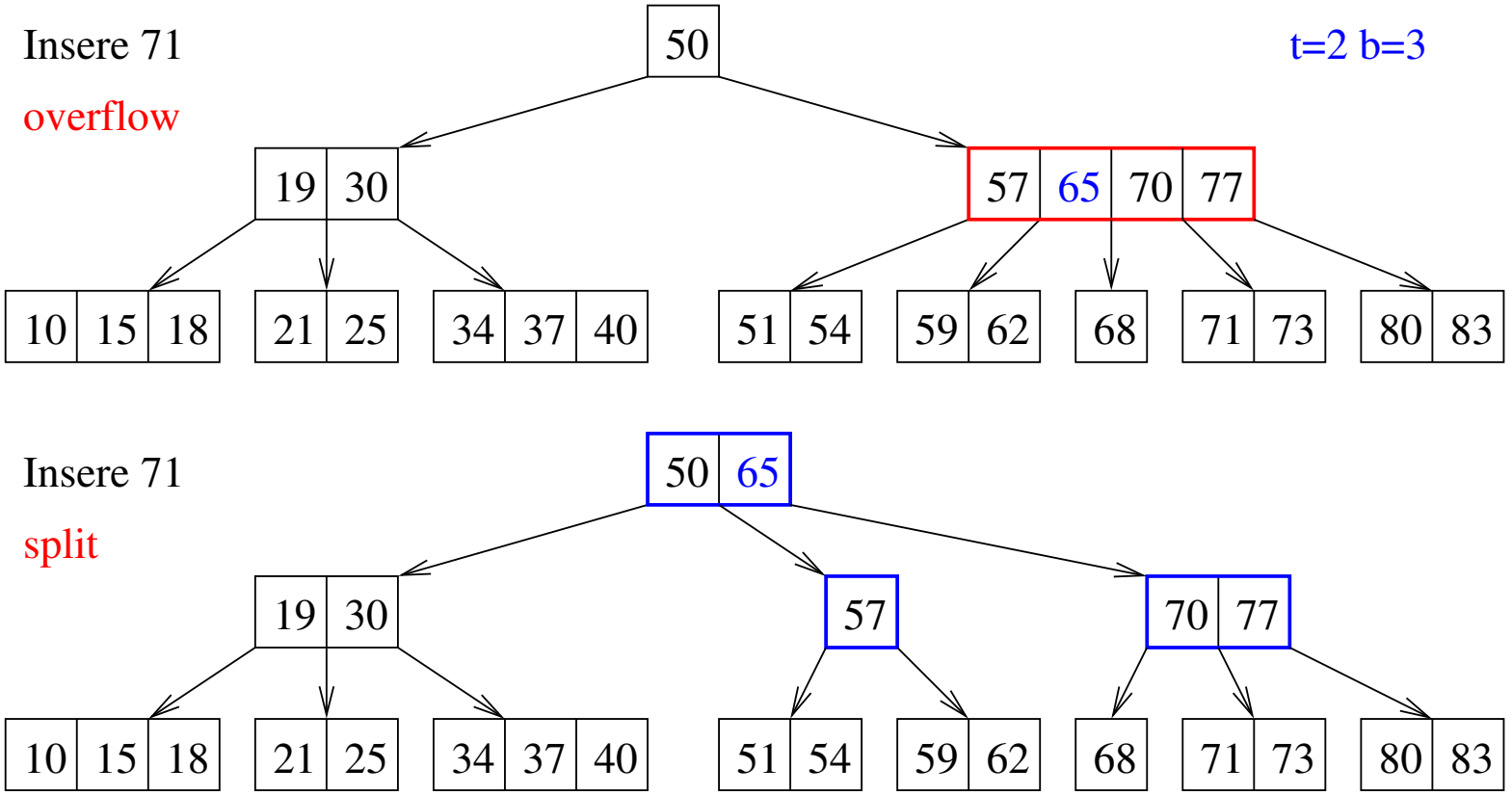


Método tradicional de inserção

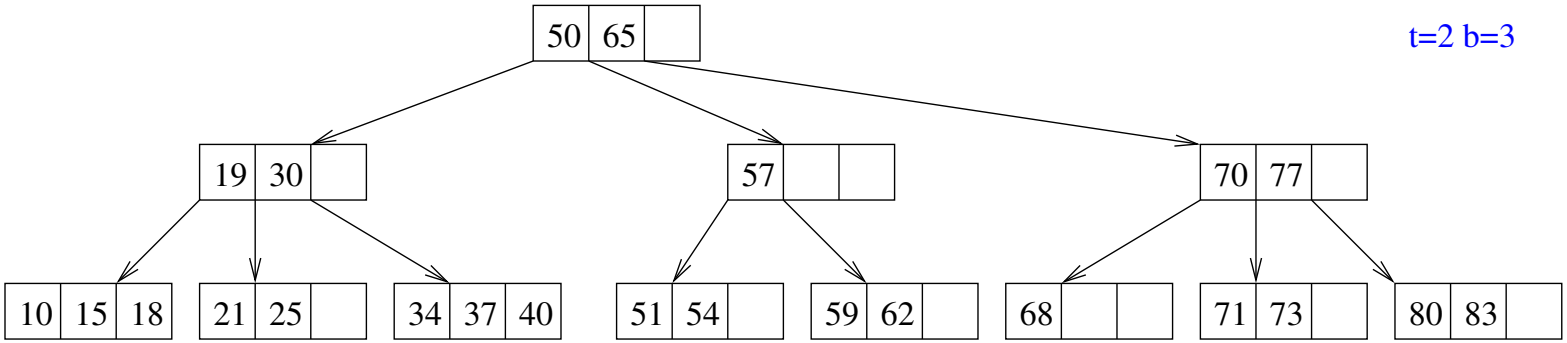




Método tradicional de inserção

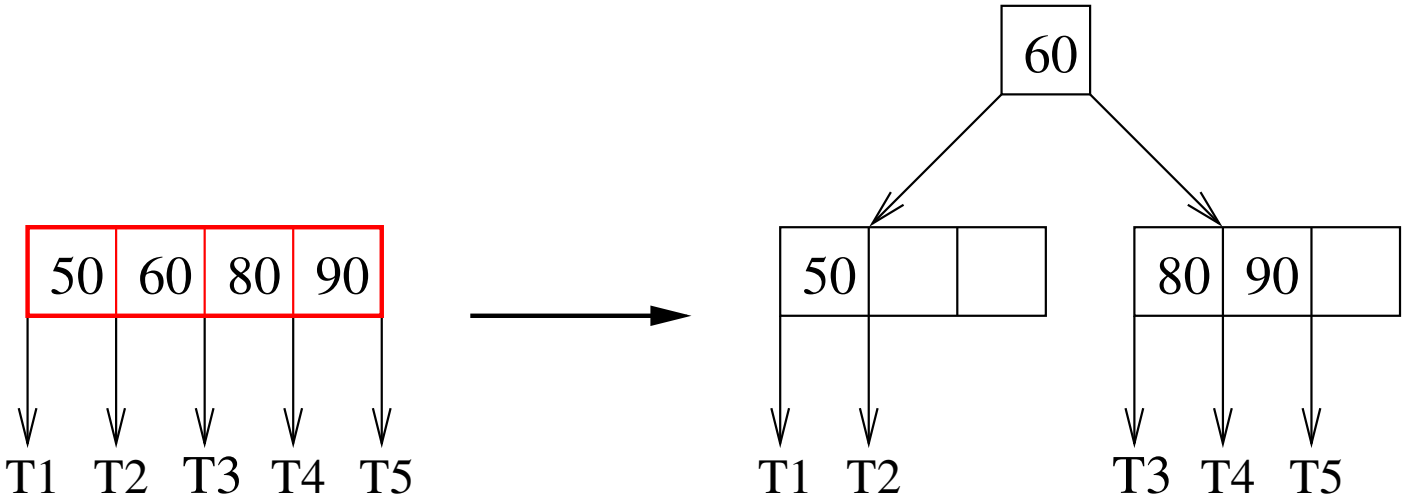


Método tradicional de inserção



Atingiu um nó não-cheio. O processo para.

Método tradicional de inserção



Se no caminho da raiz até a folha todos os nós estiverem cheios, então é feito um split na raiz e cria-se uma nova raiz com uma única chave.

Método de inserção de CLRS

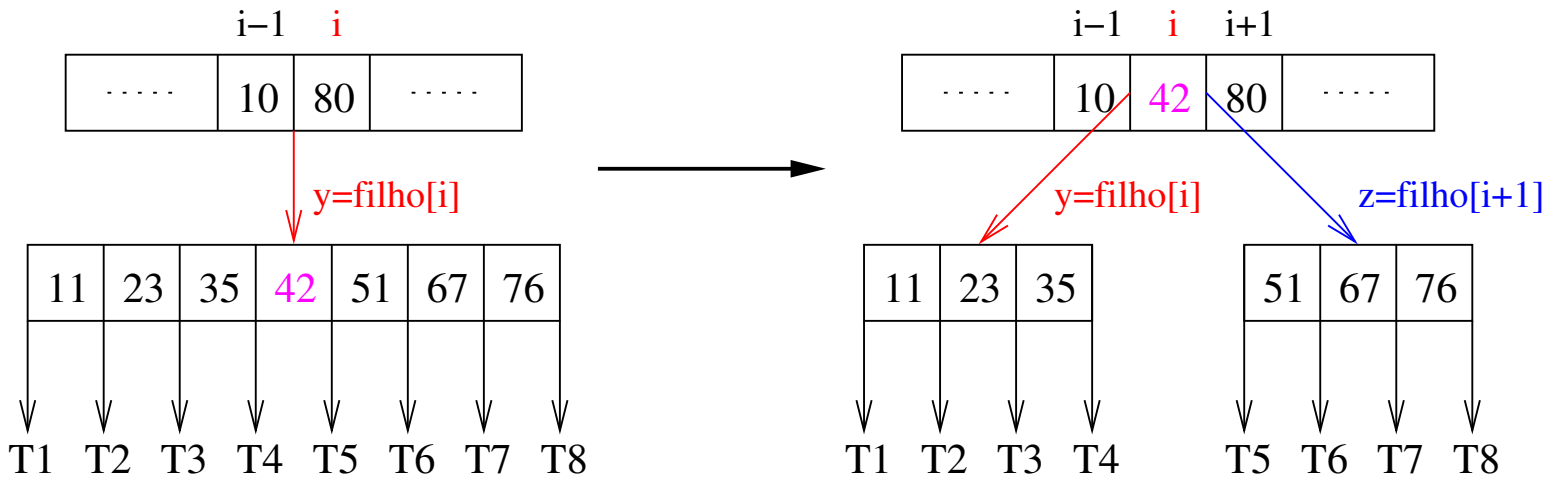
Cormen, Leiserson, Rivest e Stein propuseram outra maneira de implementar a inserção em árvore B.

À medida que o algoritmo faz a busca pela chave k, sempre que ele encontrar um nó x cheio, ele faz o split de x. Ele prossegue a busca em uma das subárvores criadas (comparando k com $chave[t]$).

Vantagem: faz apenas uma passada em cada nó, reduzindo o tempo de leitura e escrita.

Desvantagem: talvez faça splits desnecessários.

Inserção em árvore B (método de CLRS)

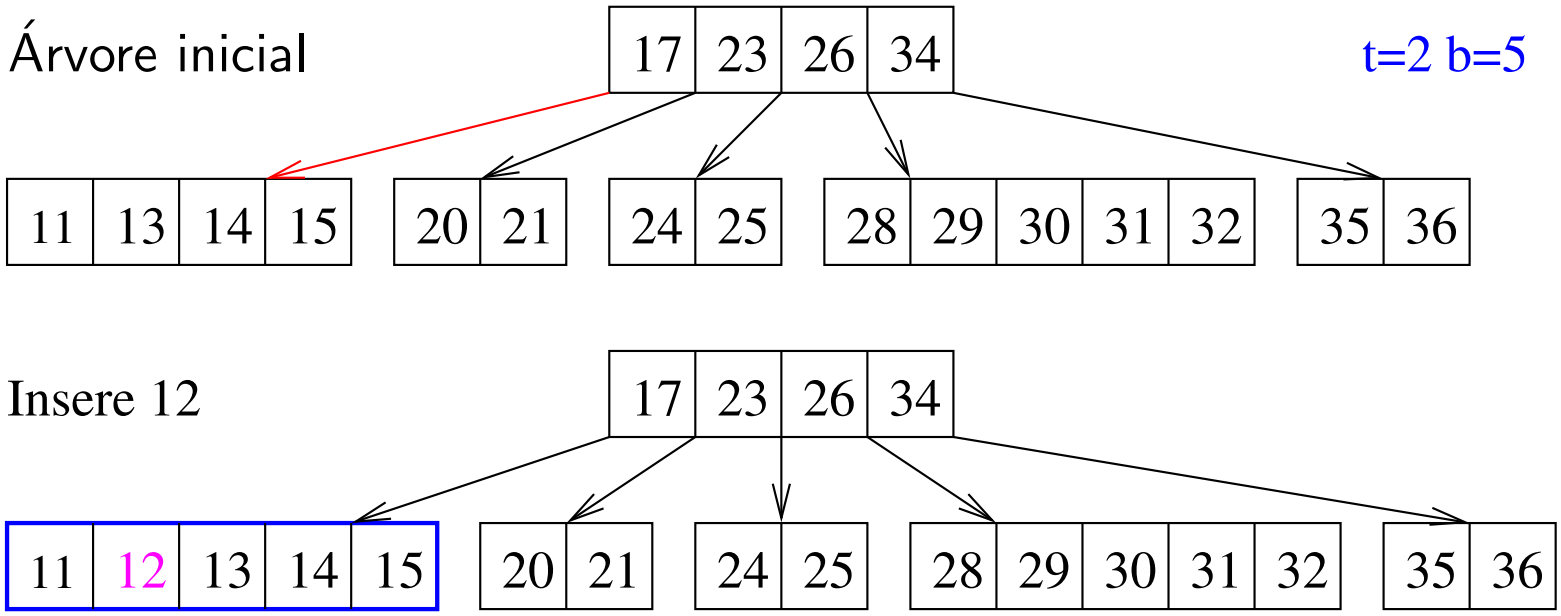


Temos $10 < k < 80$, $chave[i] = 42$ e $b = 2t - 1 = 7$.

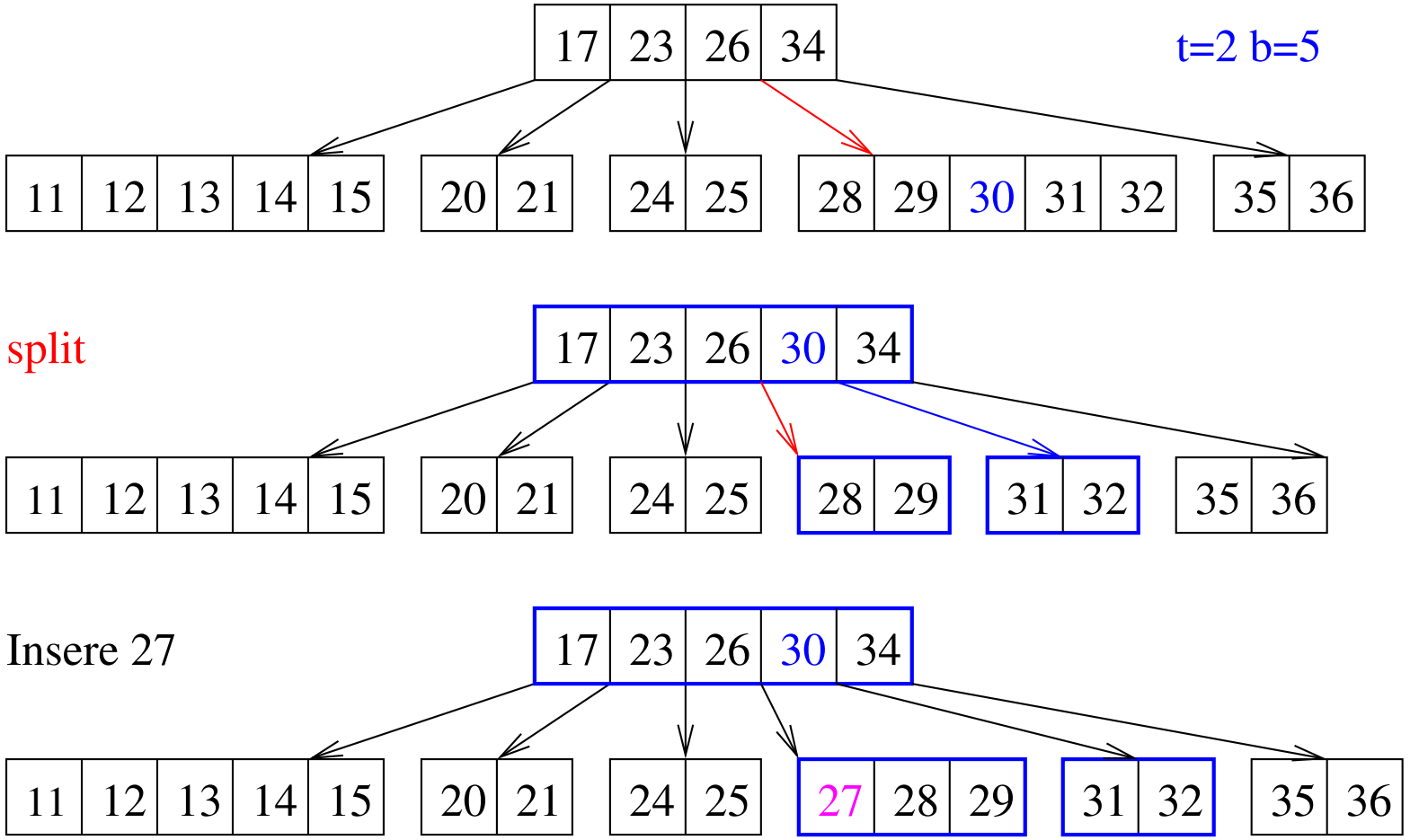
Se $k < 42$ então a busca continua em $x{->}filho[i]$.

Senão a busca continua em $x{->}filho[i+1]$.

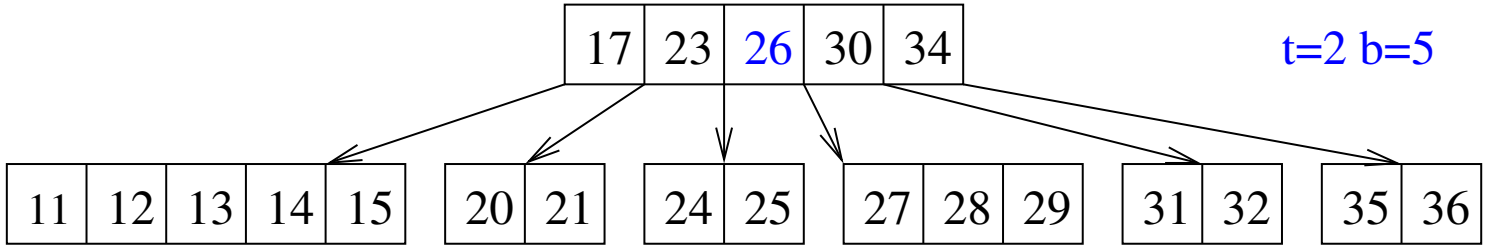
Método de inserção de CLRS — Exemplo 1

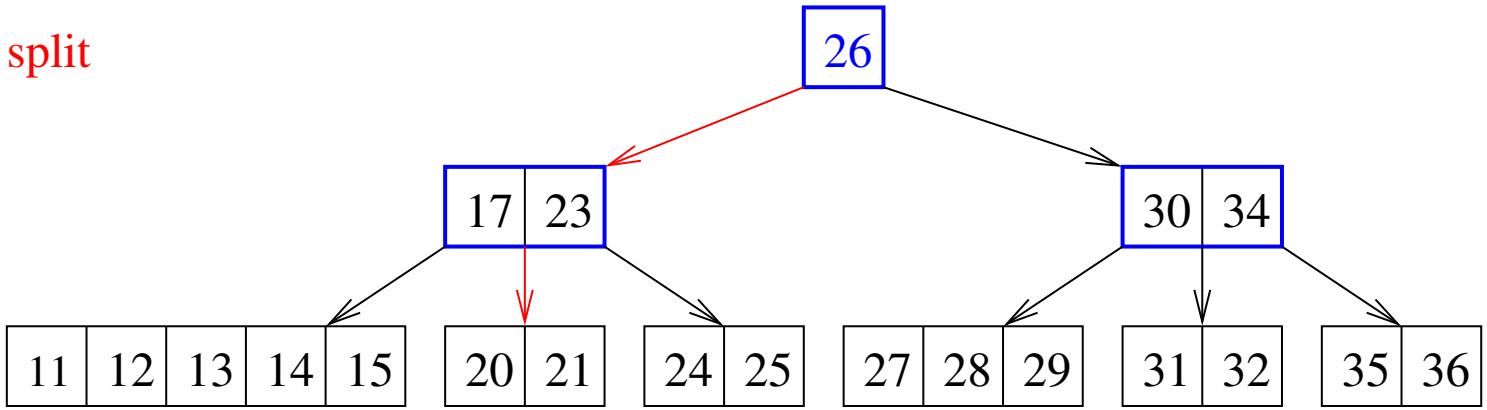


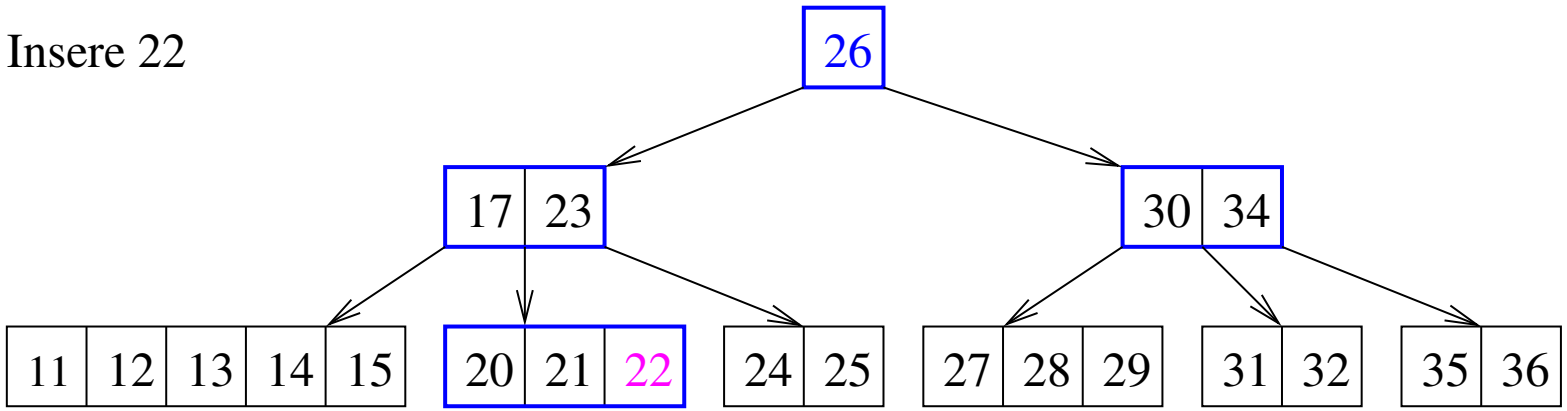
Método de inserção de CLRS — Exemplo 1



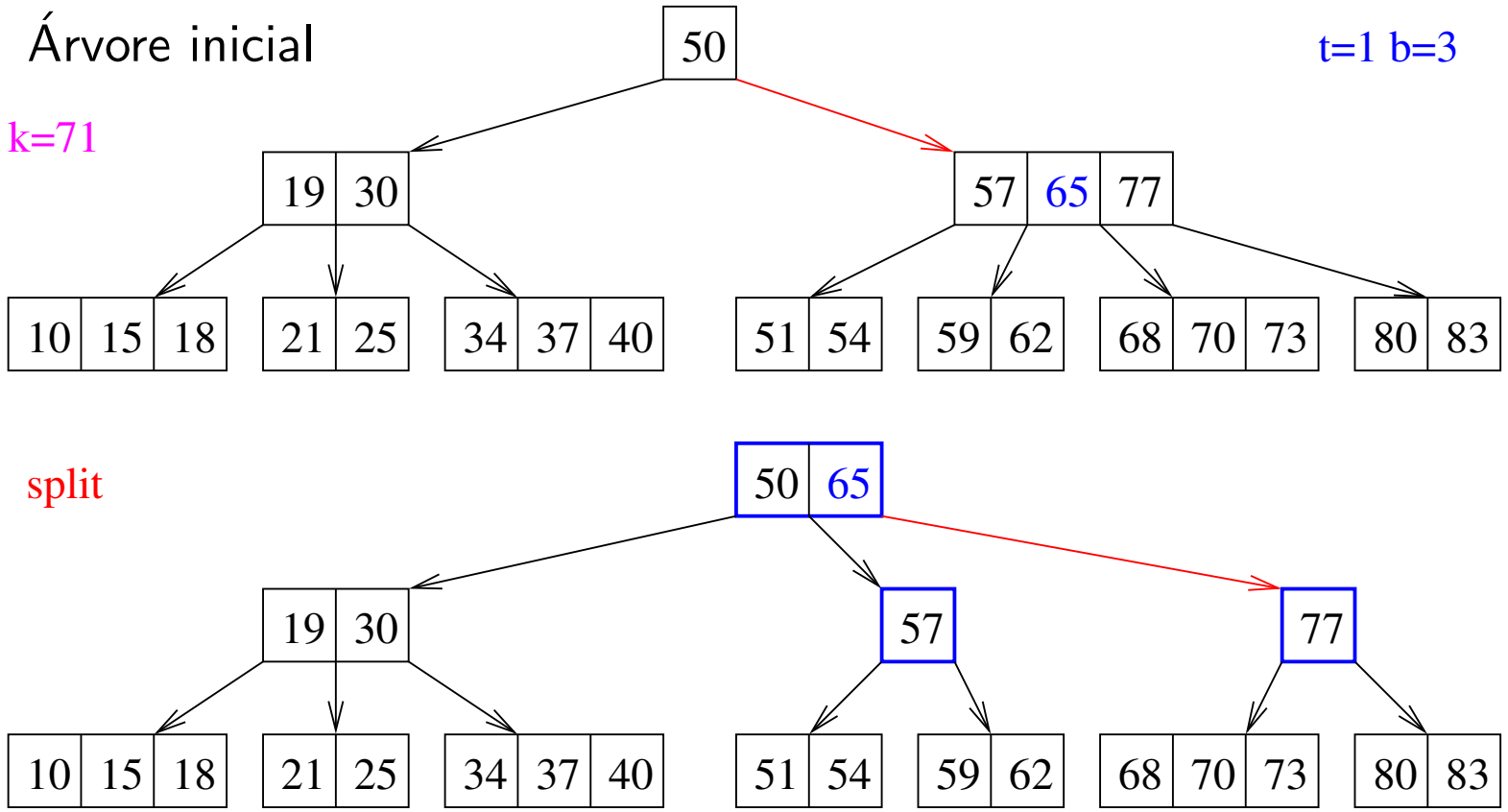
Método de inserção de CLRS — Exemplo 1



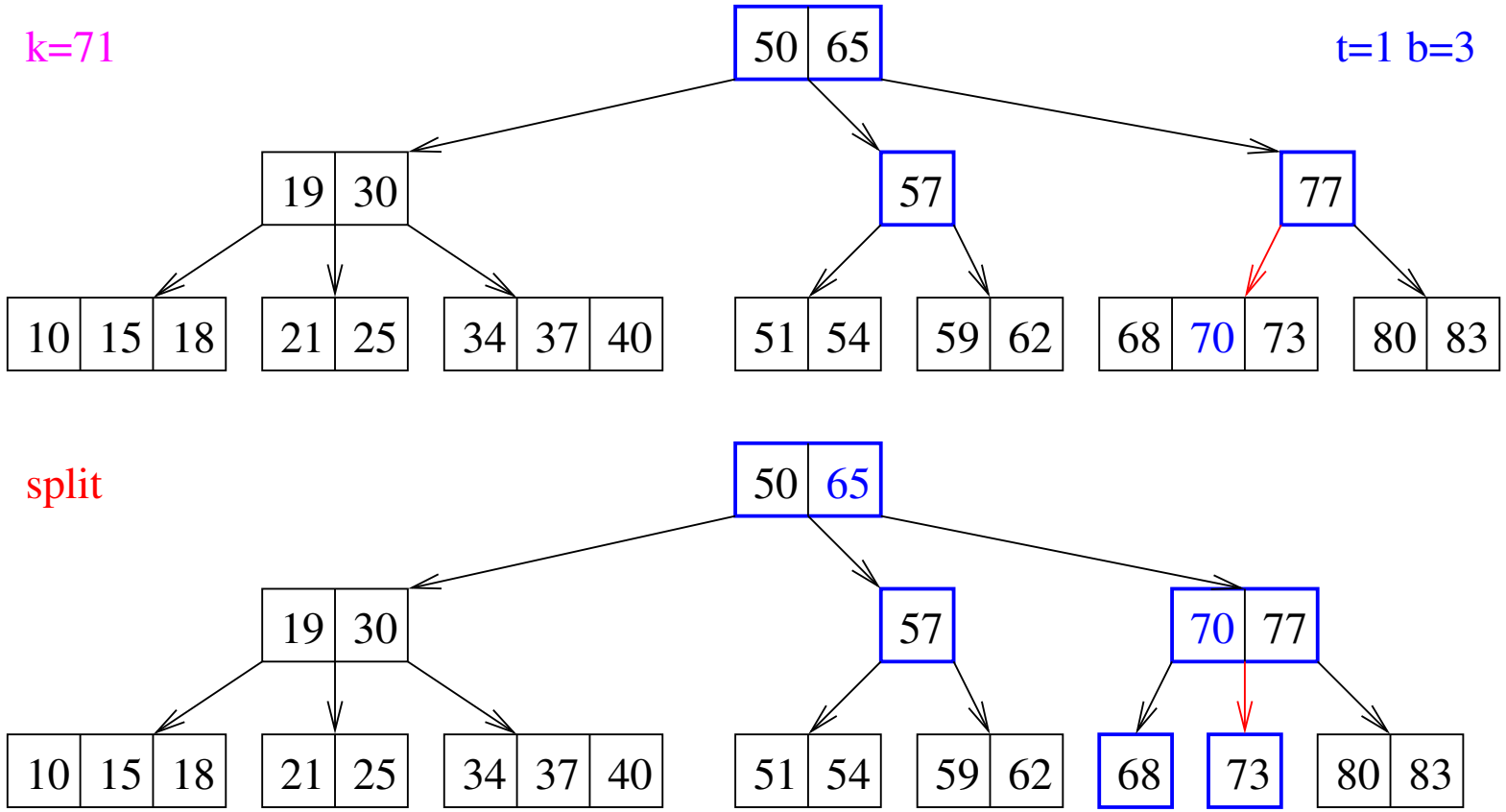




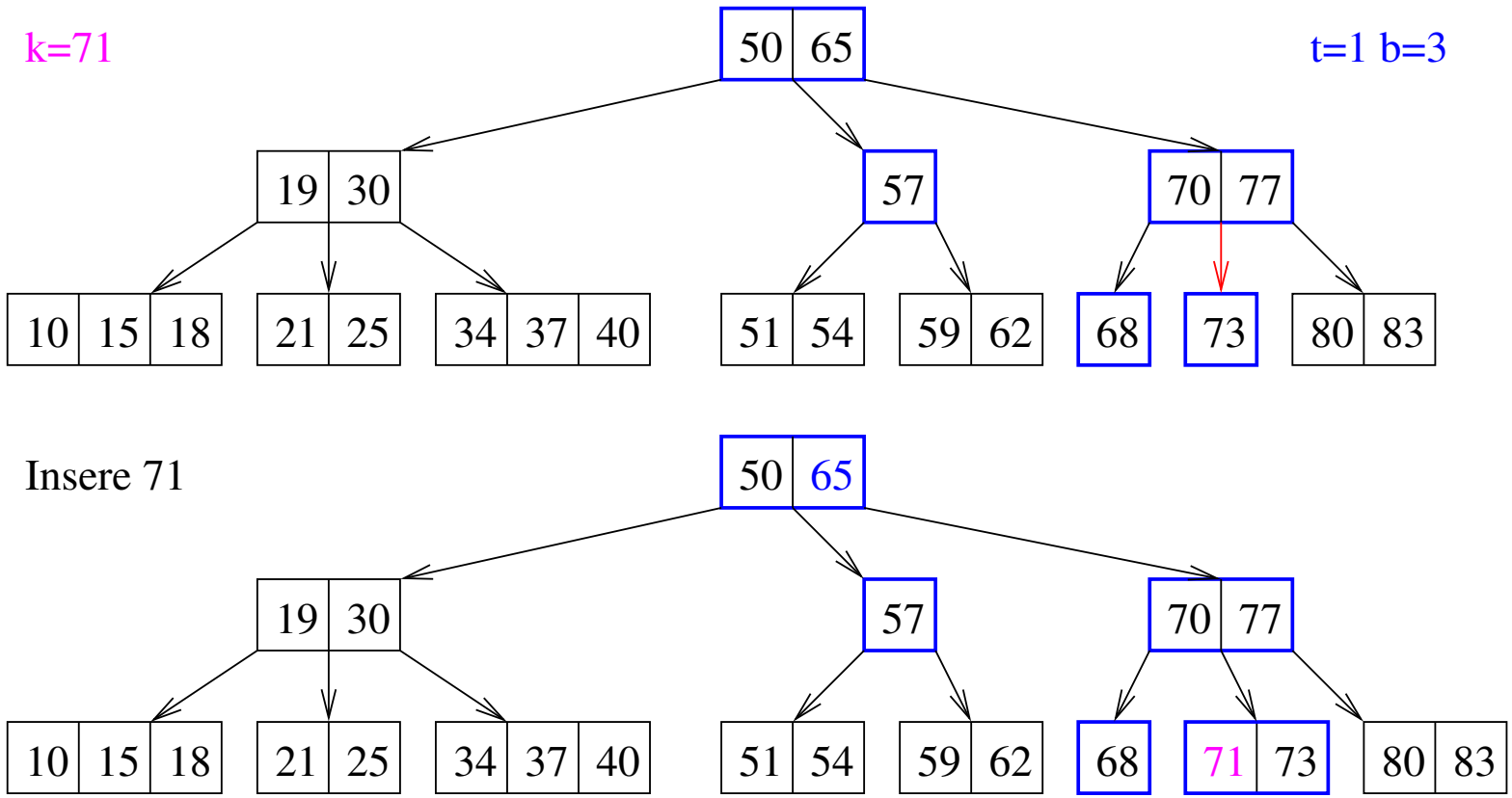
Método de inserção de CLRS — Exemplo 2



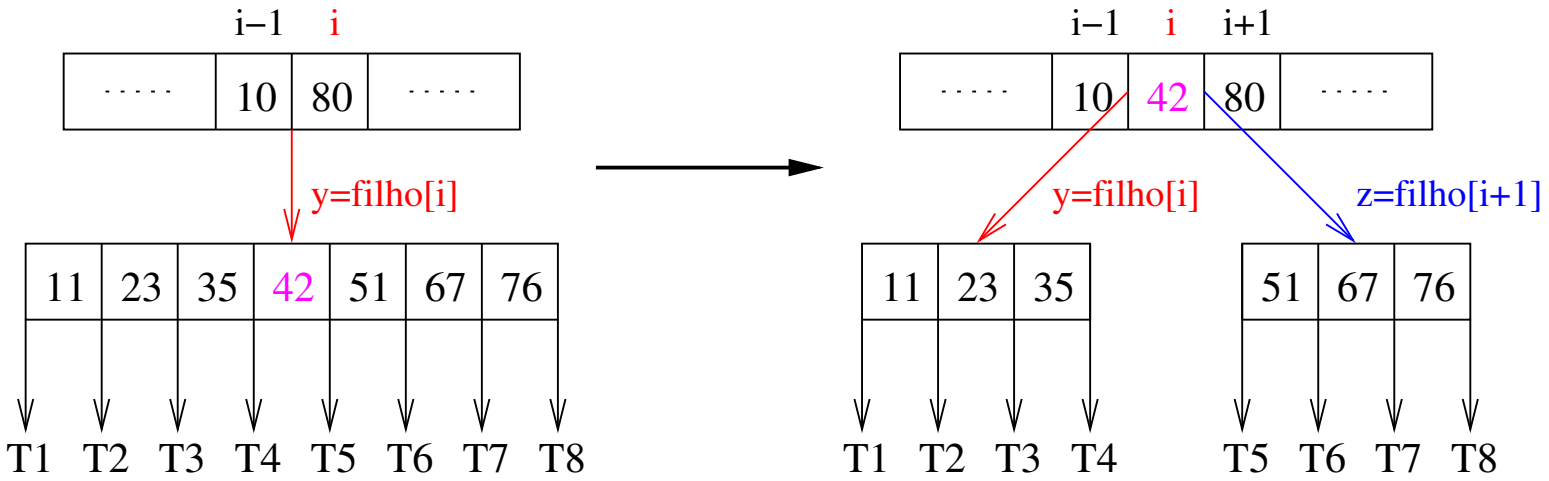
Método de inserção de CLRS — Exemplo 2



Método de inserção de CLRS — Exemplo 2



Implementação em pseudocódigo



Suponha que temos à disposição a rotina SplitChildArvoreB que recebe um nó x e um índice i tal que o nó $y = x{->}filho[i]$ é cheio. Ela faz o split de y.

Exercício. Escreva um pseudocódigo para esta função.

Inserção em árvore B (método de CLRS)

Entrada: a raiz r de uma árvore B e uma chave k
Saída: nova árvore B com a chave k inserida

```
InsereArvoreB(r, k)
 1: if r->n == 2t - 1 then
 2:     s = AllocateNode() ▷ nova raiz
 3:     s->folha = 0
 4:     s->n = 0
 5:     s->filho[1] = r
 6:     SplitChildArvoreB(s, 1)
 7:     InsereNaoCheioArvoreB(s, k)
 8: else
 9:     InsereNaoCheioArvoreB(r, k)
```

Ela verifica se a raiz é cheia. Se for o caso, é feito um split criando uma nova raiz. Então chama-se a InsereNaoCheioArvoreB. Esta função é a que faz a inserção propriamente dita e supõe que a raiz da árvore atual é não-cheia.

Inserção em árvore B (método de CLRS)

InsereNaoCheioArvoreB recebe a raiz **não-cheia** de uma árvore B e uma chave k. Ela faz o seguinte:

- se x é uma folha então insere k na posição correta
- se x é um nó interno então descobre a árvore $y = x{->}filho[i]$ em que k deve ser inserido
- se y for cheio então faz o split de y e atualiza y
- recursivamente insere k na árvore y

Inserção em árvore B (método de CLRS)

Entrada: raiz x não-cheia de uma árvore B e uma chave k
Saída: nova árvore B com a chave k inserida

```
InsereNaoCheioArvoreB(x, k)
 1: i = x->n
 2: if x->folha then
 3:     while i ⩾ 1 and k < x->chave[i] do
 4:         x->chave[i + 1] = x->chave[i]
 5:         i = i − 1
 6:         x->chave[i + 1] = k
 7:         x->n = x->n + 1
 8:         Escrita(x)
 9: else
10:     ▷ continua no próximo slide
```

Se x é folha então k é inserido na posição correta (como no insertionsort). O nó então é gravado na memória secundária.

Inserção em árvore B (método de CLRS)

```
 1: if x->folha then
 2:     ...
 9: else
10:     while i ⩾ 1 and k < x->chave[i] do
11:         i = i − 1
12:     i = i + 1
13:     LEITURA(x->filho[i])
14:     if x->filho[i].n == 2t − 1 then
15:         SPLITCHILDARVOREB(x, i)
16:         if k > x->chave[i] then
17:             i = i + 1
18:     INSERENAOCHEIOARVOREB(x->filho[i], k)
```

Se x não é folha então encontra a árvore $x{->}filho[i]$ em que k deve ser inserido. Se este nó for cheio então faz um split e atualiza i (se necessário). Recursivamente, insere k na árvore $x{->}filho[i]$.

Exercícios

Exercício 1. Mostre o resultado da inserção (nesta ordem) das chaves $16, 29, 27, 21, 13, 22, 18, 30, 32, 33, 23, 28, 24, 26, 11, 12, 34, 35, 14, 36, 15$ em uma árvore B de ordem $b = 3$ ($t = 2$). Mostre a configuração da árvore após cada inserção e imediatamente antes de cada split.

Remoção em árvore B (método de CLRS)

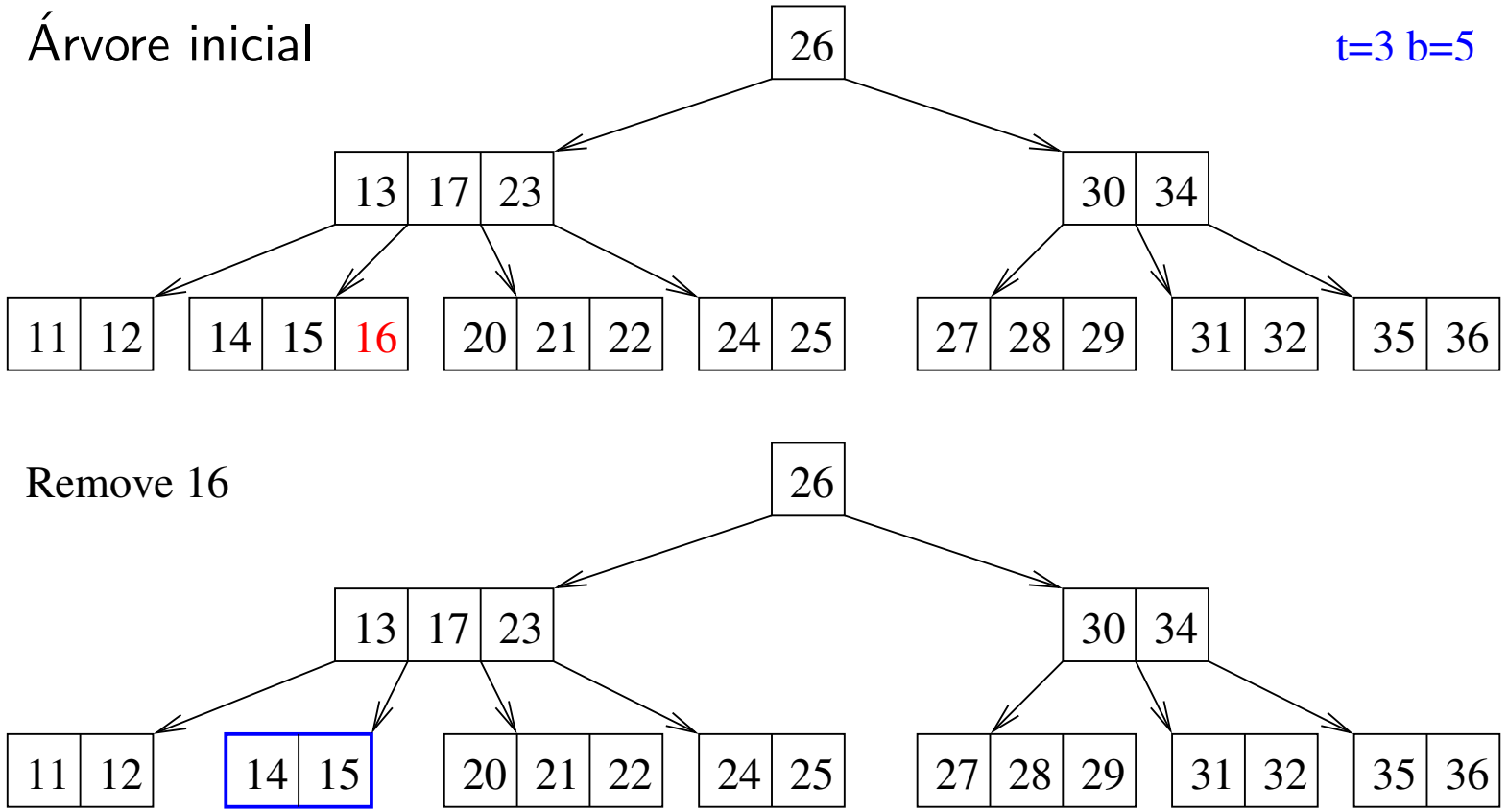
- A remoção é mais complicada que a inserção pois pode ocorrer tanto em um nó interno quanto em uma folha.
- Ao se remover uma chave de um nó interno, é preciso rearrumar os filhos deste nó.
- É preciso garantir que após a remoção, todo nó distinto da raiz tenha pelo menos $t - 1$ chaves (definição de árvore B). Isto pode ser feito remanejando chaves ou juntando dois nós.
- Como na inserção, o processo de remoção é feito em uma passada (exceto em um caso).

Remoção em árvore B (método de CLRS)

- A função $\textsc{RemoveArvoreB}(x, k)$ remove a chave k da árvore com raiz x. A função é recursiva e supõe que x **tem pelo menos t chaves**.
- Para garantir que o nó para qual queremos descer tenha pelo menos t chaves, usamos duas ideias:
  - pegamos emprestado uma chave de um dos irmãos imediatos (esquerdo ou direito)
  - juntamos (merge) dois irmãos em um novo nó juntamente com uma chave do nó atual.
- Somente quando garantimos que o nó destino tem pelo menos t chaves é que fazemos a chamada recursiva.

Remoção em árvore B

Caso 1: a chave k aparece em x e x é uma folha. Removemos k de x.



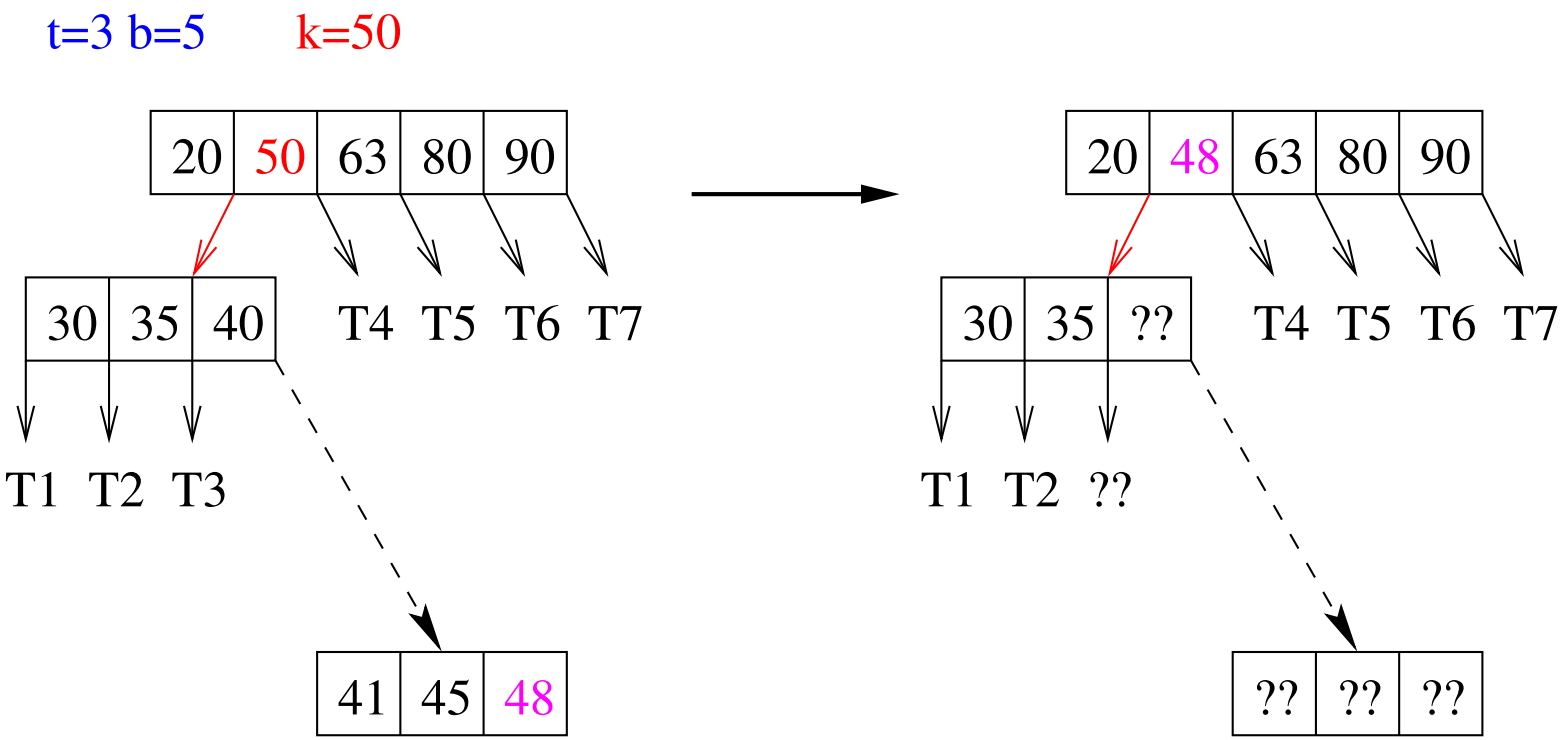
Remoção em árvore B (método de CLRS)

Caso 2: a chave k está em x e x é nó interno. Temos três subcasos.

2(a) Se o filho de y que precede k em x tem pelo menos t chaves, então encontre o predecessor k' de k na subárvore de raiz y. Recursivamente, remova k' da subárvore de raiz y e coloque a chave k' no lugar de k no nó x.

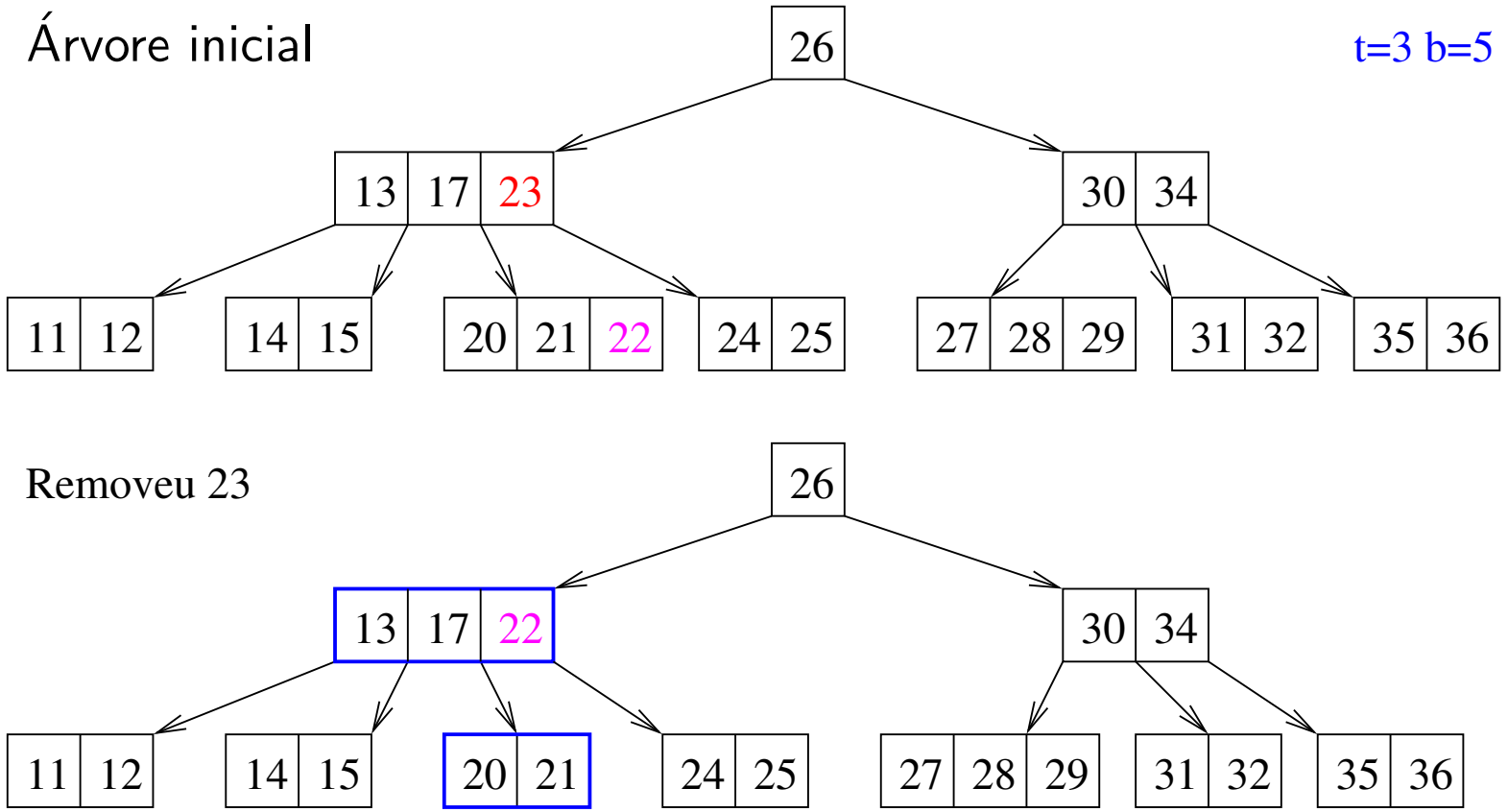
2(b) Se y tiver apenas $t-1$ chaves, então examine o filho z de x que sucede k em x. Se z tiver pelo menos t chaves, então encontre o sucessor k' de k na subárvore de raiz z. Recursivamente, remova k' da subárvore de raiz z e coloque a chave k' no lugar de k no nó x.

Esquema do Caso 2(a)



O caso 2(b) em que remove-se o sucessor de k é similar. Aqui é necessário voltar da recursão para inserir o predecessor (ou sucessor) na raiz.

Ilustração do Caso 2(a)



Remoção em árvore B (método de CLRS)

Caso 2: a chave k está em x e x é nó interno. Temos três subcasos.

2(c) Caso contrário, tanto y quanto z tem apenas $t-1$ chaves. Neste caso, junte (merge) k e todas as chaves de z em y, fazendo com que x perca a chave k e o apontador z. Agora y contém $2t-1$ chaves. Desaloque z e recursivamente remova k de y.

Esquema do Caso 2(c)

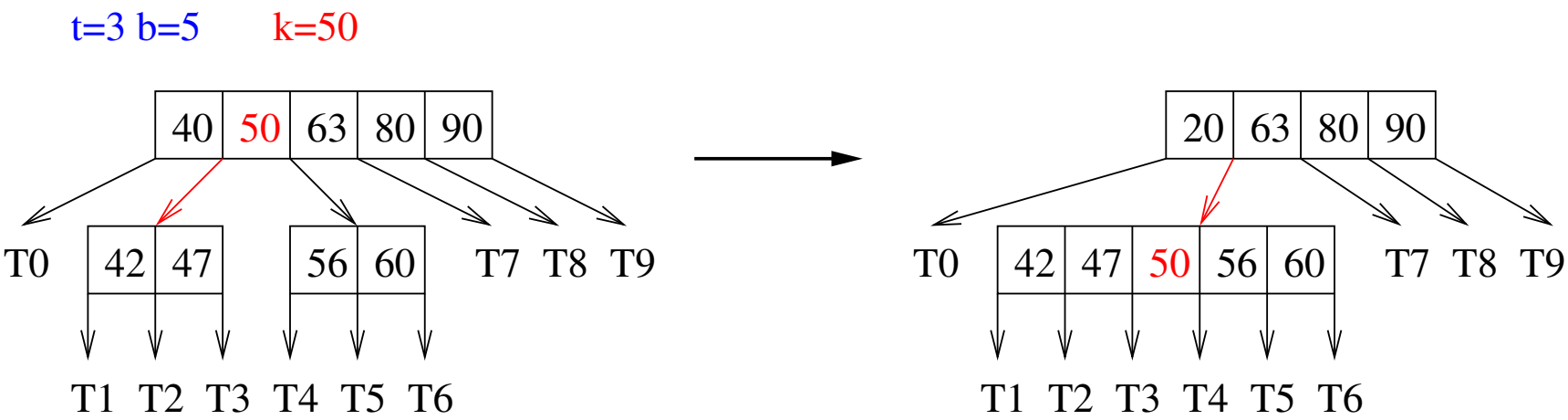
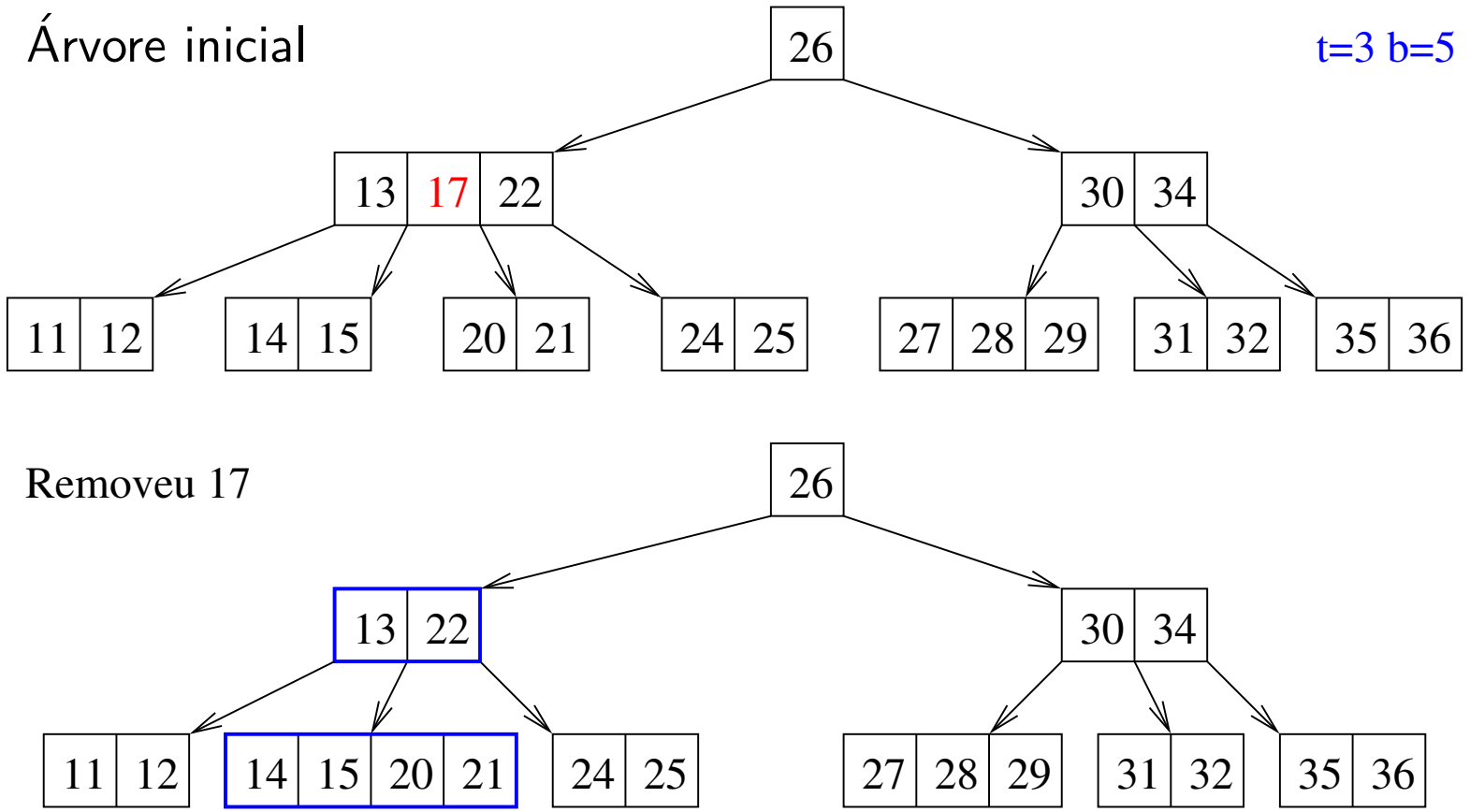


Ilustração do Caso 2(c)



Remoção em árvore B (método de CLRS)

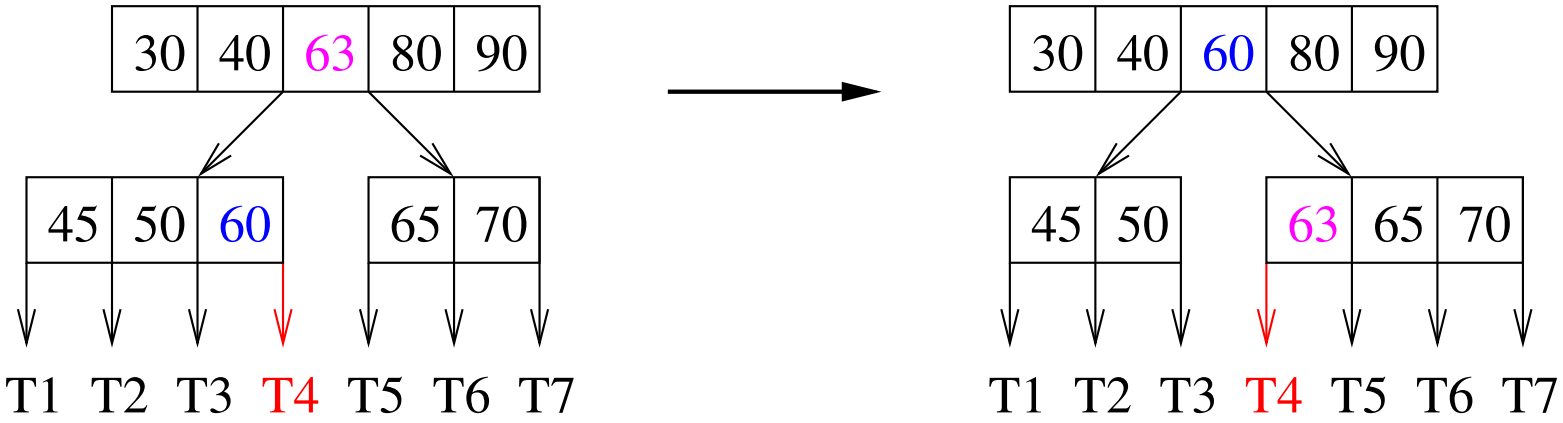
Caso 3: a chave k não está em x e x é um nó interno. Encontre a raiz $x{->}filho[i]$ da subárvore que deve conter k (se k estiver na árvore). Se $x{->}filho[i]$ tem apenas $t-1$ filhos, execute o passo 3(a) ou 3(b) para garantir que tenha pelo menos t chaves. Então recursivamente remova k da subárvore apropriada (com raiz sendo um dos filhos de x).

3(a) Se $x{->}filho[i]$ tem apenas $t-1$ chaves mas tem um irmão imediato à esquerda ou à direita com pelo menos t chaves, então desça uma chave de x para $x{->}filho[i]$ e suba uma chave do irmão para x, movendo também o apontador apropriado para $x{->}filho[i]$.

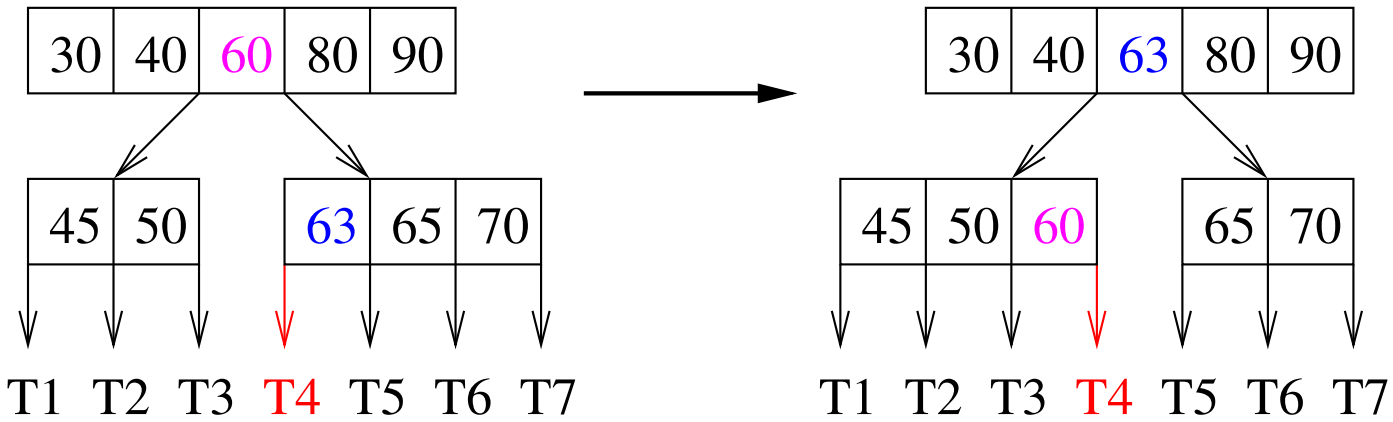
3(b) Se $x{->}filho[i]$ e seus irmão imediatos têm apenas $t-1$ filhos, junte $x{->}filho[i]$ com um dos irmãos imediatos, descendo uma chave de x para o novo nó para ser a mediana.

Esquema do Caso 3(a)

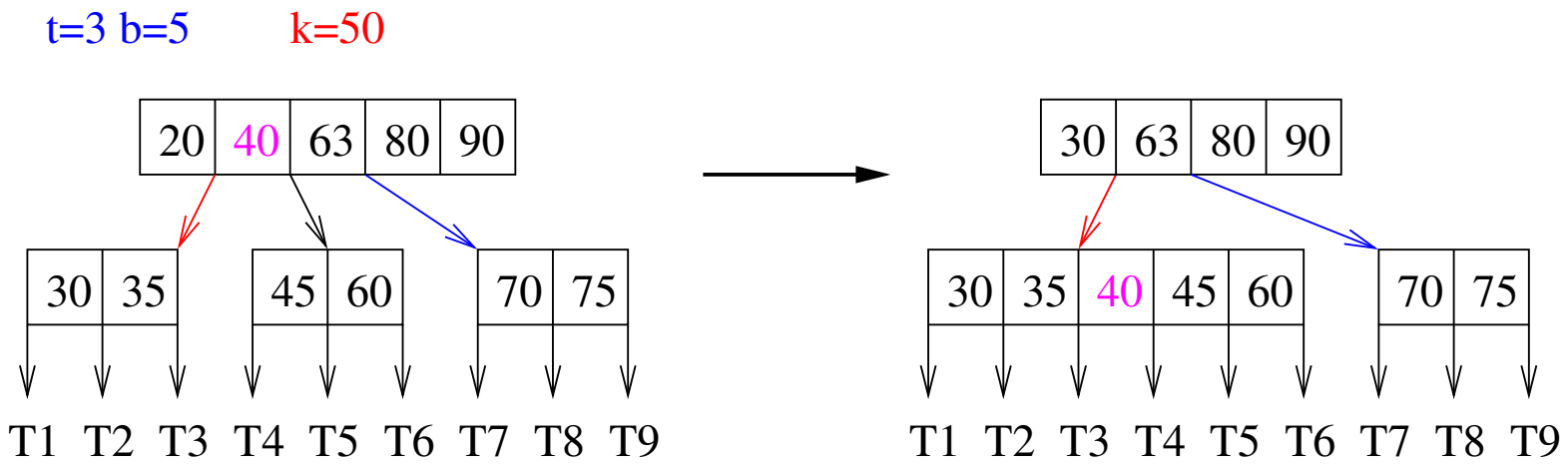
t=3 b=5 k=73



t=3 b=5 k=51



Esquema do Caso 3(b)



Note que pode não haver irmão imediato à esquerda. Neste caso, faz-se o procedimento análogo com o irmão imediato à direita.

Ilustração do Caso 3(a)

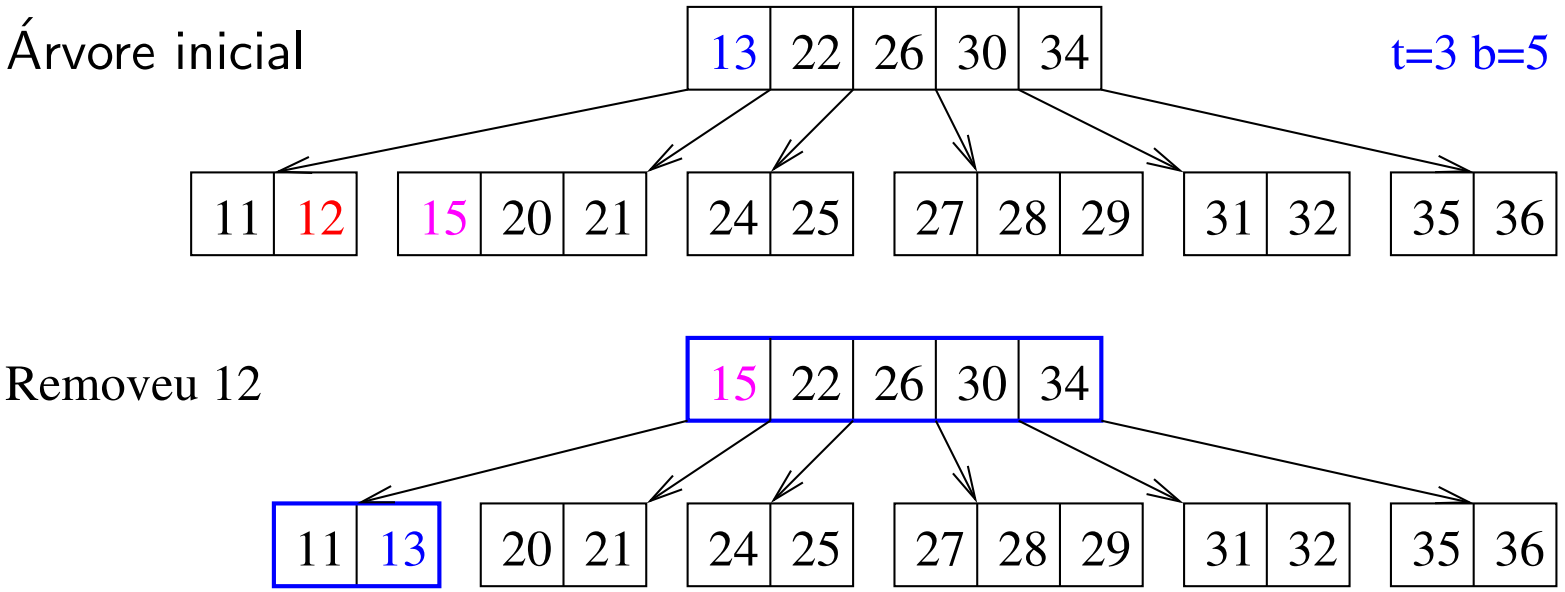
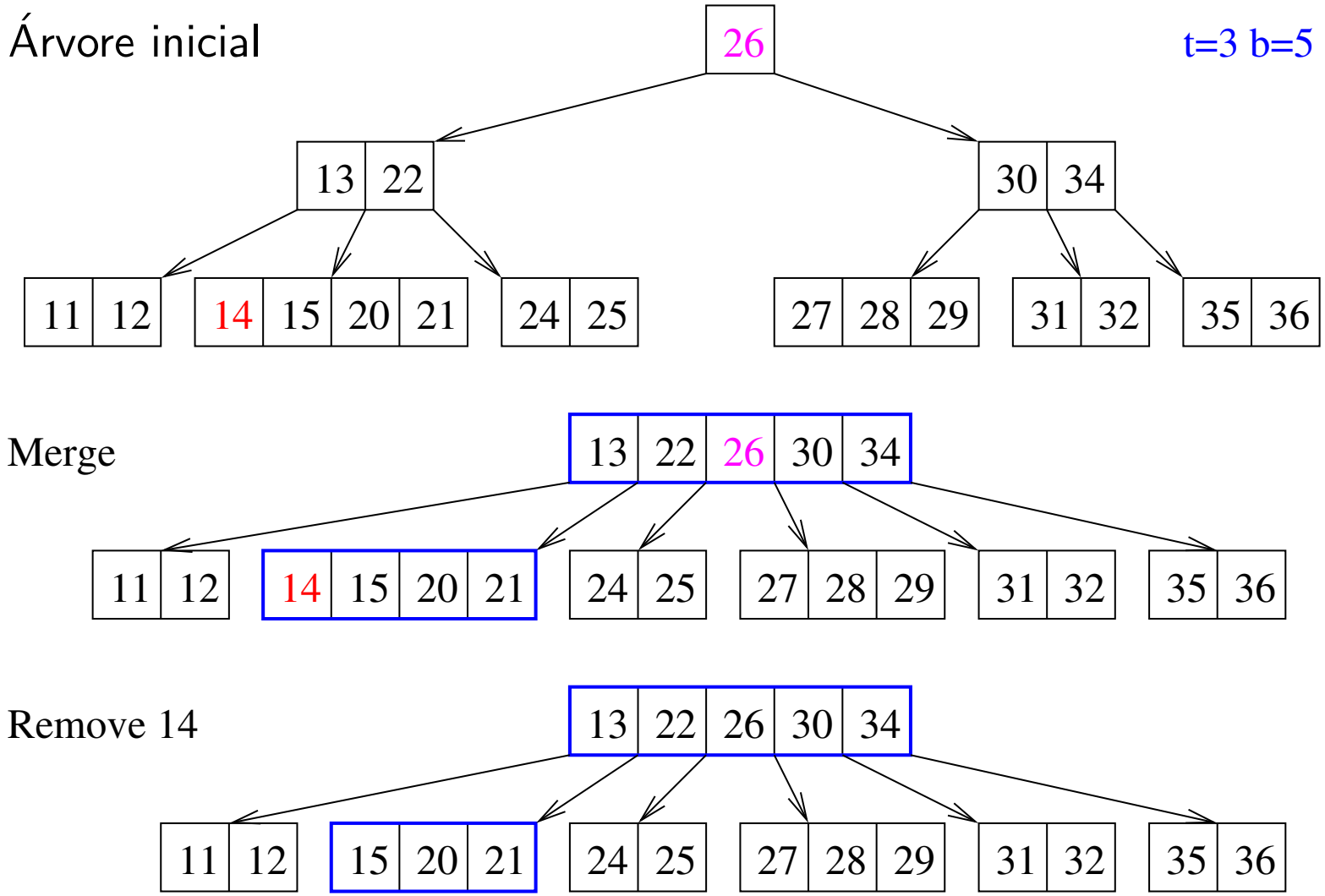
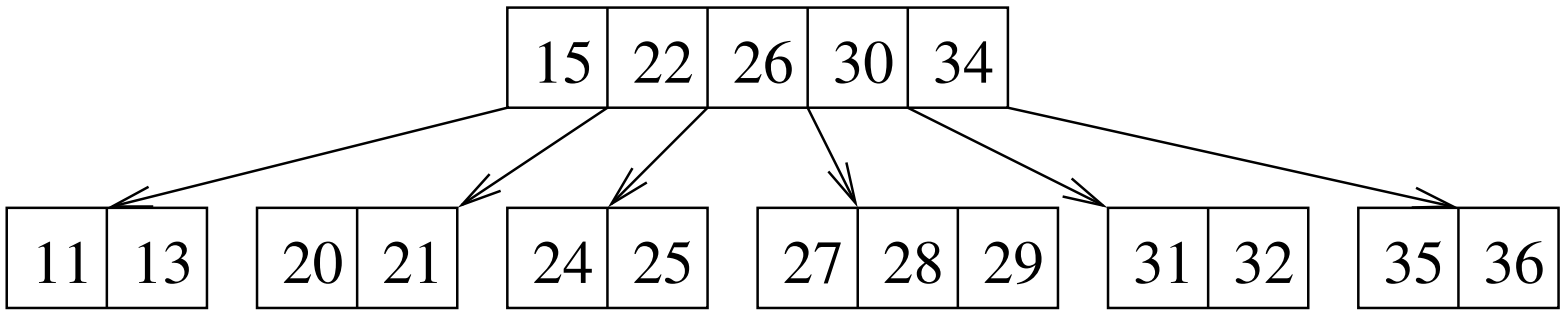


Ilustração do Caso 3(b)



Exercícios

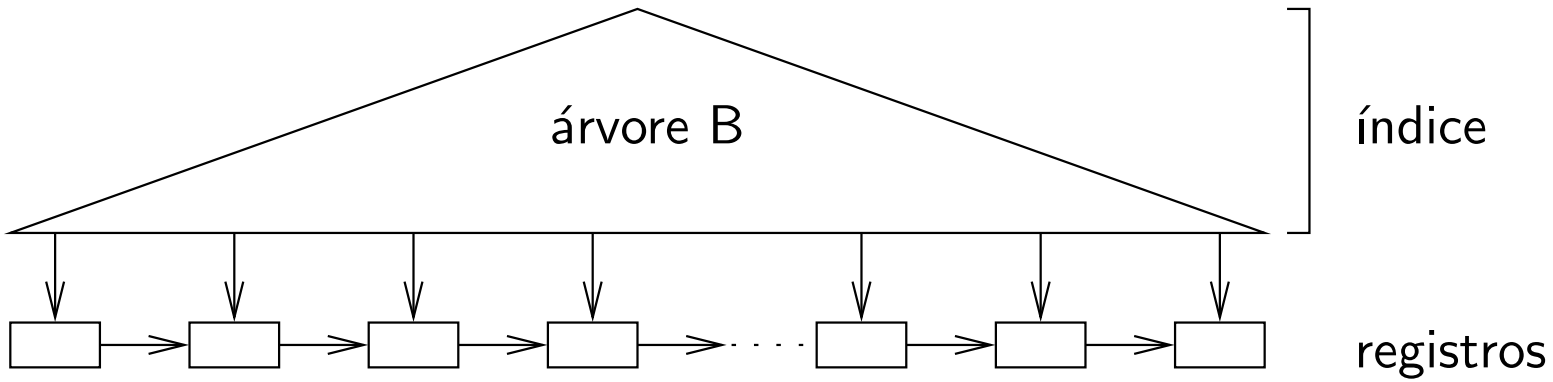
Exercício. Remova as chaves $13, 26$ e 22 nesta ordem da árvore B abaixo de ordem $b = 5$ $(t = 3)$.

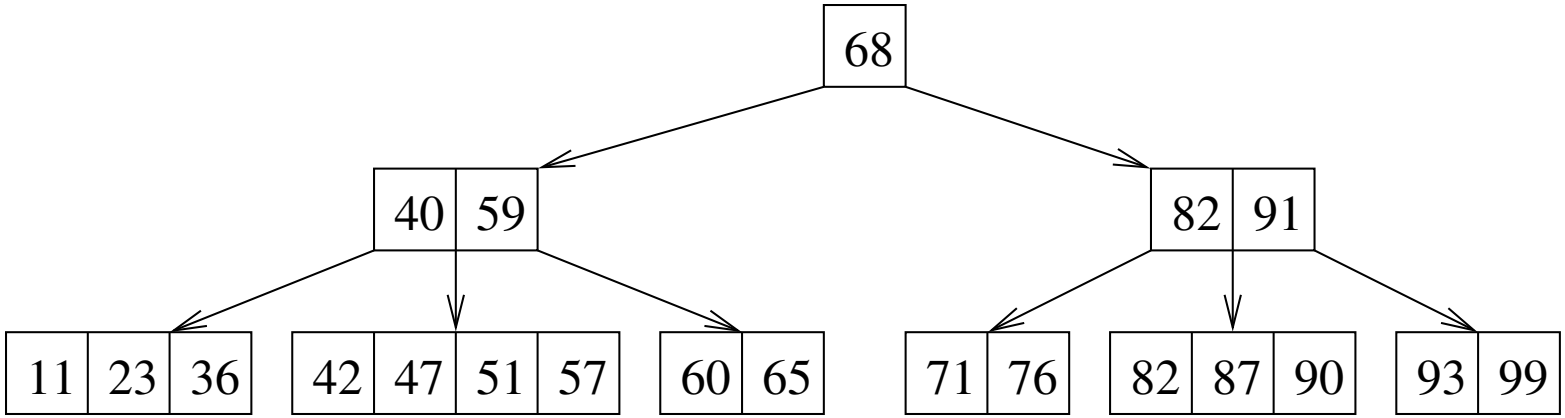


Árvores B+

Uma variação bastante usada de árvores B são as árvores B+:

- os registros (dados com as informações relevantes) são mantidos nas folhas;
- os demais níveis formam um índice (apenas com chaves) organizado como uma árvore B;
- as folhas são conectadas da esquerda para a direita, o que permite acesso sequencial.





Note que as chaves nos nós internos não precisam corresponder a chaves existentes nas folhas. Pode haver também chaves repetidas (mas não entre folhas). Precisa-se convencionar onde fica o nó com a mesma chave (esquerda ou direita).

Operações sobre árvores B+

- **Busca:** é idêntico ao método usado na árvore B, mas a busca só deve parar quando chegar a uma folha.
- **Inserção:** similar à inserção (método tradicional) em árvores B. Neste caso, no caso em que há um split de uma folha, subimos uma cópia da chave. Nos outros outros níveis fazemos como no método tradicional.

  Pode-se usar também o método de CLRS para fazer a inserção. Neste caso, mesmo que haja espaço no nó, continuamos descendo até chegar a uma folha.
- **Remoção:** é mais simples do que em árvores B porque a remoção é sempre feita em uma folha (elimina o Caso 2).