Árvores 2-4

Prof. Dr. Lucas C. Ribas

Disciplina: Estrutura de Dados II

Departamento de Ciências de Computação e Estatística

unesp
UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

IBILCE / UNESP - CÂMPUS DE SÃO JOSÉ DO RIO PRETO

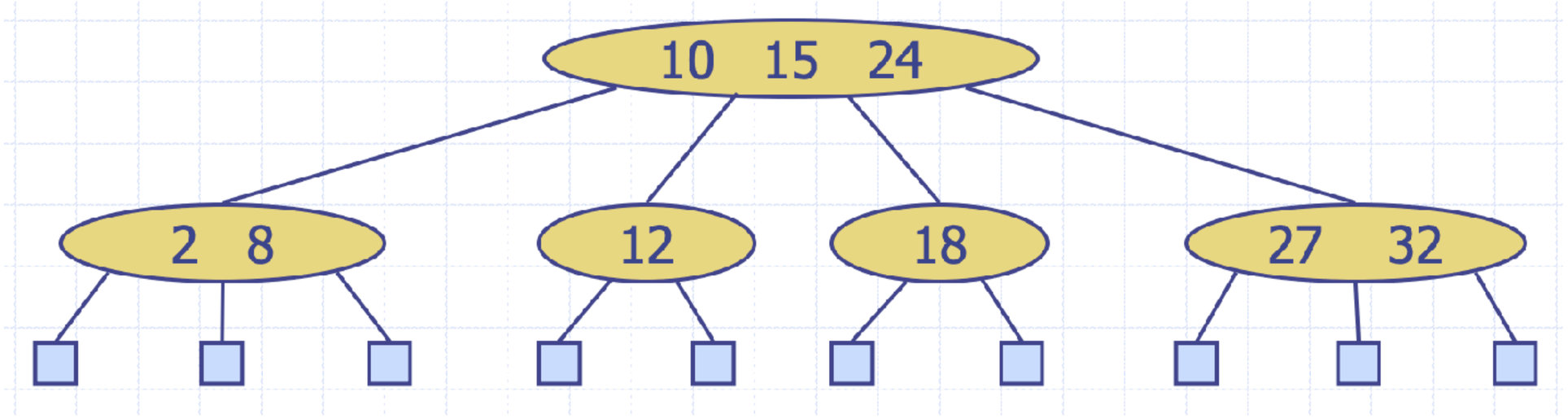
Introdução

Introdução

- Árvore 2-4 é um caso especial de árvore B
- Uma árvore B de ordem 4
- Esta árvore B foi primeiramente discutido por Rudolf Bayer, que a chamou de árvore B binária simétrica (Bayer 1972)
- Mas geralmente é chamada de **árvore 2–3–4** ou apenas **árvore 2–4**

Propriedades

- Cada nó interno tem **até 4 filhos** ou **1, 2 ou 3 elementos/items/chaves**
- Todos os nós externos (folhas) tem a mesma profundidade
- Dependendo do número de filhos, um nó interno de uma árvore 2-4 é chamado de **2 nós, 3 nós ou 4 nós**



- Uma árvore 2–4 parece não oferecer novas perspectivas, mas a verdade é exatamente o oposto
- Nas **árvores B, projetadas para HD/SSDs**, os nós são grandes para acomodar o conteúdo de um bloco lido do armazenamento secundário
- Por outro lado, em **árvores 2–4**, apenas um, dois ou no máximo três elementos (chaves) podem ser armazenados em um nó
- Isso pode parecer ineficiente para armazenamento secundário porque muitos blocos de disco seriam desperdiçados se cada nó de uma árvore 2-4 correspondesse a um bloco de disco
- Embora as árvores B tenham sido introduzidas no contexto do tratamento de dados em armazenamento secundário, isso não significa que devam ser utilizadas apenas para esse fim

- No entanto, embora as árvores B tenham sido introduzidas no contexto do tratamento de dados em armazenamento secundário, isso não significa que devam ser utilizadas apenas para esse fim
  - árvores 2-4 são muito bem balanceadas, o que as torna atrativas para certas aplicações de memória principal onde o balanceamento é crítico
  - as árvores B (e suas variantes, como as árvores 2-4) podem ser eficientes em termos de desempenho devido a propriedades como baixa altura e bom balanceamento

- Passamos um semestre inteiro discutindo árvores binárias
  - em particular árvores de busca binária, e desenvolvendo algoritmos que permitem acesso rápido às informações armazenadas nessas árvores
- As árvores B podem oferecer uma solução melhor para o problema de balanceamento de árvores binárias?
- Voltamos agora aos tópicos de árvores binárias e processamento de dados na memória

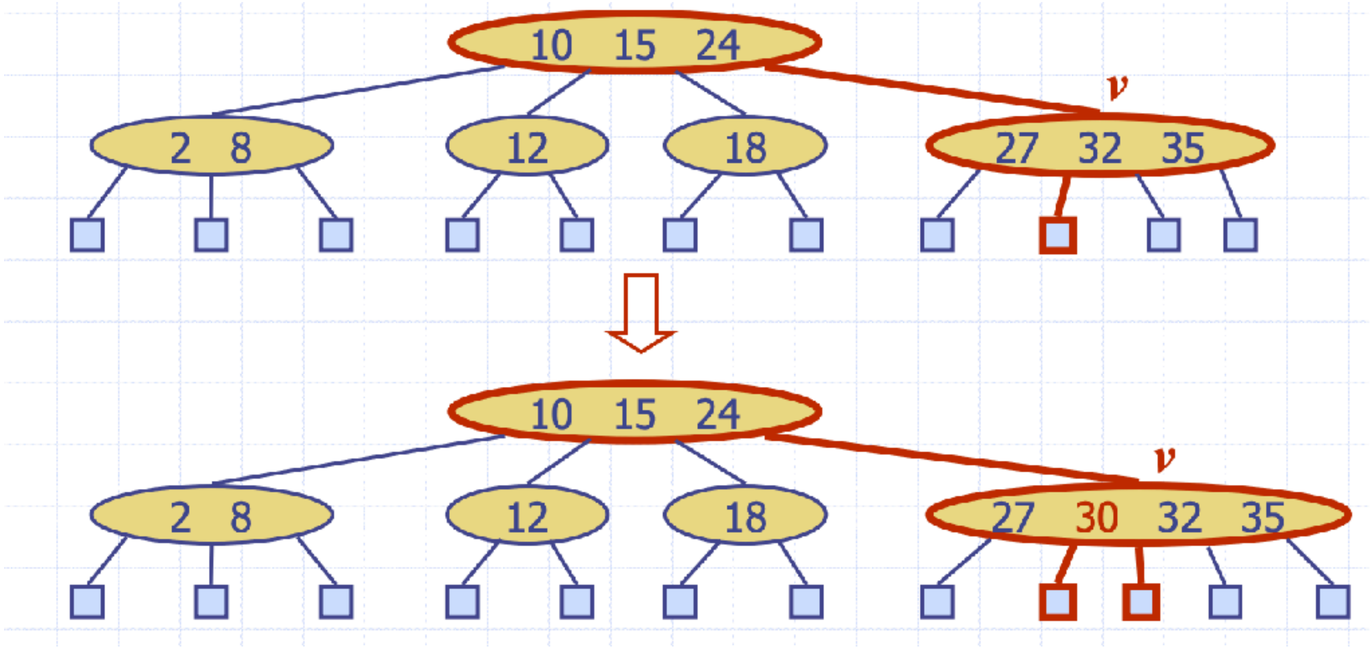
- As árvores B são adequadas para desafiar os algoritmos usados para árvores de busca binária
  - árvore B, por sua natureza, precisa ser balanceada
- Nenhum tratamento especial é necessário além de construir uma árvore:
  - construir uma árvore B equilibra-a ao mesmo tempo

- Em vez de usar árvores de pesquisa binária, podemos usar árvores B de pequena ordem, como 2–4 árvores

- Se essas árvores são implementadas como estruturas semelhantes às árvores B, existem **três locais por nó** para armazenar até **três chaves** e quatro locais por nó para armazenar até quatro ponteiros.

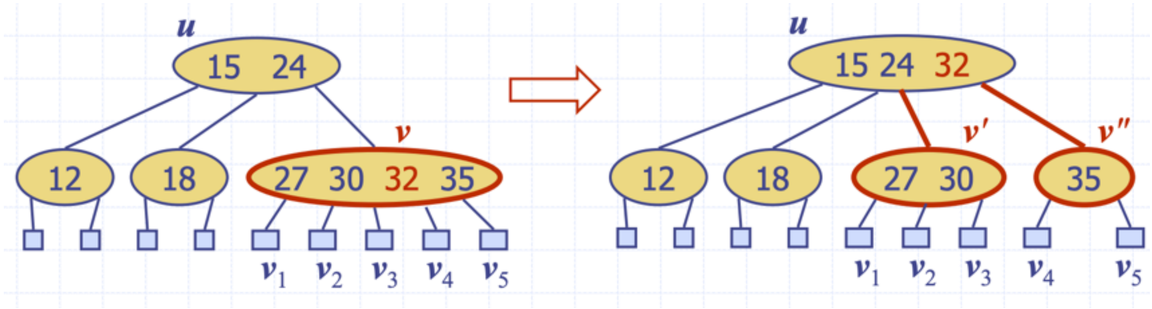
Inserção

- Inserimos um novo elemento em um nó folha após uma busca
  - Preservamos a propriedade de profundidade
  - Podemos causar *overflow* (isto é, nó pode se tornar um 5-nó)
  - Exemplo: inserindo a chave 30 causa *overflow*

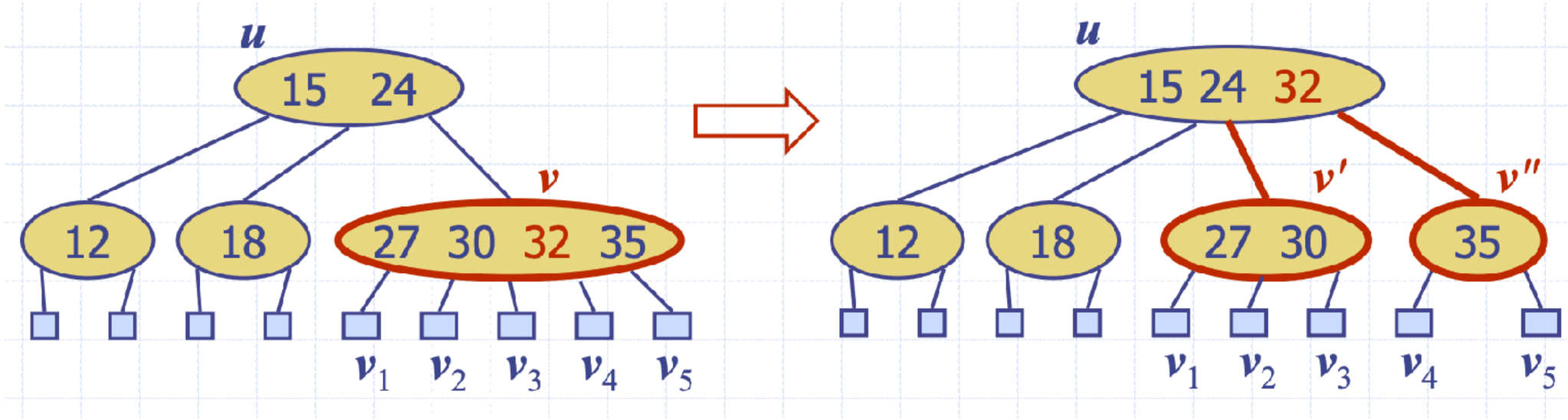


Inserção

- Lidamos com o *overflow* com um *split:*
  - *Seja os nó **v1, …, v5** filhos de **v** e **k1, …, k4** as chaves de **v***
  - *Nó v é substituído pelos nós v' e v''*
    - *v' é um 3-nó com as chaves k1 e k2 e os filhos v1, v2, v3*
    - *v'' é um 2-nó com a chave k4 e os filhos v4 e v5*
  - *A chave k3 é inserida no nó pai u de v (o nova raiz pode ser criada)*
  - *Overflow pode propagar para o nó pai u*



Inserção


u
15 24
v
12
18
27 30 32 35
v1 v2 v3 v4 v5
u
15 24 32
v'
v''
12
18
27 30
35
v1 v2 v3 v4 v5

Análise da Inserção

Algoritmo inserção(*k, o*)

1. Procuramos pela chave *k* para localizar o nó de inserção *v*
2. Adicionamos a nova chave *k* no nó *v*
3. **while** *overflow*(v)
    1. **if** *isRoot*(v)
        1. Cria uma nova raiz acima de v
    2. v <- split(v)

unesp

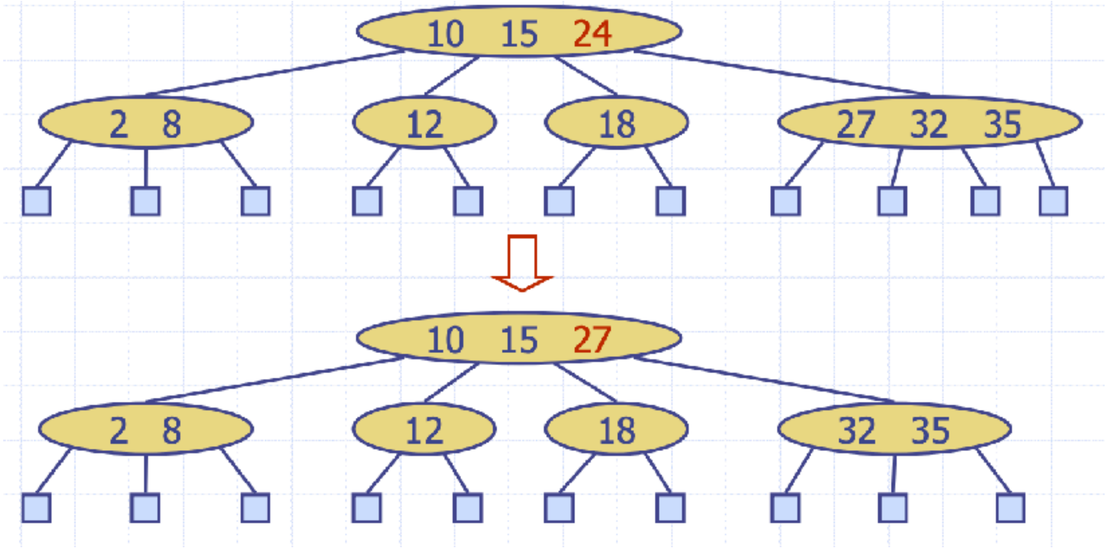
Análise da Inserção

- Considere T como sendo uma árvore (2,4) com *n* itens:
  - A altura da árvore T é **O(log n)**.
  - Passo 1: o tempo de execução do Passo 1 é O(log n) porque visitamos O(log n) nós durante a busca pelo local de inserção.
  - Passo 2: o tempo de execução do Passo 2 é O(1), pois a inserção em si é uma operação que leva tempo constante, desde que o local onde o novo item deve ser inserido já tenha sido identificado.
  - Passo 3: o tempo de execução do Passo 3 é O(log n) porque, embora cada divisão (split) de um nó seja O(1), podemos ter que realizar O(log n) divisões à medida que subimos de volta pela árvore para manter suas propriedades após a inserção.
- Assim, uma inserção em uma árvore 2-4 leva um tempo total de O(log n), pois a complexidade é dominada pelos passos que escalam com o logaritmo do número de itens na árvore

unesp

Remoção

- Reduzimos a remoção de uma chave ao caso em que o item está no nó folha
- Caso contrário, substituímos a entrada pelo seu sucessor e exclua a última entrada
  - **Exemplo:** para deletar a chave 24, substituímos por 27 (sucessor em ordem)



Underflow e fusão

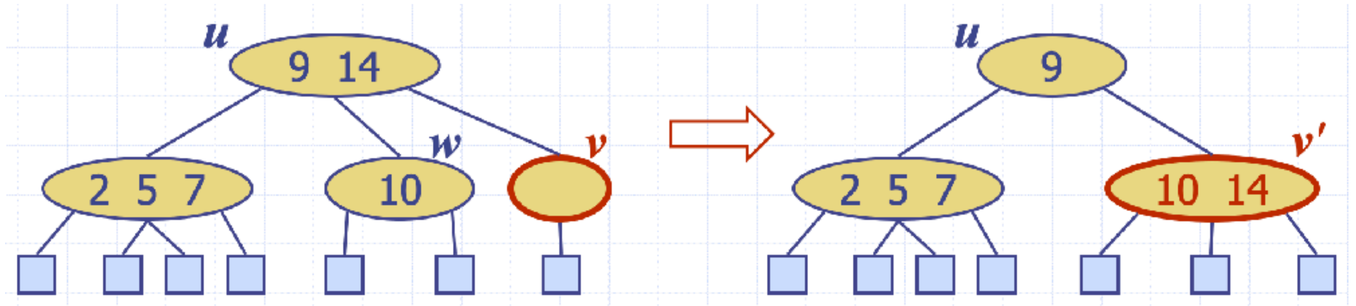
- A exclusão de uma entrada do nó *v* pode causar um *underflow*, onde o nó ***v*** se torna um 1-nó com um filho e sem chaves.

- Para tratar um *underflow* no nó ***v*** com o pai ***u***, consideramos dois casos:

- Caso 1: os irmãos adjacentes de ***v*** são 2-nós

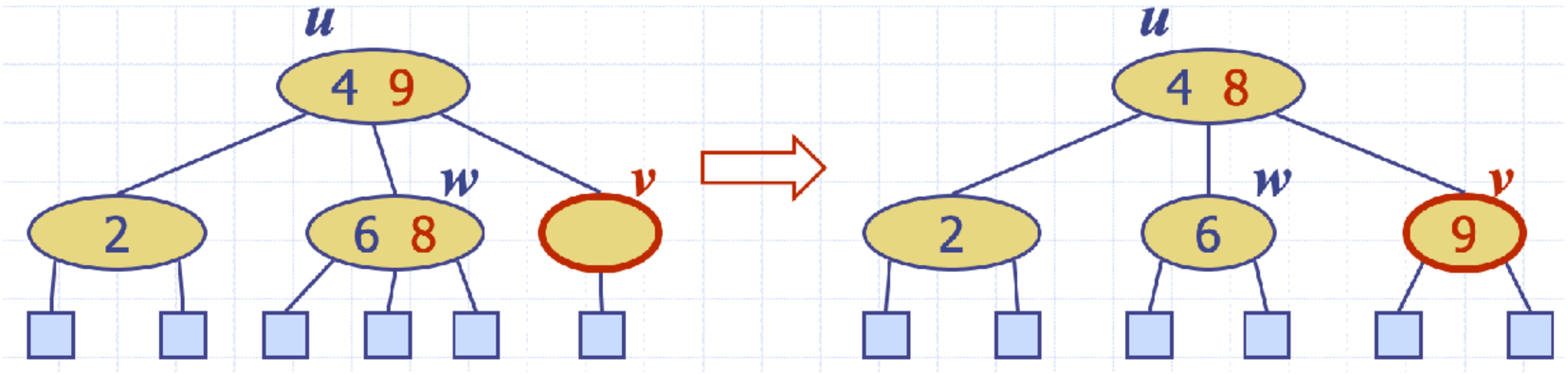
  - Operação de fusão: mesclamos v com um irmão adjacente w e movemos uma entrada de u para o nó mesclado v'.

  - Após uma fusão: o *underflow* pode se propagar para o pai u, o que pode exigir tratamento adicional.



Underflow e fusão

- Caso 2: um irmão adjacente ***w*** de ***v*** é um 3-nó ou um 4-nó
  - Operação de transferência:
    1. Movemos um filho de w para v.
    2. Transferimos uma entrada de u para v.
    3. Movemos uma entrada de w para u.
  - Após uma transferência: não ocorre underflow, pois v agora tem a quantidade suficiente de filhos/entradas para atender à propriedade da árvore 2-4.



Análise da remoção

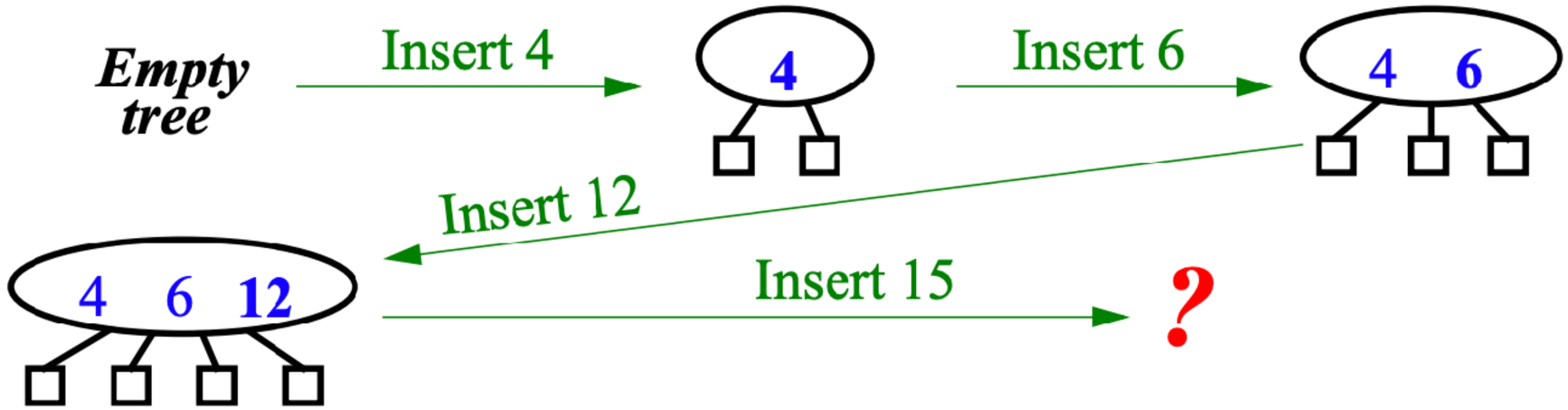
- Seja **T** uma árvore 2-4 com ***n*** itens:
  - A altura da árvore T é **O(log n)**
- Na operação de exclusão:
  - Nós visitamos O(log n) nós para localizar o nó de onde a entrada será excluída.
  - Para tratar de um underflow, realizamos uma série de fusões O(log n). Essas fusões podem propagar a necessidade de reajuste para cima na árvore, mas cada uma dessas operações de fusão leva tempo constante O(1).
- Portanto, **excluir um item de uma árvore 2-4 leva tempo O(log n)**

Exercícios

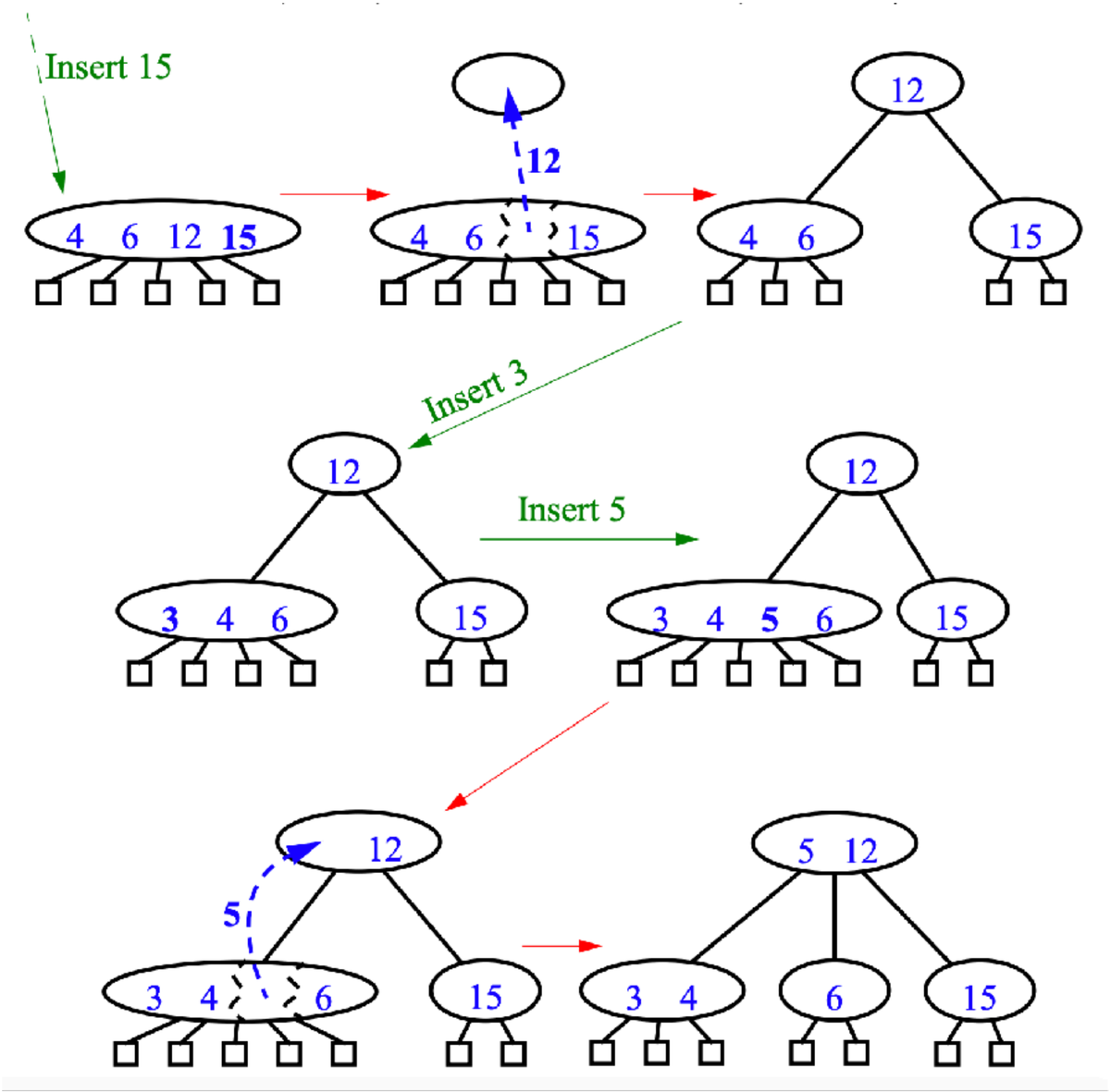
- Crie uma árvore 2-4 inserindo passo a passo as chaves: 4, 6, 12, 15, 3, 5, 17

Exercícios

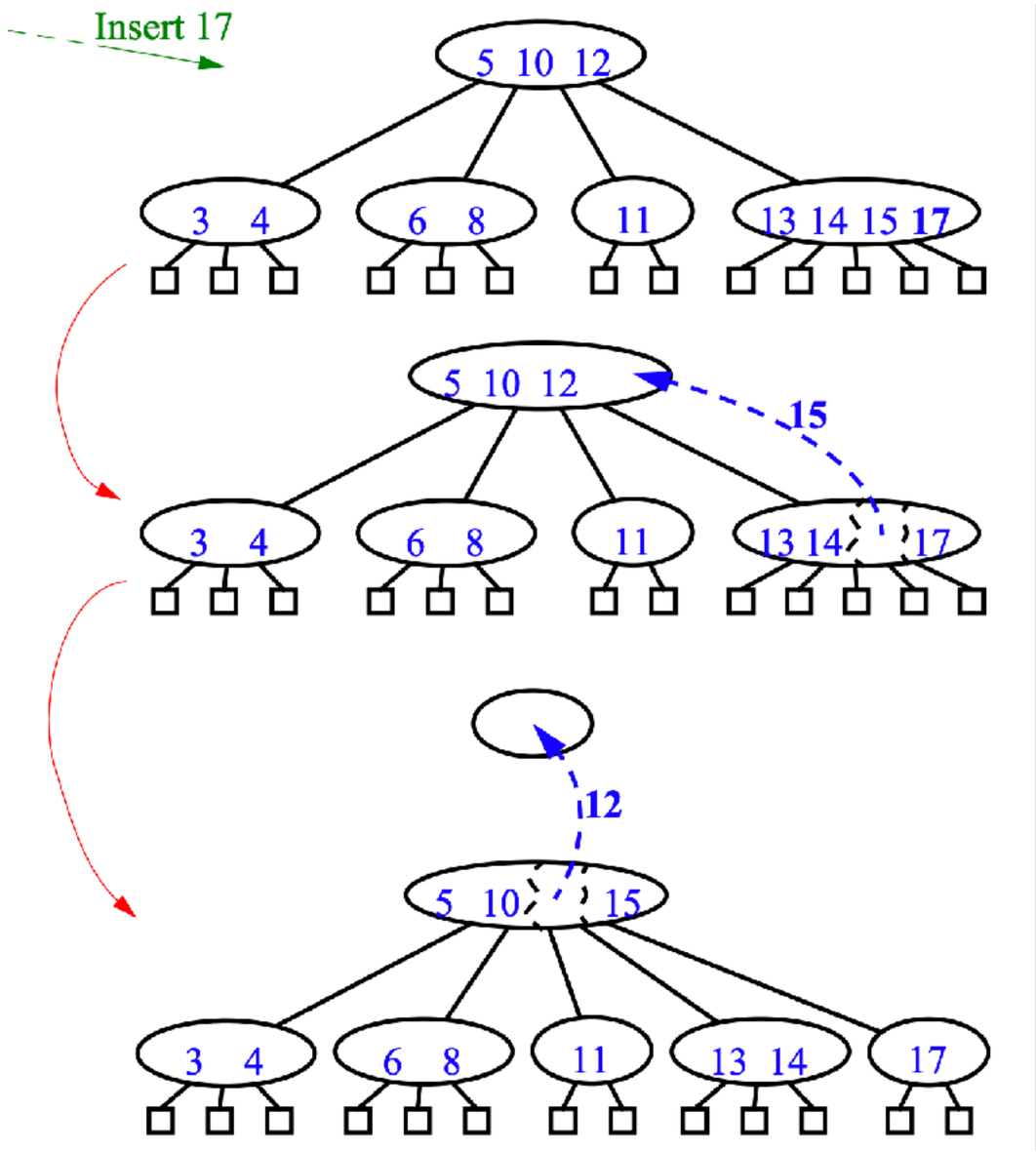
- Crie uma árvore 2-4 inserindo passo a passo as chaves: 4, 6, 12, 15, 3, 5, 17



Exercícios


Insert 15
4 6 12 15
12
4 6 15
12
4 6
15
Insert 3
12
3 4 6
15
Insert 5
12
3 4 5 6
15
12
5
3 4 6
15
5 12
3 4
6
15

Exercícios


Insert 17
5 10 12
3 4
6 8
11
13 14 15 17
5 10 12
15
3 4
6 8
11
13 14
17
12
5 10 15
3 4
6 8
11
13 14
17

Representações de árvore 2-4

- Árvores 2-4 são implementadas como estruturas semelhantes às árvores B
- Na pior das hipóteses, metade destas células não são utilizadas e, em média, **69% são utilizadas**
  - pior das hipóteses, quando cada nó (exceto folhas) tem apenas 2 filhos (o mínimo para não ser uma folha), metade dos espaços para chaves e ponteiros está vazia
  - os nós tendem a ter mais de 2 filhos em média, o que leva a uma utilização de cerca de 69%.

Representação de árvore 2-4

- Como o espaço é muito mais valioso na memória principal do que no armazenamento secundário, gostaríamos de evitar esse espaço desperdiçado
- Árvores 2 a 4 são transformadas em forma de árvore binária, na qual cada nó contém apenas uma chave
- É claro que a transformação deve ser feita de uma forma que permita uma restauração da forma original da árvore B

unesp

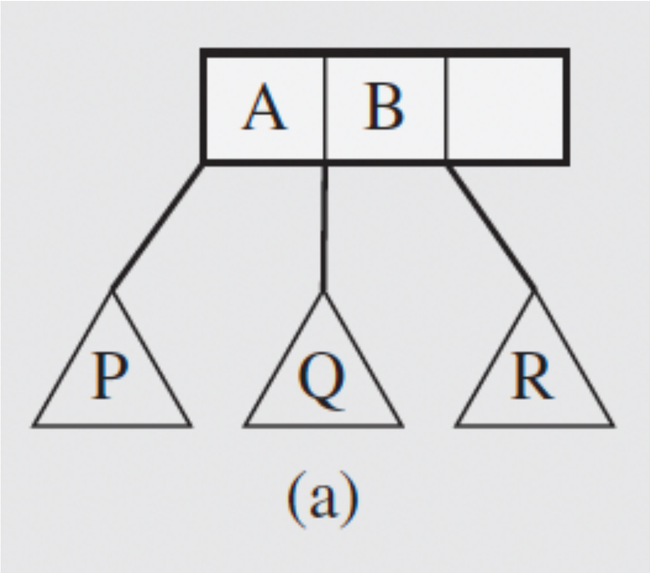
Representação de árvore 2-4

- Para representar uma árvore 2–4 como uma árvore binária, dois tipos de conexões entre nós são usados:
  - conexões entre nós que representam chaves pertencentes ao mesmo nó de uma árvore 2–4
  - conexões que representam uma relação regular pai-filhos entre nós
- *Bayer* os chamou de ponteiros horizontais e verticais ou, ponteiros *r* e ponteiros *d*
  - *vh-trees*
- *Guibas e Sedgewick* em sua estrutura usam os nomes: ponteiros vermelhos e pretos
  - *red-black trees*

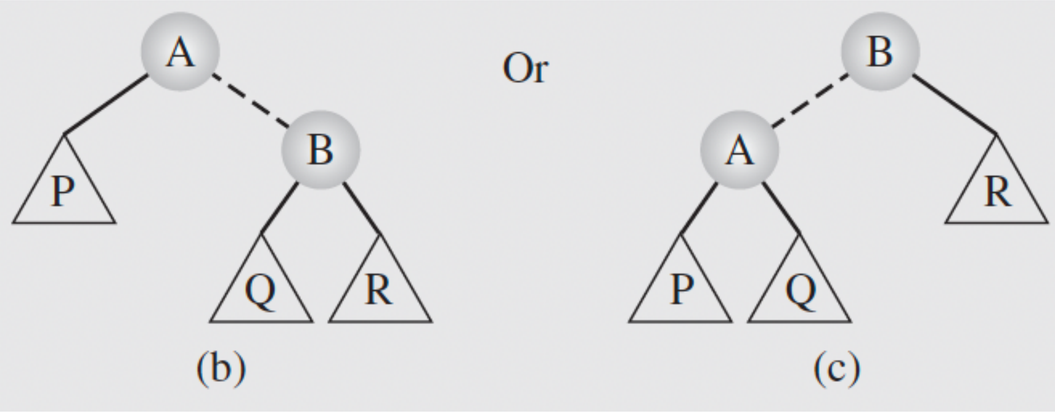
unesp

Representação de árvore 2-4

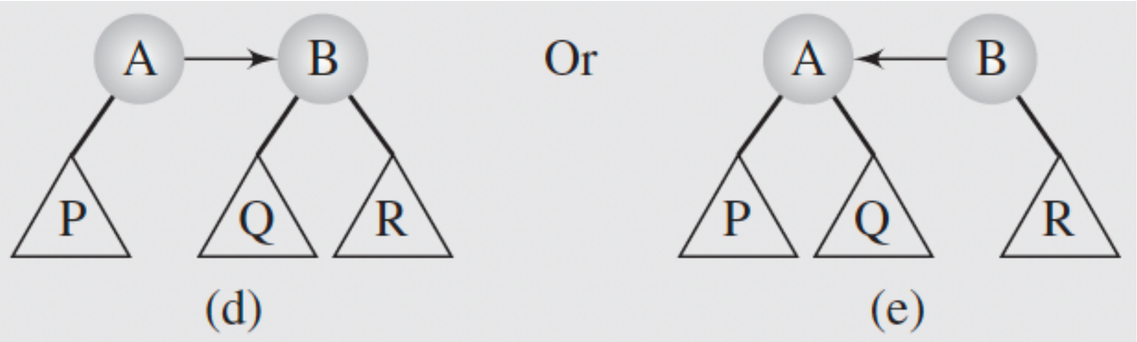
- Não apenas os nomes são diferentes, mas as árvores também são desenhadas de forma um pouco diferente. Abaixo temos (2 chaves) *3 nós:*



2-4 tree



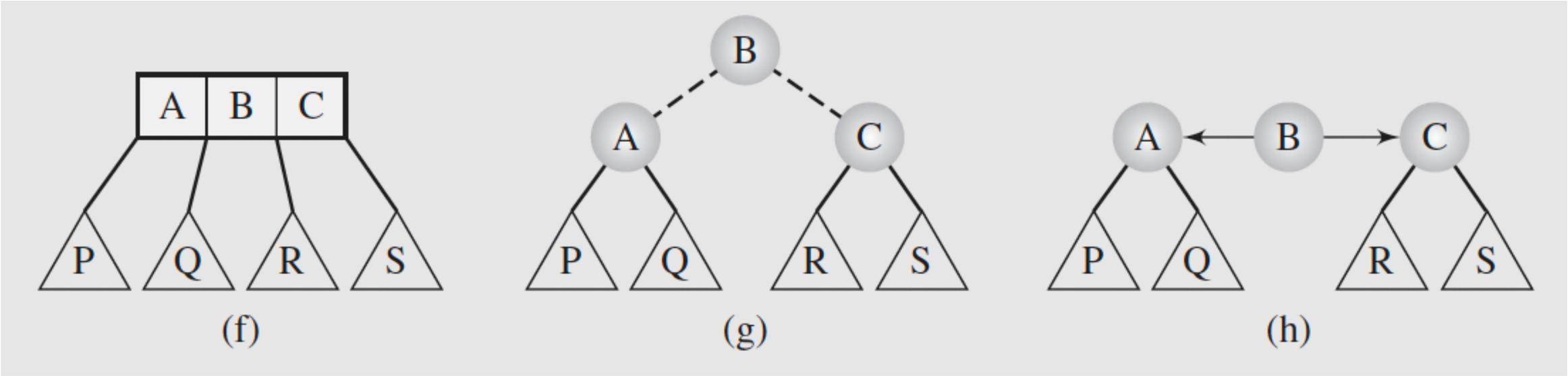
red-black trees



vh trees

Representação de árvore 2-4

- Não apenas os nomes são diferentes, mas as árvores também são desenhadas de forma um pouco diferente. Abaixo temos (3 chaves) 4 *nós:*



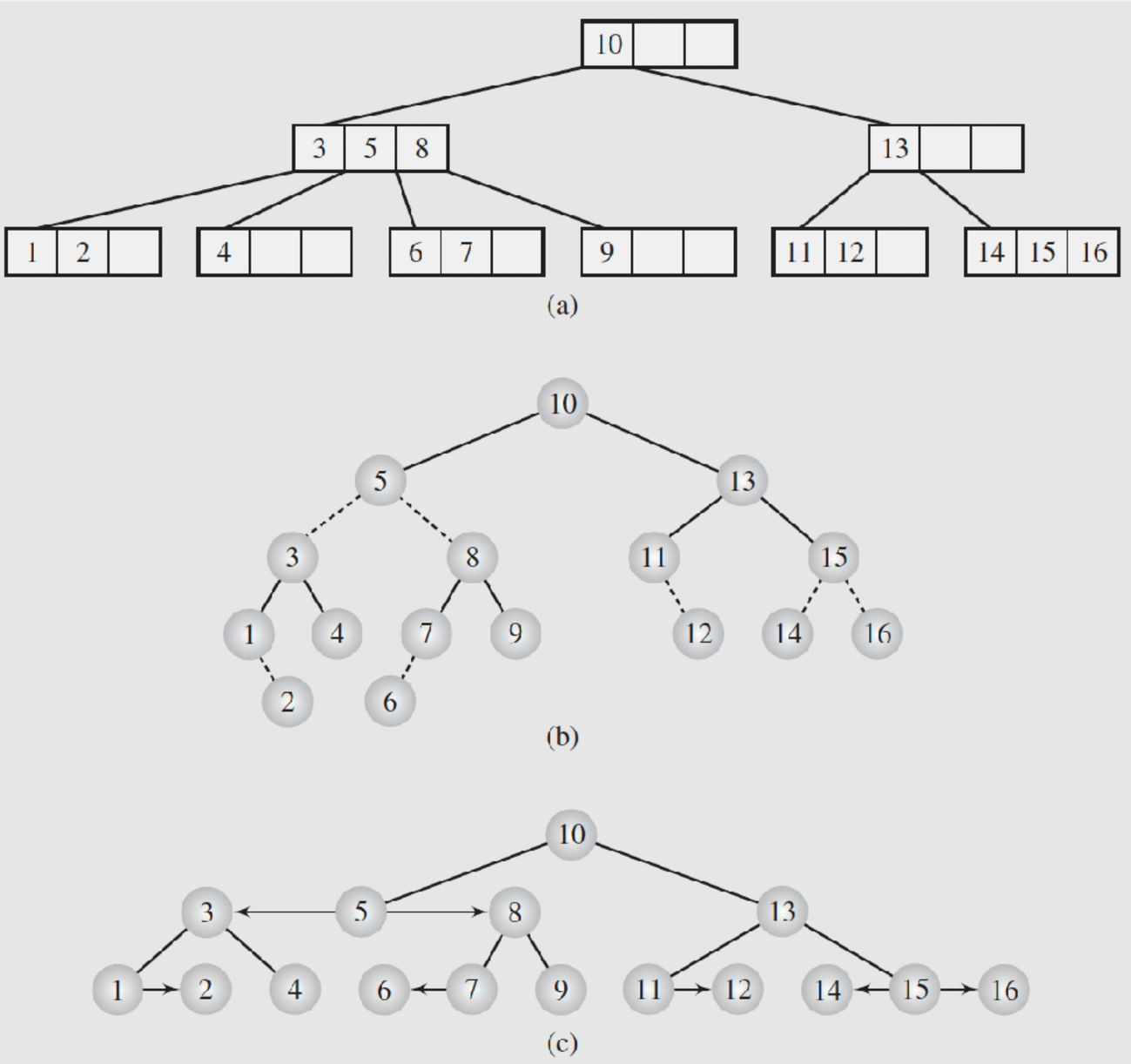
2-4 tree

red-black trees

vh trees

Representação de árvore 2-4

- Árvore 2-4 completa e suas árvores binárias equivalentes:



**Observe que as conexões vermelhas são desenhados com linhas tracejadas*

Representação de árvore 2-4

- A árvore rubro-negra (red-black) representa melhor a forma exata de uma árvore binária
- As árvores horizontais verticais ou árvores vh (*vh* tree), são melhores em manter a forma de árvores 2-4
  - e em ter folhas mostradas como se estivessem no mesmo nível
- Além disso, as árvores vh se prestam facilmente à representação de árvores B de qualquer ordem
  - as árvores rubro-negras não

unesp

Representação de árvore 2-4

- Tanto as árvores rubro-negras quanto as árvores vh são árvores binárias
- Cada nó possui dois ponteiros (conexões) que pode ser interpretado de duas maneiras
- Para fazer uma distinção entre a interpretação aplicado em um determinado contexto, um sinalizador para cada um dos ponteiros é usado
  - conexão vertical/horizontal
  - conexão vermelha/preta

unesp

Representação de árvore 2-4

- As árvores vh têm as seguintes propriedades:
  - O caminho da raiz até qualquer nó nulo contém o mesmo número de conexões verticais (ponteiro de um nó para o seu filho direto). Esta propriedade garante que a árvore permaneça balanceada em termos de altura.
  - Nenhum caminho da raiz pode ter dois links horizontais seguidos. Uma conexão horizontal em uma árvore vh é um ponteiro de um nó para outro nó que está no mesmo nível da árvore

unesp

Operações

- As operações realizadas nas **árvores vh** devem ser as mesmas que nas árvores binárias, embora sua implementação seja muito mais complicada.
- Somente a busca é a mesma: para encontrar uma chave em uma árvore vh, nenhuma distinção é feita entre os diferentes tipos de ponteiros.
- Podemos usar o mesmo procedimento de busca das árvores de busca binária:
  - se a chave for encontrada, pare.
  - Se a chave no nó atual for maior que a que procuramos, vamos para a subárvore esquerda;
  - caso contrário, iremos para a subárvore certa.

unesp

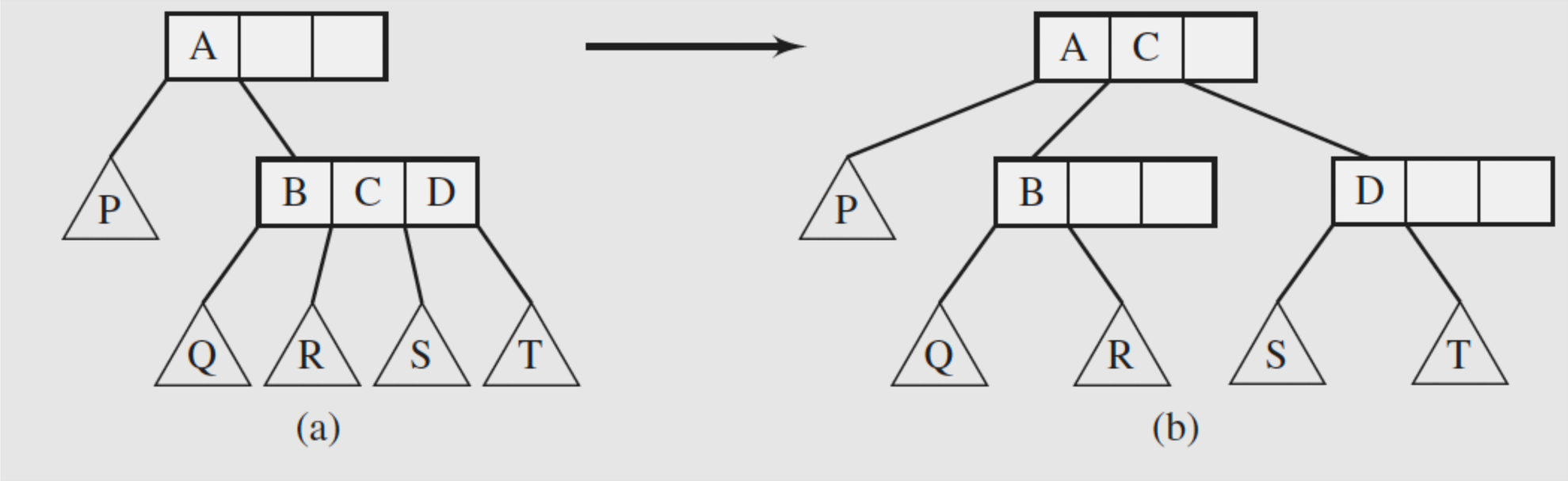
Operações - Inserção

- Uma boa ideia ao dividir árvores 2-4, é dividir os nós ao descer na árvore enquanto insere uma chave
- Se um nó 4 (com 3 chaves) nós for encontrado, ele será dividido antes de descer mais abaixo na árvore
- Como essa divisão é feita de cima para baixo, um nó 4 pode ser filho de um nó 2 ou 3 (com a exceção usual: a menos que seja a raiz)

unesp

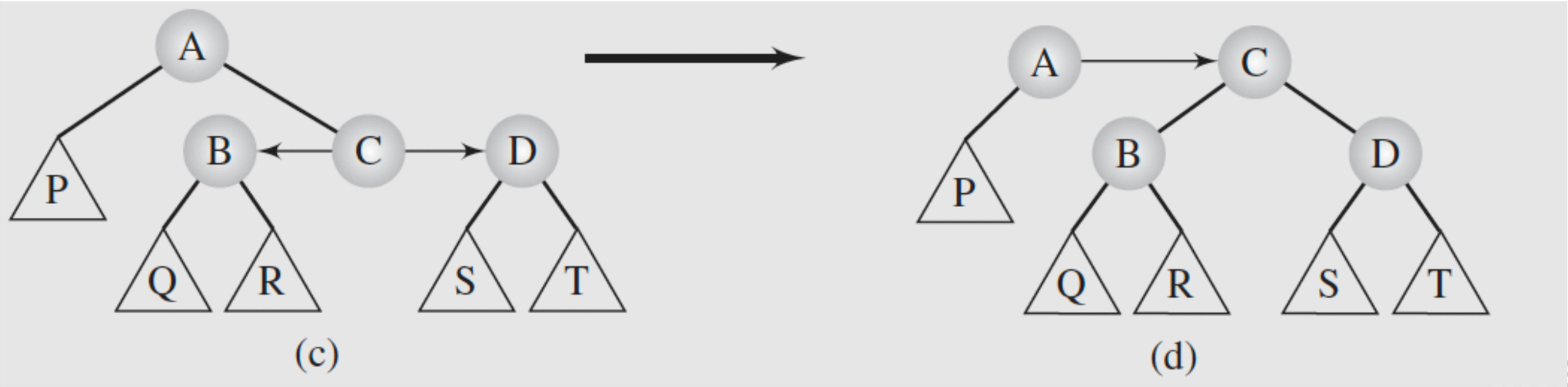
Operações - Inserção

- Dividir o nó com as chaves **B, C** e **D** requer a criação de um novo nó
  - Os dois nós envolvidos na divisão estão 4/6 cheios, após a divisão os três nós estão 4/9 cheios



Operações - Inserção

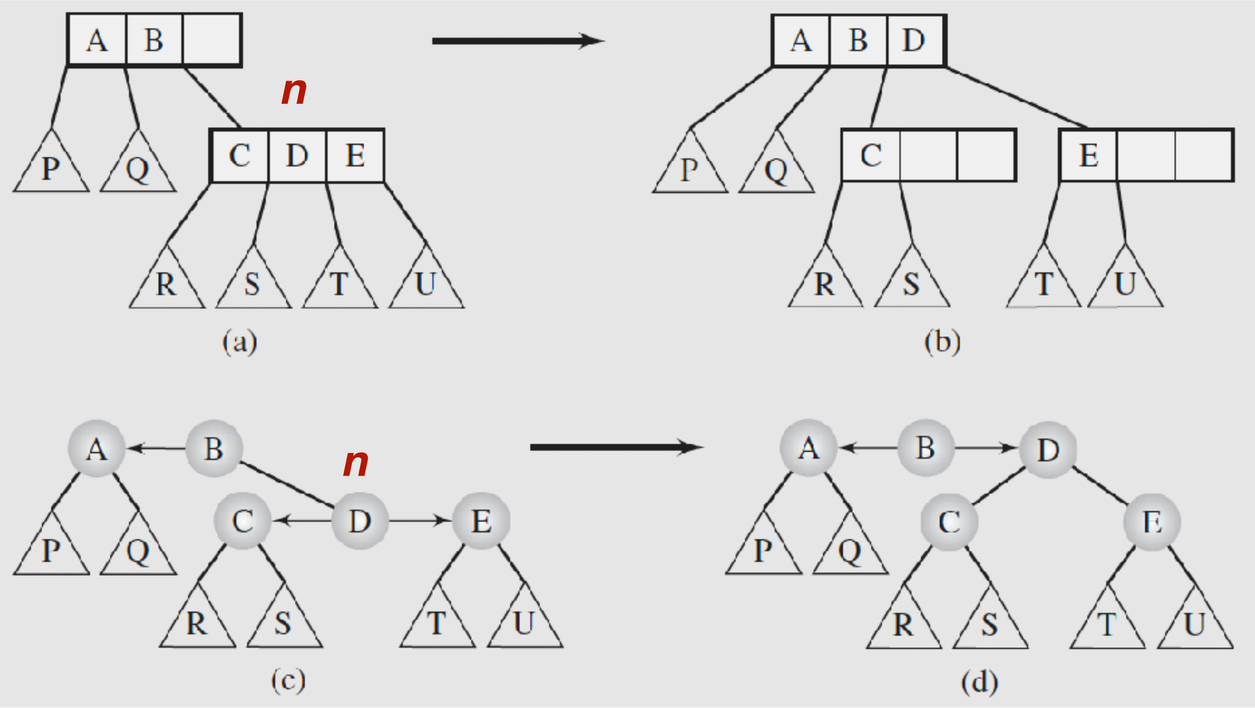
- A divisão de nós em árvores 2-4 resulta em baixo desempenho
  - No entanto, se as mesmas operações forem executadas em seus equivalentes na árvore vh, a operação será notavelmente eficiente
  - Abaixo uma mesma divisão é realizada em uma **árvore vh**, e a operação requer a mudança de apenas dois sinalizadores de horizontal para vertical e um de vertical para horizontal; assim, apenas três bits são redefinidos!



Operações - Inserção

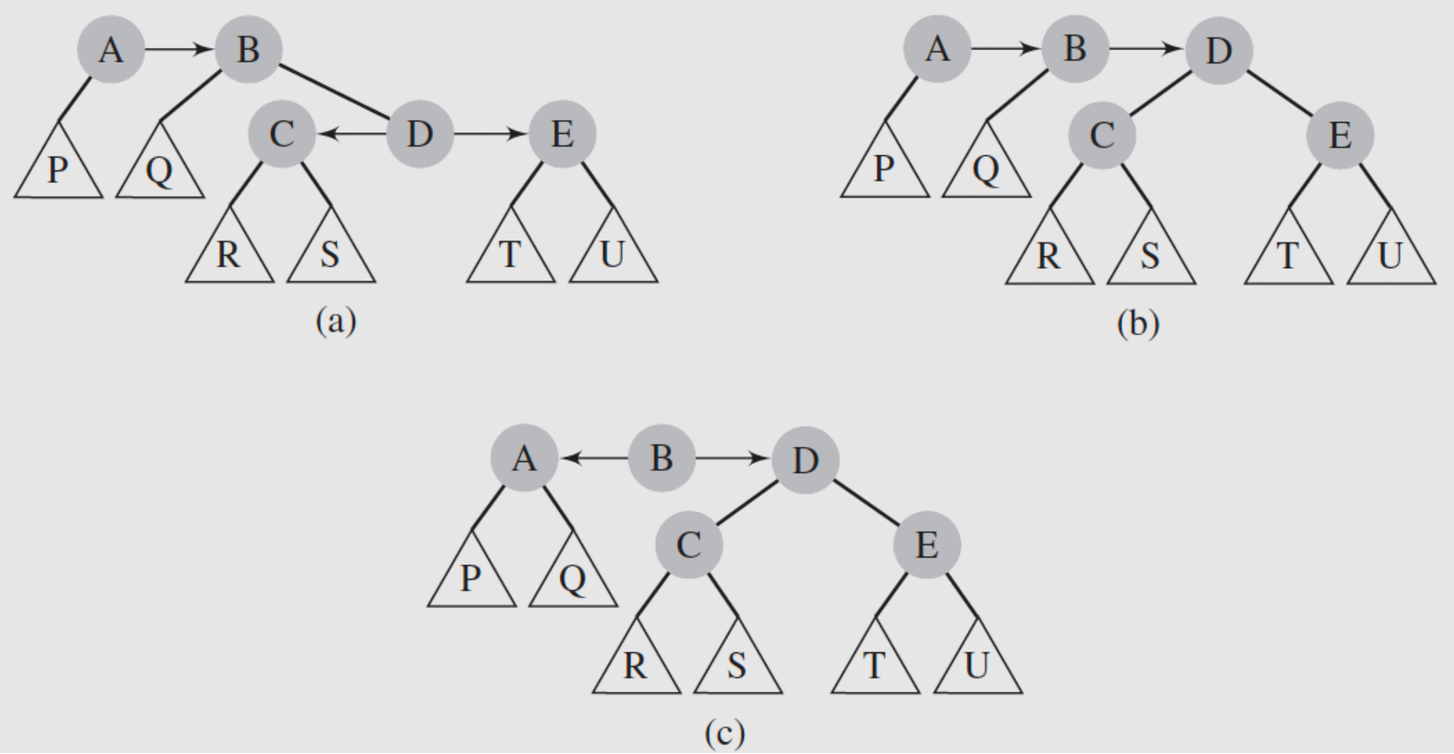
- A redefinição desses três sinalizadores sugere o algoritmo *flagFlipping*, que executa as seguintes etapas:
  - se visitarmos um nó ***n*** cujos links (conexões) são ambos horizontais, então redefinimos o sinalizador correspondente a conexão do **pai de *n*** para ***n*** para **horizontal** e ambos os sinalizadores em ***n*** para vertical

*aplicar **flagFlipping** em uma **árvore vh** requer que apenas três bits sejam redefinidos*



Operações - Inserção

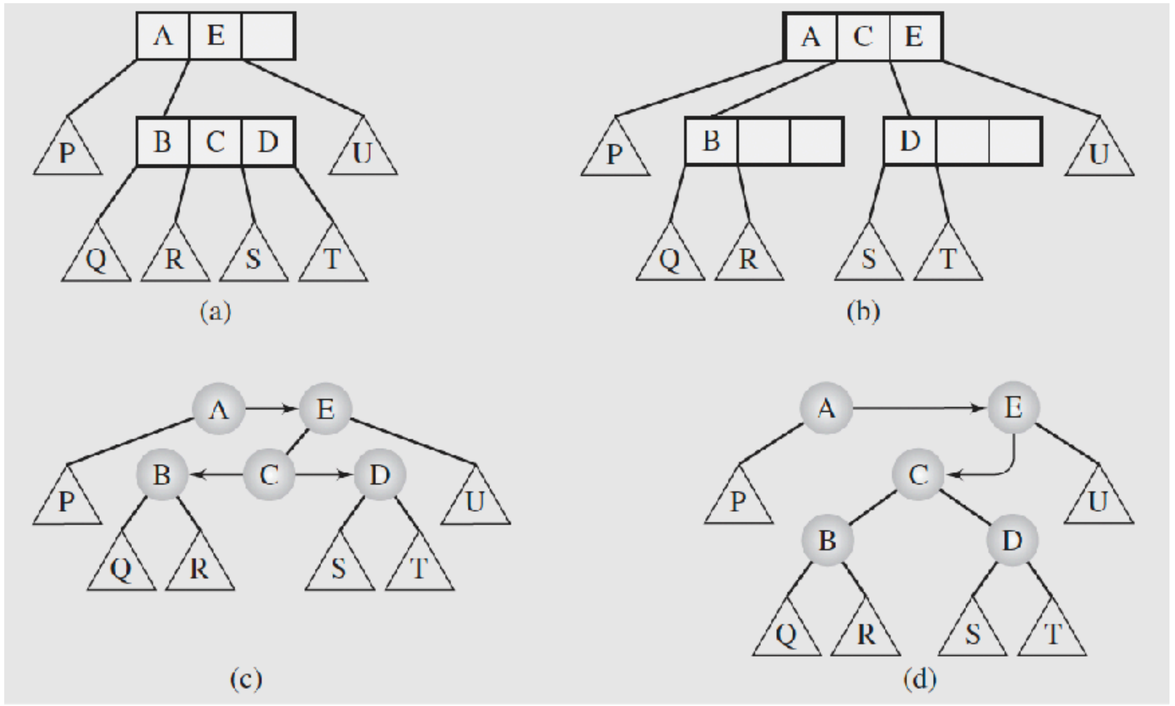
- Se procedermos como antes, alterando três sinalizadores, a árvore termina com dois links horizontais consecutivos, que não têm contrapartida em nenhuma árvore 2–4
  - Neste caso, as três inversões de flag devem ser seguidas de uma rotação; ou seja, o nó B é girado em torno do nó A (as duas flags são invertidas)



Operações - Inserção

- A aplicação do ***flagFlipping*** na árvore vh abaixo produz uma árvore com dois links horizontais consecutivos

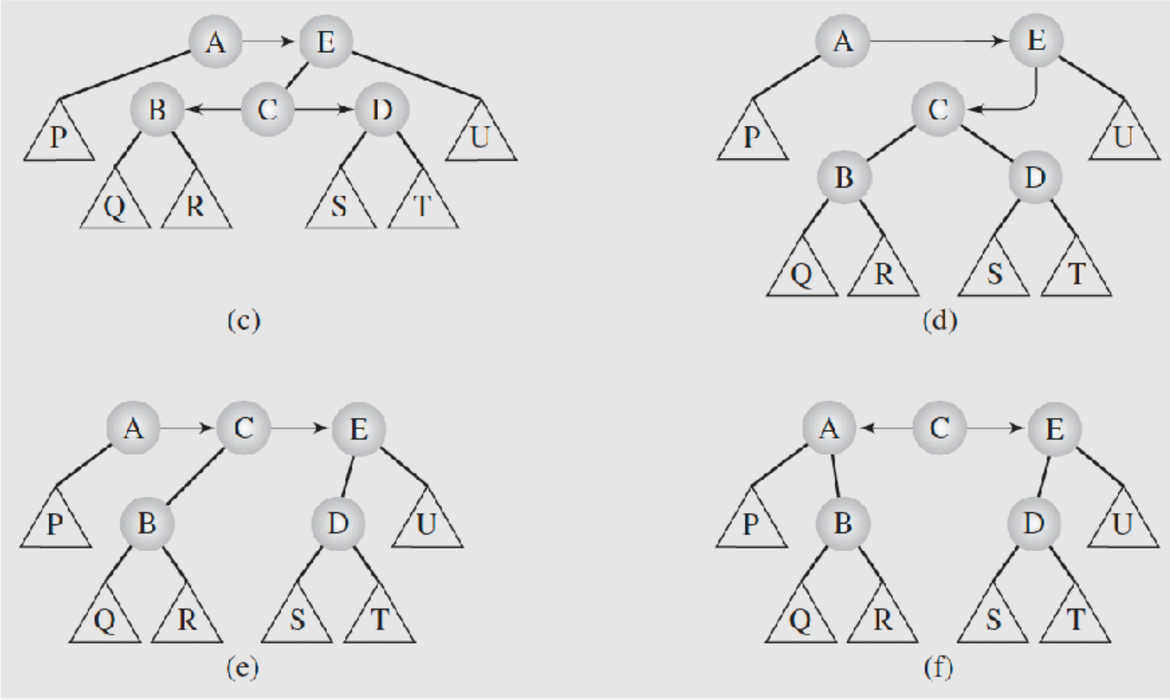
  - Para restaurar a propriedade da **árvore vh**, são necessárias duas rotações e quatro inversões de flag: o nó C é girado em torno do nó E, que é seguido por duas inversões de flag, e então o nó C em torno do nó A, que também é seguido por duas inversões de flag



Operações - Inserção

- A aplicação do ***flagFlipping*** na árvore vh abaixo produz uma árvore com dois links horizontais consecutivos

  - Para restaurar a propriedade da **árvore vh**, são necessárias duas rotações e quatro inversões de flag: o nó C é girado em torno do nó E, que é seguido por duas inversões de flag, e então o nó C em torno do nó A, que também é seguido por duas inversões de flag



Operações - Inserção

- Apresentamos configurações que levam a uma divisão
  - Este número deve ser duplicado se forem somadas as imagens espelhadas da situação que acabamos de analisar
  - O único caso em que a altura aumenta é quando a raiz tem 4 nós. Este é o nono caso de divisão

Operações - Inserção

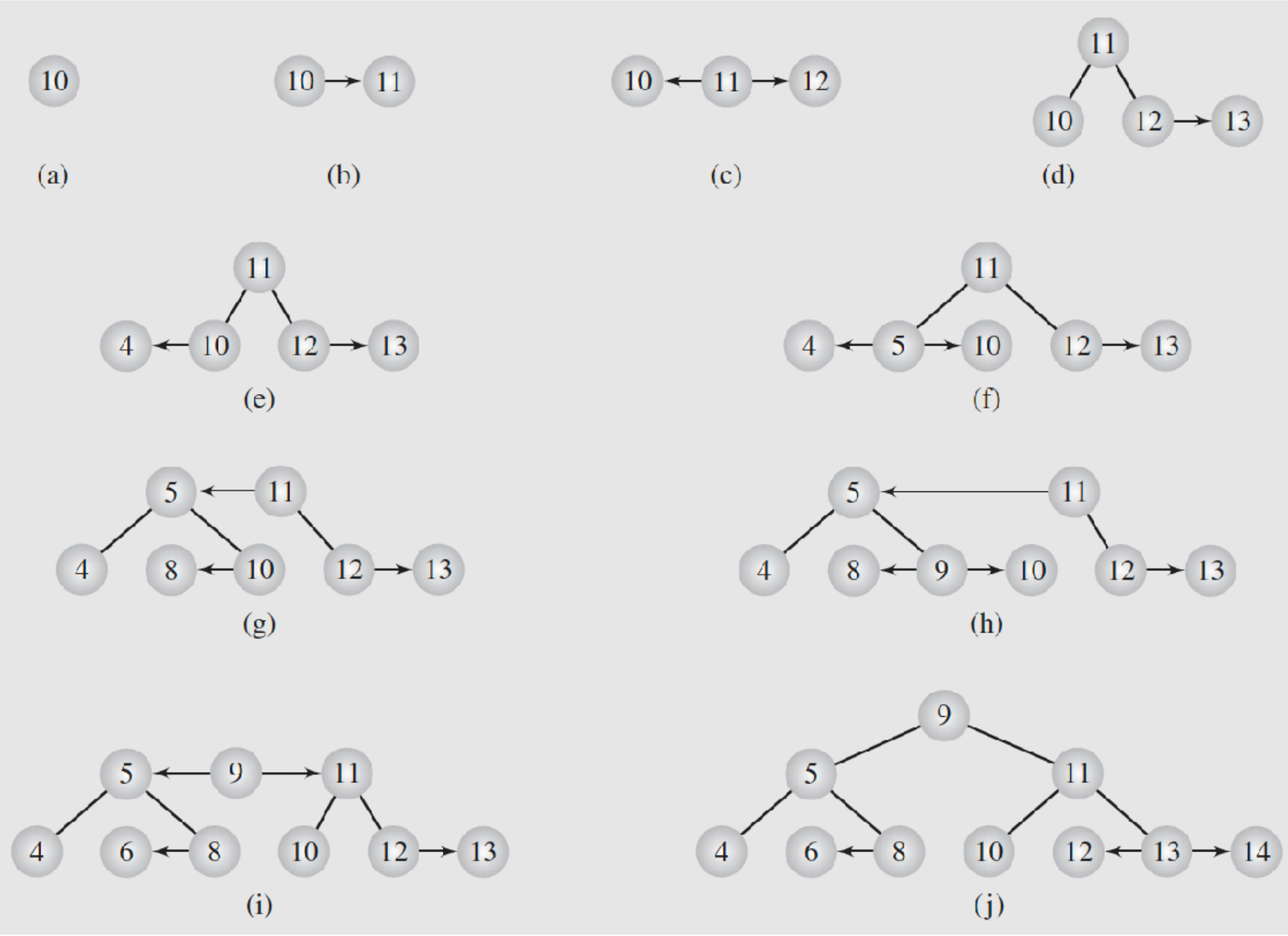
```
InsereArvoreVH(K)
    cria novoNó e inicializa
    if ArvoreVH está vazia
        raiz = novoNó
        return
         //percorre árvore fazendo divisões até o nó de inserção
    for (p = raiz, prev = null; p != null;)
        if ambos os flags de p são horizontais
            define-os para vertical // flagFlipping
            marca o link de prev (pai) conectando-o com p como horizontal
              if os links conectando o pai de prev com prev e prev com p são ambos horizontais
                    if ambos esses links são à esquerda ou ambos são à direita // Figura 7.28b (livro)
                        rotaciona prev sobre seu pai
                    else
                        rotaciona p sobre prev e depois p sobre seu novo pai // Figura 7.29d (livro)
        prev = p
        if (p->chave > K)
            p = p->esquerdo
        else
            p = p->direito

    insere o novoNó no nó prev
    marca a flag de prev correspondente ao seu link com novoNó como horizontal

    if o link do pai de prev para prev é horizontal
        rotaciona prev sobre seu pai
    else
        primeiro rotaciona novoNó sobre prev e depois novoNó sobre seu novo pai
```

Operações - Inserção

- Inserção de **10, 11, 12, 13, 4, 5, 8, 9, 6, 14.**

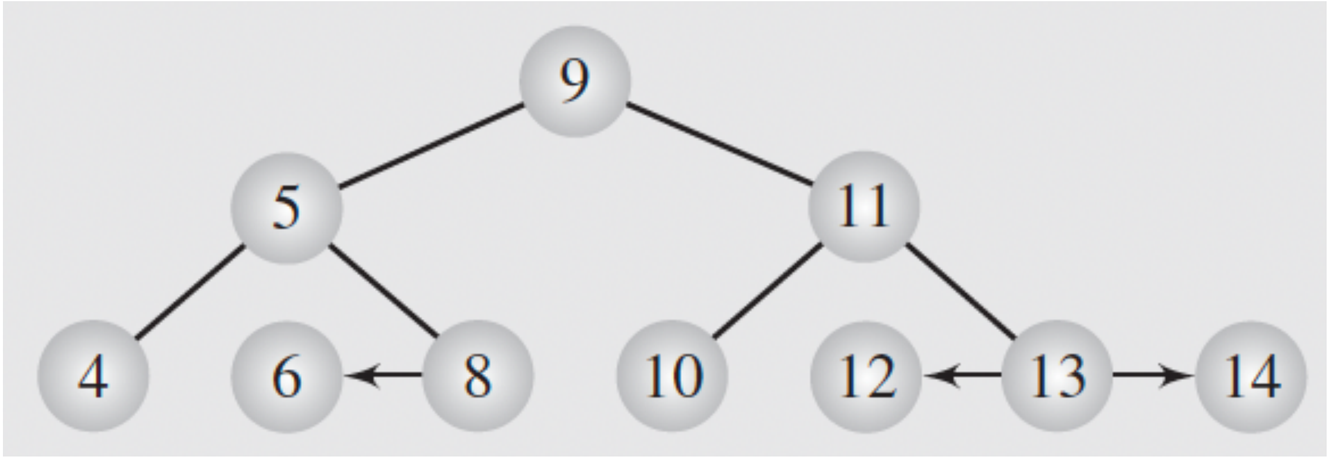


Remoção

- A remoção de um nó pode ser realizada por exclusão por cópia
  - um sucessor imediato (ou predecessor) é encontrado na árvore, copiado sobre o elemento a ser removido e o nó que contém o sucessor original é removido da árvore
  - o sucessor é encontrado indo um passo para a direita do nó que contém o elemento a ser removido e então o mais para a esquerda possível.
  - o sucessor está no último nível dos links verticais; isto é, o sucessor pode ter um descendente esquerdo acessível através de uma ligação horizontal

Remoção

- Em uma árvore de busca binária simples é fácil remover tal sucessor. Na **árvore vh**, entretanto, pode não ser assim
  - se a conexão do sucessor com o pai é estabelecido através de link vertical, então a remoção deste sucessor pode violar a propriedade da árvore vh. Exemplo: remover o nó 9 em que o sucessor é o 10.

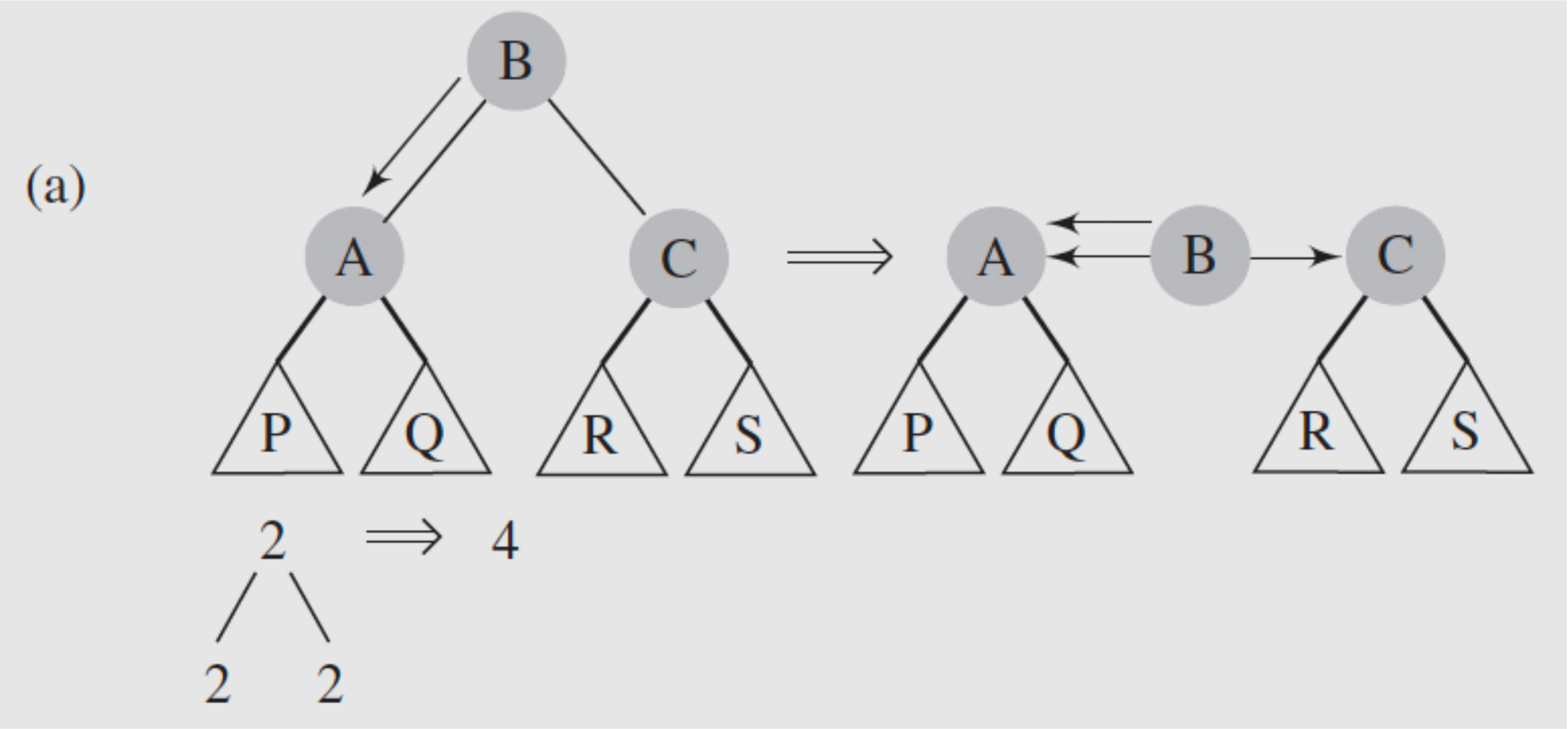


Remoção

- Uma maneira de evitar o problema é garantir que, ao procurar pelo sucessor de um determinado nó, sejam executadas transformações de árvore que tornem uma **árvore vh** uma árvore válida e façam com que o sucessor sem descendentes seja conectado ao seu pai com um ligação horizontal
  - São 5 casos

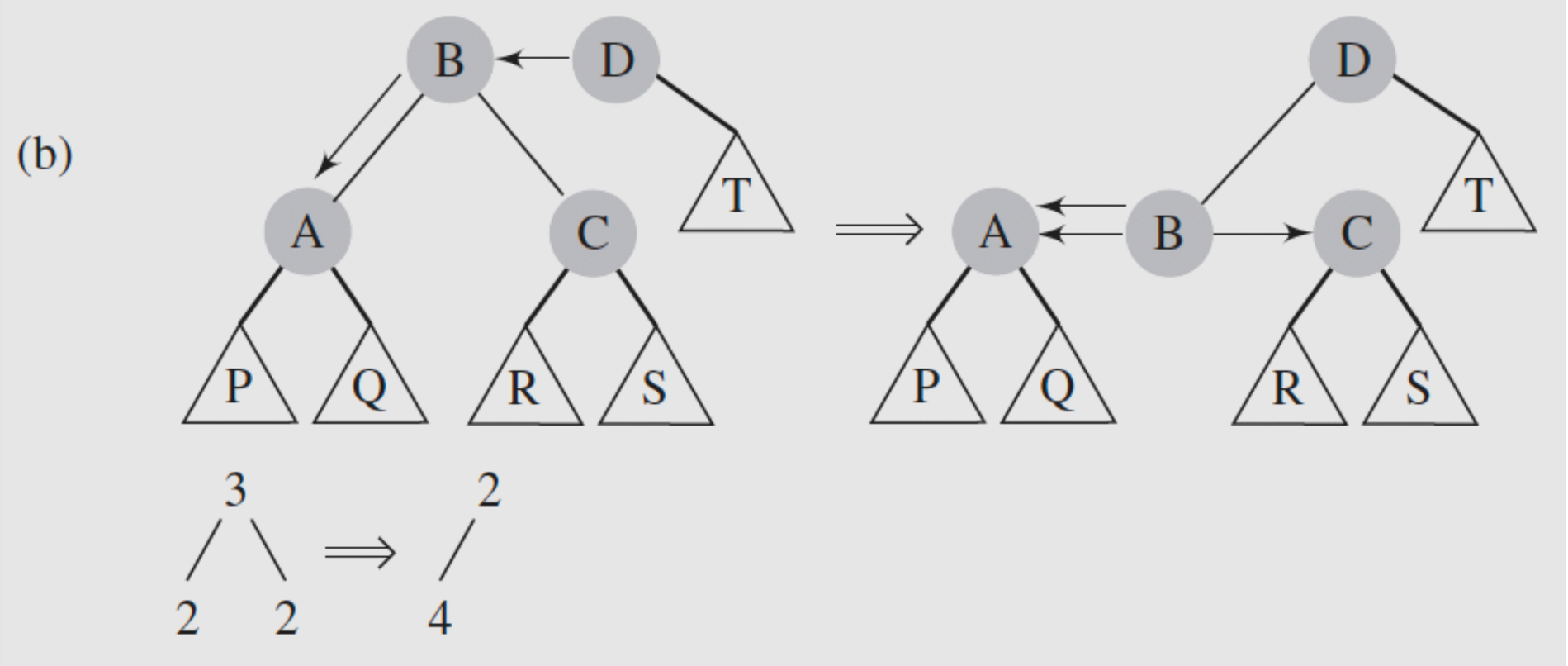
Remoção

- **Caso 1:** dois irmãos que são nós 2-nó têm um pai que é um 2-nó; o nó e seus descendentes são fundidos em um 4-nó, o que requer apenas duas mudanças de flag



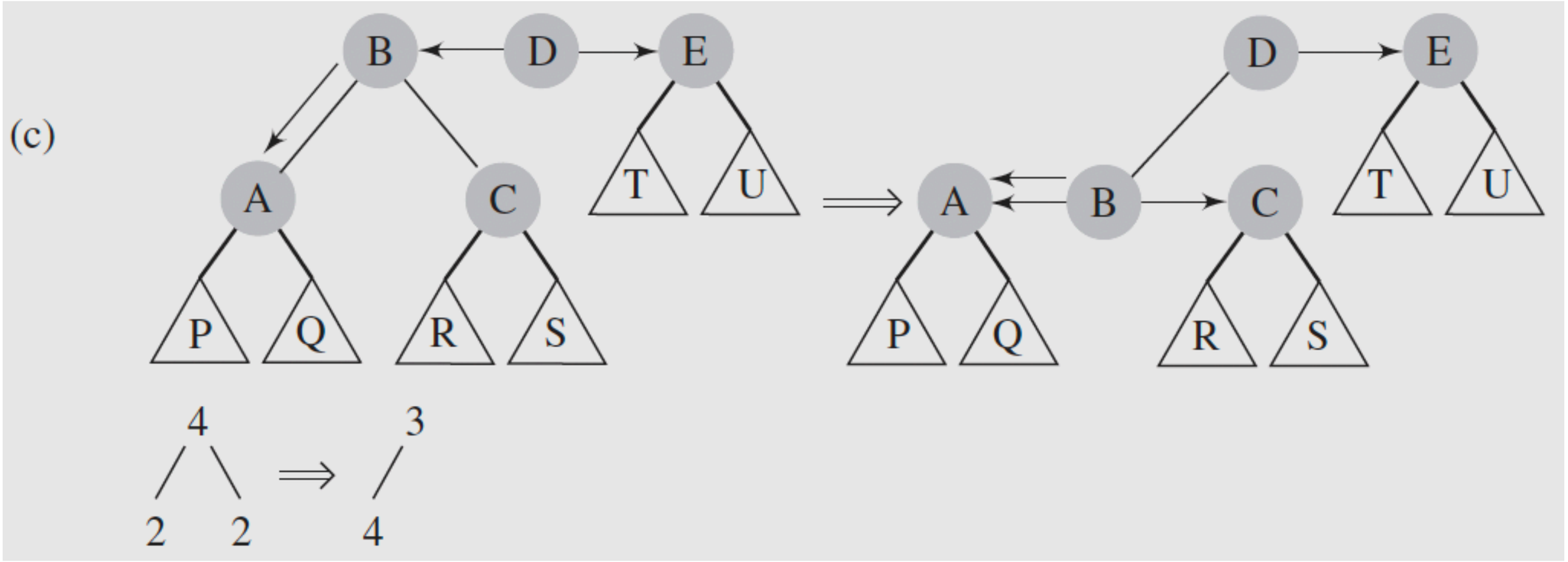
Remoção

- **Caso 2:** Um 3-nó com dois descendentes que são 2-nós é transformado dividindo o 3-nó em dois 2-nós e criando um 4-nó a partir dos três 2-nós, como indicado na figura, ao custo de três mudanças de flag



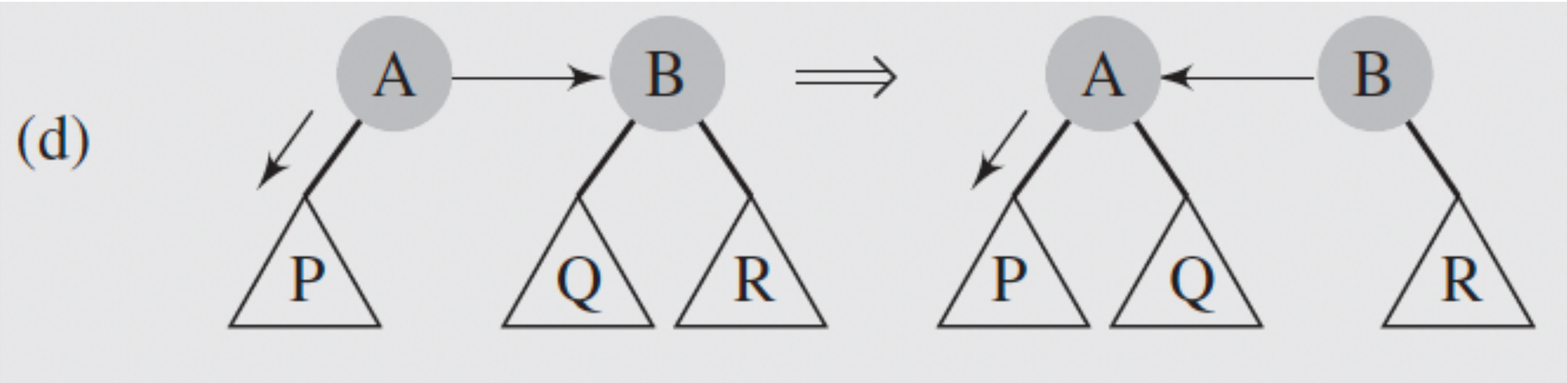
Remoção

- **Caso 2a:** Um 4-nó com dois descendentes que são 2-nós é dividido em um 2-nó e um 3-nó, e os três 2-nós são fundidos em um 4-nó. Isso requer as mesmas três mudanças de flag que no Caso 2



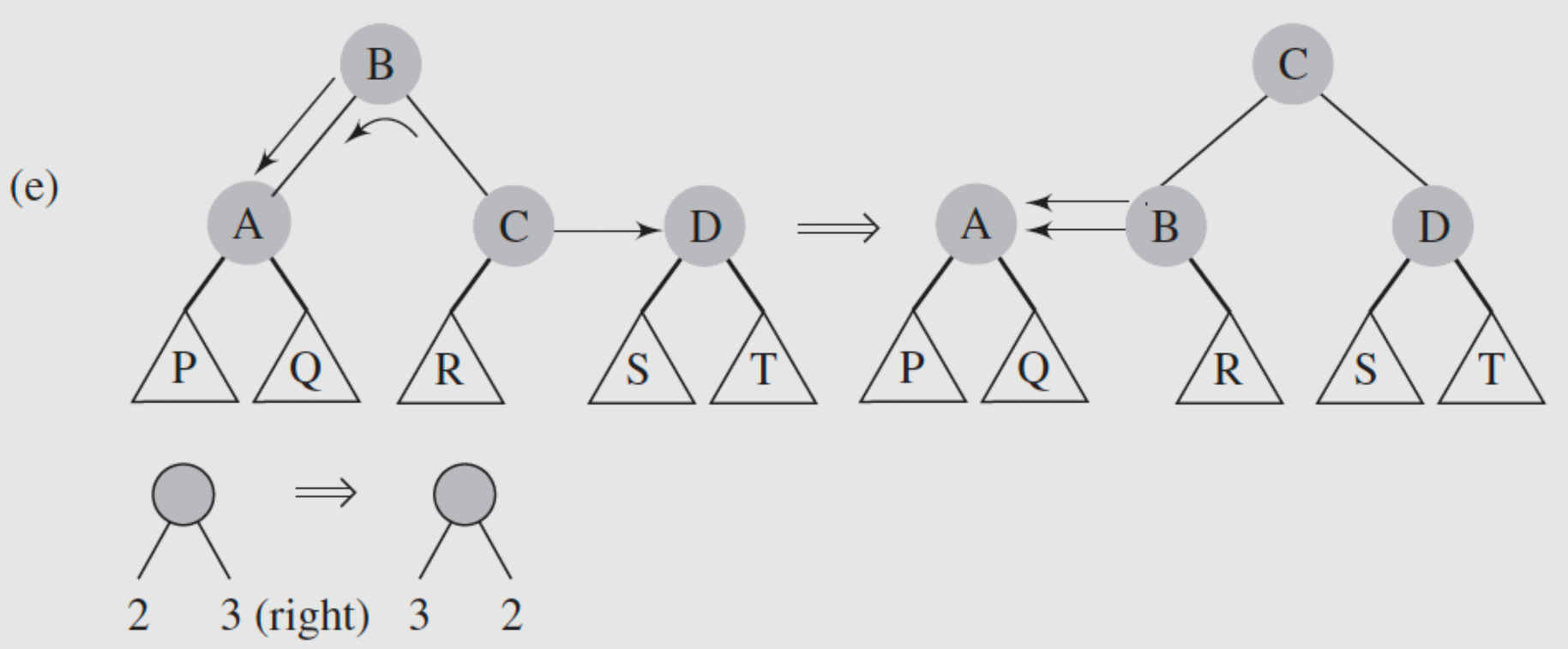
Remoção

- **Caso 3:** Quando o final de um 3-nó com um link de saída horizontal é alcançado, a direção do link é revertida por meio de uma rotação e duas mudanças de flag



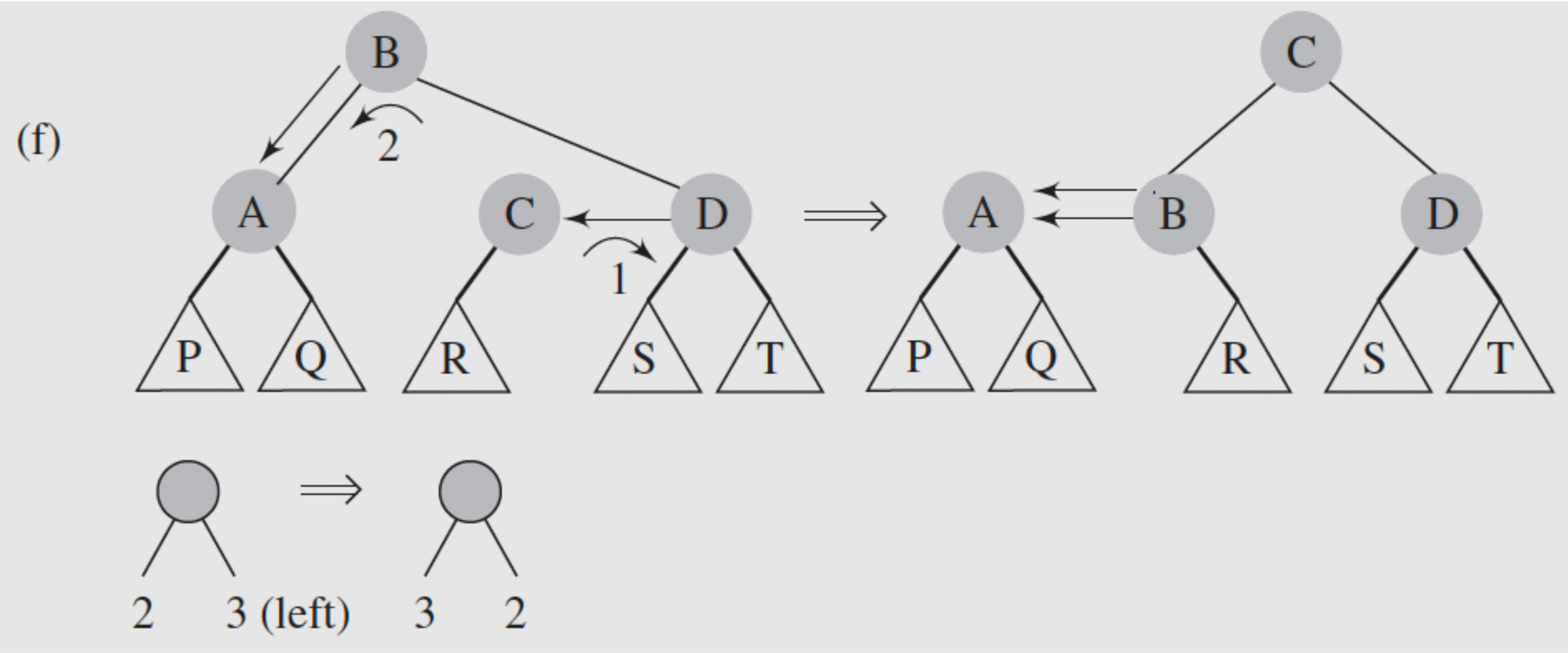
Remoção

- **Caso 4:** Um 2-nó tem um irmão que é um 3-nó (pode haver pais de qualquer tamanho). Por meio de uma rotação—C sobre B—e duas mudanças de flag, o 2-nó é expandido para um 3-nó e o irmão 3-nó é reduzido para um 2-nó



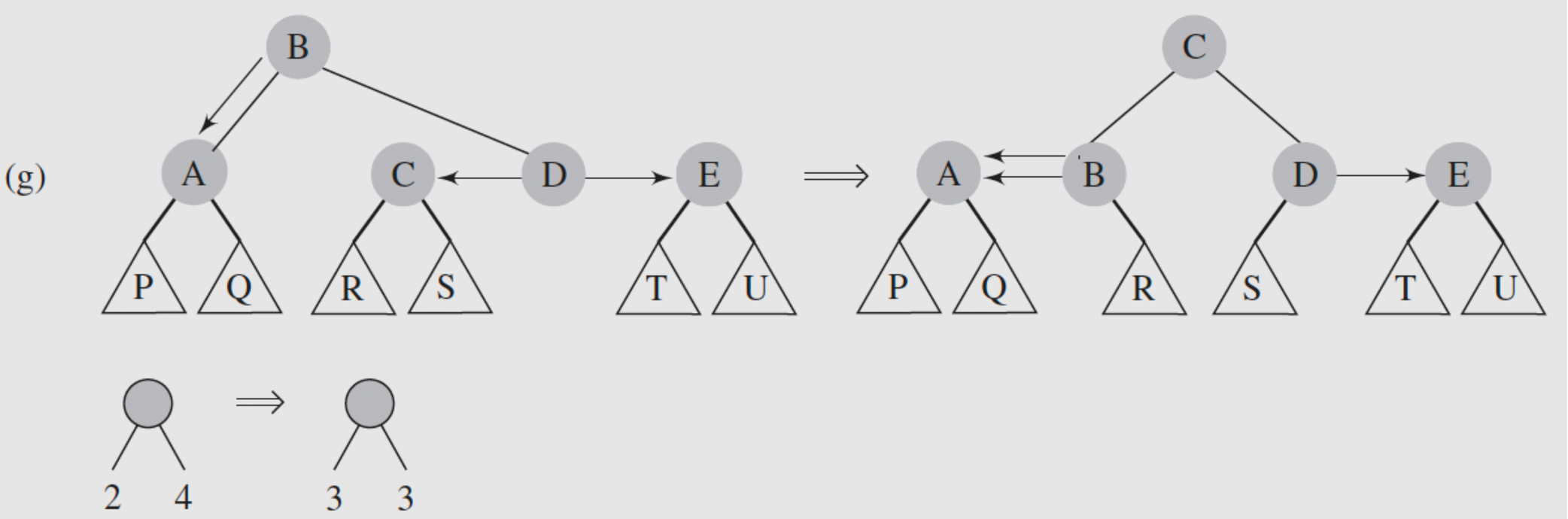
Remoção

- **Caso 5:** Semelhante ao Caso 4, exceto que o irmão 3-nó tem uma direção diferente. A transformação é realizada por meio de duas rotações—primeiro, C sobre D e depois C sobre B—e duas mudanças de sinalizador



Remoção

- **Caso 5a:** Um 2-nó tem um irmão que é um 4-nó (qualquer pai). O 2-nó é transformado em um 3-nó e o 4-nó é transformado em um 3-nó com as mesmas transformações que no Caso 5



Remoção

- Veja o exemplo de remoção da **Figura 7.32** do livro

Referências

- Adam Drozdek, Data Structures and Algorithms in C++. 4 ed. 2012