

Universidade Federal do Pará
Instituto de Ciências Exatas e Naturais
Programa de Pós-Graduação em Ciência da Computação

Árvores Balanceadas

Nelson Cruz Sampaio Neto
nelsonneto@ufpa.br

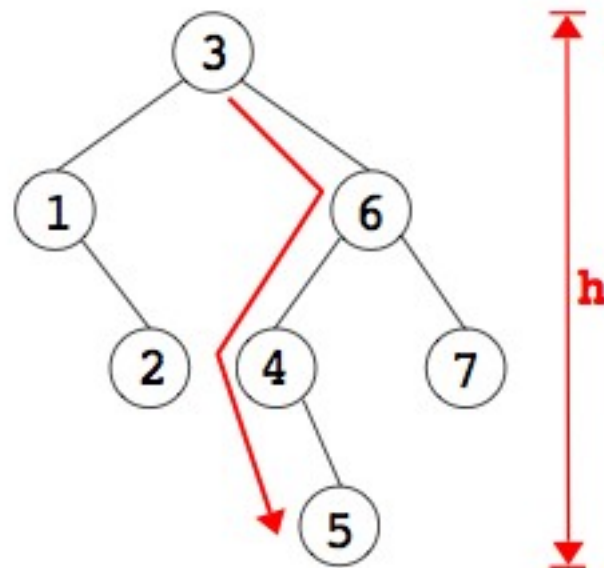
10 de abril de 2019

Introdução

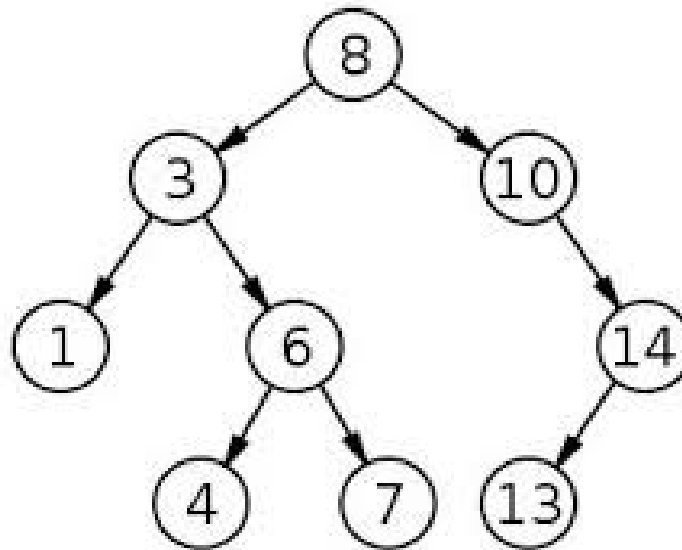
- A árvore de busca (ou pesquisa) é uma estrutura de dados muito eficiente para armazenar informação.
- Estrutura adequada quando existe necessidade de considerar todos ou alguma combinação de requisitos, tais como:
 - i. acessos direto e sequencial eficientes;
 - ii. facilidade de inserção e retirada de registros;
 - iii. boa taxa de utilização de memória; e
 - iv. utilização de memória primária e secundária.

Introdução

- As árvores de busca binária são estruturas de dados encadeadas em que cada nó é um objeto.
- Toda árvore de busca binária deve satisfazer a propriedade:
 - Seja x um nó qualquer da árvore
 - Seja y um nó da sub-árvore da esquerda, então $\text{chave}[y] < \text{chave}[x]$
 - Seja y um nó da sub-árvore da direita, então $\text{chave}[y] > \text{chave}[x]$



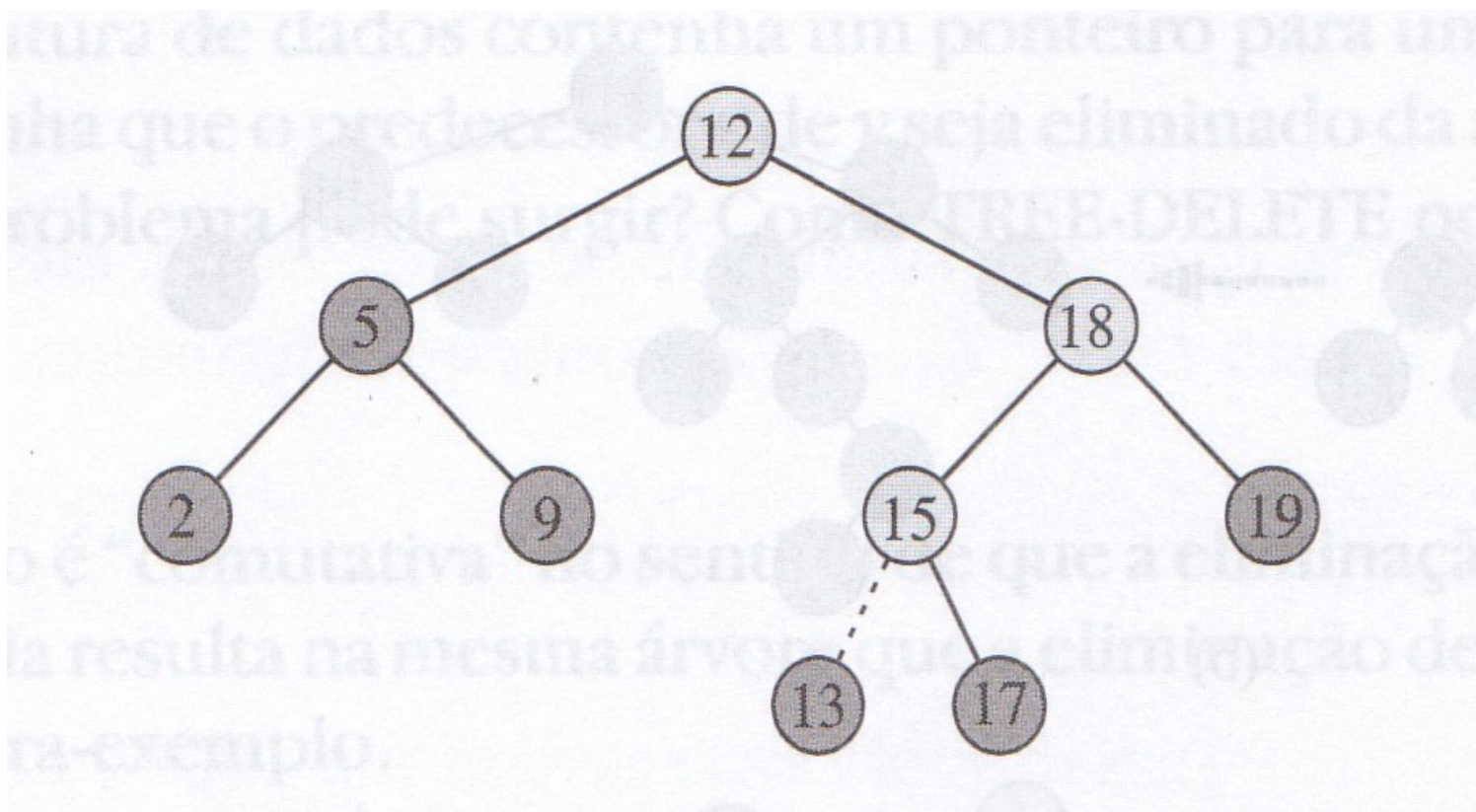
Introdução



- Busca pela chave 4 na árvore binária acima: 8 – 3 – 6 – 4.
- A chave mínima é 1, seguindo os ponteiros a esquerda a partir da raiz.
- A chave máxima é 14, seguindo os ponteiros a direita a partir da raiz.

Introdução

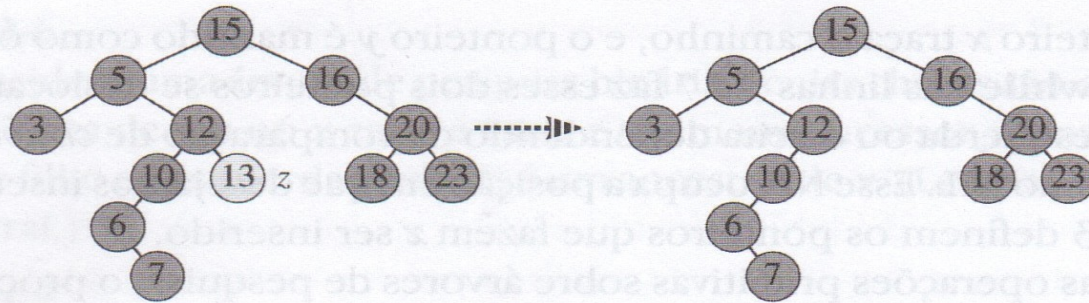
- Inserção do nó com chave 13. Os nós levemente sombreados indicam o caminho desde a raiz até a posição em que o item é inserido.



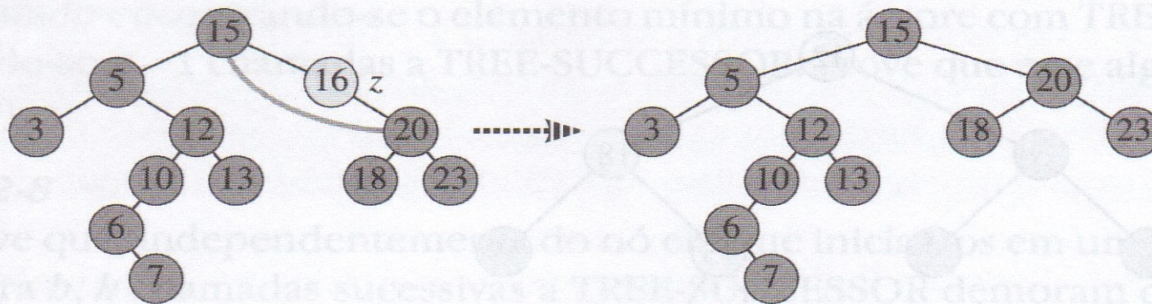
Introdução

- A operação de remoção de um nó z em uma árvore binária de busca consiste de 3 casos:
 - a) Se z não tem filhos, modificar o pai de z para que este aponte para NIL.
 - b) Se z possui apenas um filho, fazemos o pai de z apontar para o filho de z .
 - c) Se z possui dois filhos, colocamos no lugar de z o seu sucessor y , que com certeza não possui filho à esquerda.
- O sucessor de um nó x é o nó y com a menor chave que seja maior do que a chave de x .

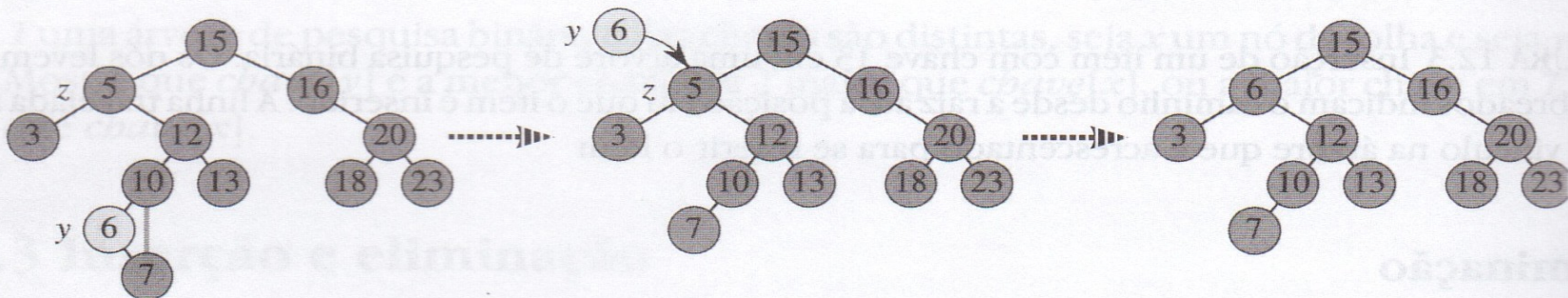
Introdução



(a)



(b)



(c)

Introdução

- Como visto, árvores de pesquisa binária admitem operações de conjuntos dinâmicos: seleção, inserção, remoção, entre outras.
- As operações levam um tempo proporcional à **altura da árvore**. Complexidade temporal no pior caso: $O(h)$.
- Para uma árvore completa de n nós, tais operações levam um tempo proporcional a $O(\log n)$. **Muito bom!**
- Mas se a árvore for uma lista de nós (ou seja, degenerada), as operações podem levar $O(n)$. **Ruim!**

Introdução

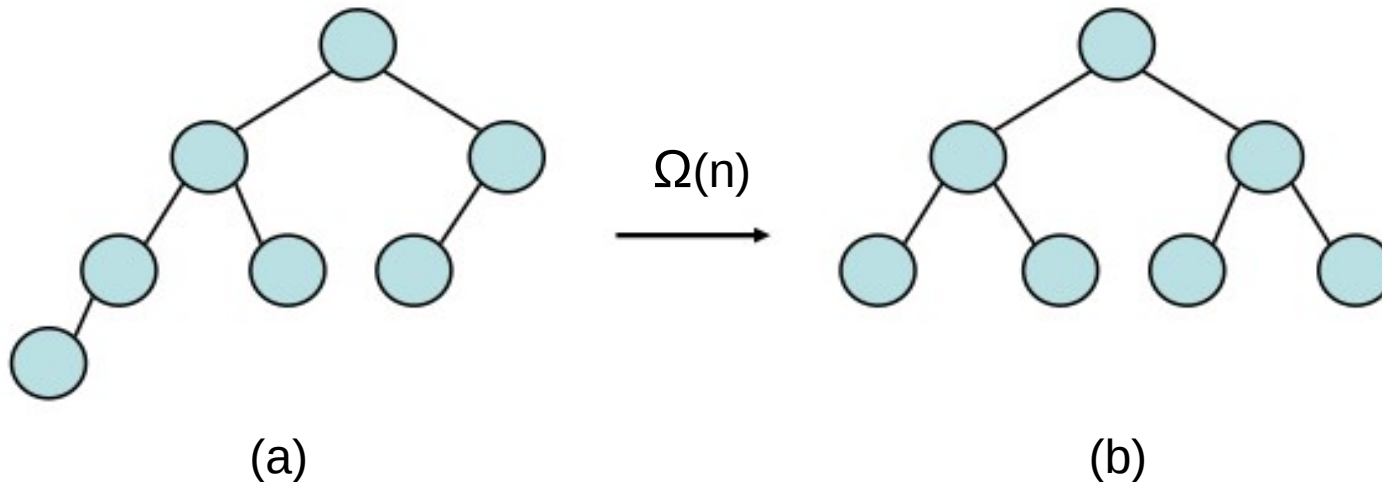
- Seja T uma **árvore binária completa** com n nós e altura h . Então,

$$2^h \leq n \leq 2^{h+1} - 1$$

- O valor $n = 2^{h+1} - 1$ ocorre quando a árvore é cheia. Nesse caso, basta observar que o número de nós em um dado nível é igual ao dobro do anterior.
- O valor $n = 2^h$ corresponde ao caso em que há apenas um nó no último nível de T .
- Com isso, conclui-se que $h \leq \log_2 n$, ou seja, $h = O(\log n)$.

Introdução

- Embora uma árvore completa possua altura proporcional a $\log n$, após uma operação de inserção, p.e., a árvore pode perder essa característica (figura(a)).
- Uma solução seria aplicar um algoritmo que tornasse a árvore novamente completa (figura (b)), porém, o custo dessa operação seria no mínimo proporcional a n , ou seja, $\Omega(n)$. **Ruim!**

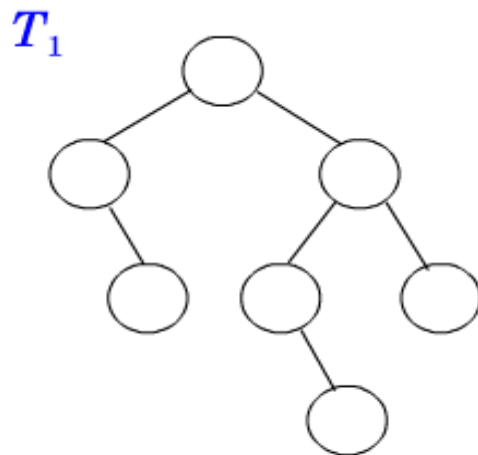


Introdução

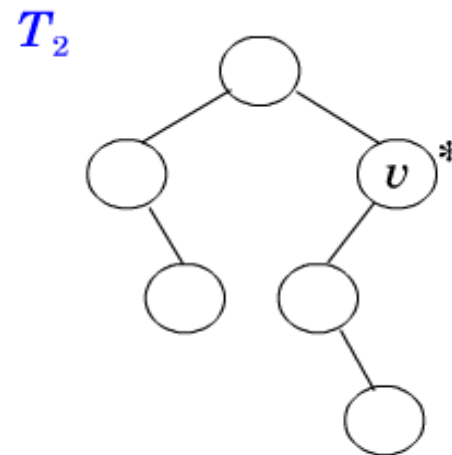
- **Objetivo:** Manter o custo das operações na mesma ordem de grandeza da altura de uma árvore completa, ou seja, $O(\log n)$, onde n é o número de nós da árvore. Em outras palavras, ter sempre uma **árvore balanceada**.
- Mas para uma árvore ser balanceada (ou seja, $h = O(\log n)$) ela precisa ser completa? **Não!**
- Há implementações de árvores de pesquisa binária balanceadas que garantem altura $O(\log n)$, tais como árvores vermelho-preto e árvores AVL, mas que não são obrigatoriamente completas.
- As árvores B também são consideradas balanceadas, mas não são binárias e possuem como característica o armazenamento de mais de uma chave por nó.

Árvores AVL

- **Definição:** É uma árvore de busca binária altamente balanceada. Em tal árvore, **as alturas das duas sub-árvores a partir de cada nó diferem no máximo em uma unidade.**
- Também chamada de árvore balanceada pela altura.
- Se uma dada árvore é dito AVL, então todas as suas sub-árvores também são AVL.



T_1 é AVL



T_2 não é AVL

(possui nó v que está desregulado)

Árvores AVL

- As operações feitas em uma árvore AVL geralmente envolvem os mesmos algoritmos de uma árvore de busca binária.
- A altura de uma árvore AVL com n nós é $O(\log n)$. Assim, suas operações levam um tempo $O(\log n)$, **inclusive a de arrumação**.
- Para definir o balanceamento é utilizado um fator específico:

$$FB(v) = h_e(v) - h_d(v)$$

$FB(v)$: fator de balanceamento do nó v

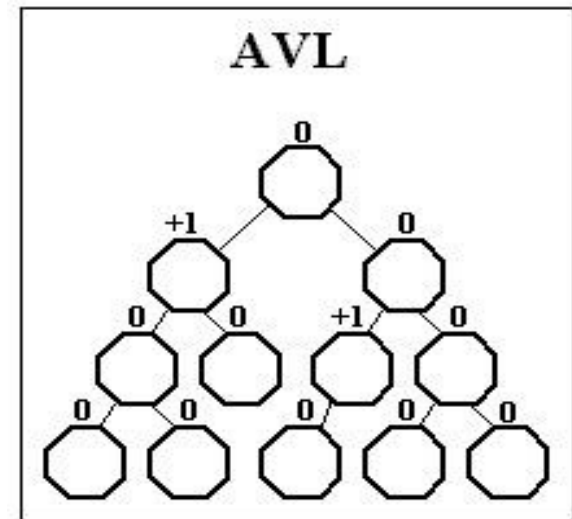
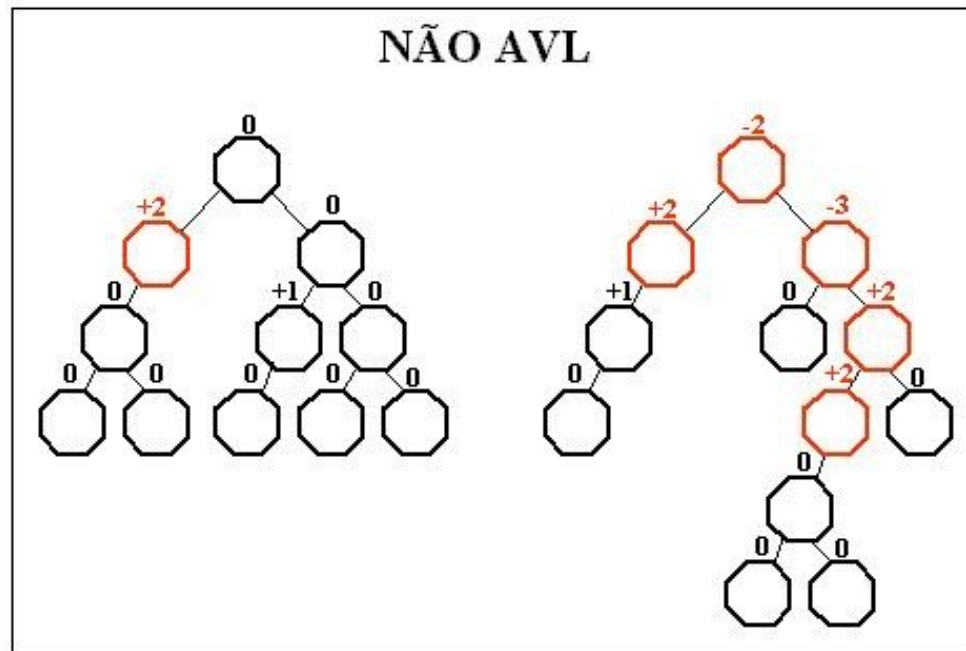
$h_e(v)$: altura da sub-árvore da esquerda

$h_d(v)$: altura da sub-árvore da direita

Árvores AVL

- Nós balanceados (ou **regulados**) são aqueles onde os valores de FB são -1, 0 ou +1.
 - 1 : sub-árvore direita mais alta que a esquerda
 - 0 : sub-árvore esquerda igual a direita
 - +1 : sub-árvore esquerda mais alta que a direita
- Qualquer nó com FB diferente desses valores é dito **desregulado**.
 - > 1 : sub-árvore esquerda está desregulando o nó
 - < -1 : sub-árvore direita está desregulando o nó
- Se todos os nós da árvore são regulados, então a árvore é AVL.

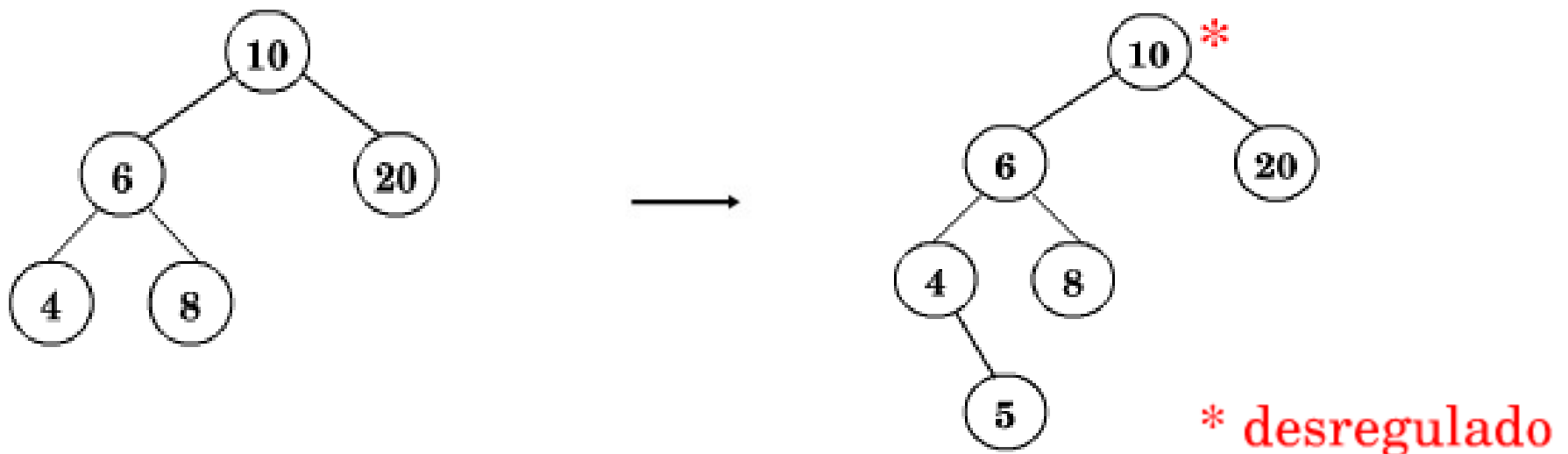
Exemplo



- Nos dois exemplos de árvore não AVL acima, percebe-se em vermelho os nós desregulados com $FB < -1$ ou $FB > +1$.
- Curiosidade: AVL = **A**delson-**V**elskii, G. e **L**andis, E. M.

Árvores AVL

- A inserção de um nó com chave 5 na árvore abaixo, faz com que ela deixe de ser AVL, com $FB(10) = +2$.
- E agora?



Árvores AVL

- Caso operações de inserção e remoção alterem o balanceamento da árvore, é necessário efetuar uma arrumação (ou **rotação**) para manter as propriedades da árvore AVL, tal que:
 - a) O percurso em ordem fique inalterado em relação a árvore desbalanceada. Em outras palavras, a árvore continua a ser uma árvore de busca binária.
 - b) A árvore modificada saiu de um estado de desbalanceamento para um estado de balanceamento.

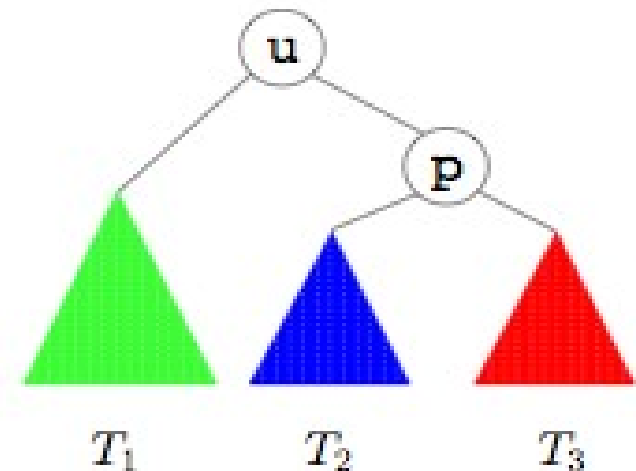
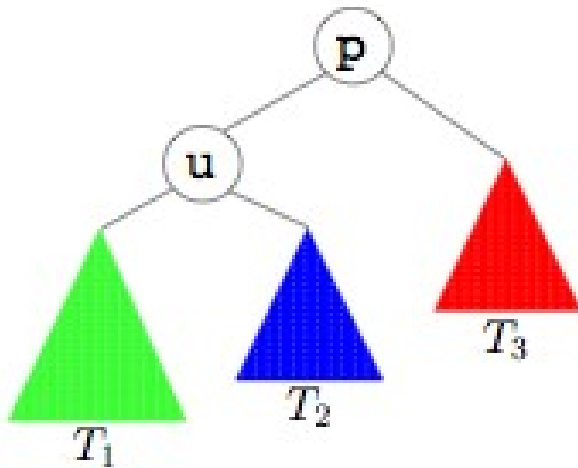
Árvores AVL

- **Definição:** A operação de rotação altera o balanceamento de uma árvore T , garantindo a propriedade AVL e a sequência de percurso em ordem.
- Podemos definir 4 tipos diferentes de rotação:
 - Rotação à direita
 - Rotação à esquerda
 - Rotação dupla à direita
 - Rotação dupla à esquerda

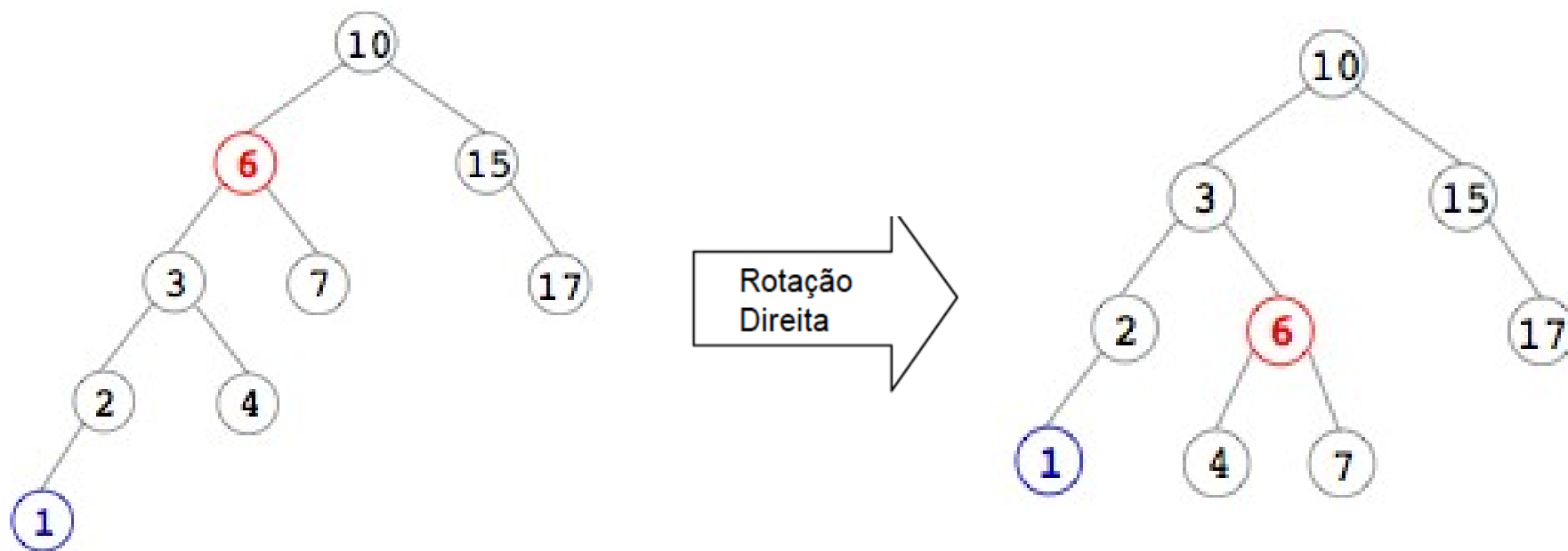
Árvores AVL

- Quando aplicar a rotação à direita?

$$\begin{aligned} h_E(p) &> h_D(p) \\ h_E(u) &> h_D(u) \end{aligned}$$



Exemplo



O nó 1 foi inserido na árvore,
causando seu desbalanceamento

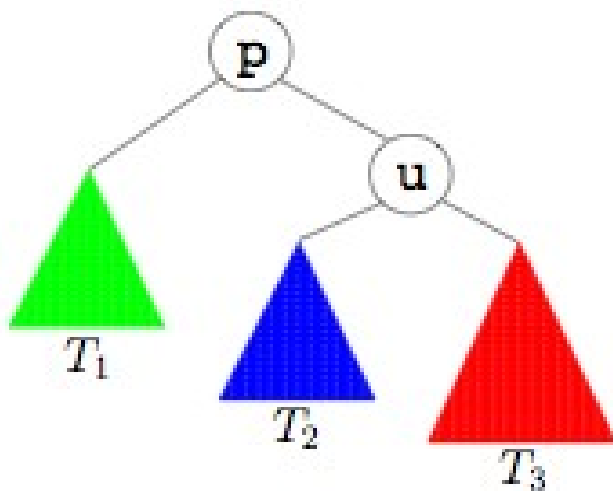
Árvore AVL

Árvores AVL

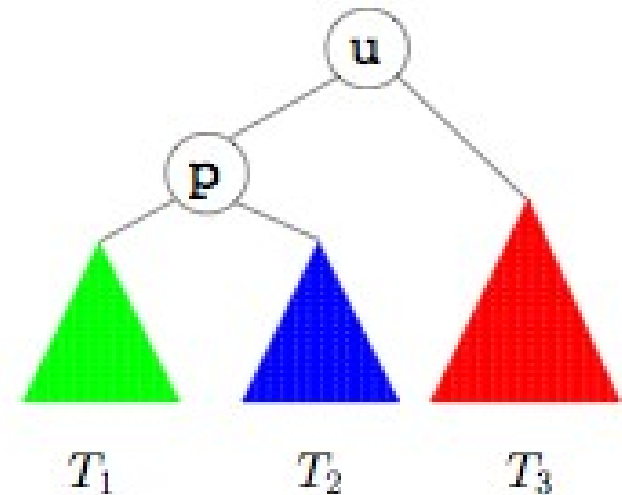
- Quando aplicar a rotação à esquerda?

$$h_D(p) > h_E(p)$$

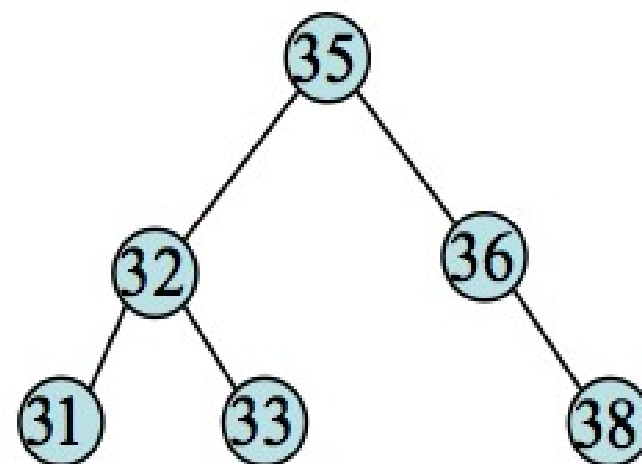
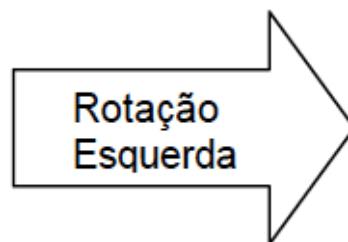
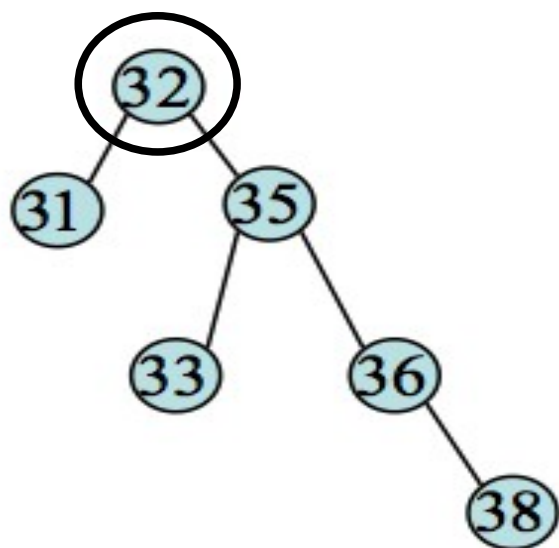
$$h_D(u) > h_E(u)$$



Rotação
Esquerda



Exemplo



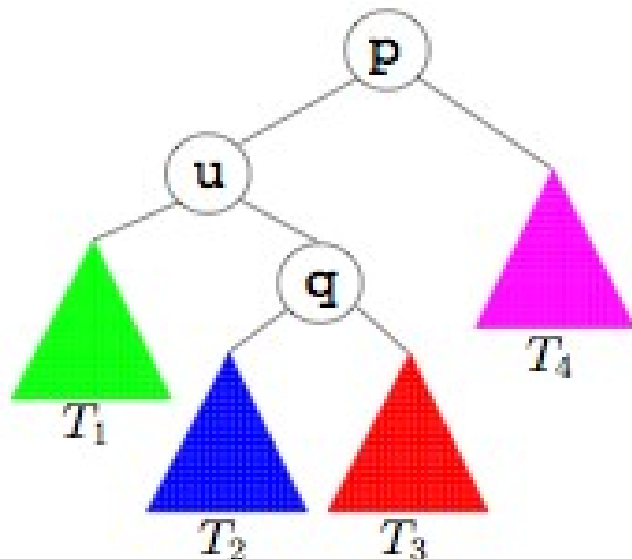
O nó 38 foi inserido na árvore,
causando desbalanceamento no nó 32

Árvore AVL

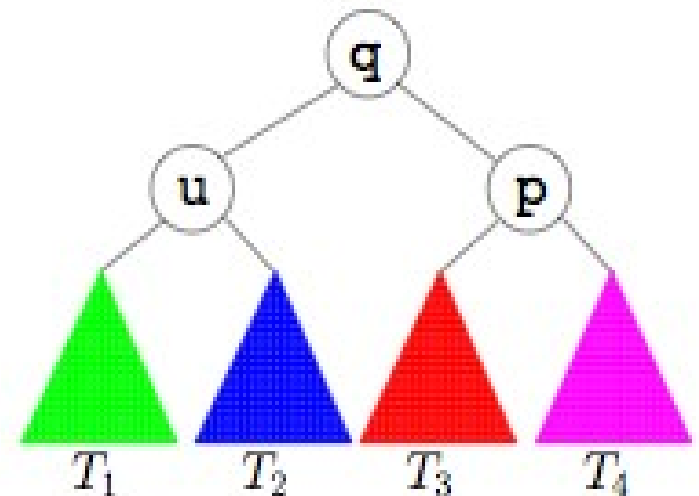
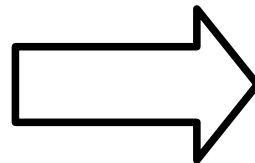
Árvores AVL

- Quando aplicar a rotação dupla à direita?

$$h_E(p) > h_D(p)$$
$$h_D(u) > h_E(u)$$

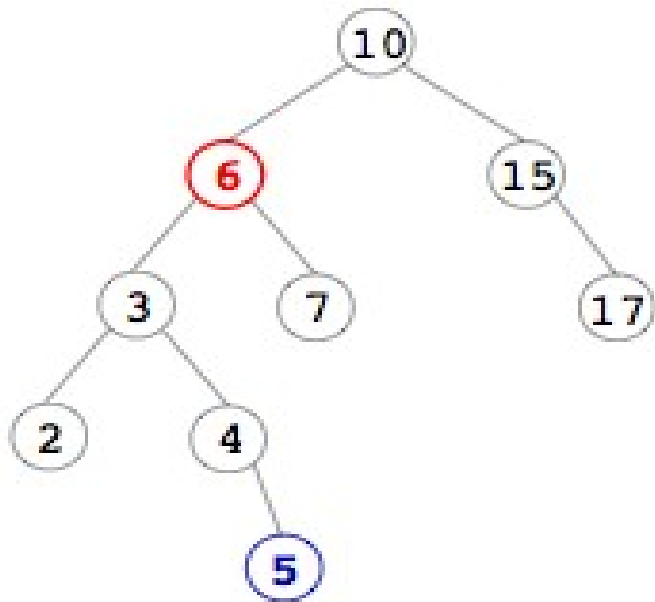


Rotação dupla
à direita

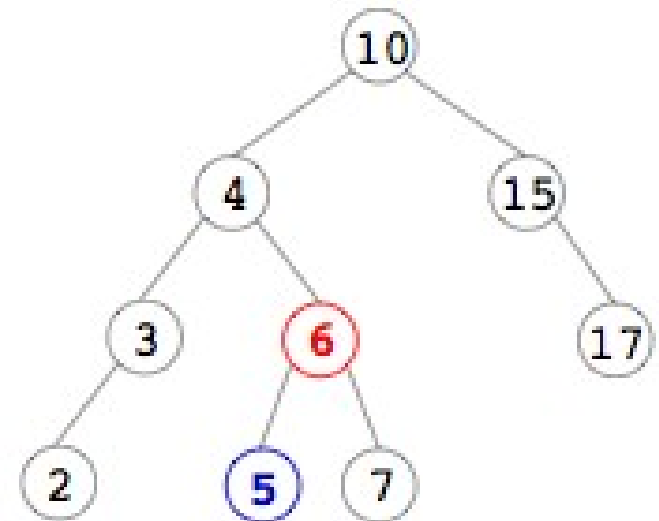
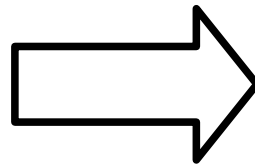


Exemplo

- Ao inserirmos o nó 5 na árvore, o nó 6 ficou desregulado.
- Então, primeiro se faz uma rotação à esquerda e depois uma rotação à direita tendo o nó 4 como pivô.



Rotação dupla
à direita

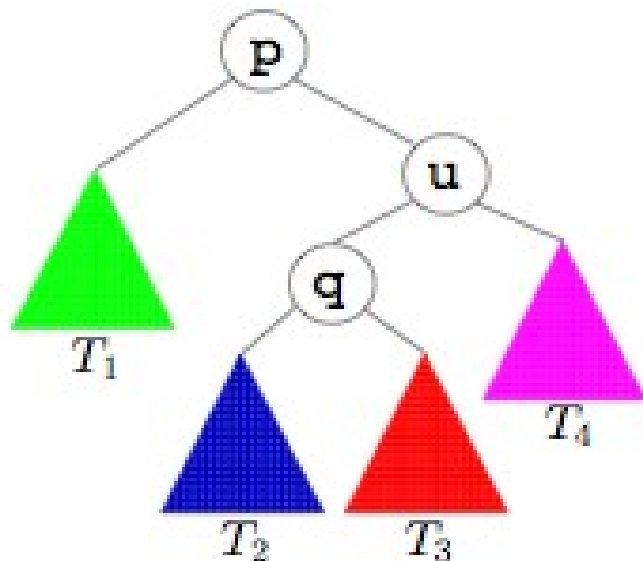


Árvores AVL

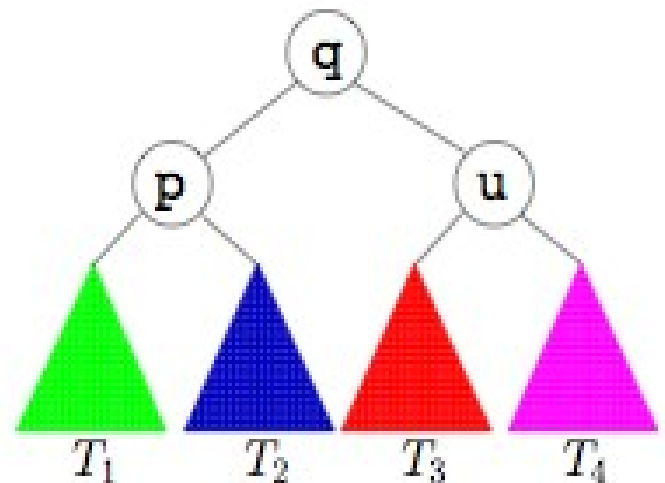
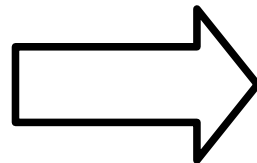
- Quando aplicar a rotação dupla à esquerda?

$$h_D(p) > h_E(p)$$

$$h_E(u) > h_D(u)$$

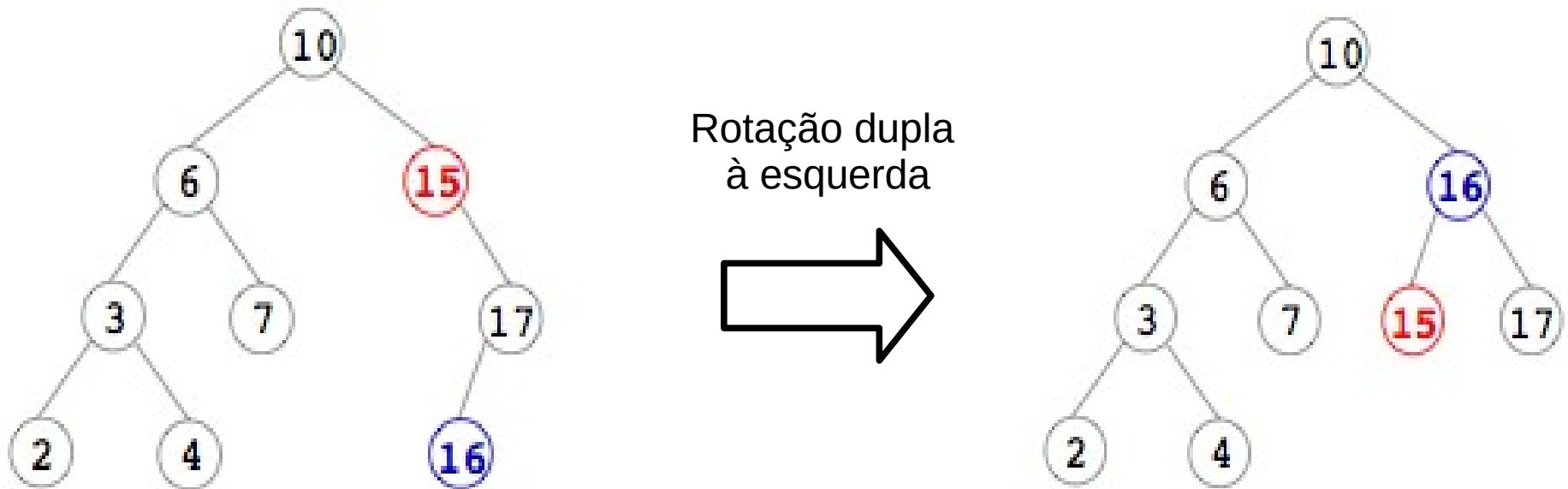


Rotação dupla
à esquerda



Exemplo

- Ao inserirmos o nó 16 na árvore, o nó 15 ficou desregulado.
- Então, primeiro se faz uma rotação à direita e depois uma rotação à esquerda tendo o nó 16 como pivô.



Árvores AVL

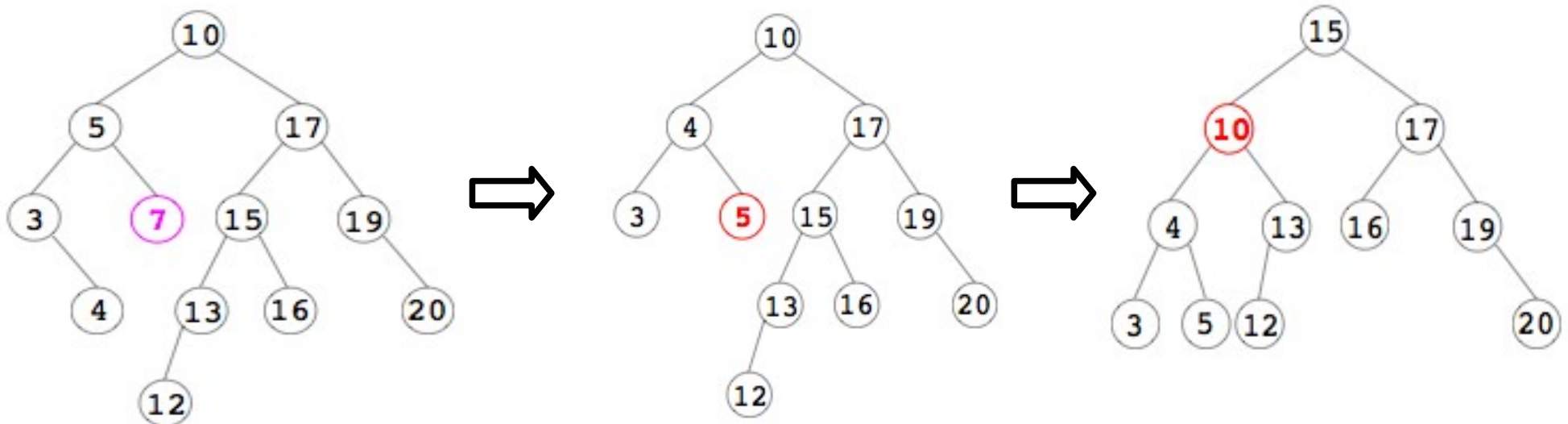
- Detalhes de implementação para inserção:
 1. Realizar a operação de inserção como em uma árvore de busca binária comum: $O(\log n)$
 2. Verificar se existem nós desregulados. Para isso, deve-se atualizar o FB dos ancestrais do nó inserido: $O(\log n)$
 3. Se existir um nó desregulado, torná-lo regulado, aplicando a rotação necessária: $O(1)$ (um simples ajuste de ponteiros)
- Complexidade temporal total no pior caso: $O(\log n)$

Árvores AVL

- Detalhes de implementação para remoção:
 1. Realizar a operação de remoção como em uma árvore de busca binária comum: $O(\log n)$
 2. Verificar se existem nós desregulados. Para isso, deve-se atualizar o FB dos ancestrais do nó removido: $O(\log n)$
 3. Percorrer o caminho desde o pai do nó removido até a raiz, fazendo as operações de rotação apropriadas (pode ser mais de uma): $O(\log n)$
- Obs: Se outro nó ocupar o lugar do nó removido, a análise deve começar no (antigo) pai desse nó.
- Complexidade temporal total no pior caso: $O(\log n)$

Exemplo

- Remova o nó com chave 7 da árvore AVL abaixo. Em seguida, aplique as rotações necessárias para garantir que a árvore resultante mantenha sua propriedade AVL.



Exercício

- Construir uma AVL com as chaves:

(10, 20, 30, 5, 3, 50, 40, 70, 60, 90)

- Depois de construída a árvore AVL acima, remova as chaves:

(20, 60, 90)

e mantenha o balanceamento da árvore.

Árvores Vermelho-Preto

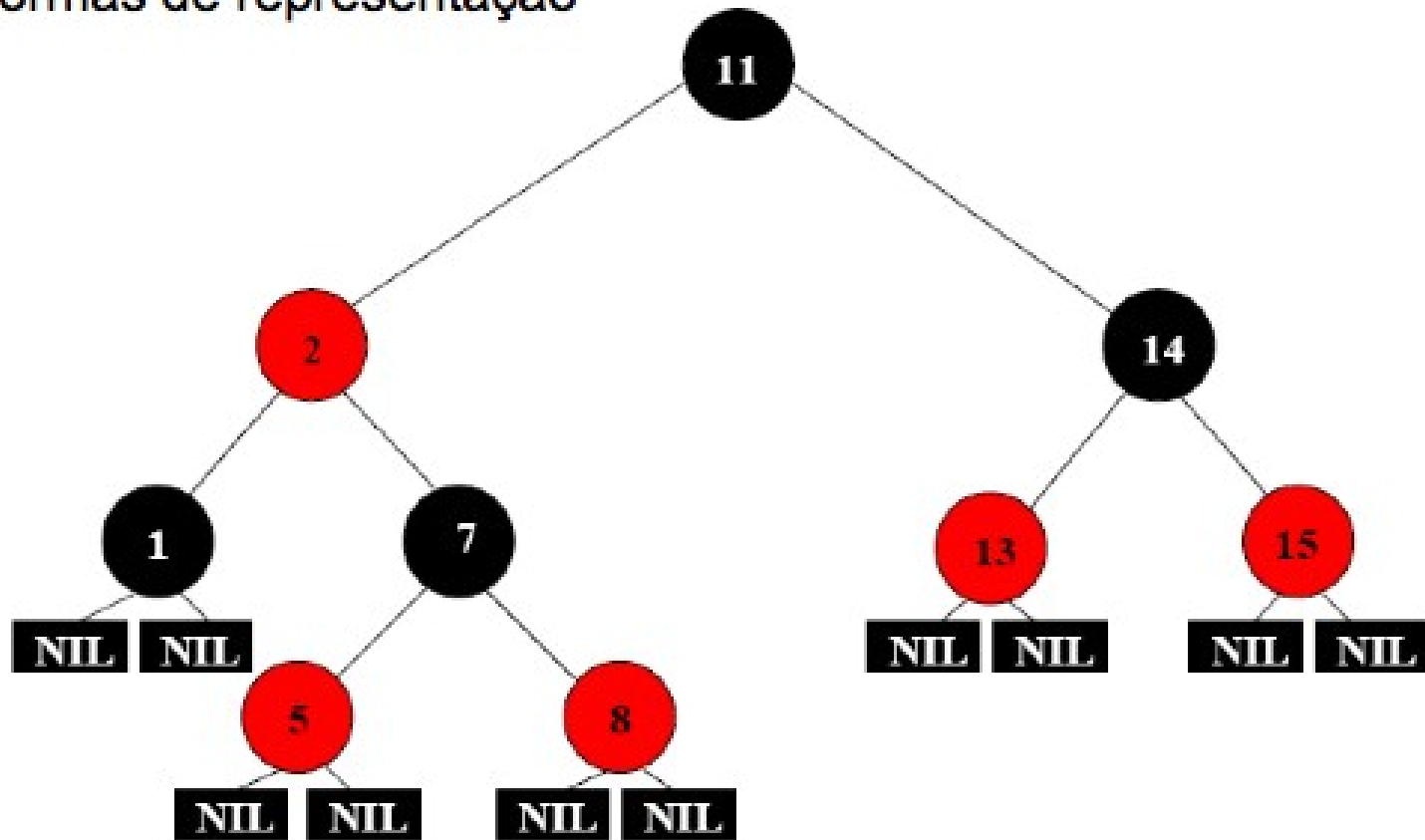
- As árvores vermelho-preto (ou rubro-negras) constituem um entre muitos esquemas de árvores de busca binária que são “balanceadas”.
- Uma árvore rubro-negra é uma árvore de busca binária com um *bit* extra de armazenamento por nó que indica sua cor: **PRETA** ou **VERMELHA**.
- Essa estrutura de dados é complexa, mas eficiente na prática, ao garantir que suas operações demorem $O(\log n)$ no pior caso.
- Foram inventadas por Bayer sob o nome “Árvores Binárias Simétricas” em 1972, 10 anos depois das árvores AVL.

Árvores Vermelho-Preto

- Uma árvore de busca binária é uma árvore rubro-negra se ela satisfaz as seguintes propriedades:
 1. Todo nó é vermelho ou preto;
 2. A raiz é preta;
 3. Todo nó externo (NIL) é preto;
 4. Se um nó é vermelho, então ambos os seus filhos são pretos;
 5. Todos os caminhos desde um nó até os nós externos descendentes contêm o mesmo número de nós pretos.
- Um nó que satisfaz as propriedades é denominado equilibrado, caso contrário é dito desequilibrado. Na árvore rubro-negra todos os nós estão equilibrados.
- Em um caminho da raiz até uma sub-árvore vazia não pode existir dois nós vermelhos consecutivos.

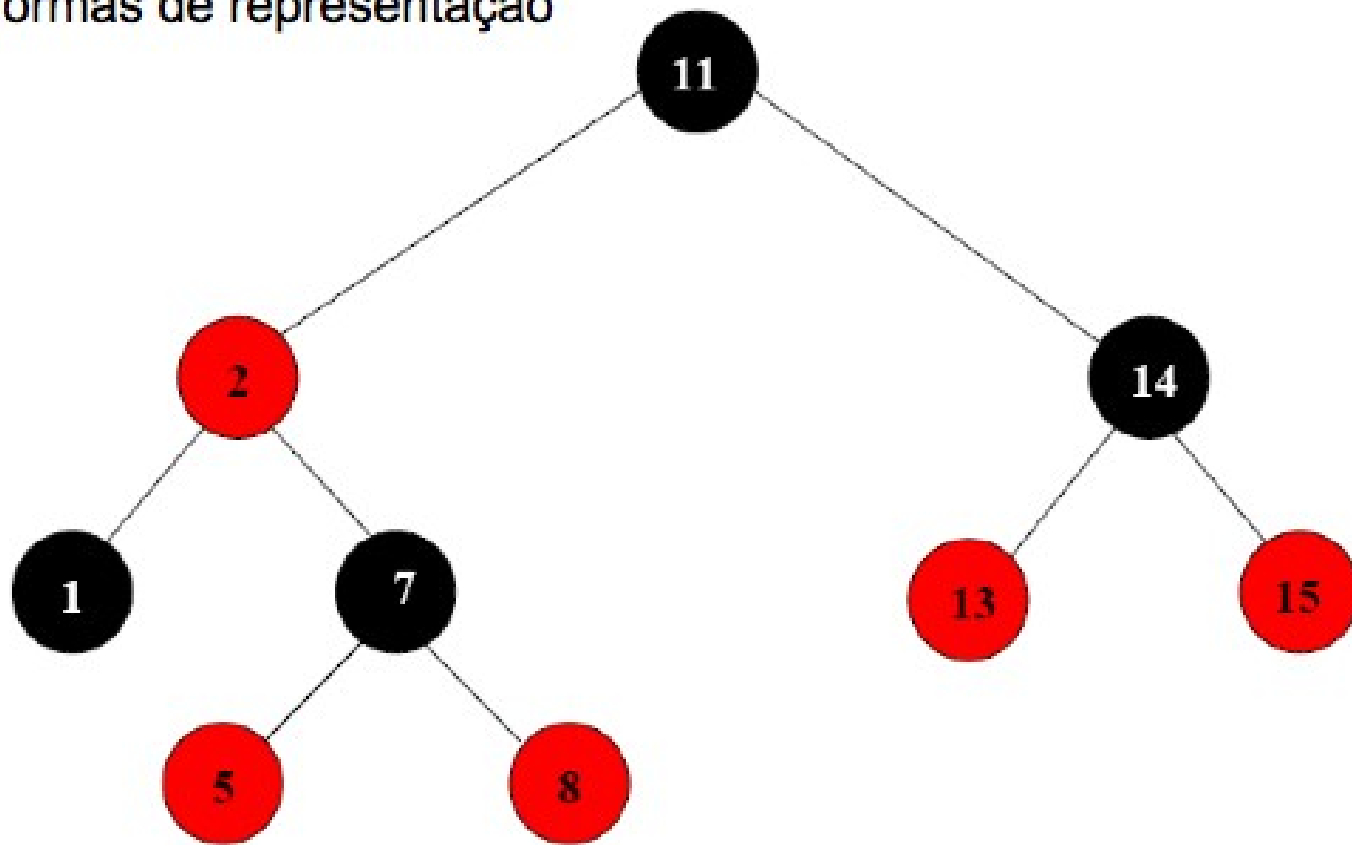
Árvores Vermelho-Preto

Formas de representação



Árvores Vermelho-Preto

Formas de representação



Árvores Vermelho-Preto

- Lema:

"Uma árvore vermelho-preto com n nós internos tem altura no máximo $2 \log (n + 1)$."

A prova do Lema pode ser conferida na página 222 do livro texto.

- O lema mostra que as árvores vermelho-preto constituem "boas" árvores de busca, visto que sua altura é $O(\log n)$.

Árvores Vermelho-Preto

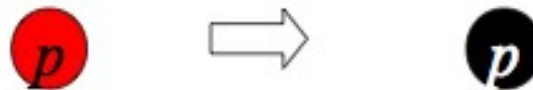
- As operações **Inserir** e **Remover** são mais complicadas nas árvores rubro-negras porque elas podem ferir alguma propriedade desse tipo de árvore.
- Como veremos, essas operações podem ser implementadas de forma bastante parecida com as respectivas operações nas árvores binárias de busca, bastando apenas modificar as cores dos nós e trocar ponteiros para que as propriedades das árvores rubro-negras sejam satisfeitas.
- Como a inserção e remoção propriamente ditas já foram vistas para árvores binárias de busca, veremos apenas o que é necessário para acertar as cores da árvore.

Árvores Vermelho-Preto

- Um nó é inserido sempre na cor **vermelha**.

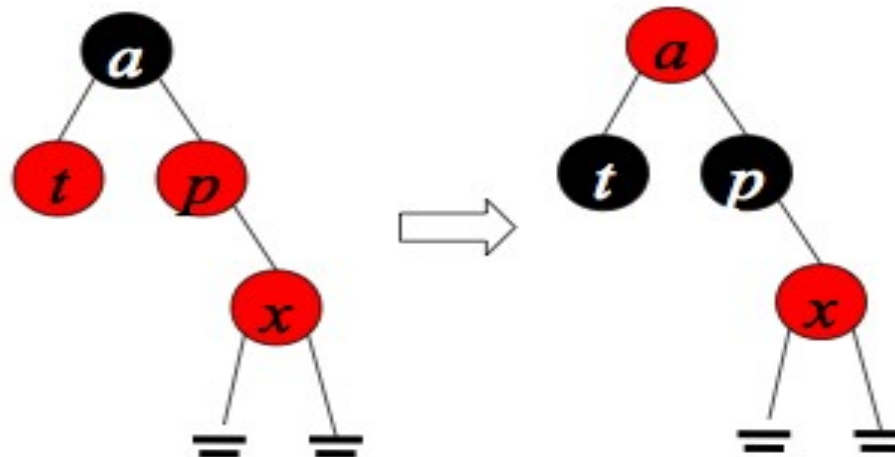


- Caso 1:** Caso a inserção seja feita em uma árvore vazia, basta alterar a cor do nó para preto, para manter a propriedade 2.



Árvores Vermelho-Preto

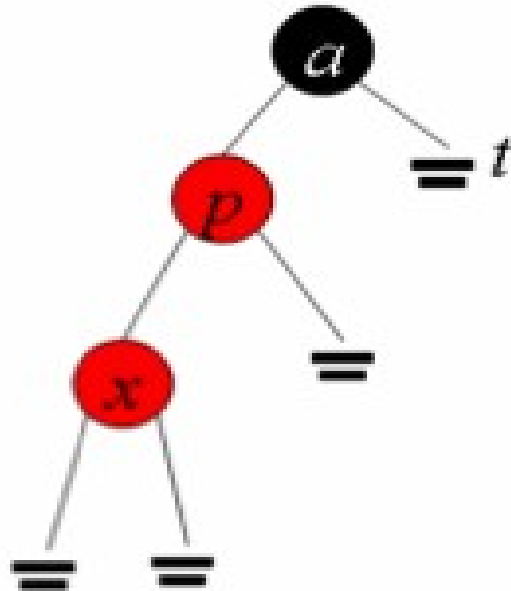
- **Caso 2:** Ao inserir x , se o tio de x é vermelho, é necessário fazer a recoloração de a , t e p .



- Se o pai do nó a é vermelho, o rebalanceamento tem que ser feito novamente, considerando o nó a como inserido.
- Se o nó a é raiz, então ele deve ser preto.

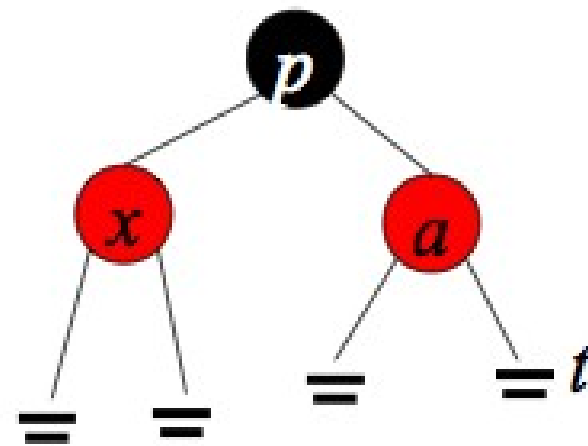
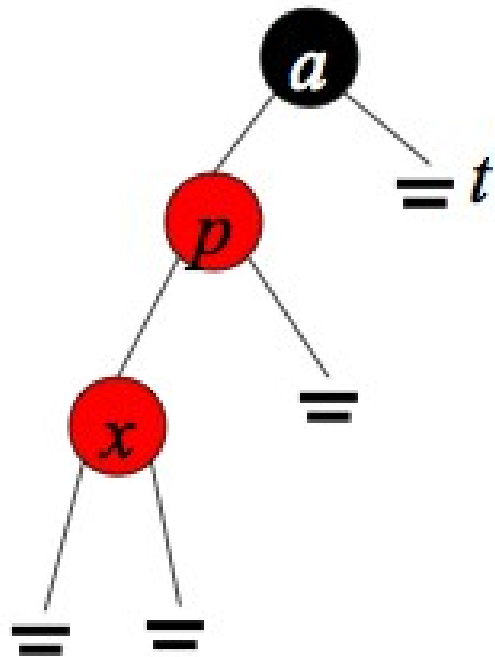
Árvores Vermelho-Preto

- **Caso 3:** Suponha que o tio do elemento inserido x seja preto. Nesse caso, para manter a propriedade 4 é preciso fazer rotações envolvendo a , t , p e x .
- Há 4 subcasos que correspondem às 4 rotações possíveis.



Árvores Vermelho-Preto

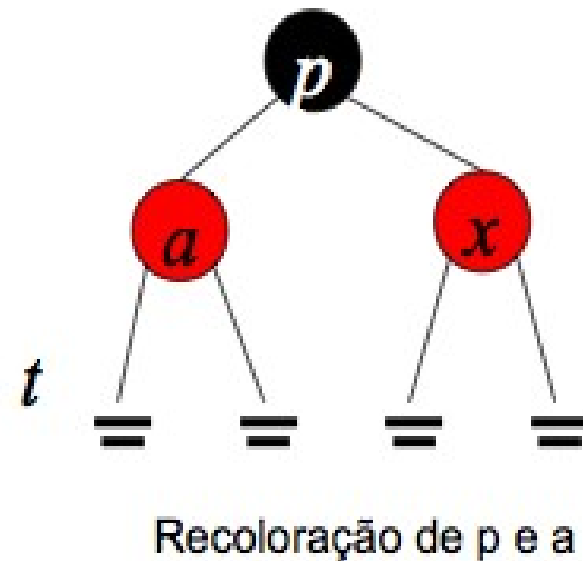
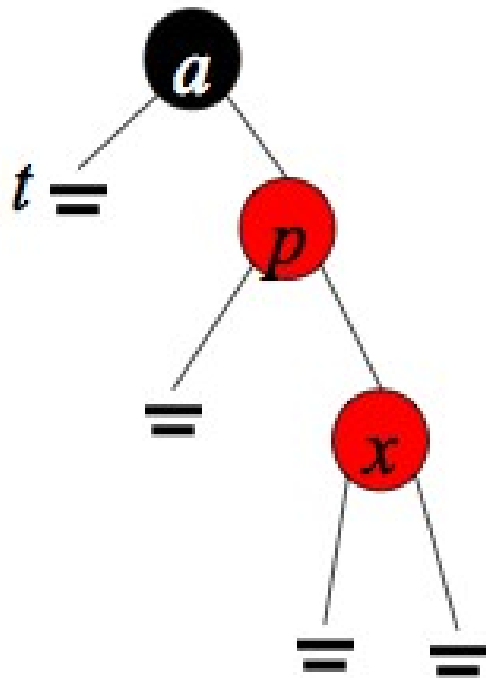
- **Caso 3a:** Rotação à direita.



Recoloração de p e a

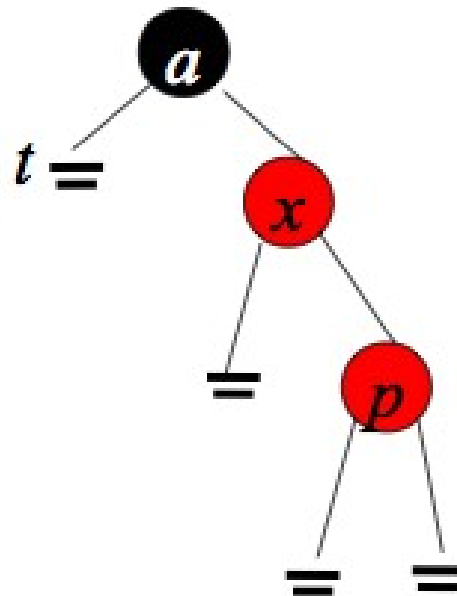
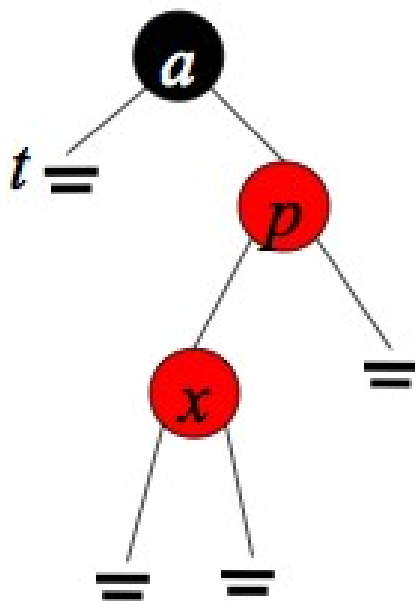
Árvores Vermelho-Preto

- **Caso 3b:** Rotação à esquerda.

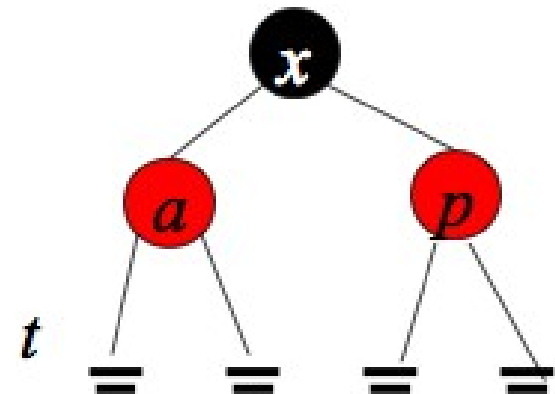


Árvores Vermelho-Preto

- **Caso 3c:** Rotação dupla à esquerda.



Rotação simples à
direita

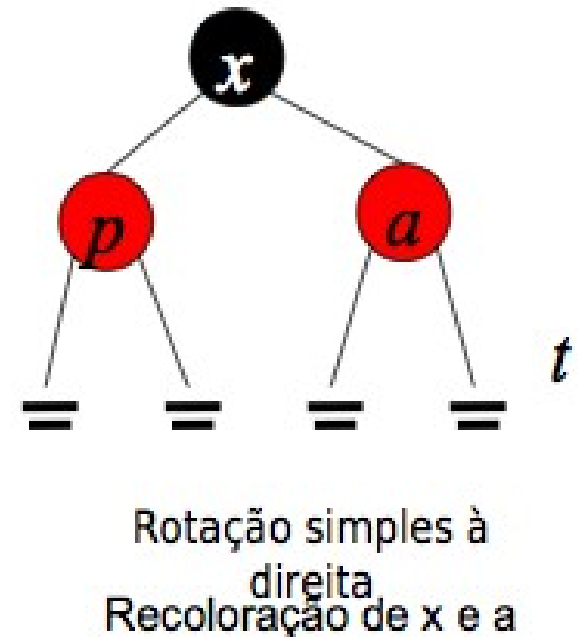
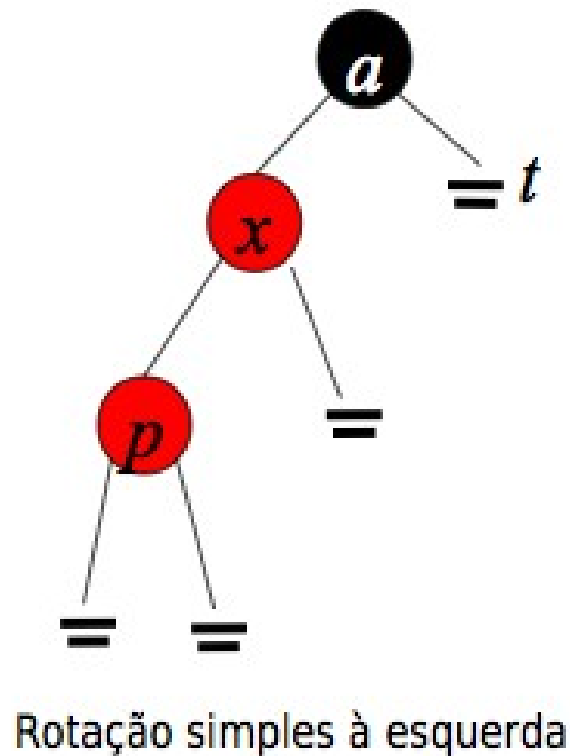
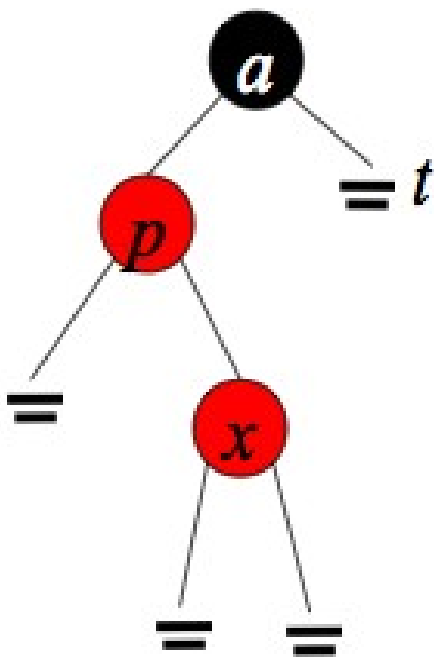


Rotação simples à esquerda

Recoloração de x e a

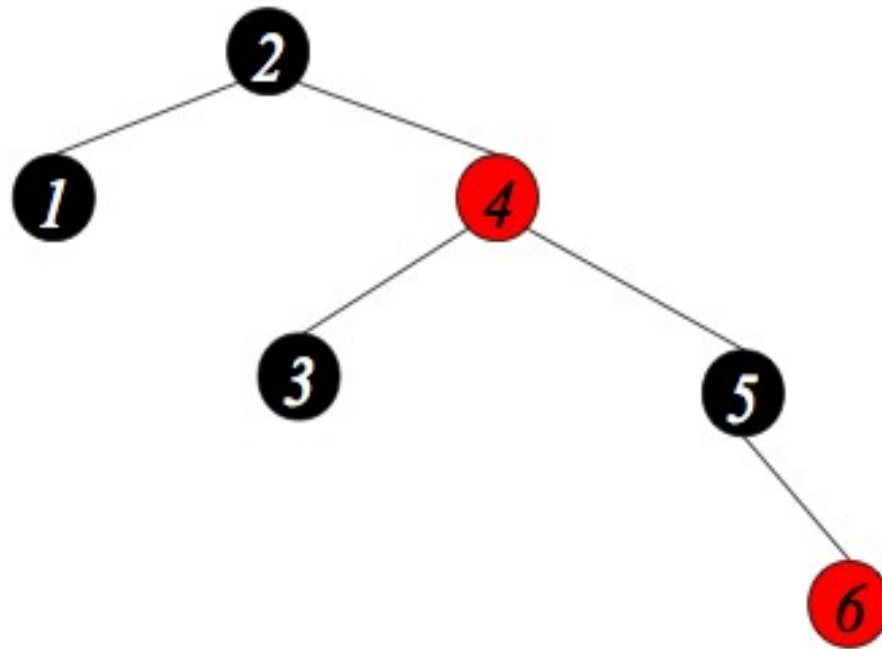
Árvores Vermelho-Preto

- **Caso 3d:** Rotação dupla à direita.



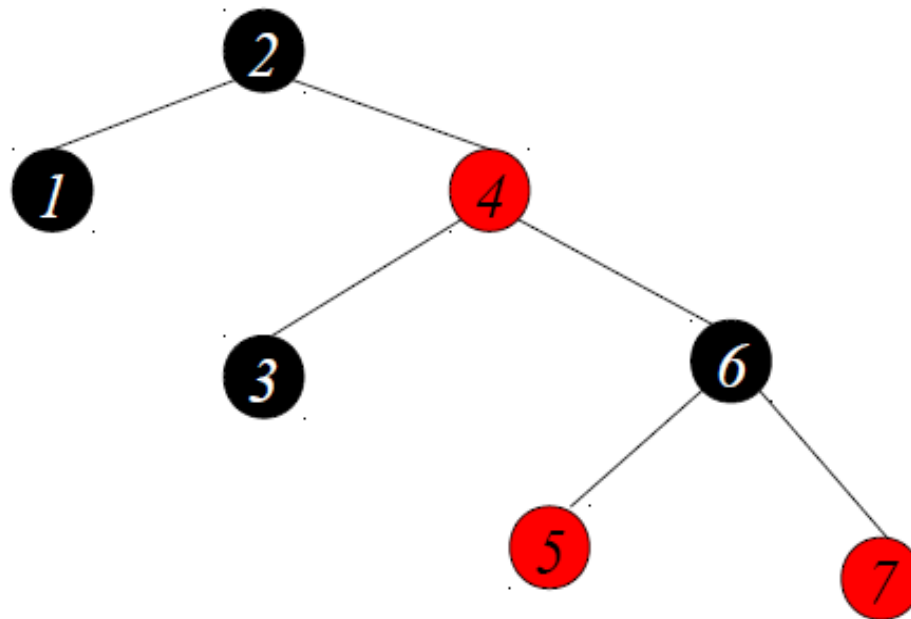
Exercício

- Insira um nó com chave 7 na árvore rubro-negra abaixo e trabalhe para manter seu balanceamento.



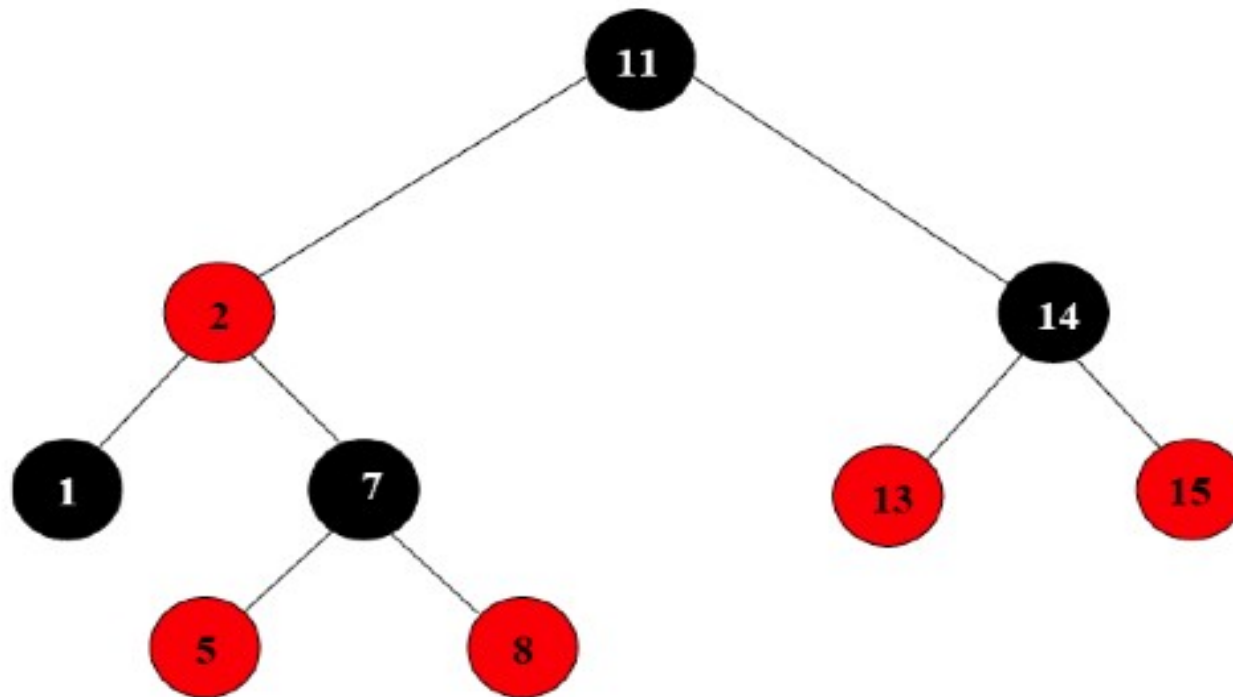
Solução

- Aplicar o **Caso 3b**:
 1. Rotação à esquerda dos nós 5, 6 e 7.
 2. Alteração da cor dos nós 5 e 6.



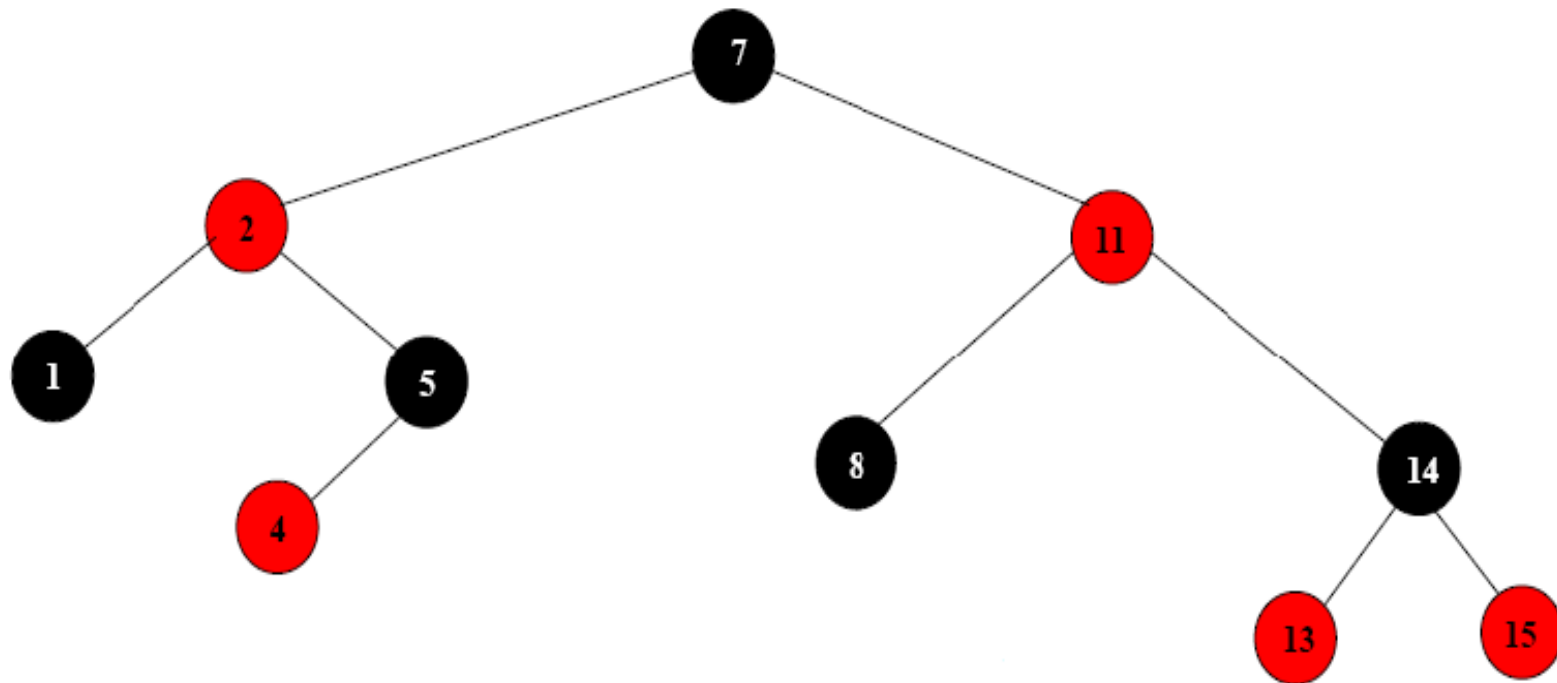
Exercício

- Insira um nó com chave 4 na árvore rubro-negra abaixo e trabalhe para manter seu balanceamento.



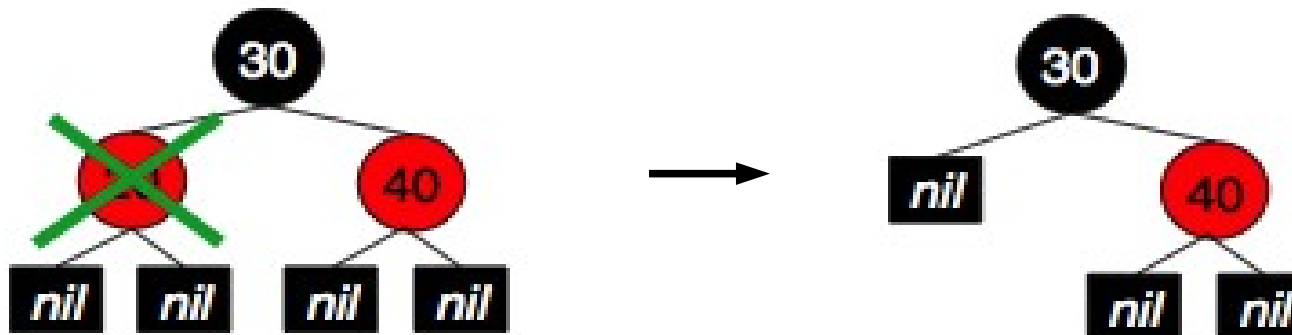
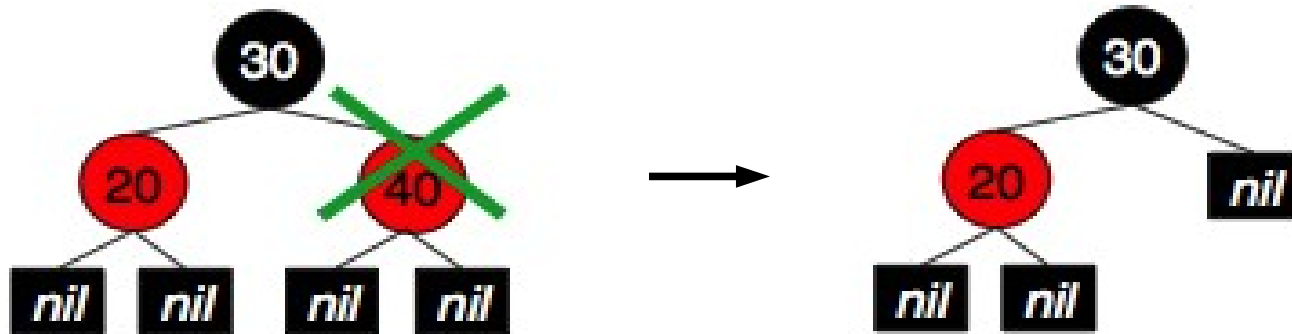
Solução

- Alteração de cor dos nós 5, 7 e 8 → **Caso 2**
- Rotação dupla à direita dos nós 2, 7 e 11 → **Caso 3d**



Árvores Vermelho-Preto

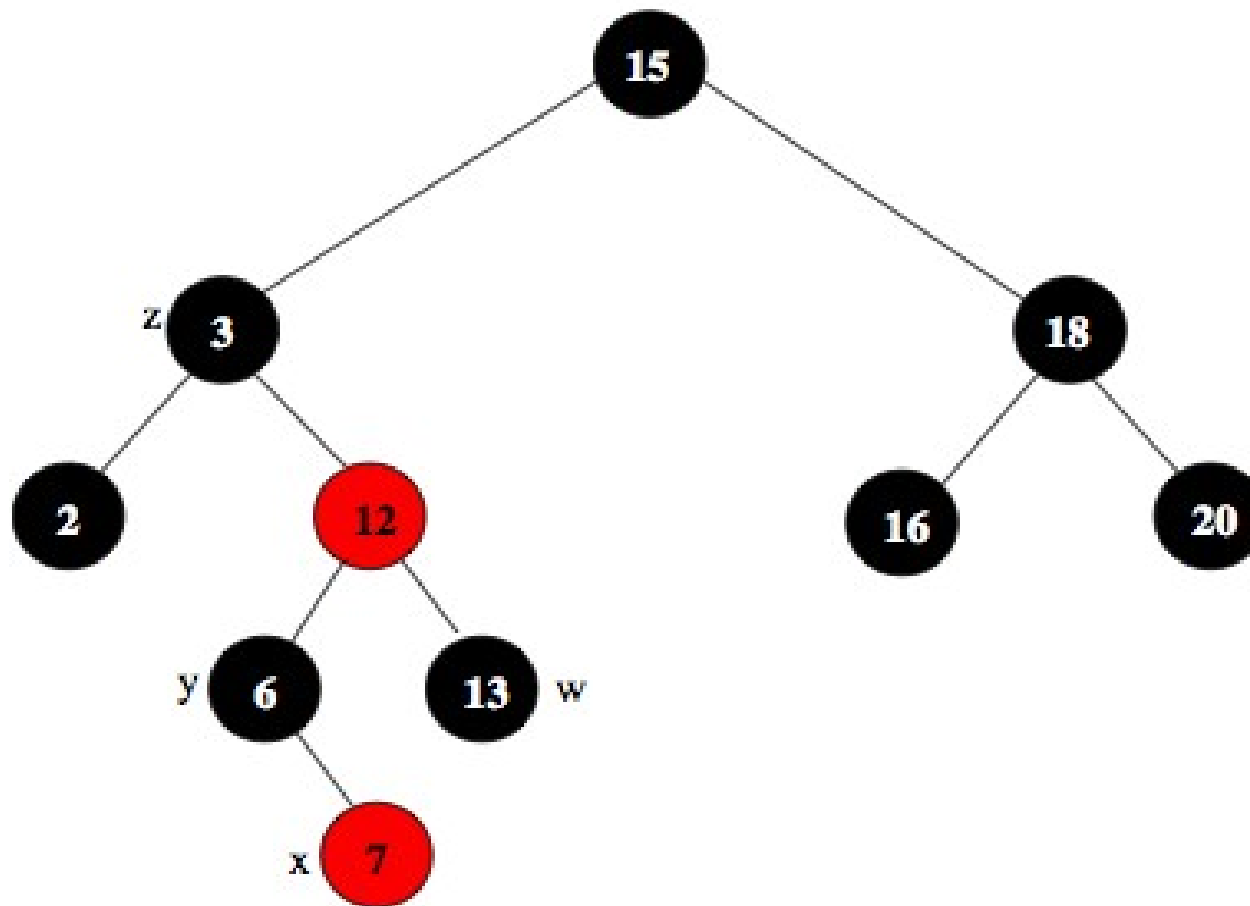
- A remoção de um nó vermelho **não** altera o balanceamento da árvore rubro-negra.



Árvores Vermelho-Preto

- Existem casos para corrigir as cores **após** a remoção de um nó preto, mas antes de defini-los, identificaremos alguns nós:
 - Seja z o nó a ser removido.
 - Seja $y = z$, se z possui um ou nenhum filho, ou $y = \text{sucessor}(z)$, se z possui dois filhos.
 - Seja x o filho de y antes da remoção de z , ou NIL, caso y não possua filho.
 - Seja w o tio de x antes da remoção de z .

Árvores Vermelho-Preto



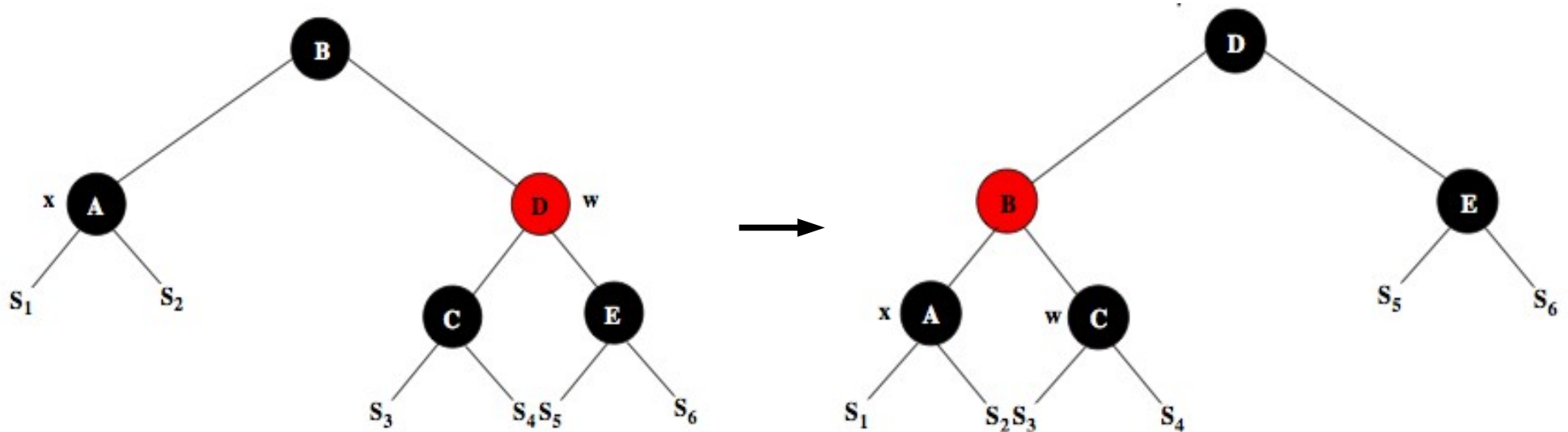
Árvores Vermelho-Preto

1. Caso o nó z tenha dois filhos e o nó y seja vermelho, basta realizar o procedimento de remoção do nó z e depois atribuir a cor preta ao nó y que a árvore estará balanceada.
2. Se após a remoção do nó z o nó x for a raiz da árvore ou da cor vermelha, então basta alterar sua cor para preto que a árvore estará balanceada. Por exemplo, na remoção do nó com chave 3 da árvore do slide anterior.
3. Mas, enquanto o nó x for diferente da raiz da árvore e sua cor for preta, 4 (quatro) casos serão repetidos no intuito de restaurar as propriedades vermelho-preto da árvore de busca.

Quando o critério de parada (descrito no passo 2) for satisfeito, a cor preta é atribuída ao nó x e a árvore estará balanceada.

Árvores Vermelho-Preto

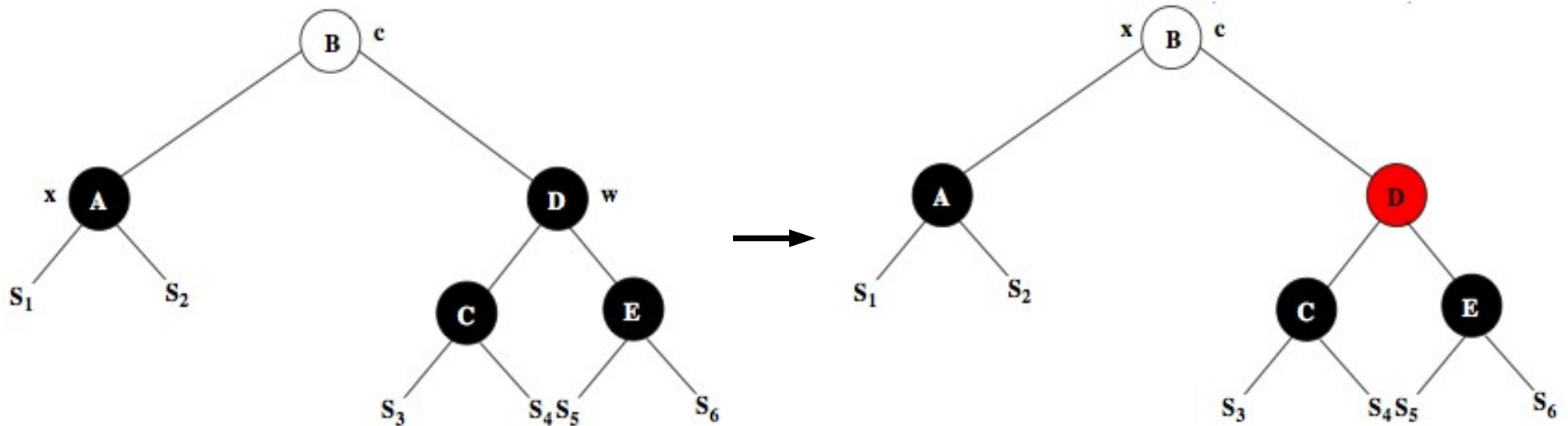
- **Caso 1:** O irmão w de x é vermelho.



- O caso 1 é transformado no caso 2, 3 ou 4 pela troca de cores dos nós B e D e pela execução de uma rotação à esquerda.

Árvores Vermelho-Preto

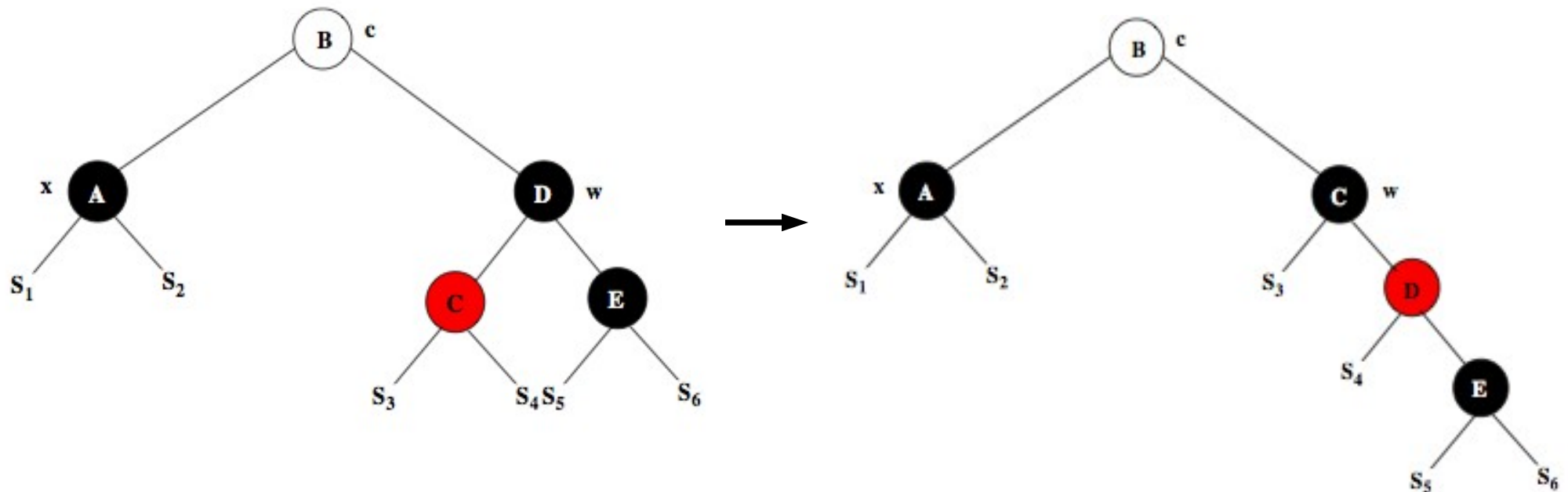
- **Caso 2:** O nó w é preto, e ambos os filhos de w são pretos.



- Se entrarmos no caso 2 vindo do caso 1, o critério de parada (o nó x será vermelho) será atendido após a transformação.

Árvores Vermelho-Preto

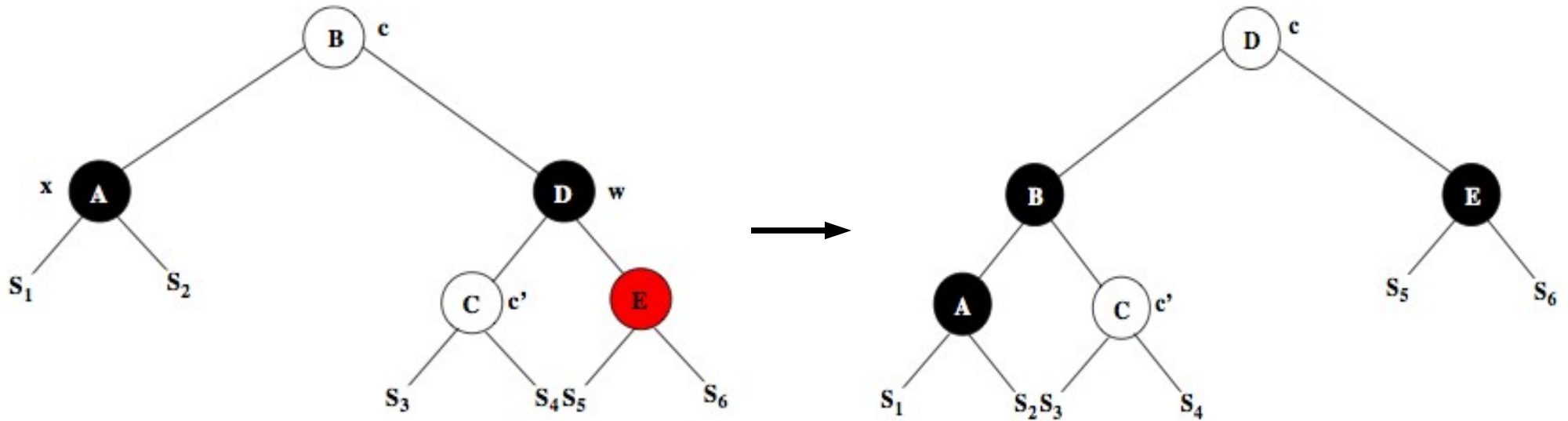
- **Caso 3:** O nó w é preto, e o filho da esquerda de w é vermelho e o filho da direita de w é preto.



- O caso 3 é transformado no caso 4 pela troca de cores dos nós C e D e pela execução de uma rotação à direita.

Árvores Vermelho-Preto

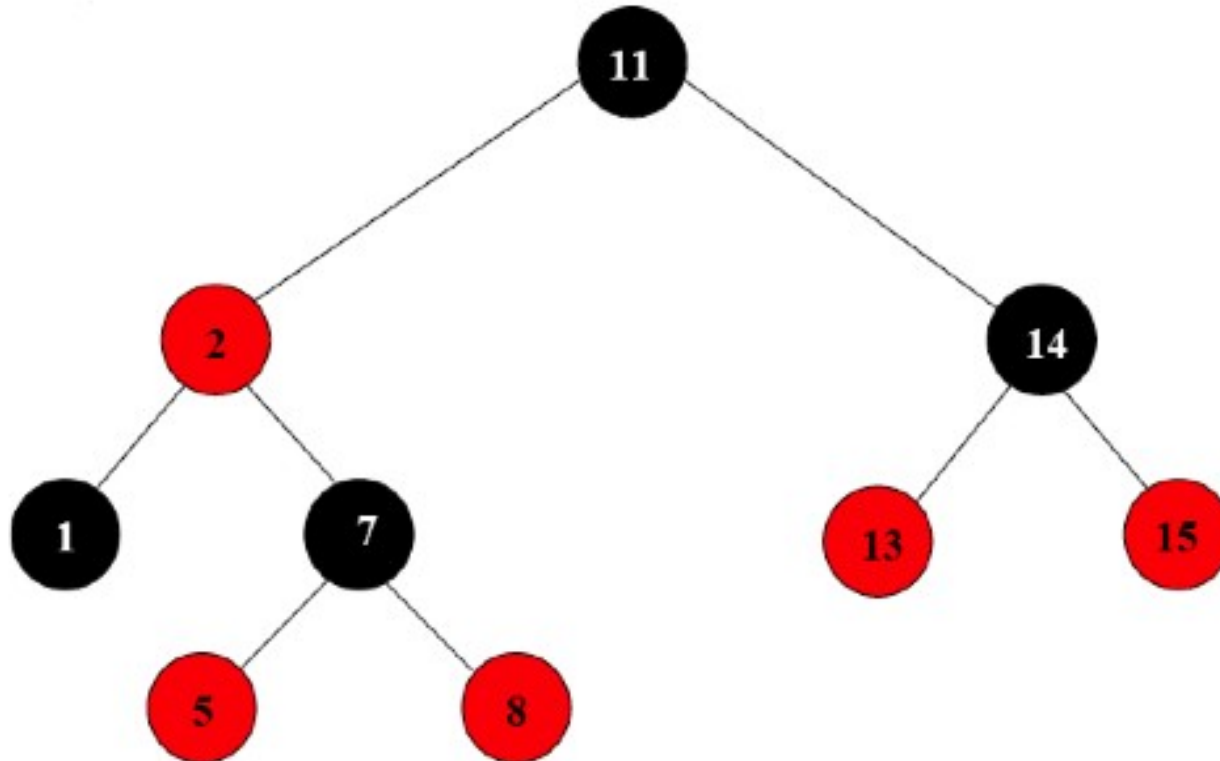
- **Caso 4:** O nó w é preto, e o filho da direita de w é vermelho.



- O critério de parada será atendido após a transformação, já que o novo x será a raiz da árvore.

Exercício

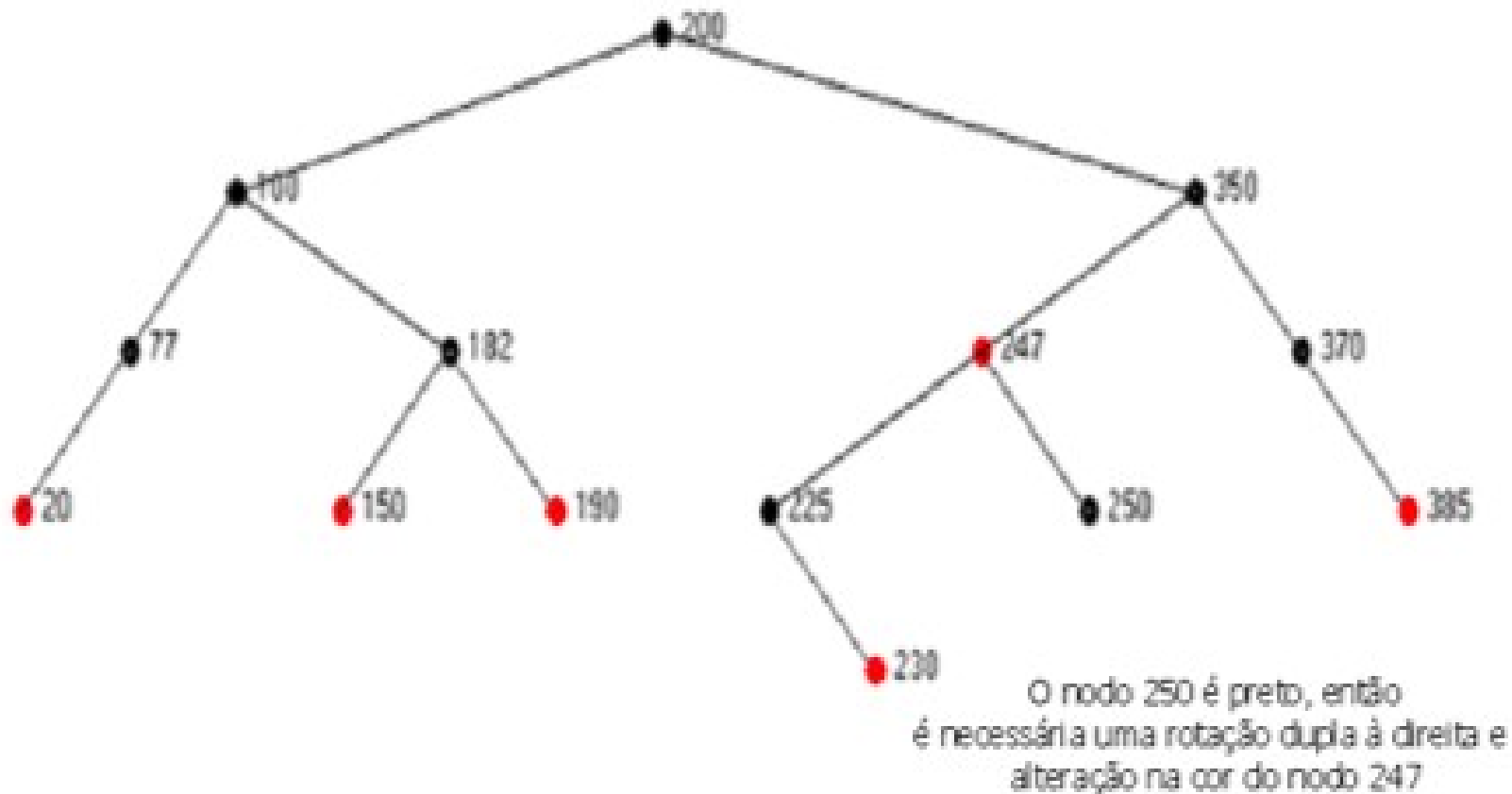
1. Remova o nó com chave 1 da árvore rubro-negra abaixo e trabalhe para manter seu balanceamento.



Rotação a esquerda e alteração na cor dos nós 2, 7 e 8

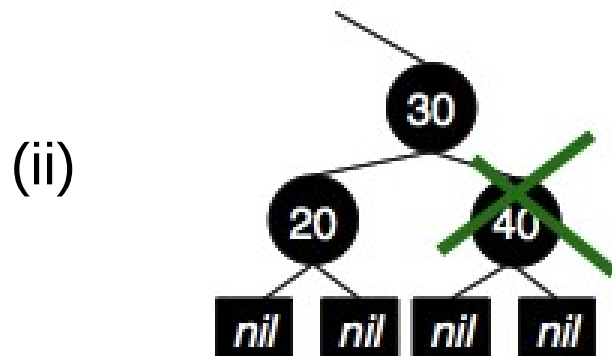
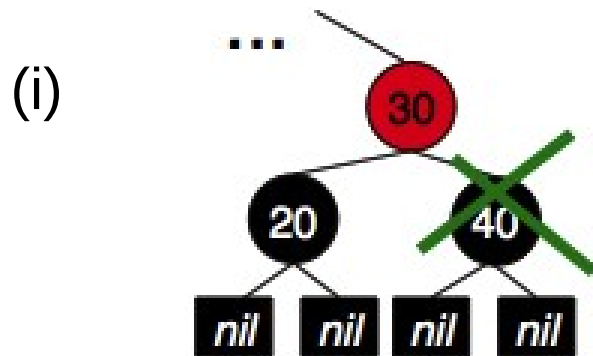
Exercício

2. Remova o nó com chave 250 da árvore rubro-negra abaixo e trabalhe para manter seu balanceamento.



Exercício

3. Explique o procedimento realizado para a remoção da chave 40 nas situações (i) e (ii) abaixo.



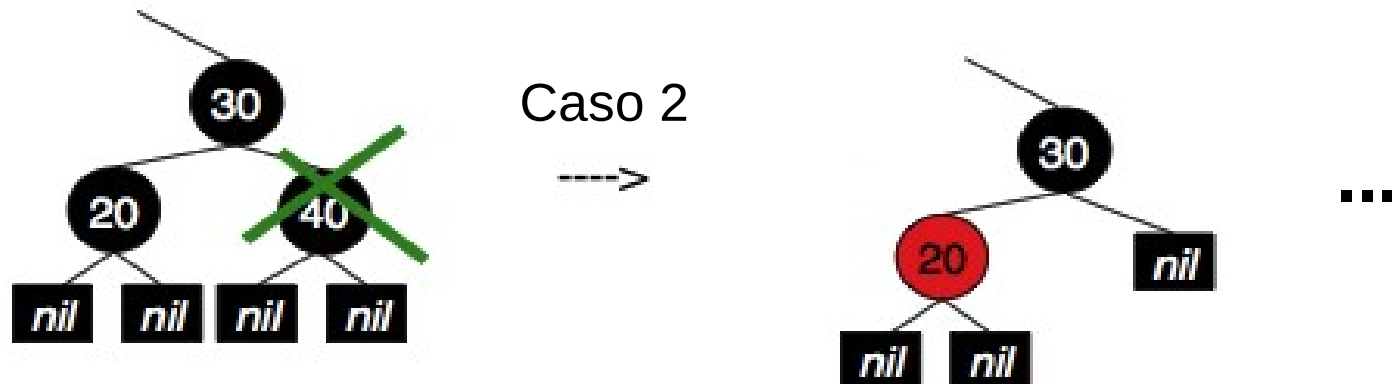
Exercício

3. Explique o procedimento realizado para a remoção da chave 40 nas situações (i) e (ii) abaixo.

(i)

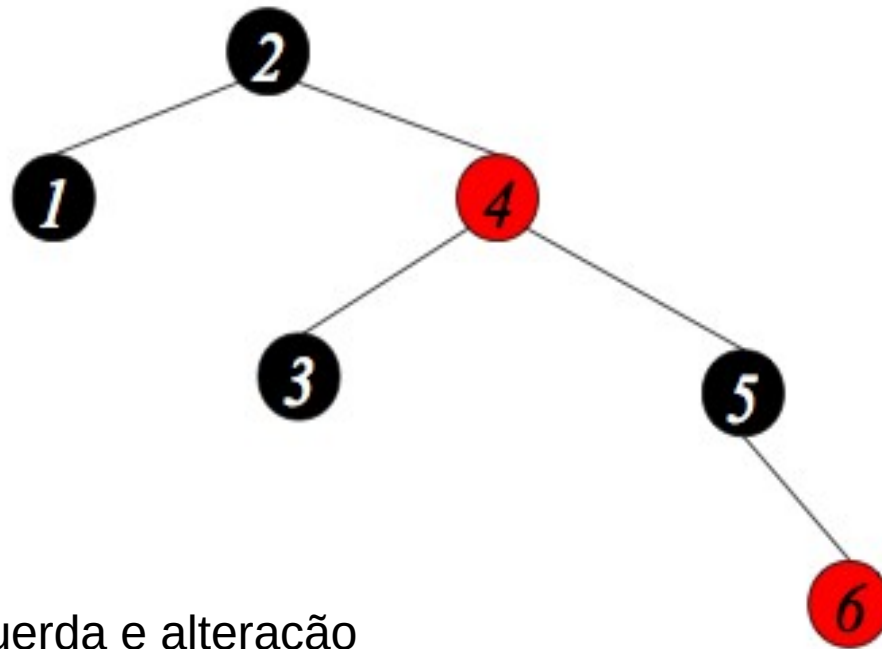


(ii)



Exercício

4. Remova o nó com chave 1 da árvore rubro-negra abaixo e trabalhe para manter seu balanceamento.



Rotação a esquerda e alteração
na cor dos nós 3 e 4.

Soluções

- **Exercício 1:** Aplicar o Caso 4.
- **Exercício 2:** Aplicar o Caso 3 e, em seguida, o Caso 4.
- **Exercício 3:**
 - (i) Aplicar o Caso 2. Como o novo x (ou seja, a chave 30) é vermelho, o critério de parada foi satisfeito. Logo, basta mudar a cor do novo x de vermelho para preto.
 - (ii) Aplicar o Caso 2 e seguir com a análise.
- **Exercício 4:** Aplicar o Caso 1 e, em seguida, o Caso 2. Após aplicar o Caso 2, o novo x (ou seja, a chave 2) é vermelho, logo, a cor do novo x muda de vermelho para preto.

Árvores B

- Em muitas aplicações a tabela empregada é muito grande, de forma que o armazenamento do conjunto de chaves não pode ser efetuado na memória principal.
- Nesse caso, torna-se necessária a manutenção da tabela em memória secundária, o que acarreta um desperdício de tempo significativo para acessar um dado da tabela.
- Para essas aplicações, é interessante criar uma estrutura que minimize o tempo de acesso. A ideia é que ao invés de buscar um dado de cada vez, consiga-se transferir, em cada acesso, uma quantidade maior de dados.

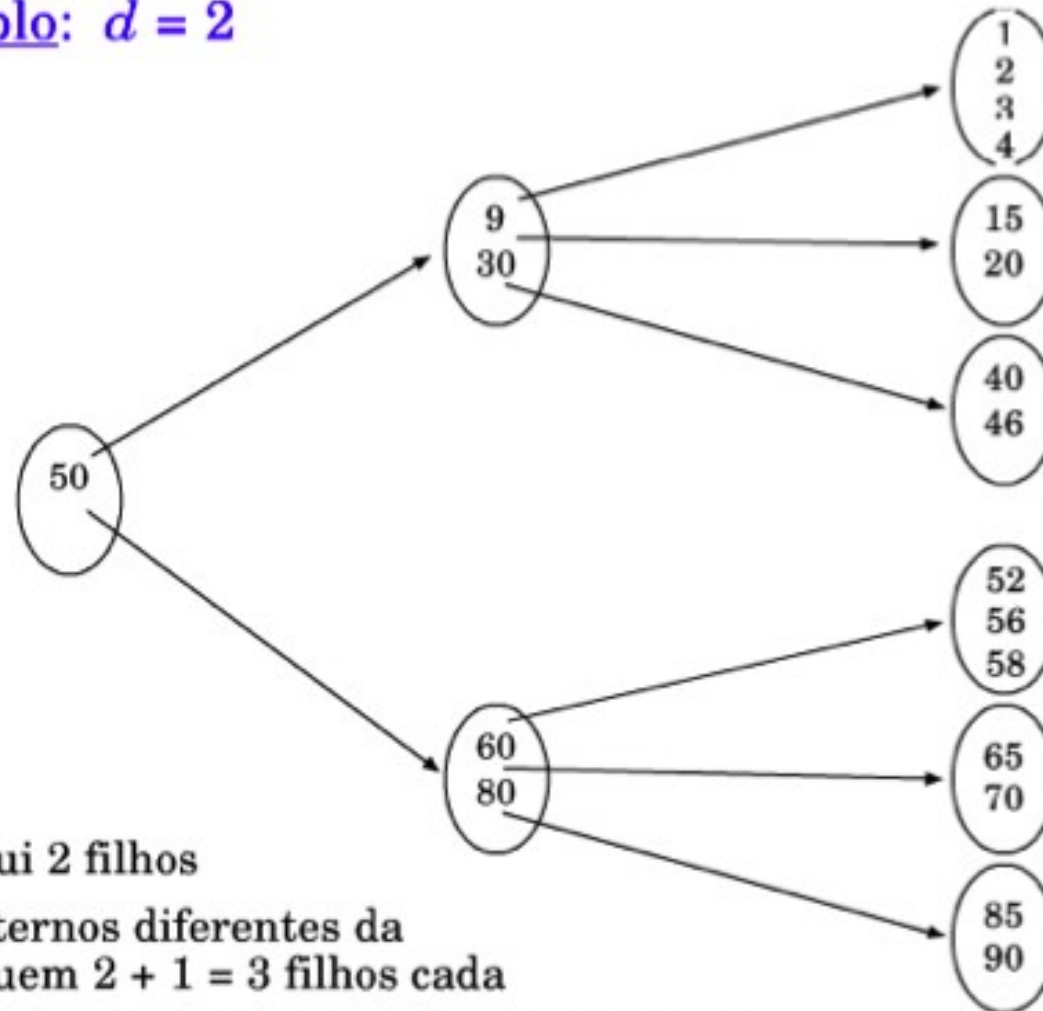
Árvores B

- As árvores B, utilizando o recurso de manter mais de uma chave em cada nó da estrutura, chamado aqui de **página**, proporcionam uma organização de ponteiros de tal forma que as operações de busca, inserção e remoção são executadas em tempo $O(\log n)$, onde n é o número de chaves na árvore.
- Sua construção assegura que as páginas-folha se encontram todas no mesmo nível, não importando a ordem de entrada dos dados.
- As árvores “tipo” B são largamente utilizadas como solução de armazenamento em memória secundária. Elas são empregadas em sistemas comerciais de BD, como Oracle, Sybase, entre outros.

Árvores B

- Seja d um número natural. Uma **árvore B de ordem d** é uma árvore ordenada que é vazia, ou que satisfaz as seguintes condições:
 - (i) a raiz é uma folha ou tem no mínimo dois filhos;
 - (ii) cada nó diferente da raiz e das folhas possui no mínimo $d + 1$ filhos;
 - (iii) cada nó diferente das folhas tem no máximo $2d + 1$ filhos;
 - (iv) todas as folhas estão no mesmo nível;
 - (v) o número mínimo (1 para raiz e d para as demais) e máximo ($2d$) de chaves por página devem ser respeitados.
- **Obs:** Uma árvore 2-3 é uma árvore B de ordem 1.

Exemplo: $d = 2$



Observe:

- 1) raiz possui 2 filhos
- 2) os nós internos diferentes da raiz possuem $2 + 1 = 3$ filhos cada
- 3) cada nó possui no máximo $2 \times 2 + 1 = 5$ filhos
- 4) todas as folhas estão no nível 2

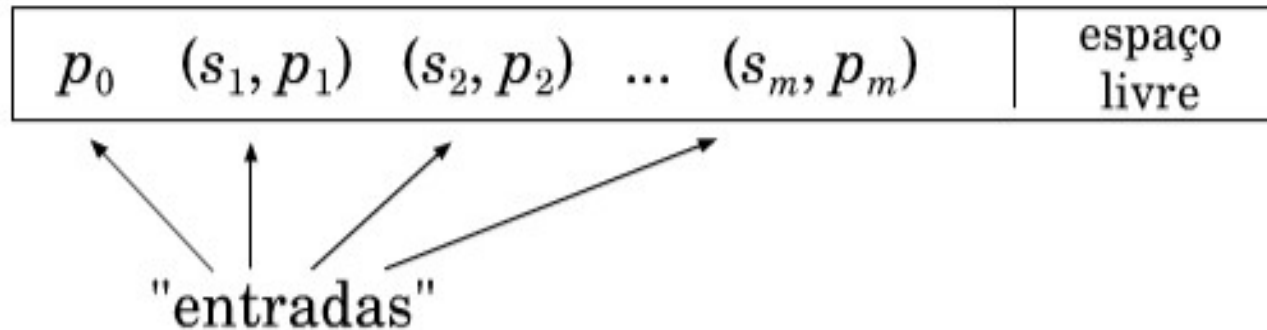
Árvores B

- Propriedades:

a) Seja m o número de chaves em uma página P não folha. Então, P tem $m + 1$ filhos.

b) Em cada página P , as chaves estão ordenadas: s_1, \dots, s_m , sendo $d \leq m \leq 2d$, exceto para a página raiz onde $1 \leq m \leq 2d$. Além disso, P contém $m + 1$ ponteiros p_0, p_1, \dots, p_m para os filhos de P .

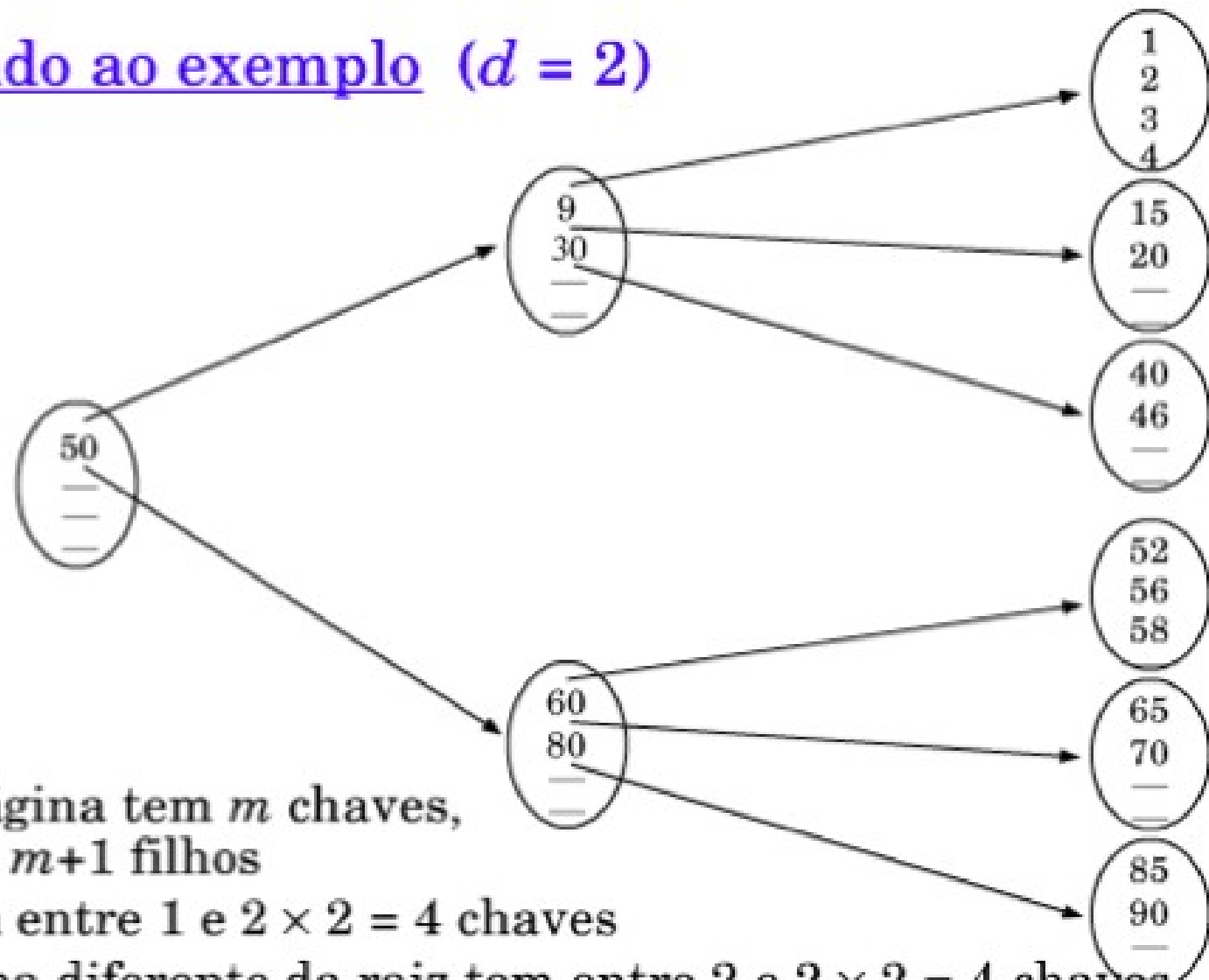
Árvores B



c) Seja uma página P com m chaves:

- Para qualquer chave y , pertencente à página apontada por p_0 , $y < s_1$
- Para qualquer chave y , pertencente à página apontada por p_k , $1 \leq k \leq m - 1$, $s_k < y < s_{k+1}$
- Para qualquer chave y , pertencente à página apontada por p_m , $y > s_m$

Voltando ao exemplo ($d = 2$)



Observe:

- a) se uma página tem m chaves, então tem $m+1$ filhos
- b) a raiz tem entre 1 e $2 \times 2 = 4$ chaves
- c) cada página diferente da raiz tem entre 2 e $2 \times 2 = 4$ chaves
- d) em cada página as chaves estão ordenadas
- e) de cada página com m chaves partem $m+1$ ponteiros (as folhas têm ponteiros nulos)

Exercícios

1. Seja uma árvore B de ordem d e altura h . Deduza o número máximo de páginas e chaves que essa estrutura pode conter.
2. Com base nos resultados da questão anterior, deduza cotas extremas para a altura h , em termos do número de chaves n .
3. Desenhe uma árvore B de ordem 3 que contenha as seguintes chaves:
1, 3, 6, 8, 14, 32, 36, 38, 39, 41, 43.

Soluções

- Exercício 1:

$$P_{max} = \frac{(2d+1)^h - 1}{2d}, \text{ para } h \geq 1$$

$$n_{max} = (2d + 1)^h - 1, \text{ para } h \geq 1$$

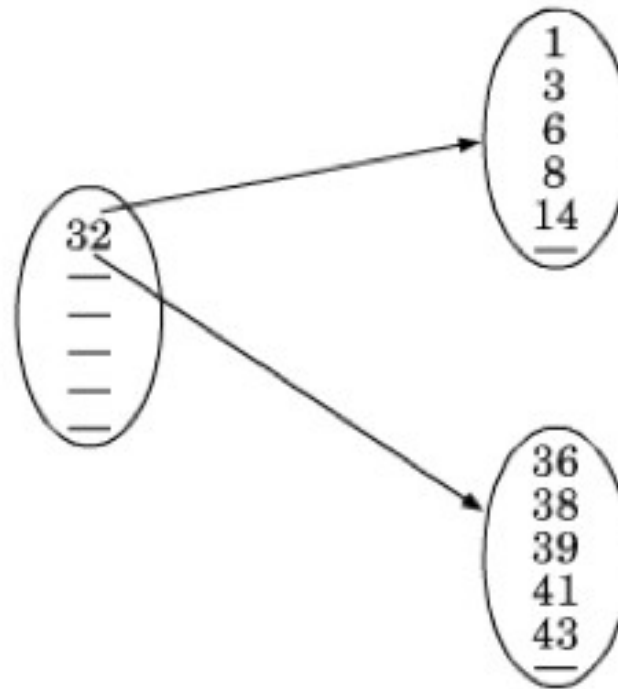
- Exercício 2:

$$\log_{2d+1}(n + 1) \leq h \leq 1 + \log_{d+1}\left(\frac{n+1}{2}\right), \text{ para } n \geq 1$$

Soluções

- Exercício 3

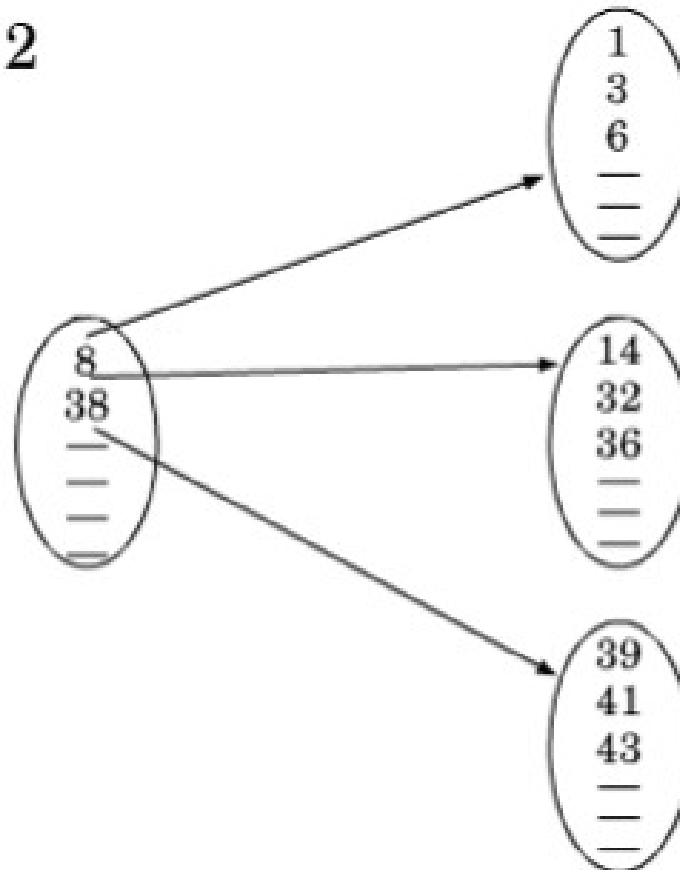
Solução 1



Soluções

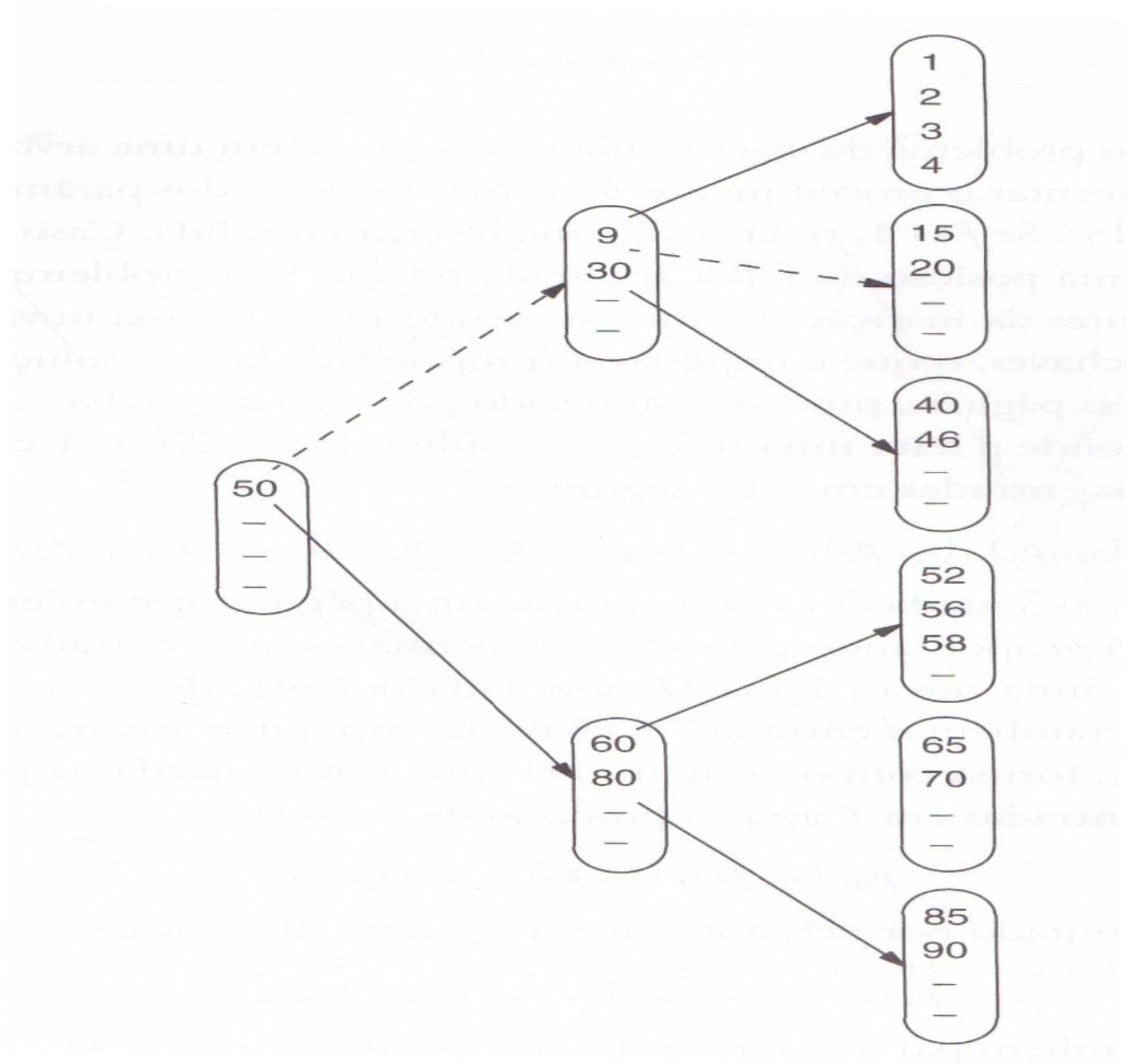
- Exercício 3

Solução 2



Operação de busca

- O algoritmo de busca compara a chave x , a chave procurada, com a chave (ou chaves) na raiz.
- Caso a chave não seja encontrada na página em questão, a busca deve prosseguir em um certo filho dessa página, o qual é escolhido observando-se a propriedade (c).
- Exemplo: O slide seguinte apresenta uma árvore B de ordem 2. As linhas pontilhadas marcam o caminho percorrido em buscas pelas chaves de valor 10 (sem sucesso) e 20 (sucesso).

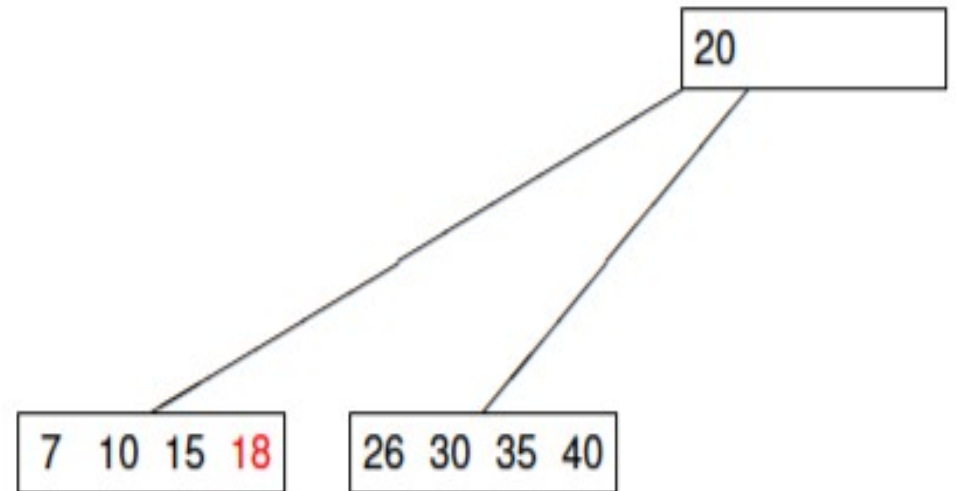
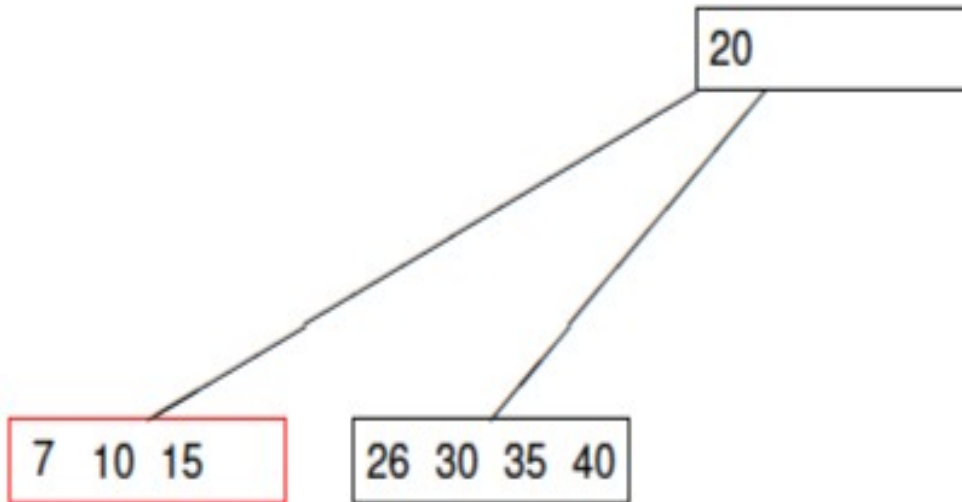


Operação de inserção

- Seja x a chave a ser inserida.
- Inicialmente, fazemos a busca da chave x na árvore.
- Se x for encontrado, nada a fazer.
- Se x não for encontrada, insere-se x na **página-folha** onde a busca se encerrou, mantendo a ordenação correta da lista de chaves.
- Se eventualmente a página-folha contém mais de $2d$ chaves, é preciso efetuar seu **particionamento** ou **cisão** (lembrando que $2d$ é o limite máximo para o número de chaves em uma página).

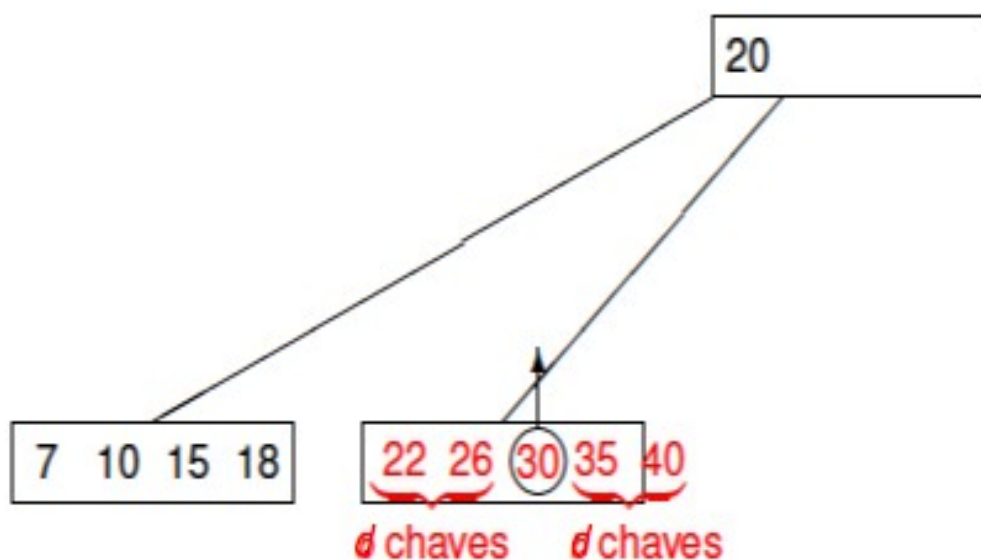
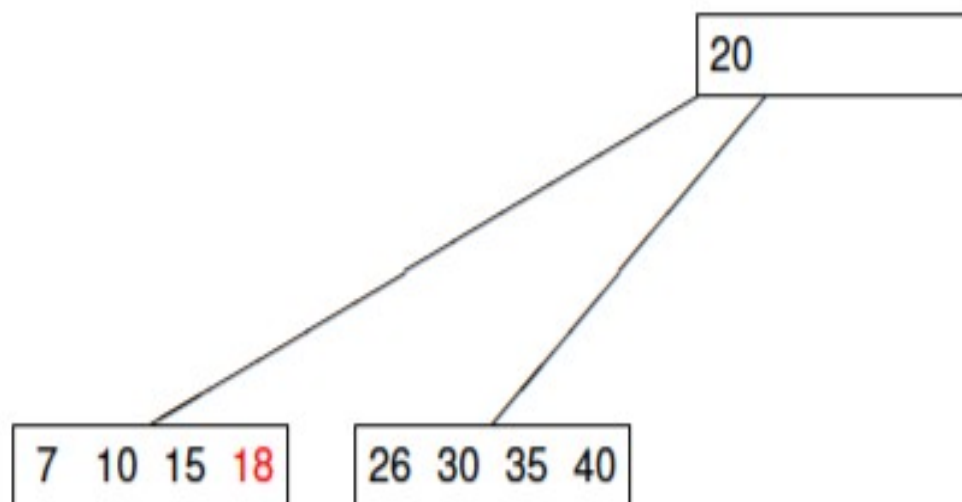
Operação de inserção

- Exemplo: Inserir a chave 18 na árvore B de ordem 2 abaixo.



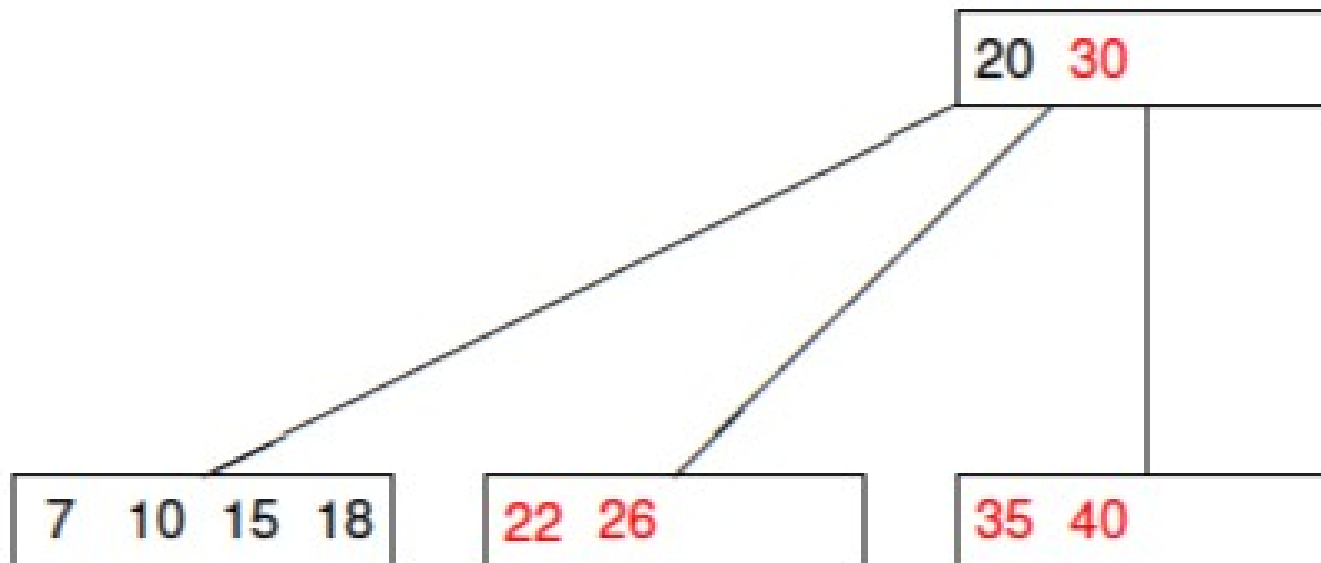
Operação de inserção

- Exemplo: Inserir a chave 22 na árvore B abaixo. Percebe-se a necessidade de uma **cisão** na página-folha onde a chave foi inserida.



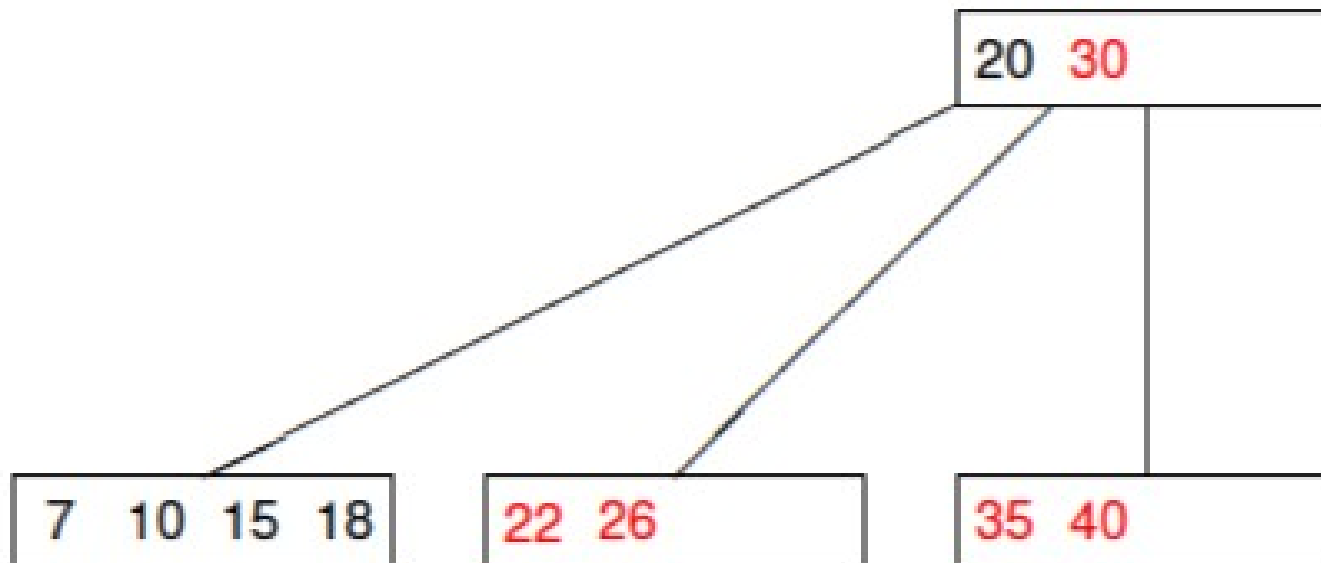
Operação de inserção

- Árvore B após a inserção da chave 22.



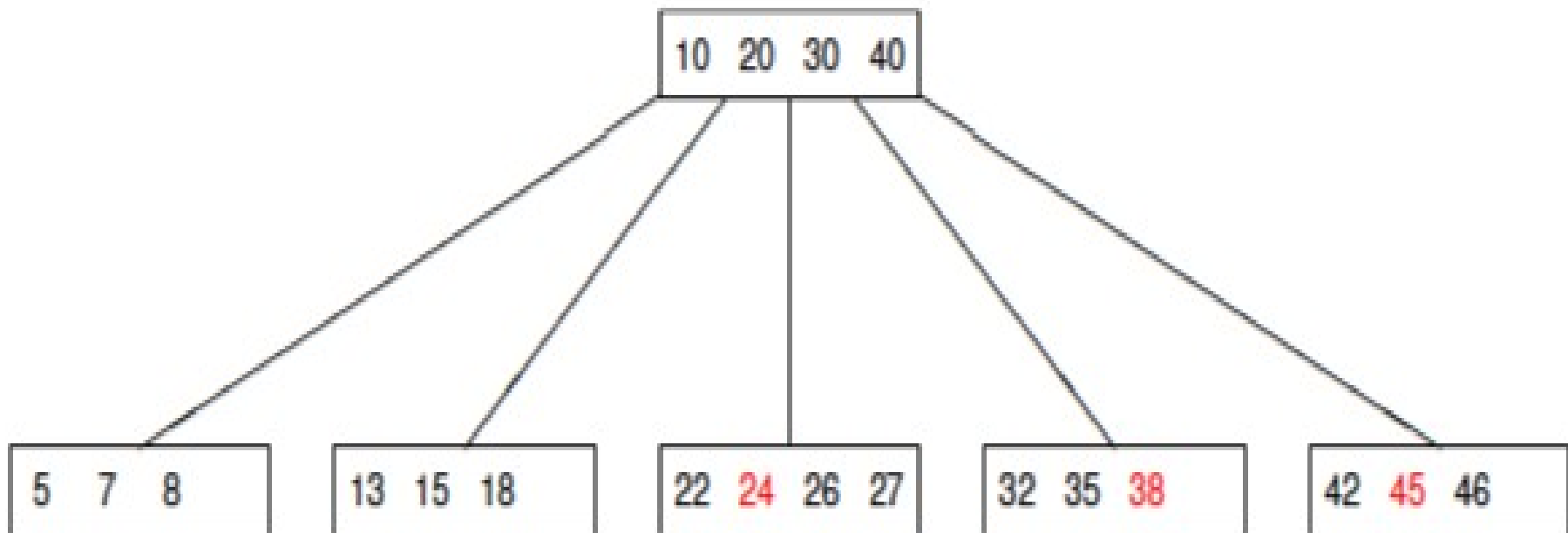
Operação de inserção

- Exemplo: Considerando a árvore B abaixo, adicione as chaves: 5, 42, 13, 46, 27, 8, 32, 38, 24, 45.



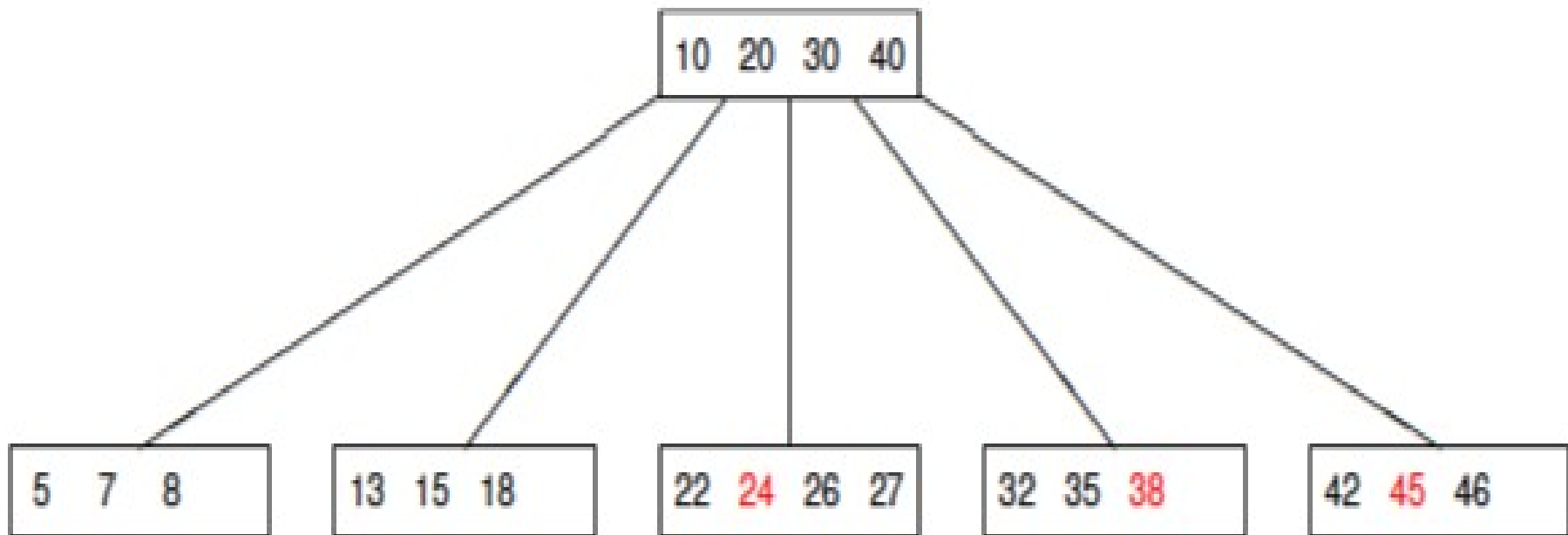
Operação de inserção

- Árvore B após as inserções pedidas no slide anterior.



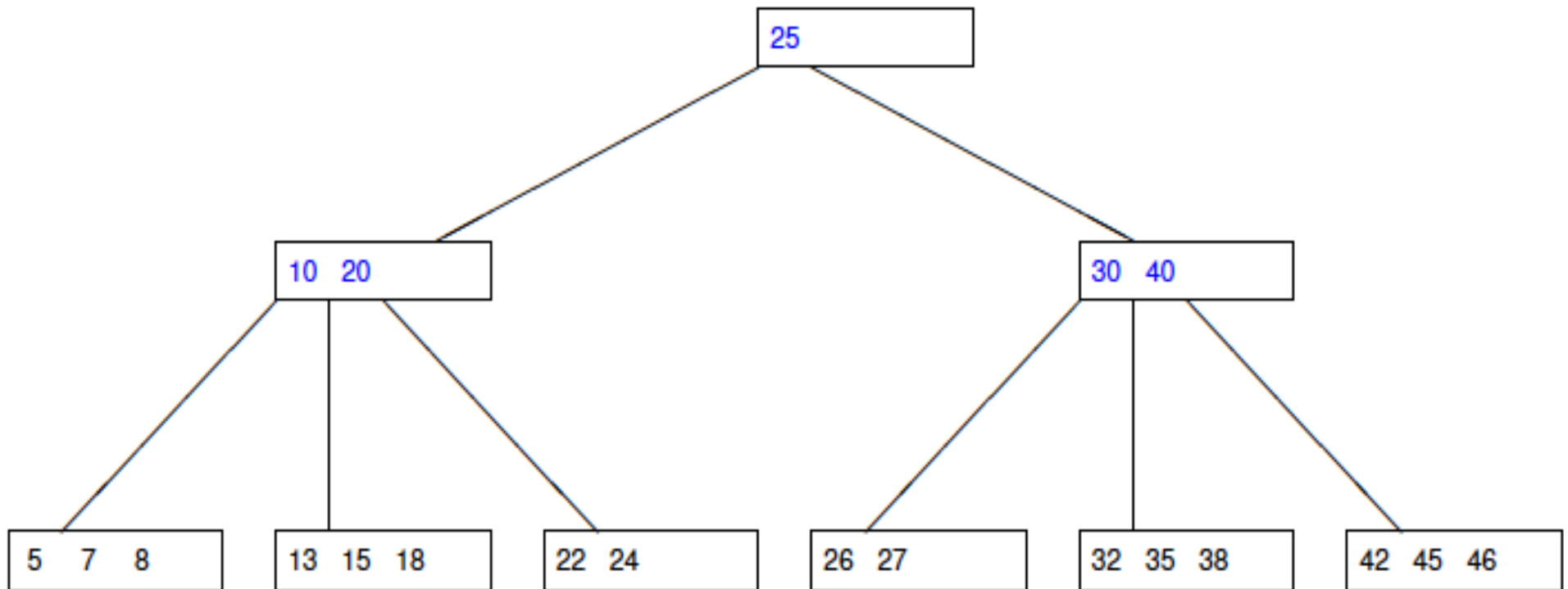
Operação de inserção

- Exemplo: Inserir a chave 25 na árvore B abaixo.



Operação de inserção

- Árvore B após a inserção da chave 25. Houve necessidade de duas cisões. O processo de cisão é **propagável**.

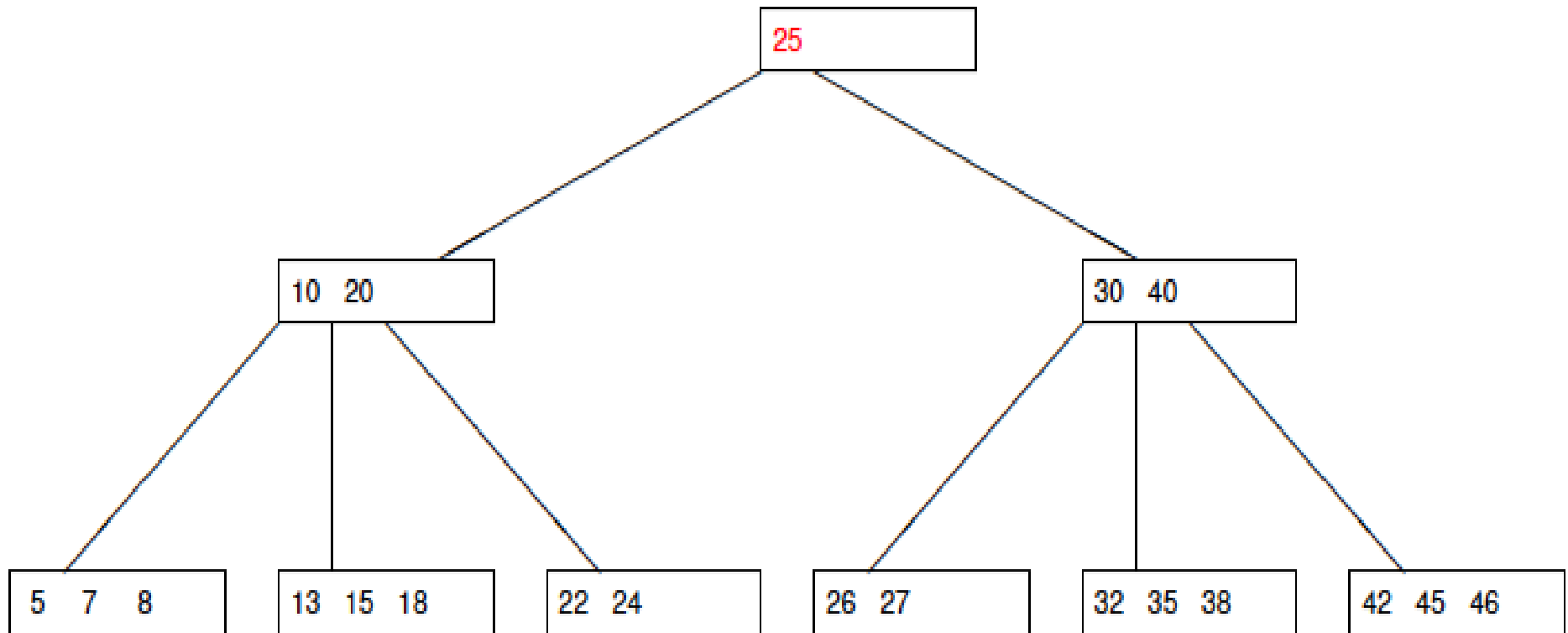


Operação de remoção

- Primeiro localizamos a página P onde se encontra a chave x a ser removida. Sabe-se que P pode ser uma folha ou interna.
- Caso 1: P é uma página-folha. Quando a chave é retirada, o número de chaves da página pode ficar menor que d , o que não é permitido. Dois tratamentos: **concatenação** e **redistribuição**.
- Duas páginas podem ser concatenadas se são irmãs adjacentes e juntas possuem menos de $2d$ chaves. Senão, redistribuir.
- Caso 2: P é uma página-interna. **Substitui-se chave x pela chave y de maior valor na subárvore esquerda**. A retirada de y de uma folha pode recair no Caso 1.

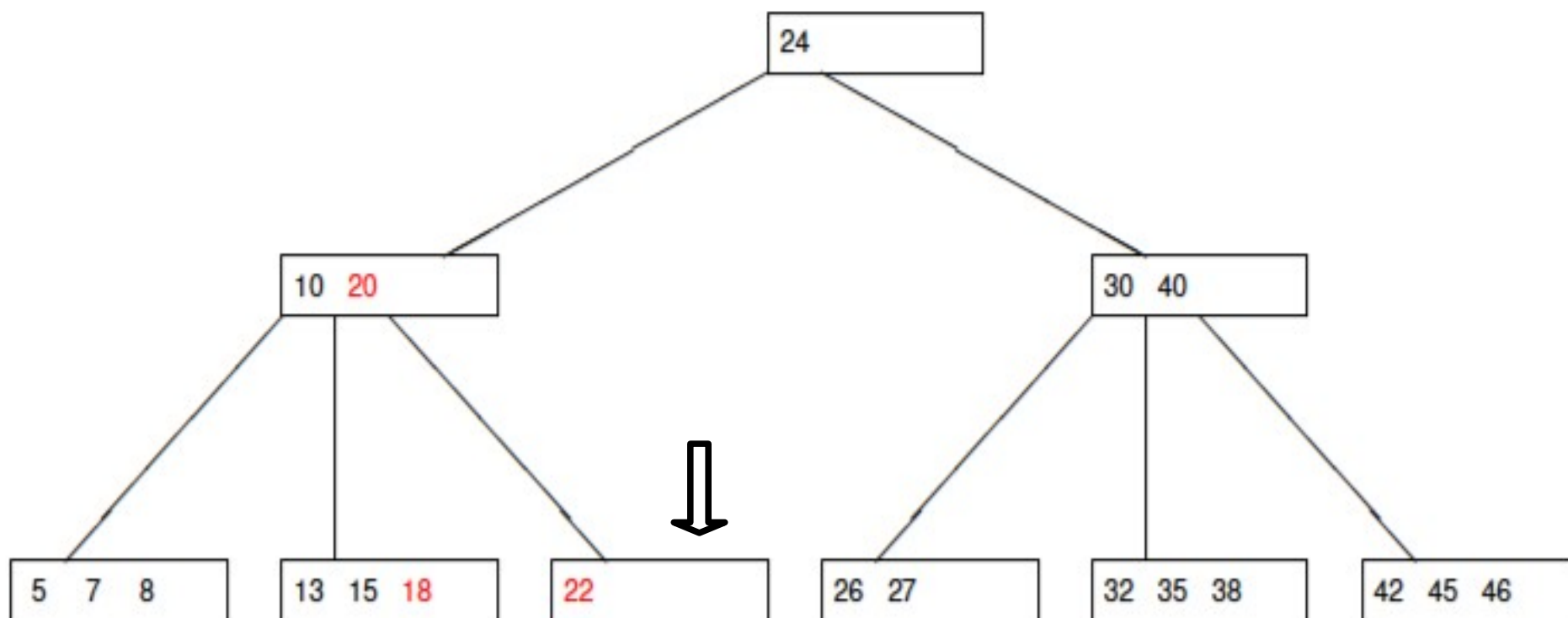
Operação de remoção

- Exemplo: Remover a chave de valor 25 da árvore B de ordem 2 abaixo.



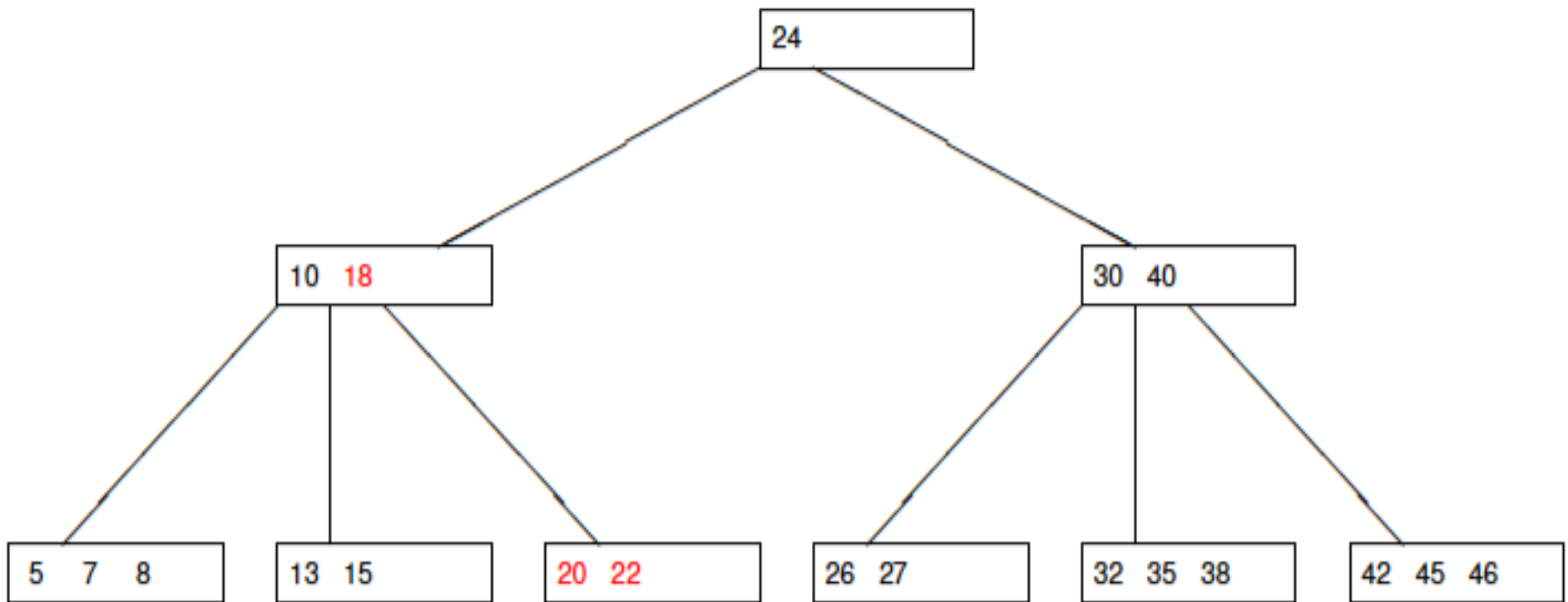
Operação de remoção

- A página com a chave 22 ficou com poucas chaves.
Redistribuir as chaves com a página-irmã.



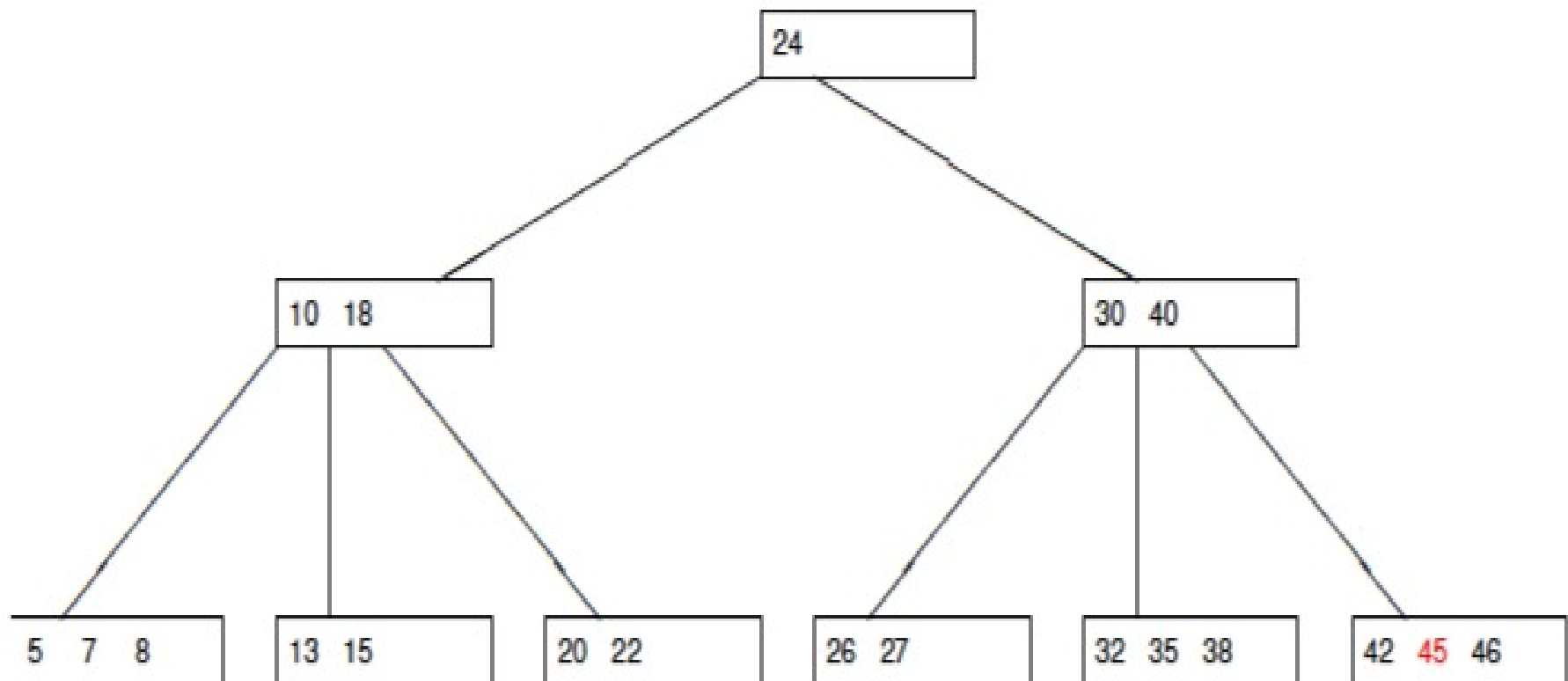
Operação de remoção

- Agora a página tem número aceitável de chaves.



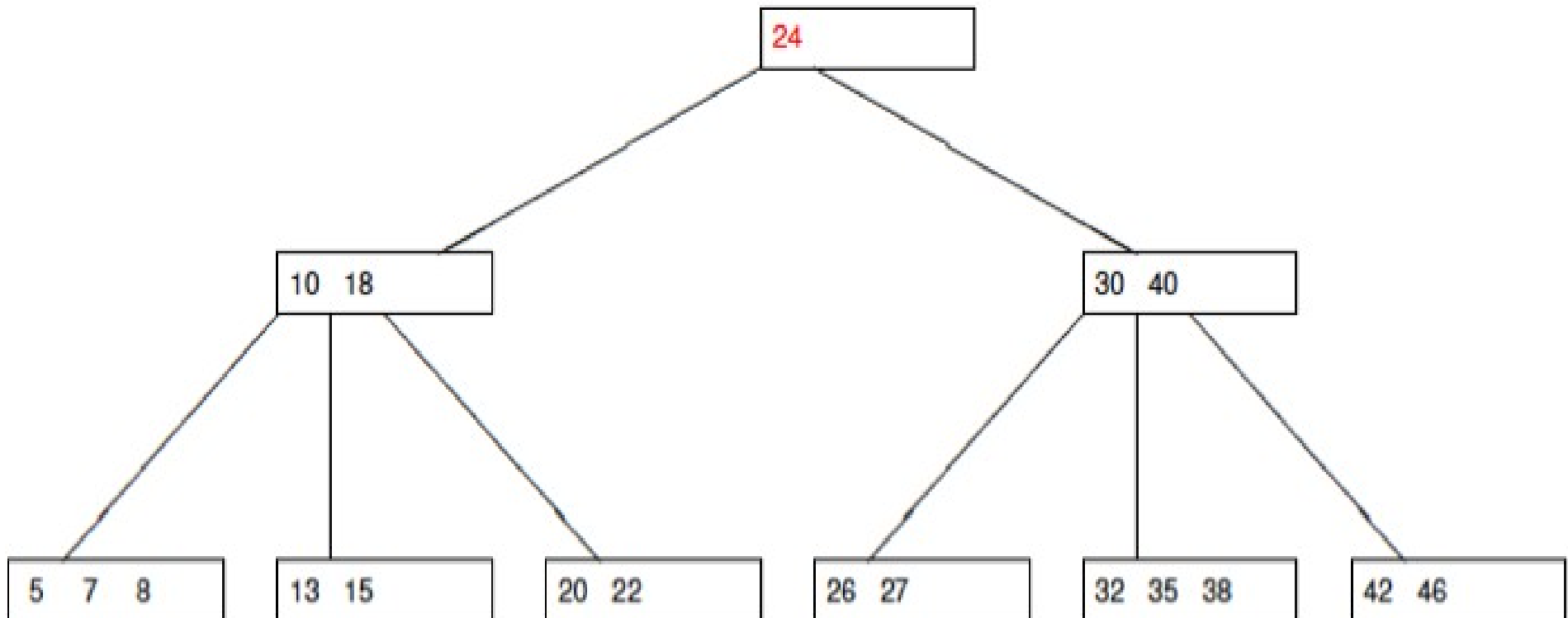
Operação de remoção

- Exemplo: Remover a chave 45 da árvore B de ordem 2 abaixo.



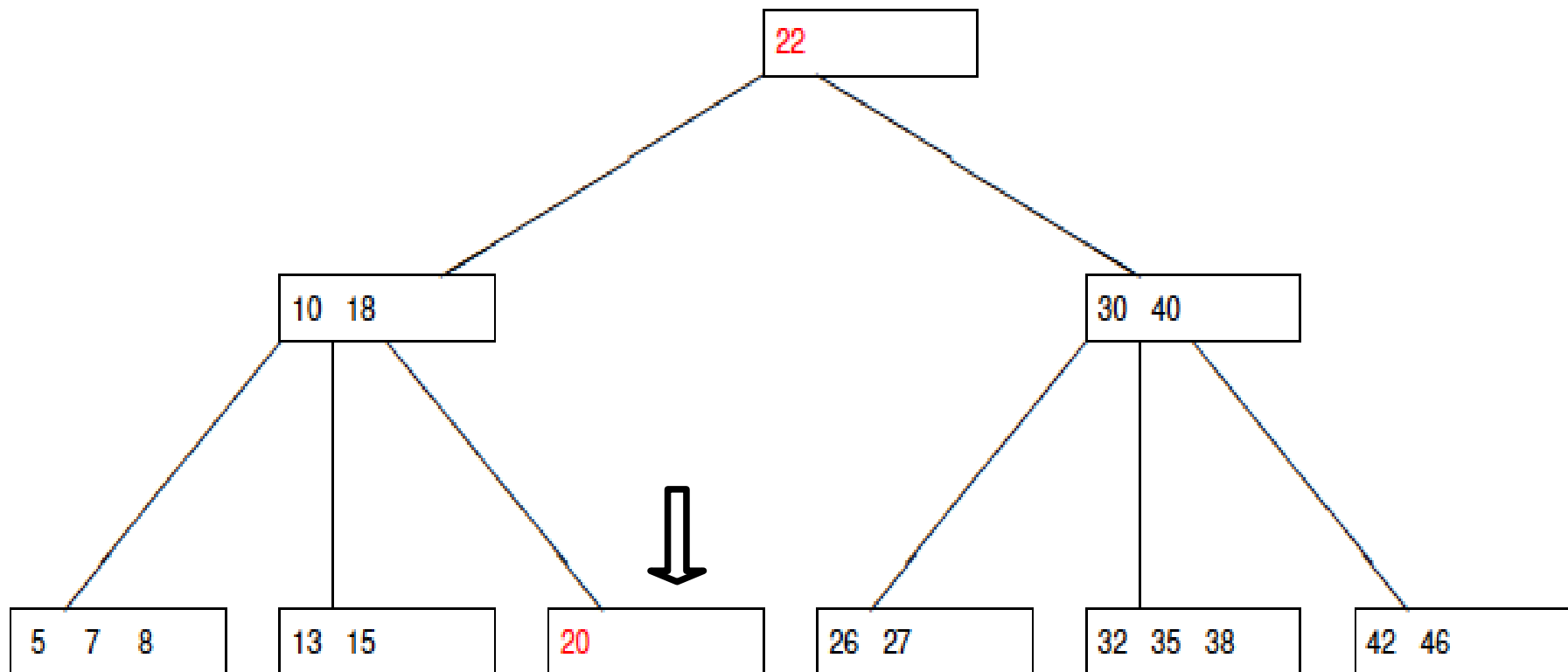
Operação de remoção

- Exemplo: Remover a chave 24.



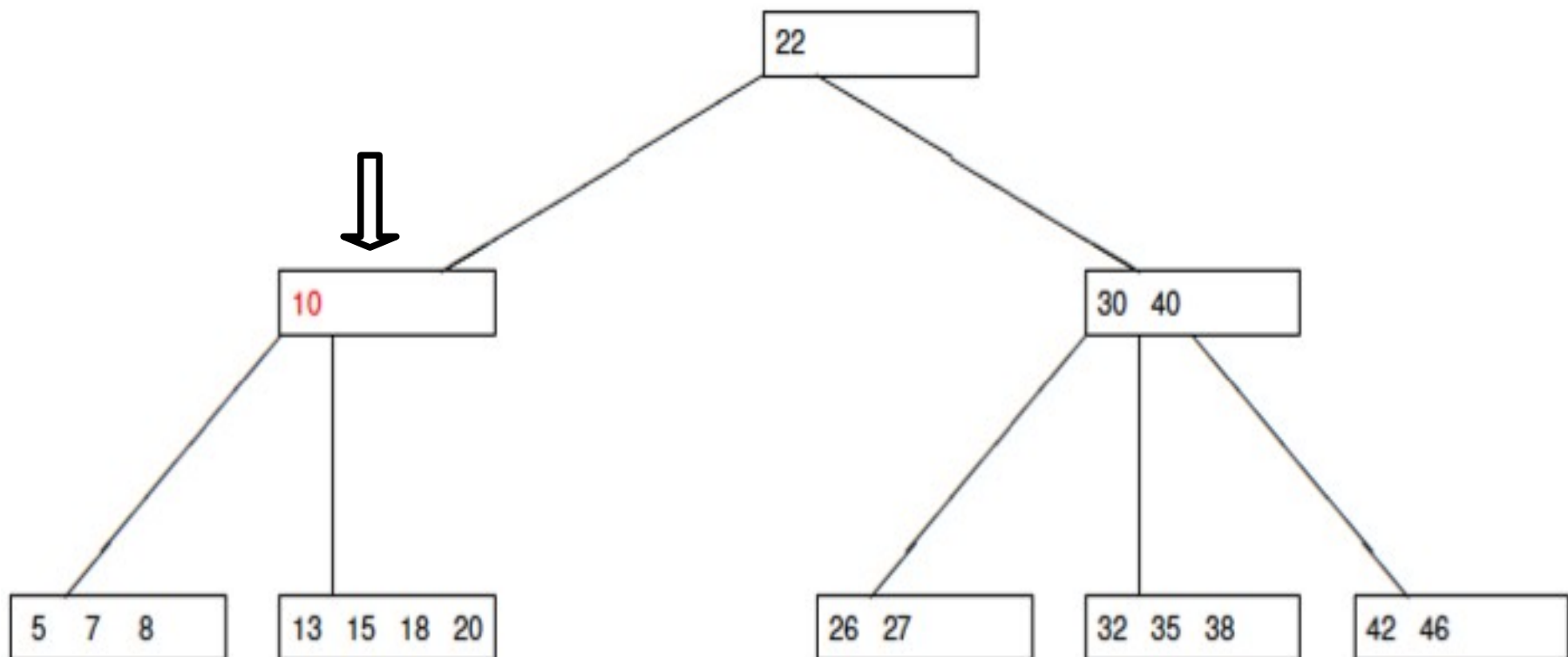
Operação de remoção

- A página com a chave 20 ficou com poucas chaves.
Concatenar com as chaves da página-irmã.



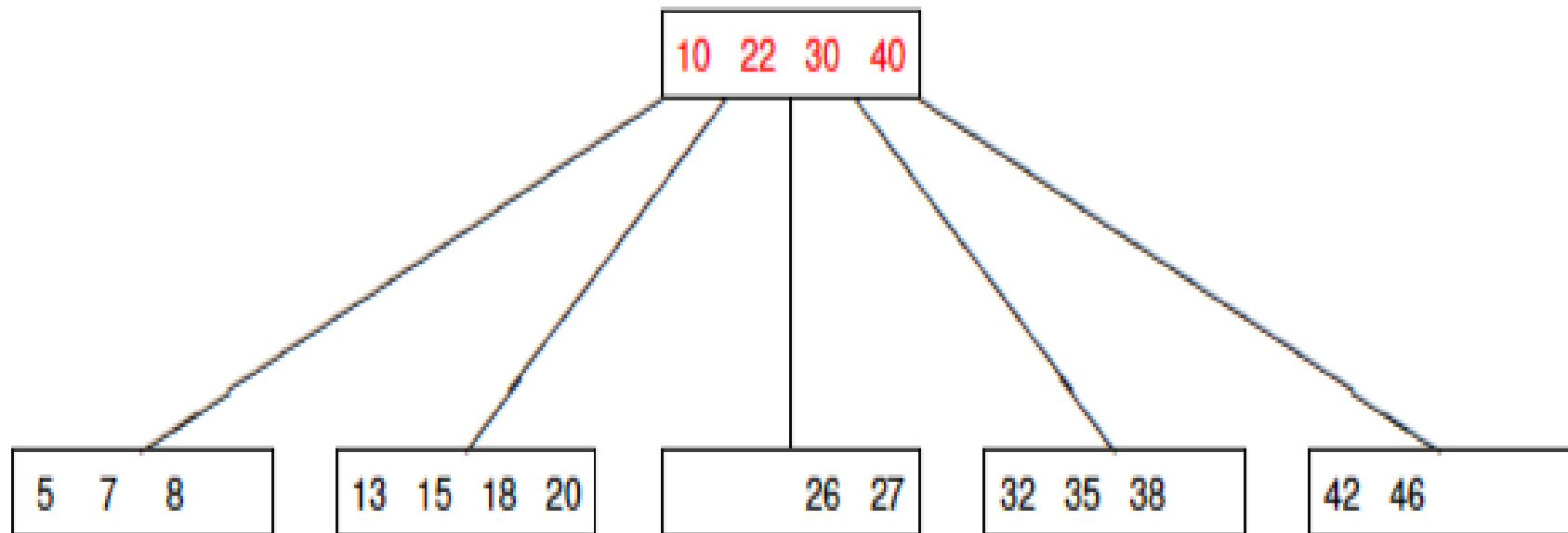
Operação de remoção

- A página-folha ficou um número aceitável de chaves. Mas surgiu outro problema. Uma das páginas-internas possui apenas a chave 10. **Concatenar** de novo.



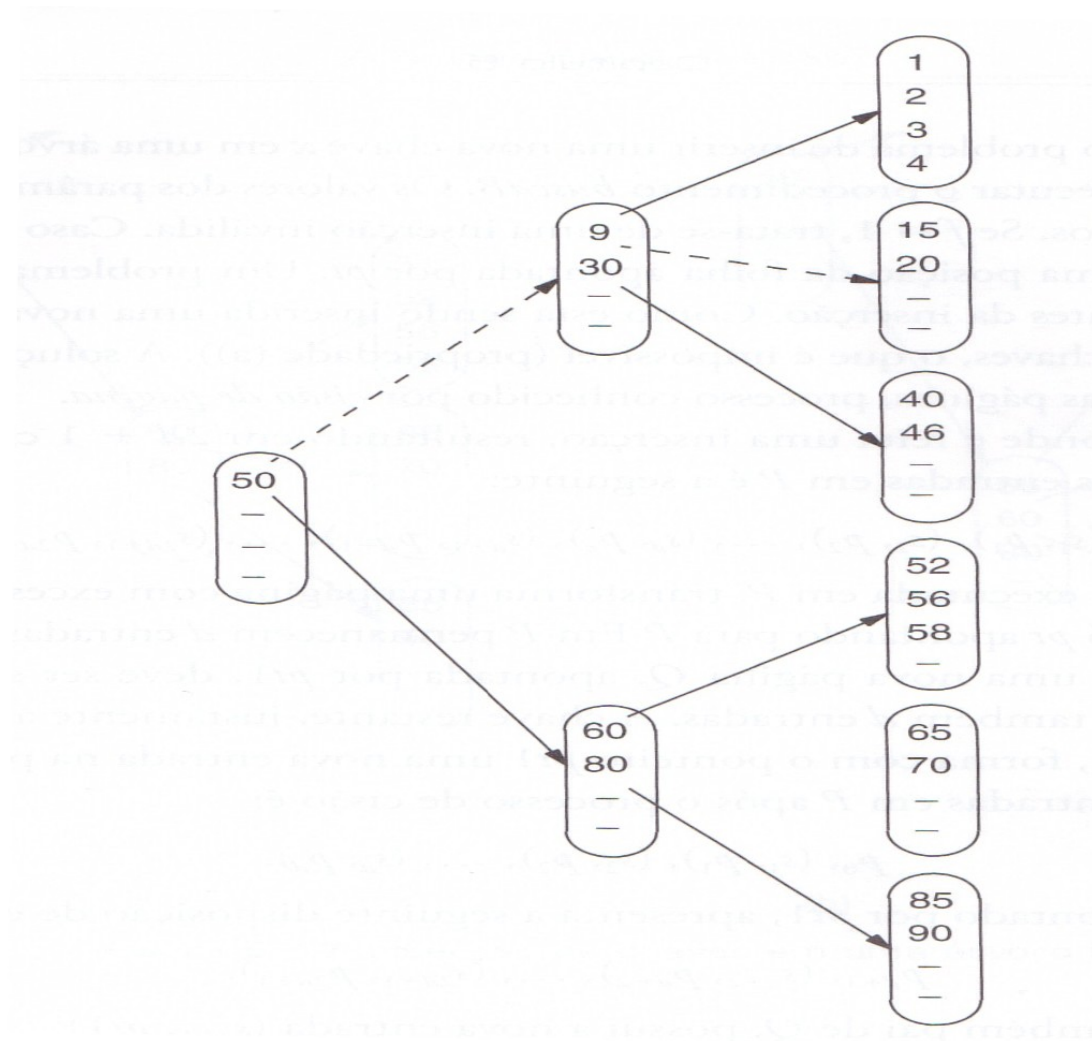
Operação de remoção

- Agora todas as páginas da árvore B possuem um número aceitável de chaves.

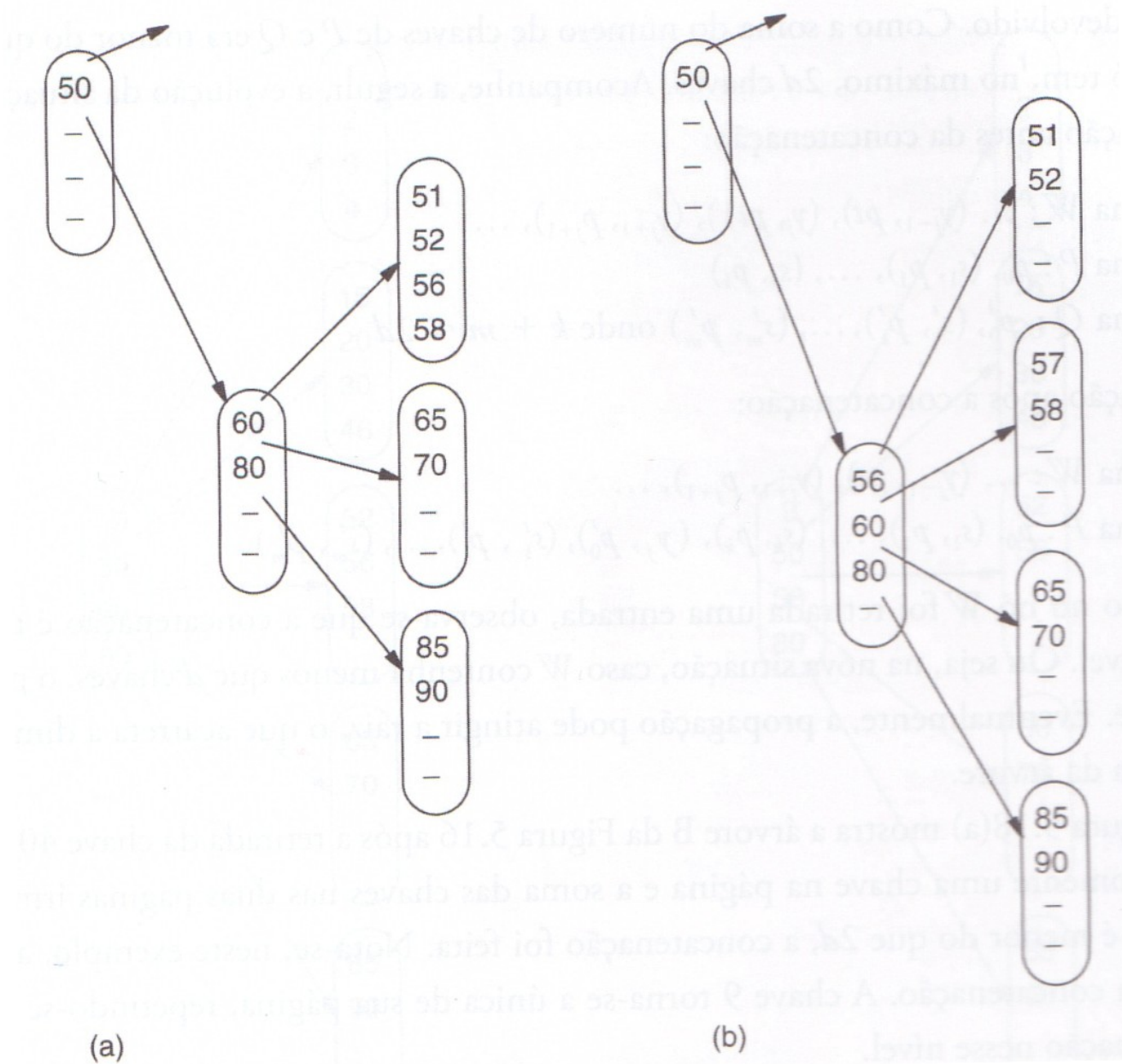


Exercício

Dada a árvore B de ordem 2 abaixo, mostre sua representação após a inserção das chaves 51 e, em seguida, 57.

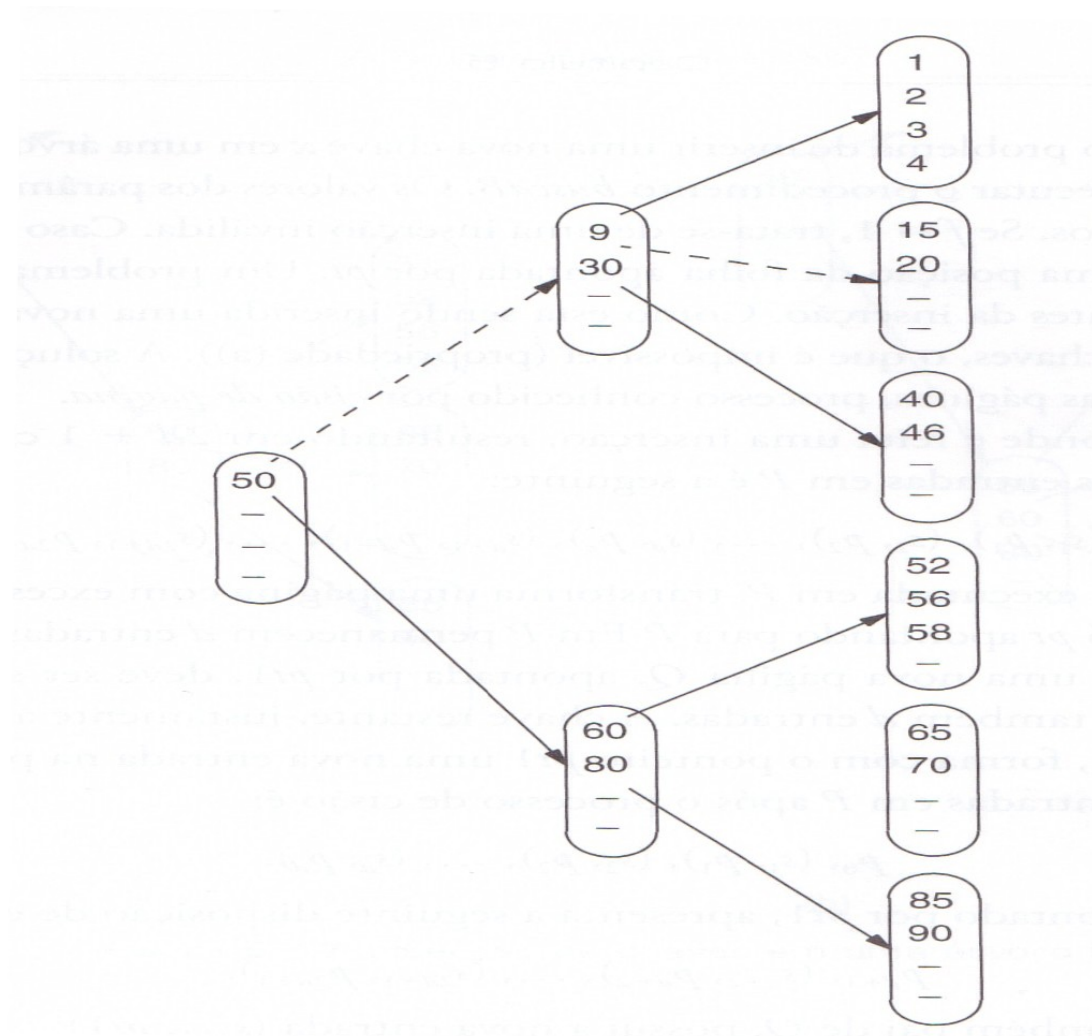


Solução

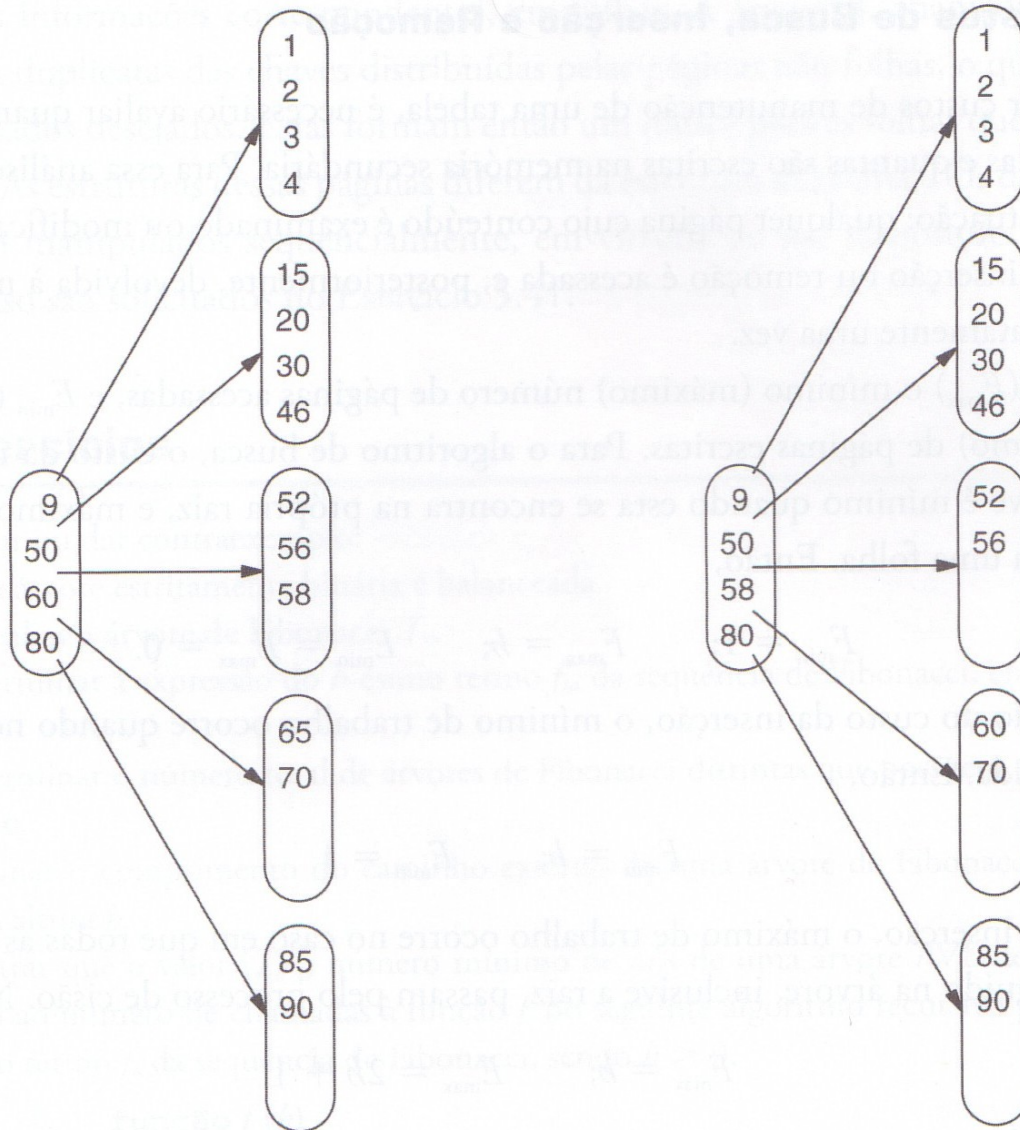


Exercício

Dada a árvore B de ordem 2 abaixo, mostre sua representação após a remoção das chaves 40 e, em seguida, 65.



Solução



(a)

(b)

A concatenação pode acarretar na diminuição da altura da árvore, no entanto, é propagável. Já a redistribuição não é propagável.

Conclusões

- Árvore de busca binária completa: inserções/remoções não são viáveis. Alto custo para mantê-la completa.
- Árvore binária balanceada: alternativa suficientemente boa.
- Busca, inserção, remoção em árvores balanceadas: $O(\log n)$.
- Árvores balanceadas são muito usadas na prática: *TreeMap* e *TreeSet* do Java.util. Em C++, *Map* e *Set* do STL.

Conclusões

- A árvore rubro-negra, assim como a AVL, necessita no máximo de 1 (uma) rotação para balancear na inserção. Contudo, pode ter $\log(n)$ trocas de cores.
- Na operação de remoção, a rubro-negra faz no máximo 1 (uma) rotação (simples ou dupla), enquanto a AVL pode fazer $\log(n)$ rotações.

Conclusões

- A AVL é uma estrutura “mais balanceada” que a rubro-negra, o que leva a AVL a ser mais lenta na inserção e remoção, porém, mais rápida na busca.
- Nesse sentido, a rubro-negra é mais utilizada em aplicações de tempo real (críticas).
- A árvore rubro-negra pode ser usada para construir blocos em outras estruturas de dados que precisam de garantias no pior caso, p.e., estruturas de dados em geometria computacional podem ser baseadas nesse tipo de árvore.

Conclusões

- Diversas variações interessantes existem na literatura para as árvores B. Uma das mais conhecidas é a **Árvore B+**. A ideia desta variação é manter todas as chaves de busca em seus nós folha de modo que o acesso sequencial ordenado das chaves de busca seja um processo mais eficiente do que em árvores B.

