Agenda

Unidade 4: Introdução a árvores

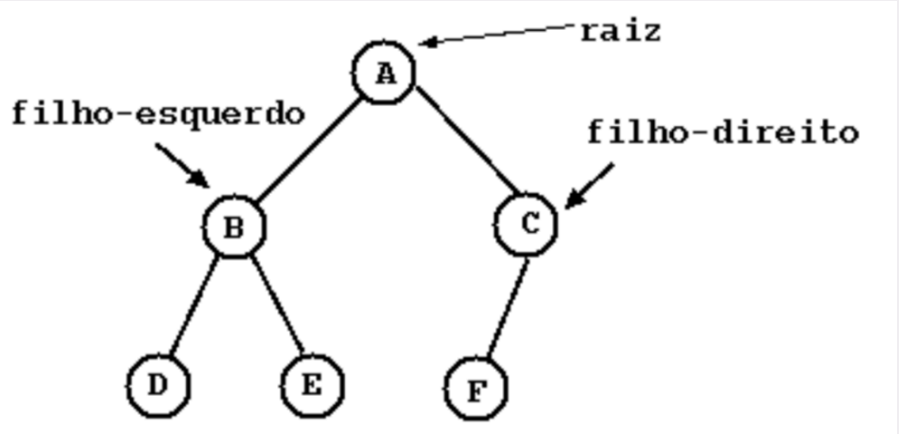
1. Definições e representações básicas
2. Pesquisa sequencial e binária
3. **Árvores binárias**
4. Percurso em árvores binárias
5. Árvores binária de busca
6. Implementações

Árvores Binárias

- Qualquer árvore pode ser transformada numa árvore binária
  - Focaremos em árvore binárias pois elas são mais fáceis de armazenar e manipular
- Uma **árvore binária** é constituída de um conjunto finito de nós.
- Cada nó pode ter no **máximo dois** filhos.

Árvores Binárias

- Uma **árvore-binária** é constituída por um conjunto finito de nós. Se o conjunto for vazio, a árvore diz-se **vazia**, caso contrário obdece às seguintes regras:
  - possui um nó especial, a **raiz** da árvore.
  - cada nó possui no máximo dois filhos, **filho-esquerdo** e **filho-direito**.
  - cada nó, excepto a raíz, possui exactamente um **nó-pai**.
- ou dito de forma mais simples,
  Árvores binárias são árvores em que cada nó tem 0, 1 ou 2 filhos



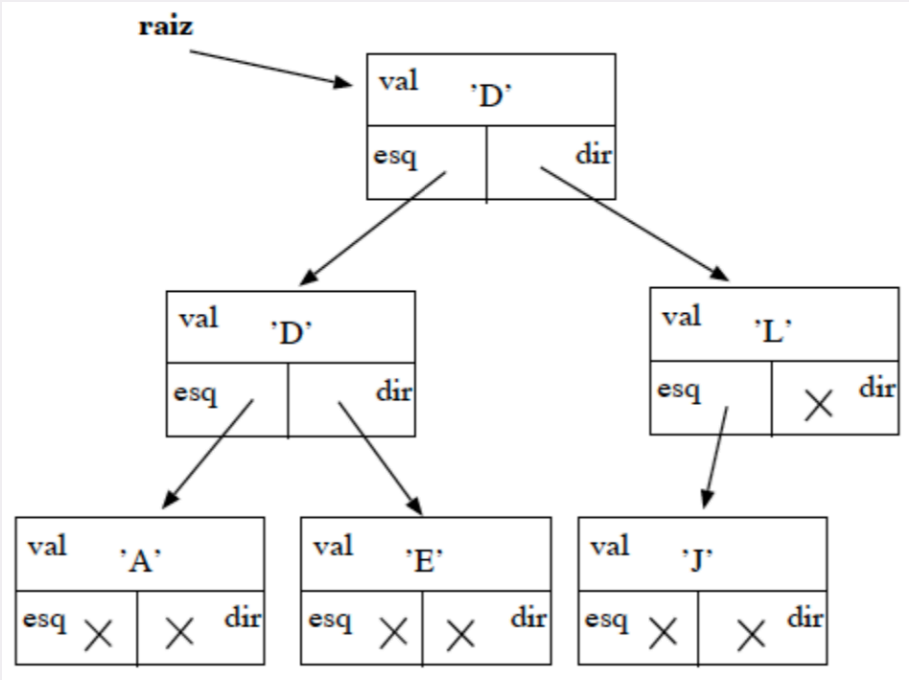
Árvores Binárias

- De **maneira recursiva**, podemos definir uma árvore binária como sendo:
  - uma árvore **vazia**; ou
  - um nó **raiz** tendo **duas sub-árvores**, identificadas como a sub-árvore da direita e a sub-árvore da esquerda.

Criando uma árvore vazia

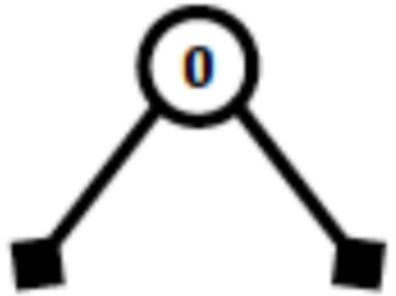
- Representação encadeada de árvores binárias
- Um nó será formado por um registro composto de:
  - Campo de informação
  - Ponteiro para nó esquerdo
  - Ponteiro para nó direito

```
struct arvore {
     struct arvore *esq;
     struct arvore *dir;
     int dado;
};
```



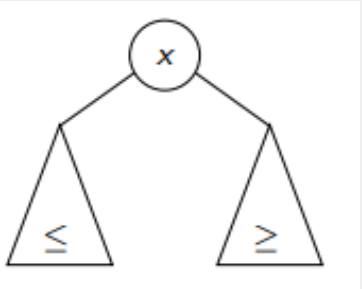
Criando uma árvore vazia

```
struct arvore * cria_arvore(void) {
    struct arvore *novo;
    novo = malloc(sizeof(struct arvore));
    novo->esq = NULL;
    novo->dir = NULL;
    novo->dado = 0;
}
```



Árvores Binárias

- O que é árvore binária de pesquisa?
  - Toda árvore binária que satisfaz a propriedade árvore-binária de pesquisa:
    - Seja x um vértice qualquer da árvore binária de pesquisa
    - Se y é um nó da sub árvore da esquerda, então key[y] ≤ key[x]
    - Se y é um nó da sub árvore da direita, então key[y] ≥ key[x]



Árvores Binárias

- Operações: seleção, inserção, remoção, entre outras
- As Operações levam tempo proporcional à altura da árvore.
- Para uma árvore binária completa de *n* nós, tais operações levam $\Theta(\lg n)$.
- Todavia, se a árvore é uma lista de nós, as operações podem levar $\Theta(n)$ no pior caso.

Inserindo um elemento em uma árvore binária de pesquisa

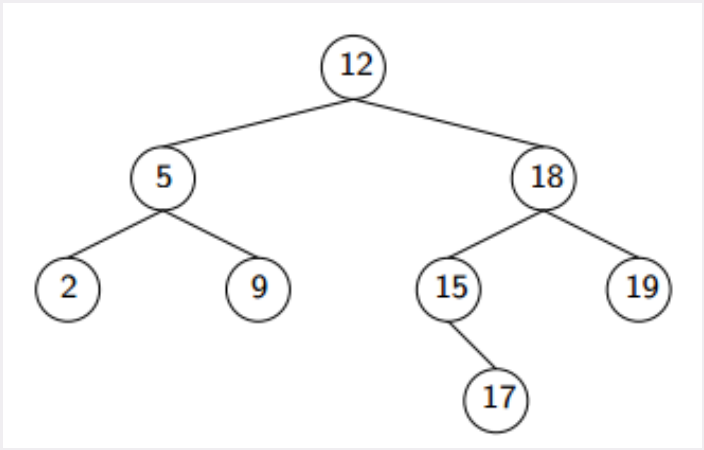
- O primeiro nó a ser inserido será a raiz da árvore.
- A cada inserção de um novo nó será verificado se ele é maior ou menor que o nó raiz.
  - Se for menor, será direcionado para o filho da esquerda.
  - Se for maior, para o filho da direita.
- Uma nova verificação será feita seguindo o mesmo critério, até que uma folha da árvore tenha sido percorrida, quando então o processo é interrompido.

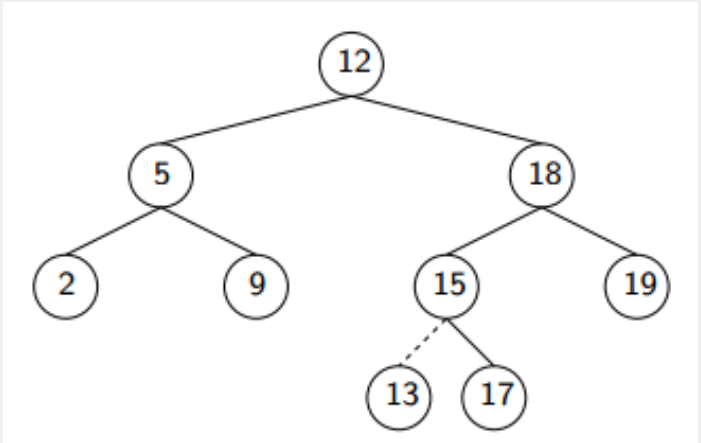
Inserindo um elemento em uma árvore binária de pesquisa

```
void insere_elemento(struct arvore *t, int D) {
    if(D < t->dado) {
        if(t->esq) { insere_elemento(t->esq, D); }
        else { /* achou local de inserção */
            struct arvore *novo = cria_arvore();
            novo->dado = D;
            t->esq = novo;
        }
    } else if(D > t->dado) {
        if(t->dir) { insere_elemento(t->dir, D); }
        else {
            struct arvore *novo = cria_arvore();
            novo->dado = D;
            t->dir = novo;
        }
    } else {
        printf("elemento já existe na árvore\n");
    }
}
```

Inserindo um elemento em uma árvore binária de pesquisa

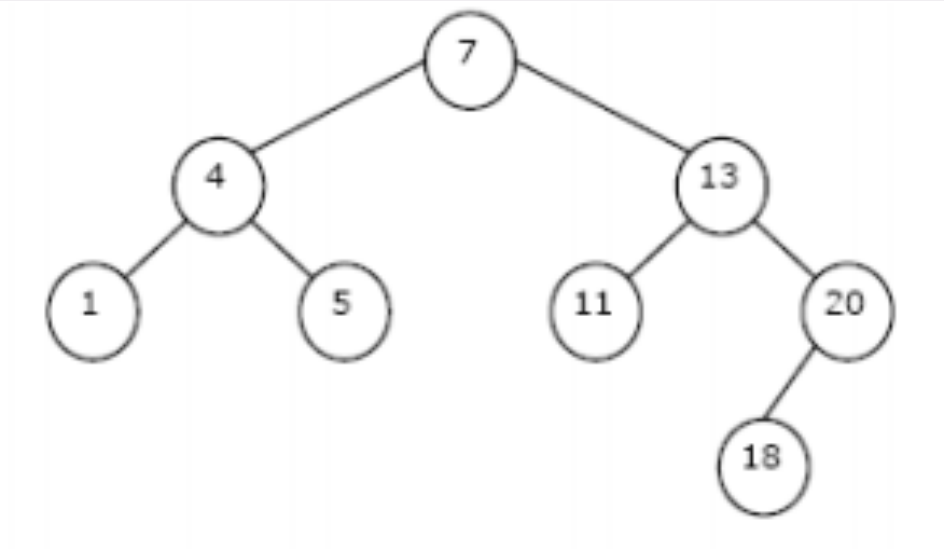
- Exemplo:
  - Inserir o valor 13 na árvore abaixo





Inserindo um elemento em uma árvore binária de pesquisa

- Exemplo:
  - Inserir os seguintes elementos: 7, 13, 20, 4, 1, 18, 5, 11



Exercício

- Construir uma arvore com as chaves:
  - (10, 20, 30, 5, 3, 50, 40, 70, 60, 90)
- Depois de construída a árvore acima, insira os nós:
  - (1, 15, 65)
- Considerando a árvore do exercício acima, mostre o resultado dos três tipos de caminho (ordem, pré-ordem e pós-ordem)

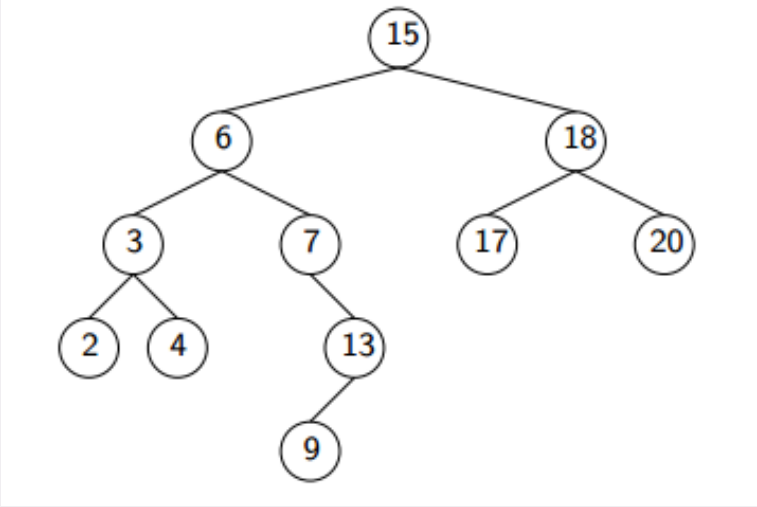
Procurando um elemento em uma árvore binária de pesquisa

- Dados um ponteiro para a raiz T e uma chave k, o procedimento Search(T,k) retorna um ponteiro para um objeto com a chave k se um existe, ou NIL caso contrário.

```
Search(T, k)
        if x = NIL or key[x] = k
                return x
        if k < key[x]
                Search(left[x], k)
        else
                Search(right[x], k)
```

Procurando um elemento em uma árvore binária de pesquisa

- O procedimento de procura por elementos em uma árvore binária é grandemente facilitado pela sua arquitetura.
- Tomemos como exemplo a árvore abaixo.



Procurando um elemento em uma árvore binária de pesquisa

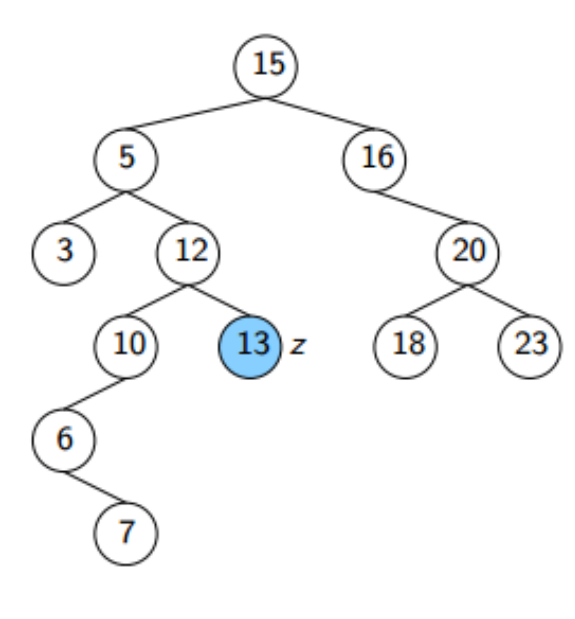
- A busca pela chave 9, segue o caminho 15 → 6 → 7 → 13 -> 9
- Na realidade, nesta árvore qualquer número pode ser encontrado com um total de 5 comparações no máximo, pois este valor corresponde ao número de níveis da árvore mais um.
- A procura em uma árvore binária é realizada em O(logN).

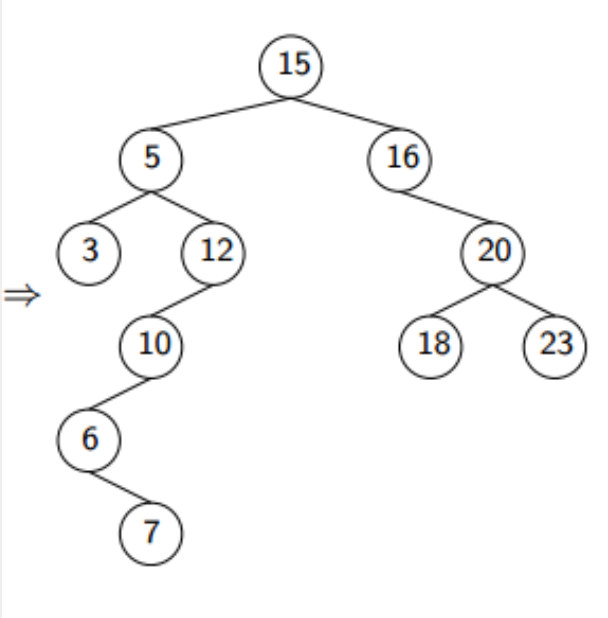
Deletando um elemento em uma árvore binária de pesquisa

- O procedimento de deleção de um nó z consiste em 3 casos.
  1. Se z não tem filhos, modificar o pai de z, p[z], para que este aponte para NIL.
  2. Se z possui apenas um filho, fazemos o pai de z apontar para o filho de z.
  3. Se z possui dois filhos, colocamos no lugar de z o seu sucessor, o que com certeza não possui filho à esquerda.

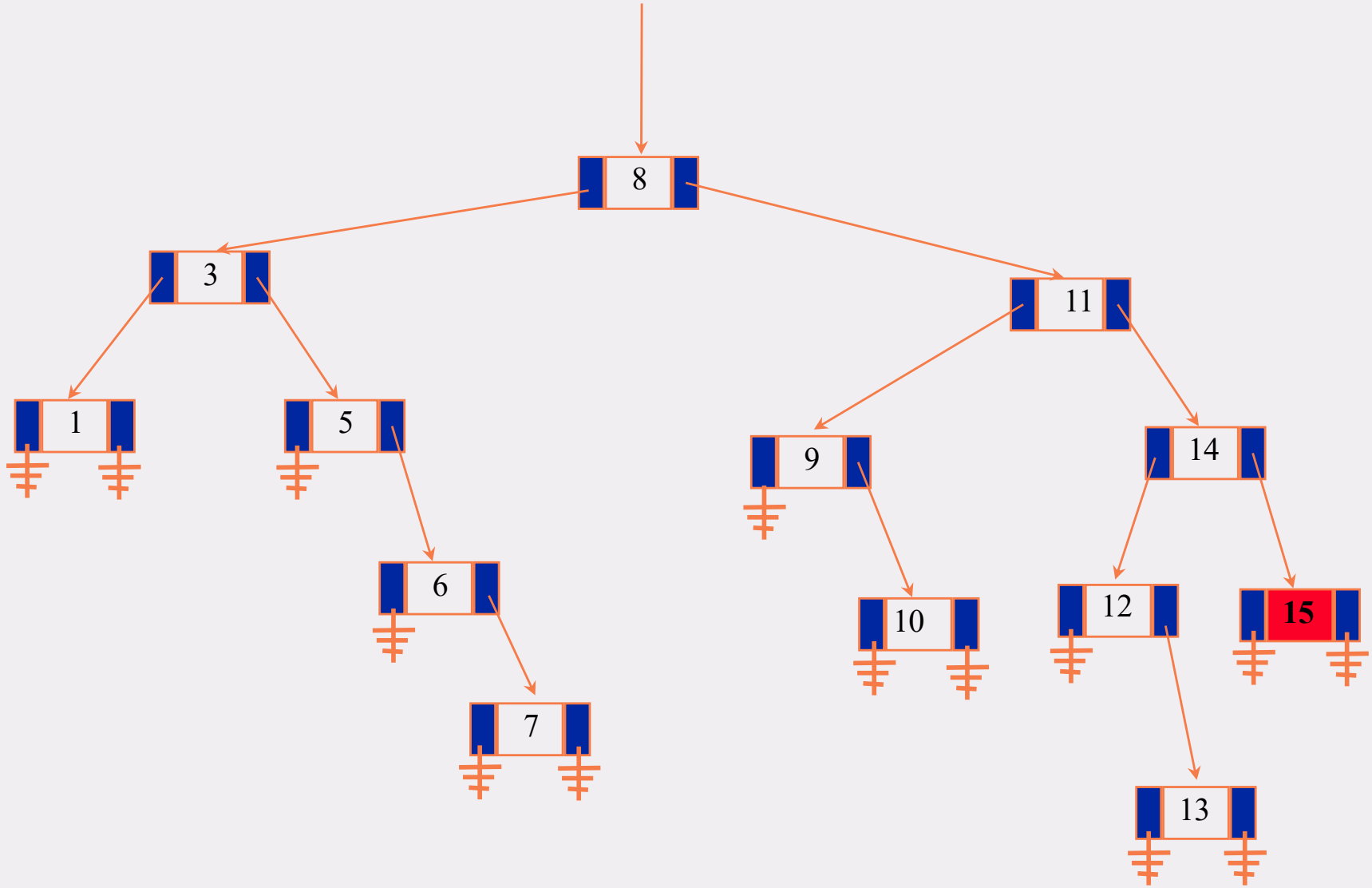
Deletando um elemento em uma árvore binária de pesquisa

- Caso 1: z não possui filhos

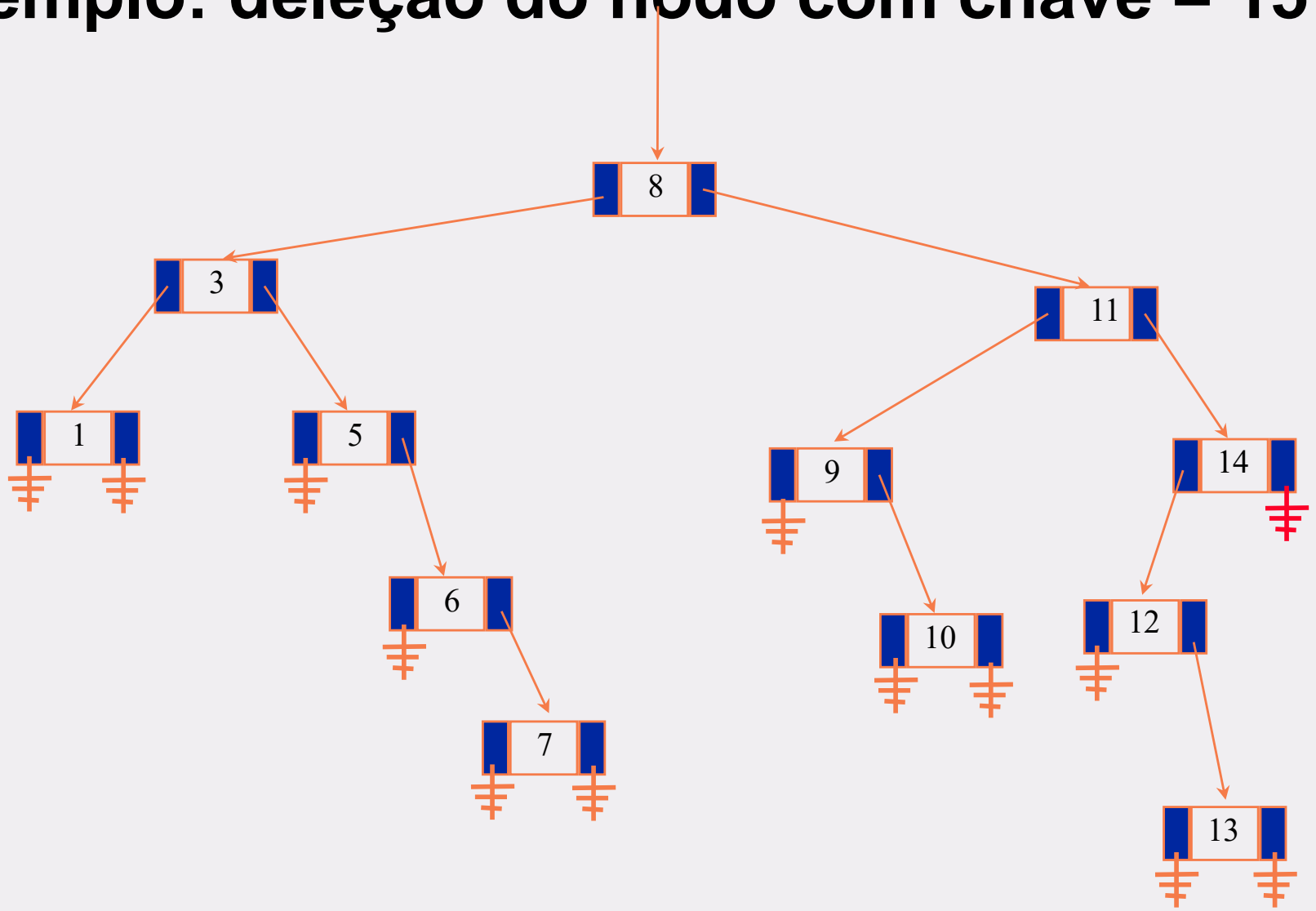




Exemplo: deleção do nodo com chave = 15

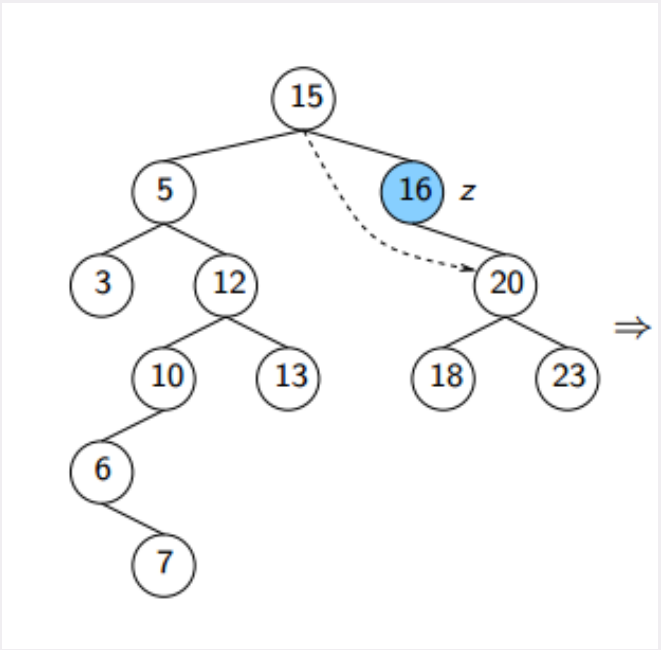


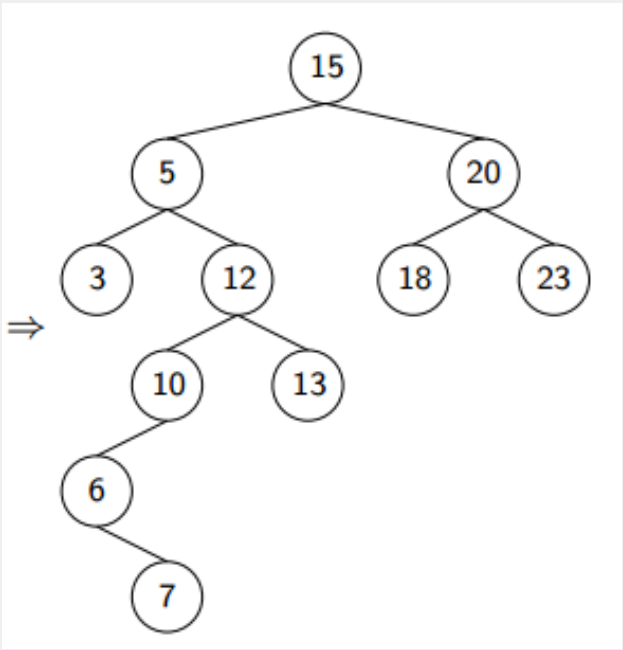
Exemplo: deleção do nodo com chave = 15



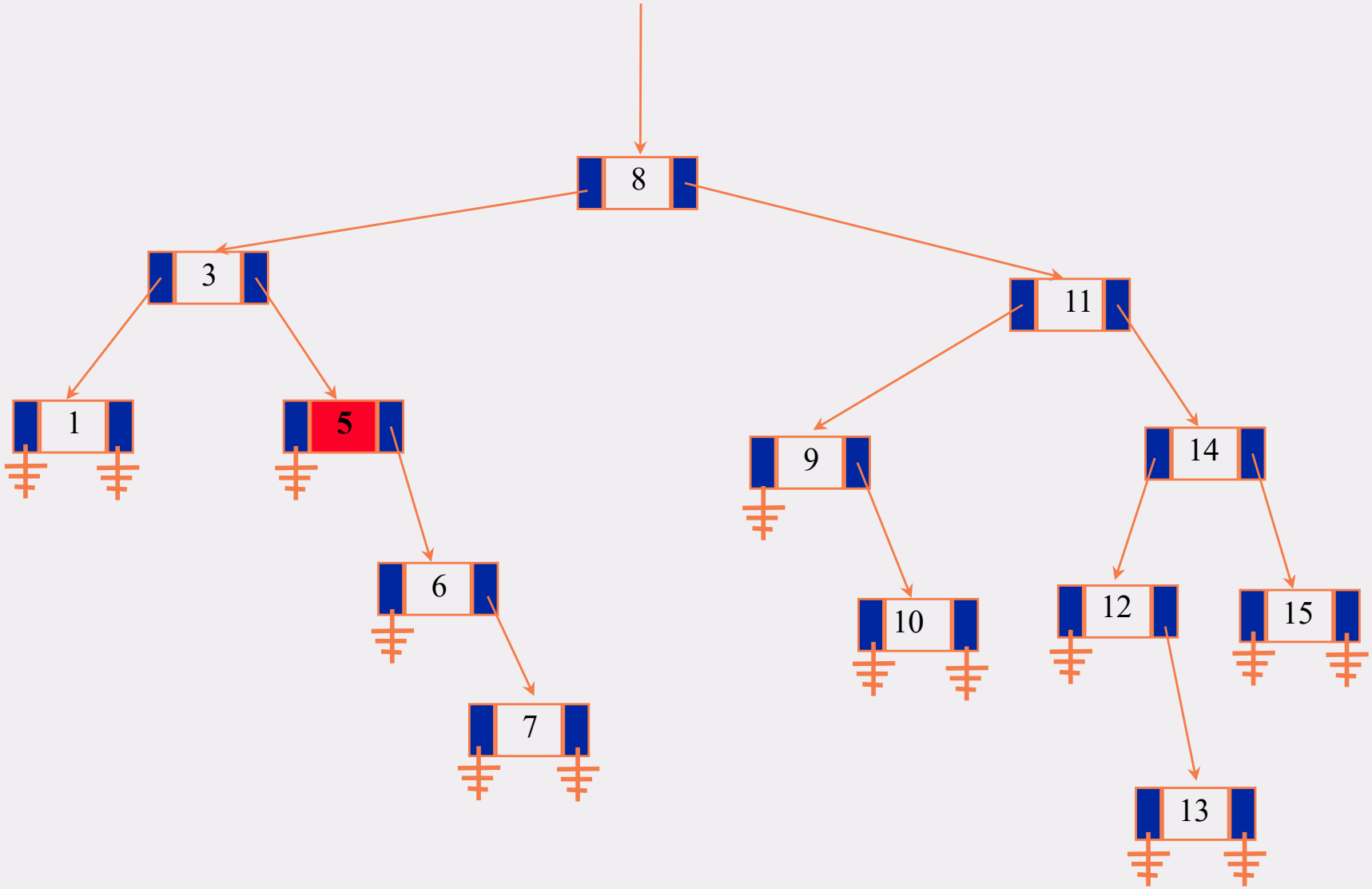
Deletando um elemento em uma árvore binária de pesquisa

- Caso 2: z possui 1 filho

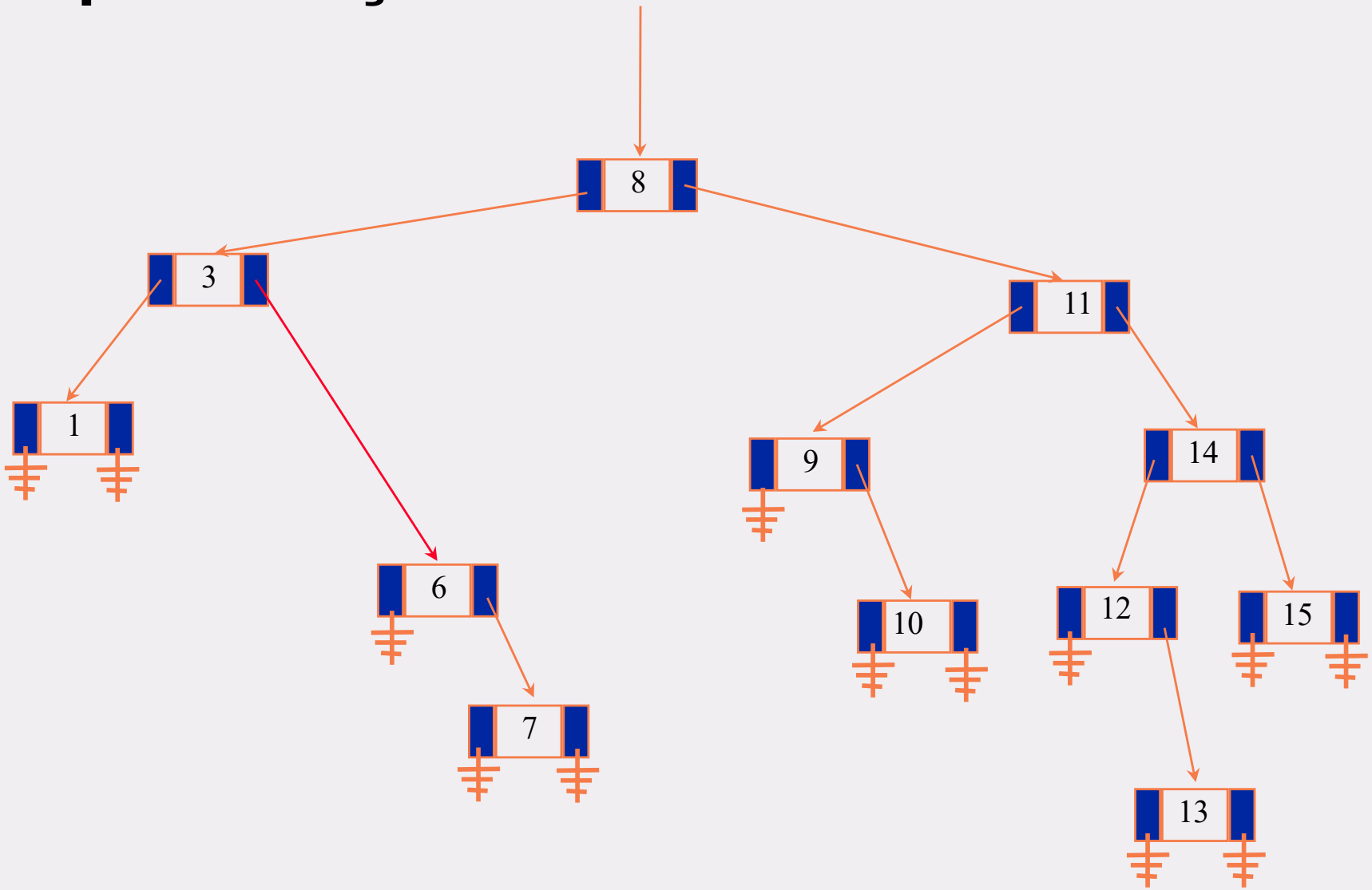




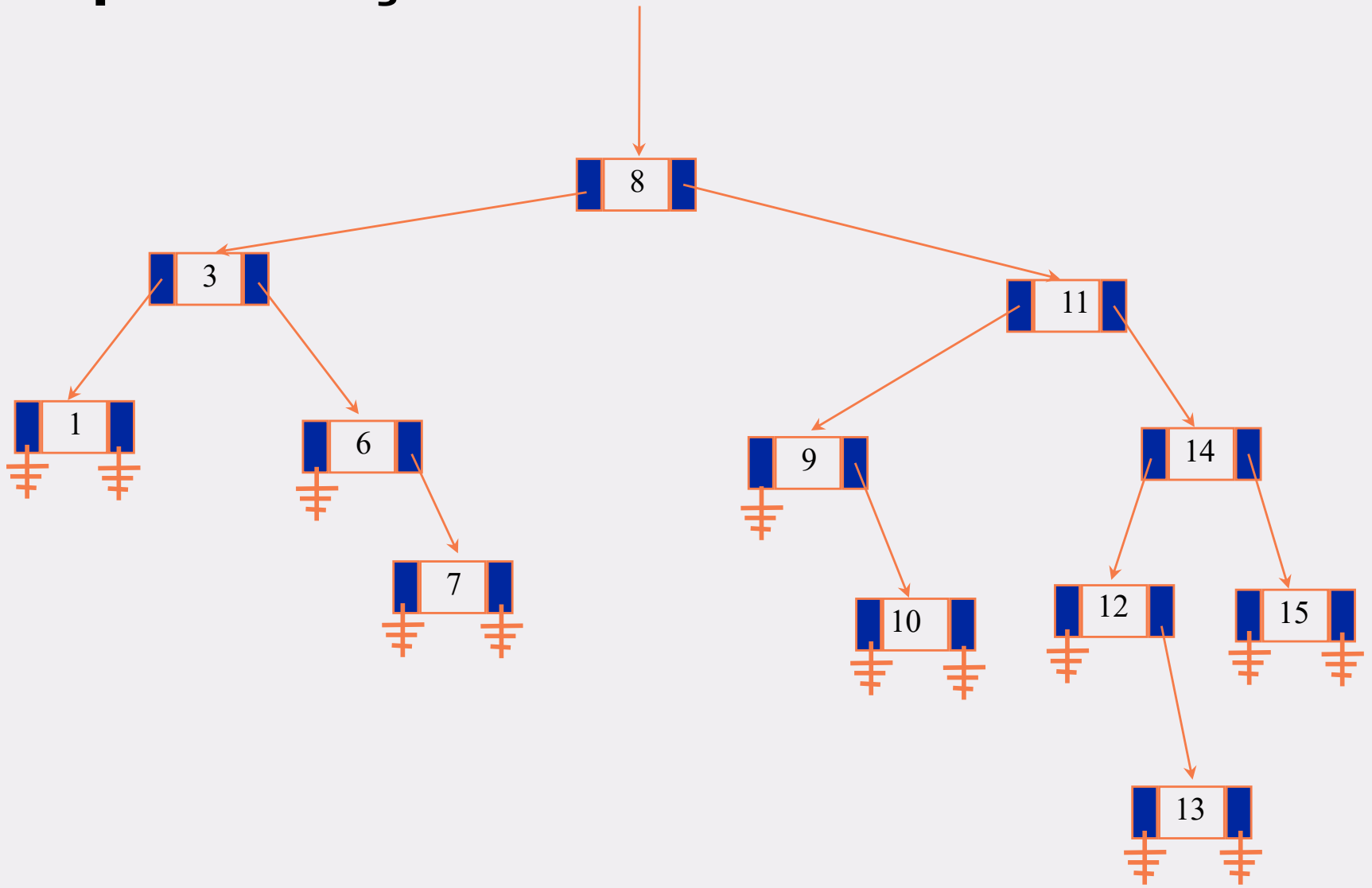
Exemplo: deleção do nodo com chave = 5



Exemplo: deleção do nodo com chave = 5



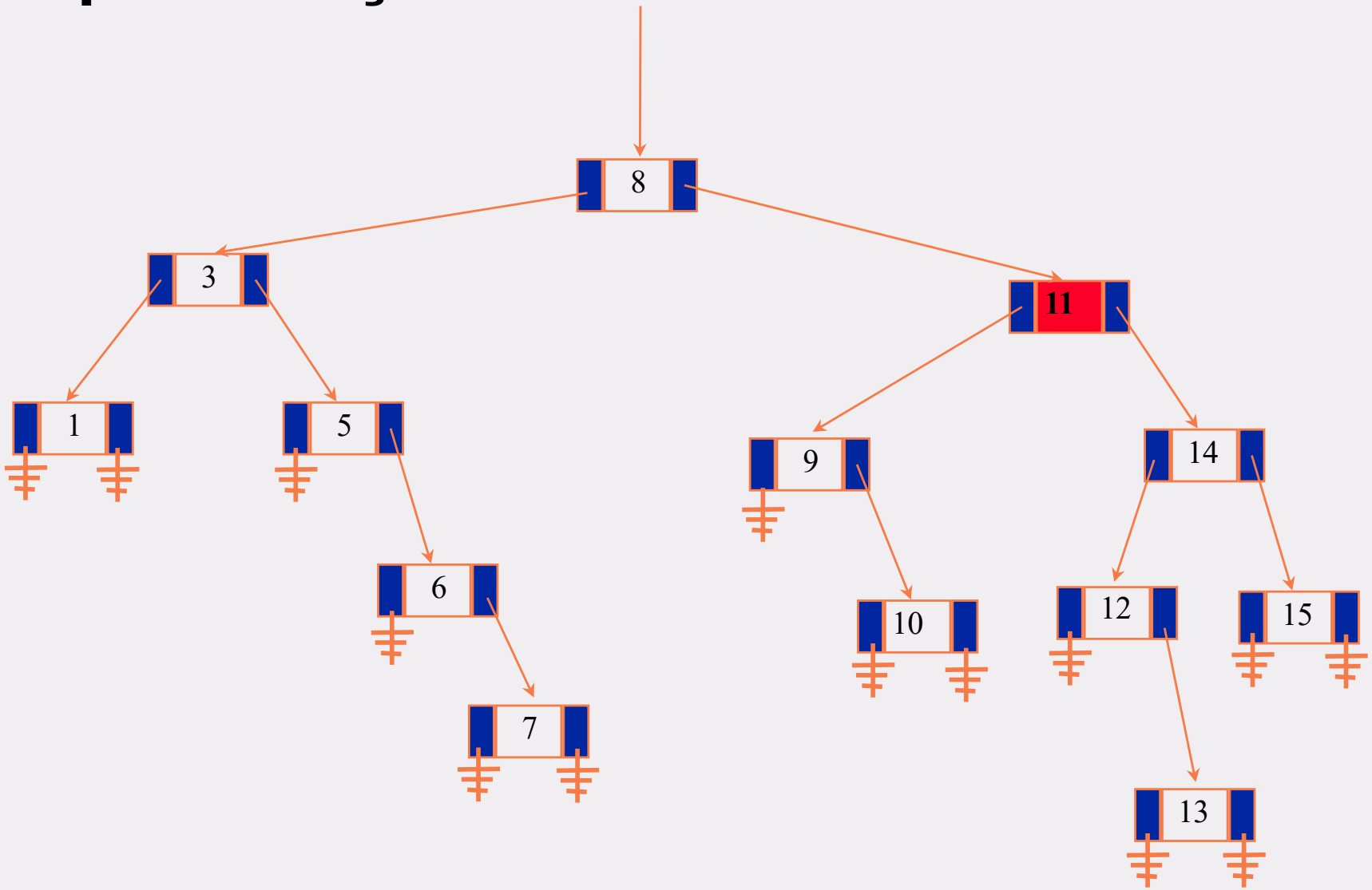
Exemplo: deleção do nodo com chave = 5



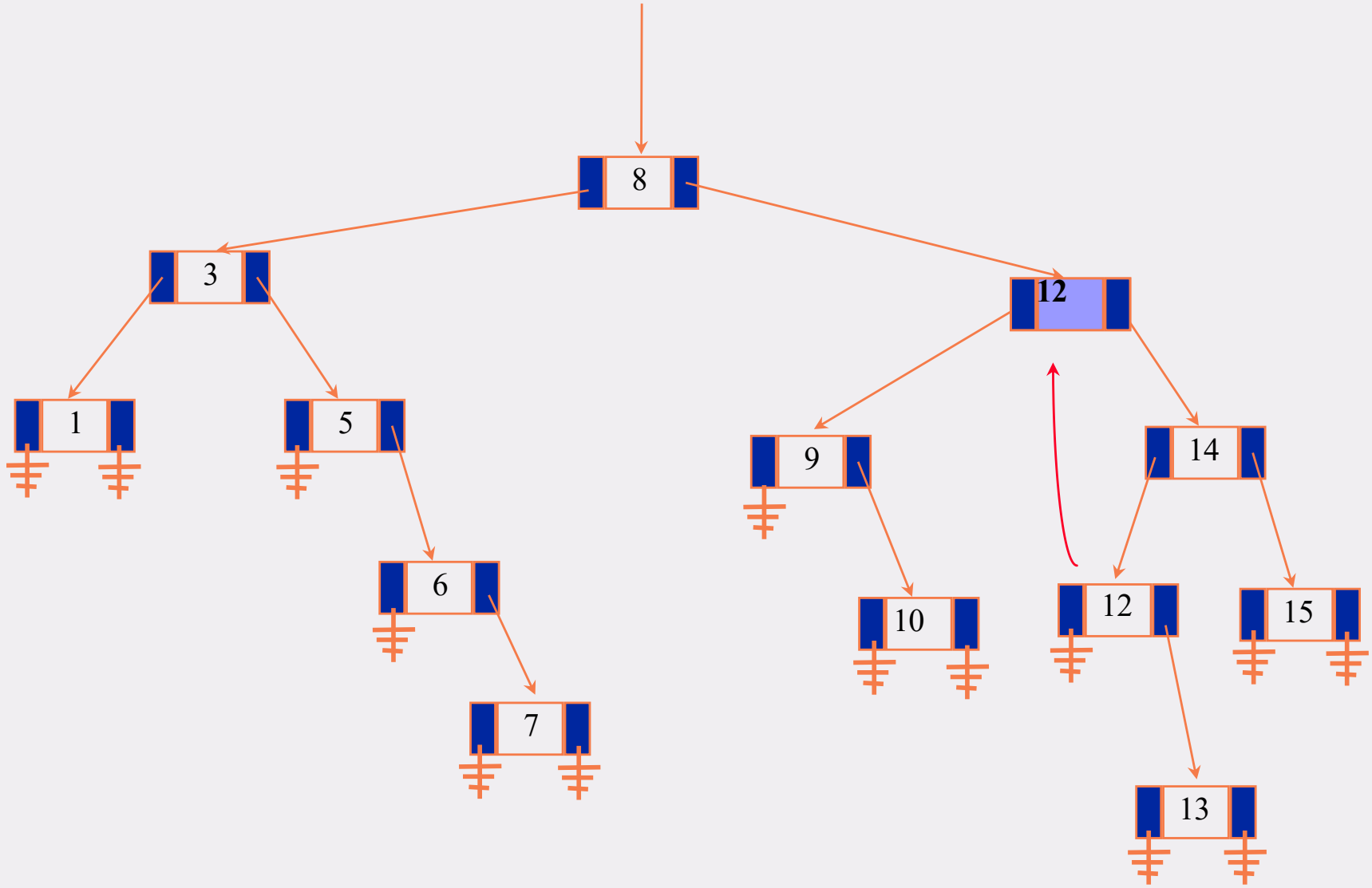
Deletando um elemento em uma árvore binária de pesquisa

- Caso 3: z possui 2 filho
  - se o filho à direita não possui subárvore esquerda, é ele quem ocupa o seu lugar;
  - se possuir uma subárvore esquerda, a raiz desta será movida para cima e assim por diante;
  - a estratégia geral (Mark Allen Weiss) é sempre substituir a chave retirada pela menor chave da subárvore direita.

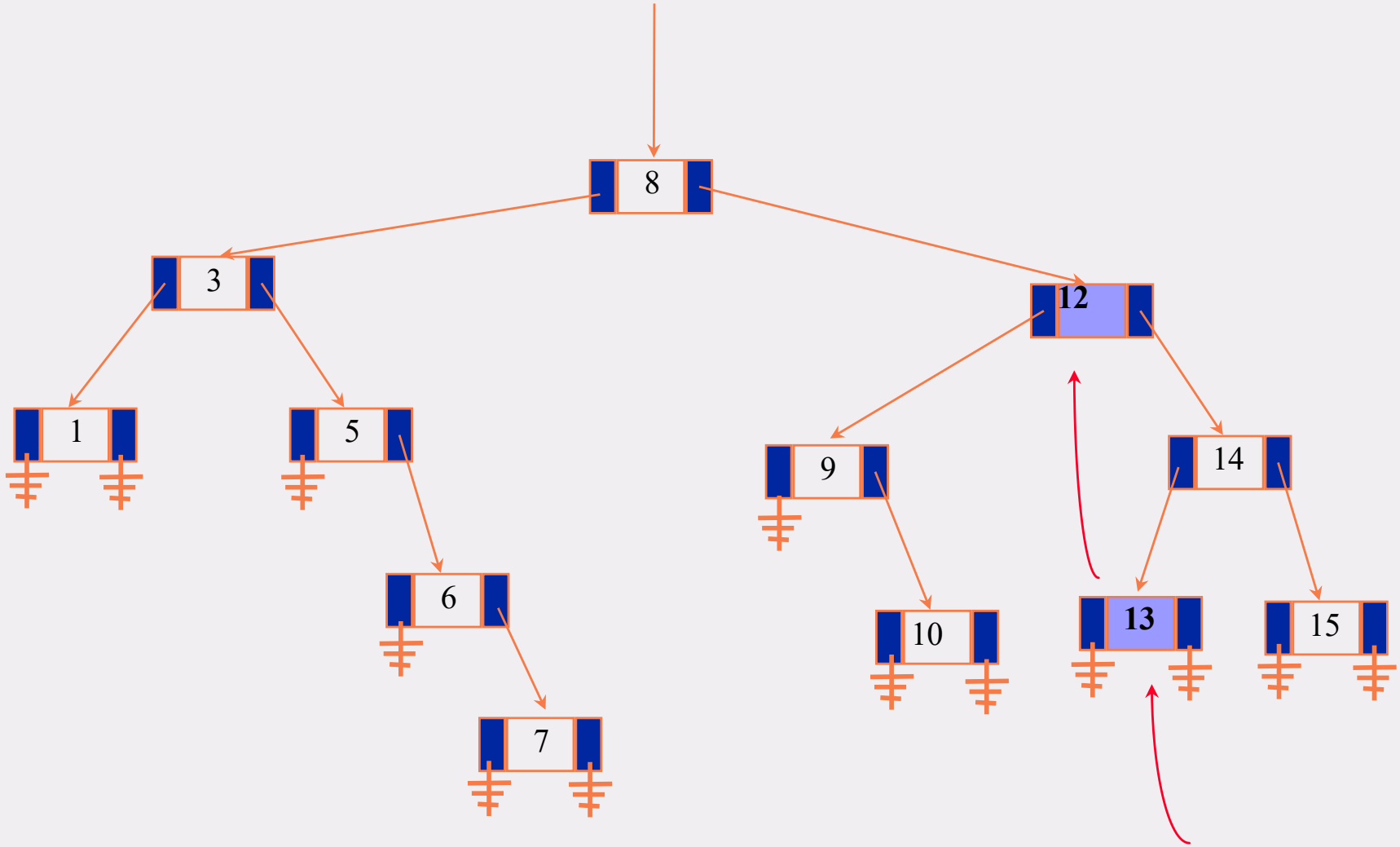
Exemplo: deleção do nodo com chave = 11



Exemplo: deleção do nodo com chave = 11

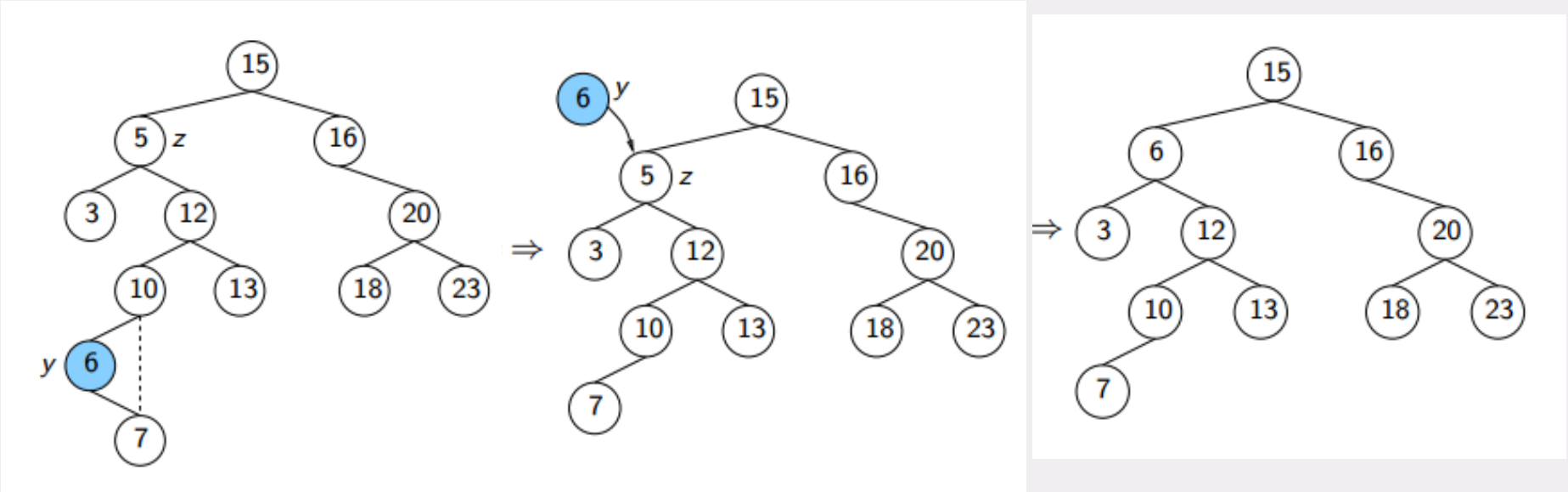


Exemplo: deleção do nodo com chave = 11



Deletando um elemento em uma árvore binária de pesquisa

- Caso 3: z possui 2 filho
  - colocamos no lugar de z o seu sucessor, o que com certeza não possui filho à



- Construir uma arvore com as chaves:
  - (10, 20, 30, 5, 3, 50, 40, 70, 60, 90)

Exercício

- Depois de construída a árvore acima, insira os nós:
  - (1, 15, 65)

Exercício

- Depois de construída a árvore acima, remova os nós:
  - (3, 30, 70)

Exercício

- Implemente uma árvore binária
- Inserir as seguintes chaves:
  - (10, 20, 30, 5, 3, 50, 40, 70, 60, 90)
- Depois de construída a árvore acima, insira os nós:
  - (1, 15, 65)
- Depois de construída a árvore acima, remova os nós:
  - (3, 30, 70)
- Considerando a árvore do exercício acima, mostre o resultado considerando o caminho em ordem

Perguntas?