# Árvores Binárias de Busca

Fábio Henrique Viduani Martinez

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Estruturas de Dados

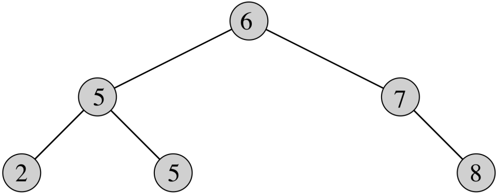
# Introdução

- Uma árvore binária de busca suporta operações (entre outras): busca, mínimo, máximo, predecessor, sucessor, inserção e remoção
- Podemos utilizá-la para implementar um dicionário ou uma lista de prioridades
- Para uma árvore com $n$ elementos, as operações levam tempo proporcional à sua altura $h$:
  - $O(\lg n)$ para uma árvore completa
  - $O(n)$ para uma árvore zigue-zague/degenerada

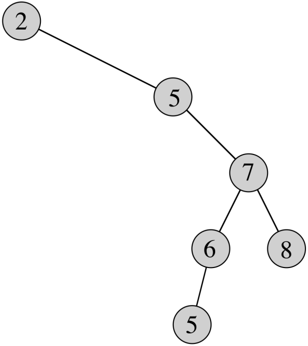
# Introdução

- A altura esperada de um árvore construída aleatoriamente é $O(\lg n)$, mas não podemos garantir que isso ocorra
- Existem outros tipos de árvore que garantem uma altura $O(\lg n)$

# Exemplos




(a)

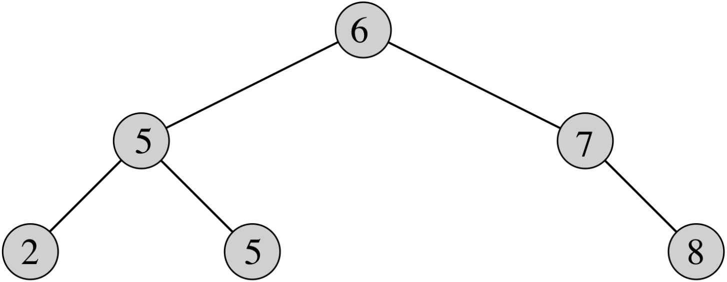



(b)

# Definições

- Podemos representar uma árvore como uma estrutura ligada onde cada nó é um objeto
- Cada nó $x$ contém além de outros eventuais dados, os atributos:
  - Uma chave $x.chave$
  - Um ponteiro para o filho esquerdo $x.esq$
  - Um ponteiro para o filho direito $x.dir$
  - Um ponteiro para seu pai $x.pai$
- Se não há um pai ou filho, o valor do atributo referente é $\mathrm{NIL}$
- Por simplicidade, consideramos que a própria chave é o dado armazenado pelo nó

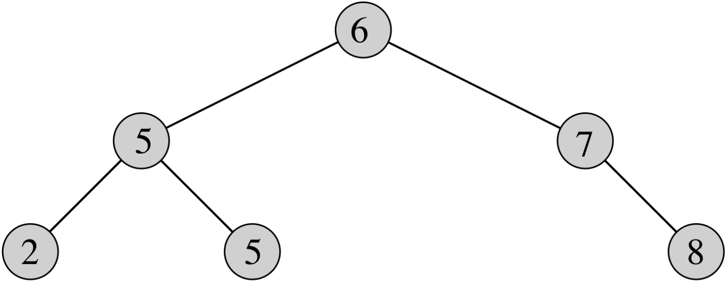
# Exemplo



# Definições (2)

- As chaves de uma árvore binária de busca são sempre armazenadas de forma a satisfazer a seguinte propriedade, chamada propriedade árvore-binária-de-busca. Seja $x$ um nó em uma árvore binária de busca:
  - Se $y$ é um nó na subárvore esquerda de $x$, então $y.chave \leq x.chave$
  - Se $y$ é um nó na subárvore direita de $x$, então $y.chave \geq x.chave$
- Uma árvore $T$ possui um atributo: $T.raiz$, que é uma referência ao nó raiz, ou NIL se a árvore é vazia

# Exemplo



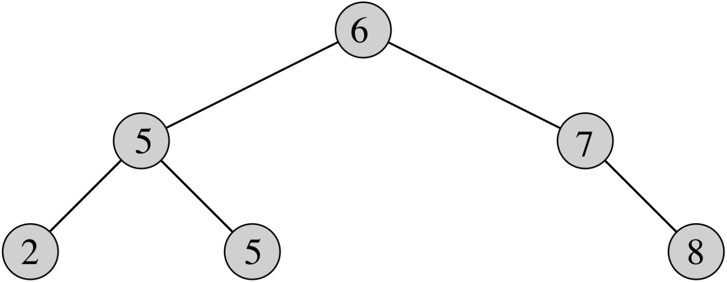
# Visita aos nós da árvore

- Podemos escrever os elementos da árvore realizando algum dos percursos estudados anteriormente
- Para escrever os elementos em ordem crescente, basta utilizar o percurso em-ordem ($e$-$r$-$d$)
- Para cada nó, percorremos sua subárvore esquerda, visitamos esse nó, depois percorremos a subárvore direita
- Tempo de execução $O(n)$

# Percurso em-ordem – algoritmo

```
EM-ORDEM(x)
    se x ≠ NIL
        EM-ORDEM(x.esq)
        escreva x.chave
        EM-ORDEM(x.dir)
```

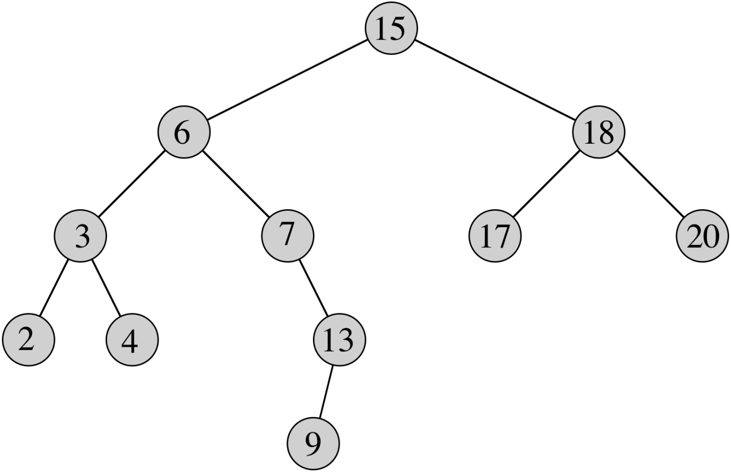
# Percurso em-ordem – exemplo



# Busca de um elemento

- Dados um ponteiro $x$ para a raiz da árvore e uma chave $k$, a busca devolve um ponteiro para um nó com chave $k$, se existir, ou NIL caso contrário
- Tempo de execução $O(h)$, onde $h$ é a altura da árvore
- O algoritmo segue um caminho simples da raiz até a parte inferior da árvore (no máximo um caminho da raiz até uma folha)

# Busca – exemplo



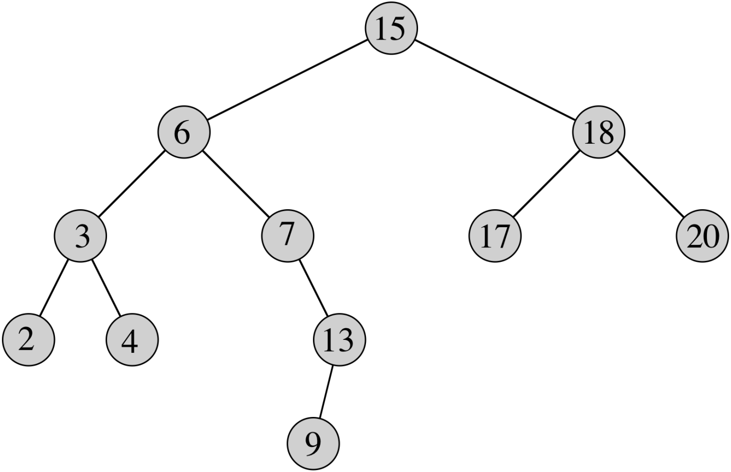
# Busca – algoritmo

```
BUSCA(x, k)
    se x = NIL ou k = x.chave
        devolva x
    se k < x.chave
        devolva BUSCA(x.esq, k)
    senão
        devolva BUSCA(x.dir, k)
```

# Elementos mínimo e máximo

- Podemos encontrar o mínimo e o máximo seguindo os ponteiros dos filhos esquerdos e direitos, respectivamente
- A propriedade da árvore binária de busca em si garante a corretude dos algoritmos
- Esses procedimentos têm tempo $O(h)$, onde $h$ é a altura da árvore
- Da mesma forma que a busca, os algoritmos seguem um caminho da raiz até a parte inferior da árvore

# Mínimo e Máximo – exemplo



# Mínimo e Máximo – algoritmos

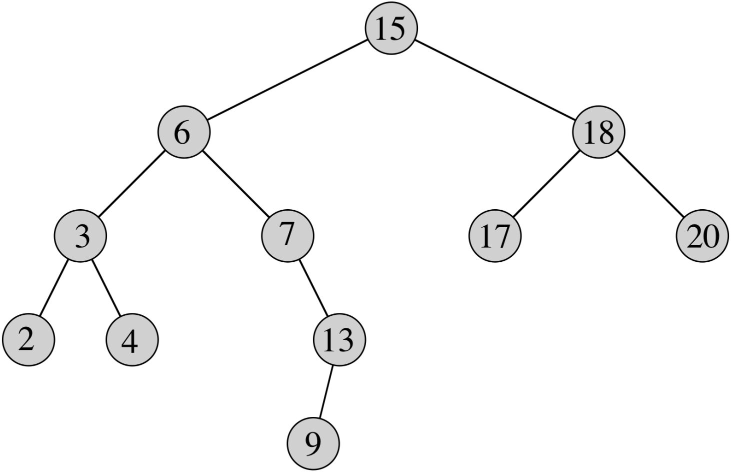
MÍNIMO($x$)
```
    enquanto x.esq ≠ NIL
        x ← x.esq
    devolva x
```

MÁXIMO($x$)
```
    enquanto x.dir ≠ NIL
        x ← x.dir
    devolva x
```

# Sucessor e predecessor de uma chave

- O sucessor de um nó $x$ é o nó com a menor chave maior que $x$
- Devolvemos o nó contendo o sucessor ou $\text{NIL}$ se ele não existir
- Se $x$ tem filho direito, seu sucessor é o elemento mínimo da subárvore direita
- Se $x$ não tem filho direito, seu sucessor é seu ancestral mais "baixo" na árvore, cujo filho esquerdo também é ancestral de $x$
- O predecessor é análogo ao sucessor, isto é, o predecessor de um nó $x$ é o nó com a maior chave menor que $x$
- Tempo de execução $O(h)$, da mesma forma que os procedimentos anteriores, onde percorremos um caminho simples (subindo ou descendo)

# Sucessor e predecessor – exemplo



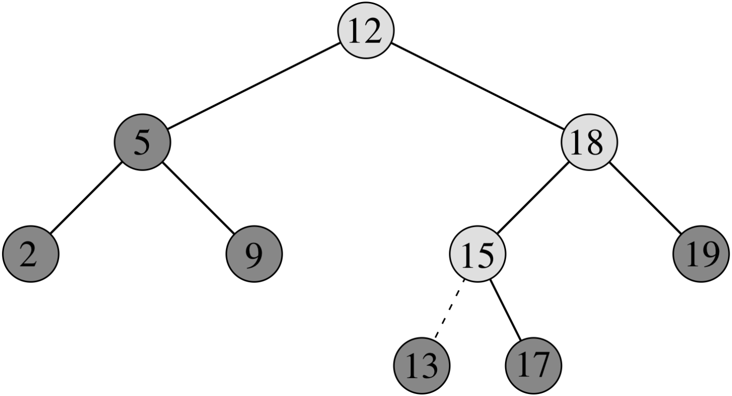
# Sucessor – algoritmo

```
SUCESSOR(x)
    se x.dir ≠ NIL
        devolva MÍNIMO(x.dir)
    y ← x.pai
    enquanto y ≠ NIL e x = y.dir
        x ← y
        y ← y.pai
    devolva y
```

# Inserção

- Para inserir um valor $v$ em uma árvore binária de busca, devemos modificar o conjunto de chaves da árvore mantendo suas propriedades
- O algoritmo recebe a árvore $T$ e um novo nó $z$ com $z.chave = v$ e $z.esq = z.dir = \text{NIL}$
- Tempo de execução $O(h)$, onde $h$ é a altura da árvore
- O algoritmo segue um caminho simples da raiz até a parte inferior da árvore

# Inserção – exemplo



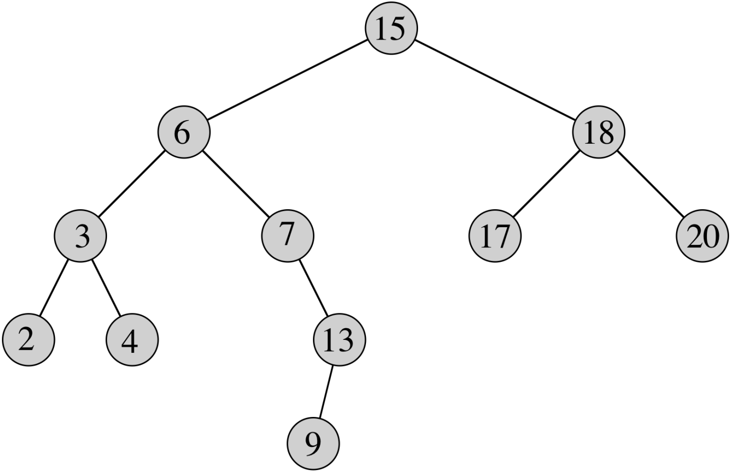
# Inserção – algoritmo

```
INSERE(T, z)
   y ← NIL
   x ← T.raiz
   enquanto x ≠ NIL
      y ← x
      se z.chave < x.chave
         x ← x.esq
      senão
         x ← x.dir
   z.pai ← y
   se y = NIL
      T.raiz ← z
   senão
      se z.chave < y.chave
         y.esq ← z
      senão
         y.dir ← z
```

# Remoção de um elemento

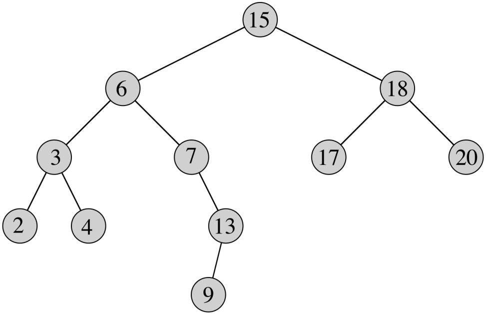
- Um pouco mais complicada que a inserção
- Usamos o procedimento auxiliar TRANSPLANTE, que substitui uma subárvore de raiz $u$ por outra com raiz $v$, ajustando o pai de $u$
- Não altera os filhos de $u$ ou $v$

# Transplante – exemplo



# Transplante – algoritmo

```
TRANSPLANTE(T, u, v)
    se u.pai = NIL
        T.raiz ← v
    senão
        se u = u.pai.esq
            u.pai.esq ← v
        senão
            u.pai.dir ← v
    se v ≠ NIL
        v.pai ← u.pai
```

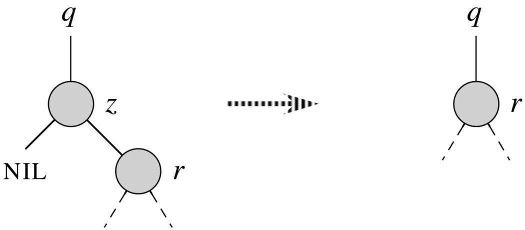


# Remoção – casos

Podemos ter 3 casos ao remover um nó $z$

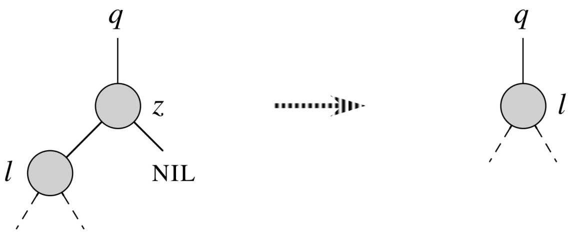
# Remoção – 1º caso

**1.** $z$ tem apenas o filho direito (ou nenhum filho): removemos $z$ e posicionamos o filho em seu lugar



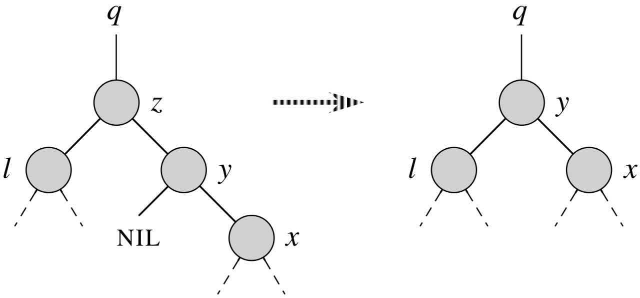
# Remoção – 2º caso

**2.** $z$ tem apenas o filho esquerdo: removemos $z$ e posicionamos o filho em seu lugar



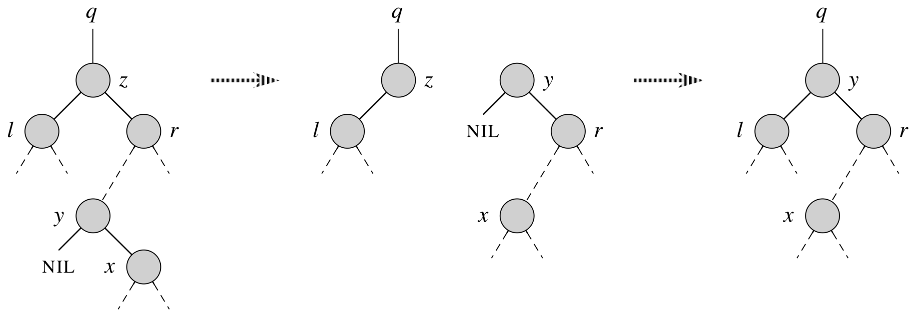
# Remoção – 3º caso (a)

**3(a).** $z$ tem 2 filhos e seu sucessor $y$ é seu filho direito: colocamos $y$ no lugar de $z$, então a subárvore esquerda de $z$ passa a ser a subárvore esquerda de $y$, e a subárvore direita de $y$ permanece inalterada



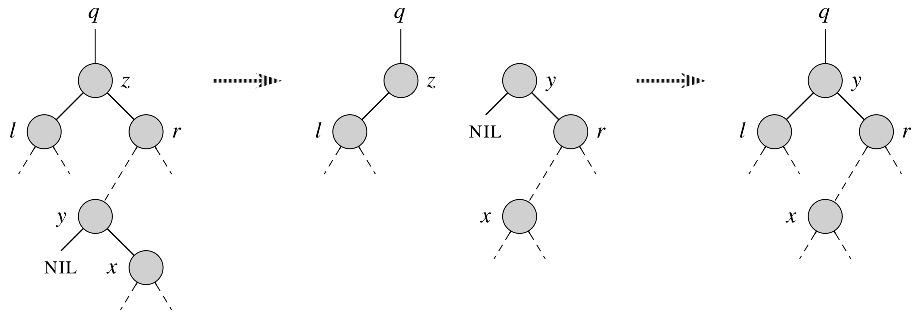
# Remoção – 3º caso (b)

**3(b).** $z$ tem 2 filhos e seu sucessor $y$ está abaixo de seu filho direito $r$ (em $T_r^E$): substituímos $y$ por seu filho $x$ e colocamos $r$ como filho de $y$, caindo no caso 3(a)



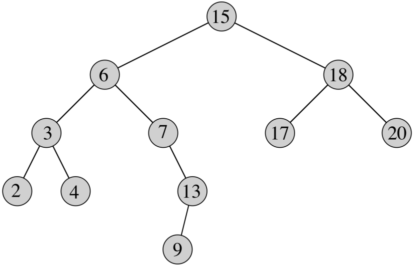
# Remoção – 3º caso (b)

**3(b).** este caso pode ser visto como substituição de $z$ pelo seu sucessor (é possível realizar a mesma operação usando a idéia de substituição pelo predecessor)

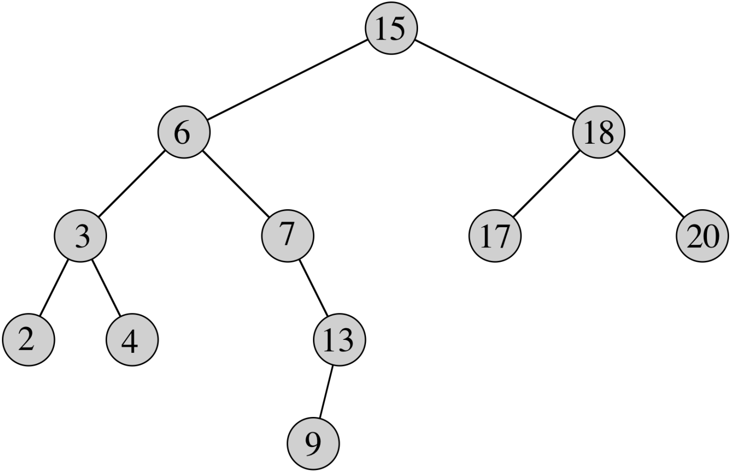


# Remoção – algoritmo

```
REMOVE(T, z)
se z.esq = NIL            // 1º caso
    TRANSPLANTE(T, z, z.dir)
senão
    se z.dir = NIL        // 2º caso
        TRANSPLANTE(T, z, z.esq)
    senão                 // 3º caso
        y ← MÍNIMO(z.dir)
        se y.pai ≠ z      // (b)
            TRANSPLANTE(T, y, y.dir)
            y.dir ← z.dir
            y.dir.pai ← y
        TRANSPLANTE(T, z, y) // (a)
        y.esq ← z.esq
        y.esq.pai ← y
```



# Remoção – exemplo



# Remoção – tempo

- Todas operações usam tempo constante, menos a chamada de MÍNIMO, que consome tempo $O(h)$
- Portanto, tempo $O(h)$

# Exercícios

5.1 Desenhe uma dentre as possíveis árvores binárias de busca contendo as seguintes letras: M, C, T, F, D, V, O.

5.2 Considere uma árvore binária de busca com as seguintes chaves: 60, 37, 12, 3, 1, 40, 18, 38, 25, 50, 42, 80, 100, 90, 83

(a) Construa a árvore inserindo os elementos na ordem dada
(b) Qual é o nó raiz?
(c) Qual é a subárvore direita do nó 40?
(d) Qual é a subárvore esquerda do nó 37?
(e) Quais nós não possuem subárvore direita?
(f) Quais nós não possuem subárvore esquerda?
(g) Qual é o pai do nó 40?
(h) Qual é o pai do nó 60?
(i) Quais são os filhos do nó 60?
(j) Quais são os filhos do nó 38?
(k) Indique 3 pares de nós irmãos
(l) Quais nós são folhas?

# Exercícios

5.2 (continuação)

(m) Quais são os ancestrais do nó 38?
(n) Quais são os ancestrais do nó 60?
(o) Qual é o sucessor do nó 42?
(p) Quais são os descendentes do nó 42?
(q) Quais são os descendentes do nó 12?
(r) Quais é o predecessor do nó 80?
(s) Qual é o nível ou profundidade do nó 60?
(t) Quais são os nós do nível 2?
(u) Qual é a altura da árvore?
(v) Se inserirmos os nós 15 e 17, a profundidade da árvore será alterada?
(w) Essa árvore é estritamente binária ou completa?
(x) Aponte a posição do elemento mínimo
(y) Aponte a posição do elemento máximo
(z) Considerando as 4 possibilidades de remoção (casos 1, 2, 3(a) e 3(b)), encontre um nó em cada uma dessas situações e remova-o