FSE598 前沿计算技术

模块 3　算法设计与分析
单元 3　高级数据结构
第 2 讲　红黑树

本课程的部分内容是基于 Thomas H.Cormen、Charles E.Leiserson 等人的“算法简介”教材

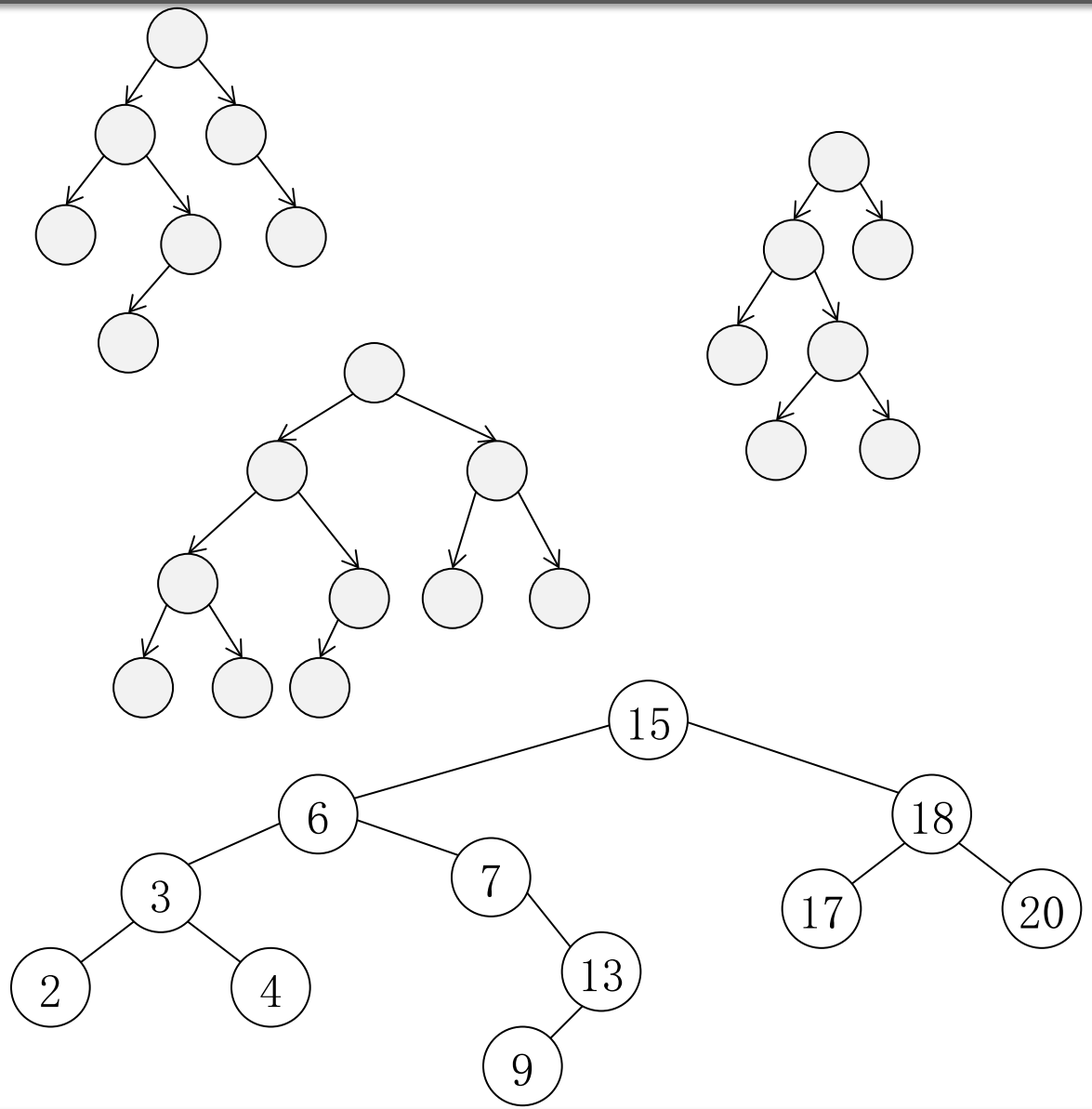
课程概要

学习内容

- 红黑树的定义
- 红黑树的属性
- 红黑树操作的复杂度
- 红黑树中的旋转
- 红黑树中的操作算法

二叉树回顾

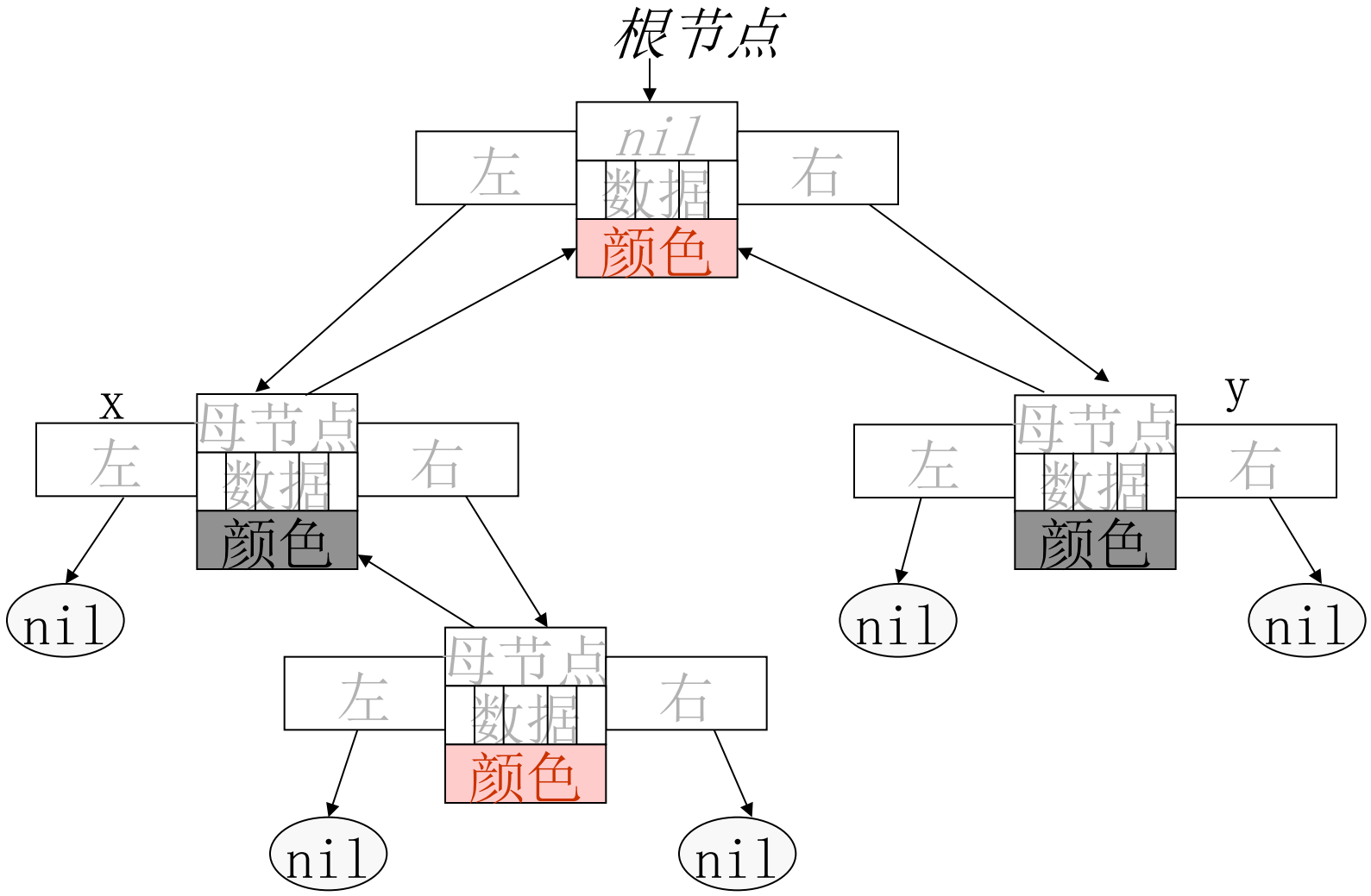
- 如果树的任意节点最多只能有两个相邻节点（子节点），则称其为“二叉树”。
- 在满二叉树中，其节点要么没有子节点，要么有两个子节点。
- 平衡的二叉树是指任何两个叶节点的高度（深度）之差都不会超过 1。对于 n 个节点的树，其高度为 $O(\lg n) + 1$
- 二叉搜索树
- 我们能否有平衡的二叉搜索树



红黑树

- 红黑树是指具有某些额外属性，使其是一种接近平衡的二叉搜索树。
- 该树有一个额外的二进制位来存储颜色：红或黑；
- 红色节点的子节点必须为两个黑色节点；
- 指向 nil 的指针被视为“外部节点”并且是黑色的，而携带数据的节点均为内部节点。所有内部节点都有两个子节点。

红黑树示例


根节点
nil
左
数据
右
颜色
x
母节点
左
数据
右
颜色
nil
母节点
左
数据
右
颜色
nil
nil
y
母节点
左
数据
右
颜色
nil
nil

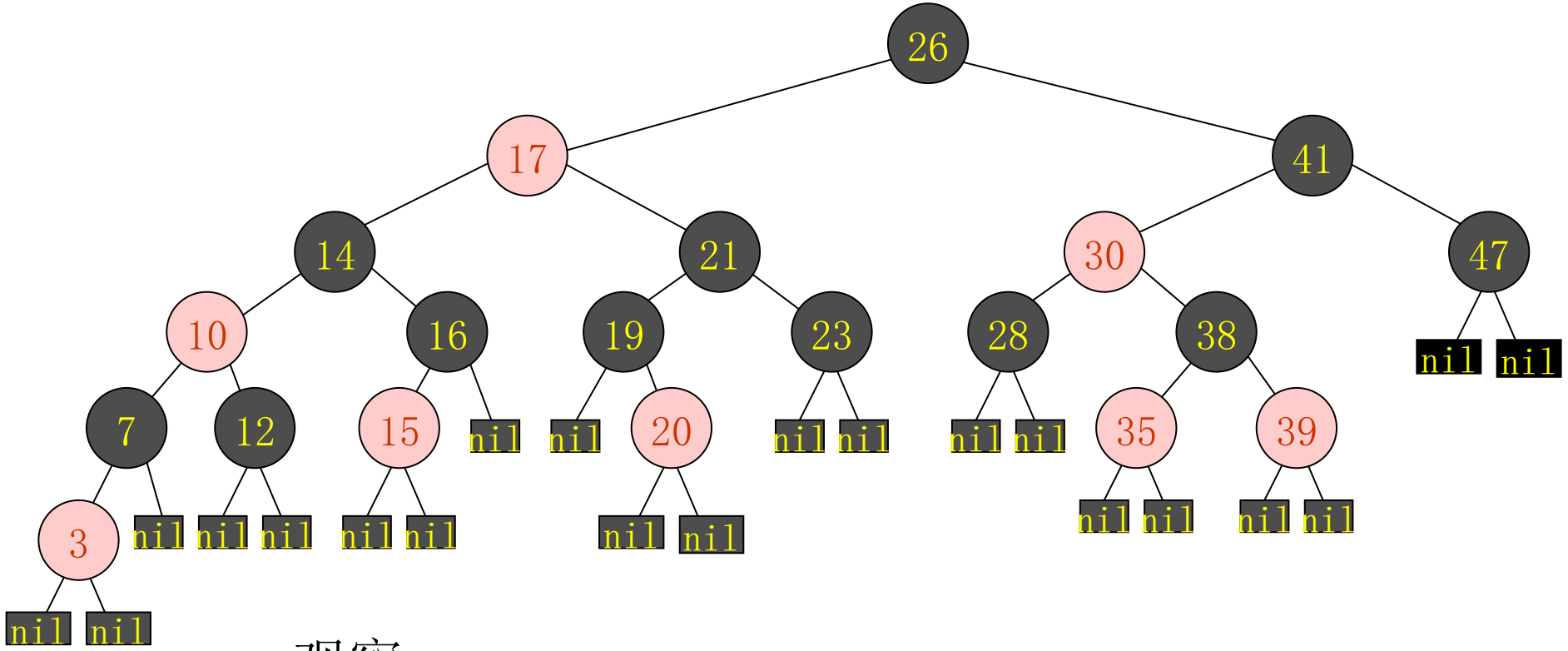
红黑树定义

如果一个二叉搜索树满足以下属性，则称其为“**红黑树**”

1. 所有节点都是红或黑之一；
2. 根节点和所有叶节点（nil）均为黑色；
3. 如果一个节点是红色的，则其子节点均为黑色；
4. 如果一个节点是黑色的，则其子节点可以是红色，也可以是黑色；
5. 从一个节点到后继叶节点的每条简单路径都包含相同数量的黑色节点，使该树接近于平衡。

- 从节点 x（但不包括 x）到叶节点的任意路径上的黑色节点数，即为节点 x 的**黑色高度**。
- **红黑树**根节点的黑色高度即为该树的**黑色高度**。

红黑树示例



观察：

节点 26：到达任意叶节点均有 3 个黑色节点，黑色高度 = 3

节点 17：到达任意叶节点均有 3 个黑色节点，黑色高度 = 3

节点 14：到达任意叶节点均有 2 个黑色节点，黑色高度 = 2

节点 41：到达任意叶节点均有 2 个黑色节点，黑色高度 = 2

红黑树的高度

定理：带有 n 个内部节点的红黑树，其高度 h 最大为 $2*\lg(n+1)$

证明的思路是表明：

1. 黑色节点的数量 $\geq n/2$
2. 黑色高度 $bh \geq h/2$
3. $n \geq 2^{bh} - 1$（这是有难度的一步）
4. $n \geq 2^{bh} - 1 \geq 2^{h/2} - 1$，或
5. $\lg(n+1) \geq h/2$，或
6. $h \leqslant 2 \lg(n+1)$

红黑树操作的运行时间

该定理的直接结果是

以下操作：

- 搜索
- 最大值
- 最小值
- 后继节点
- 前置节点

均可在 $O(h) = O(\lg n)$ 时间内完成。对于以下操作：

- 插入
- 删除

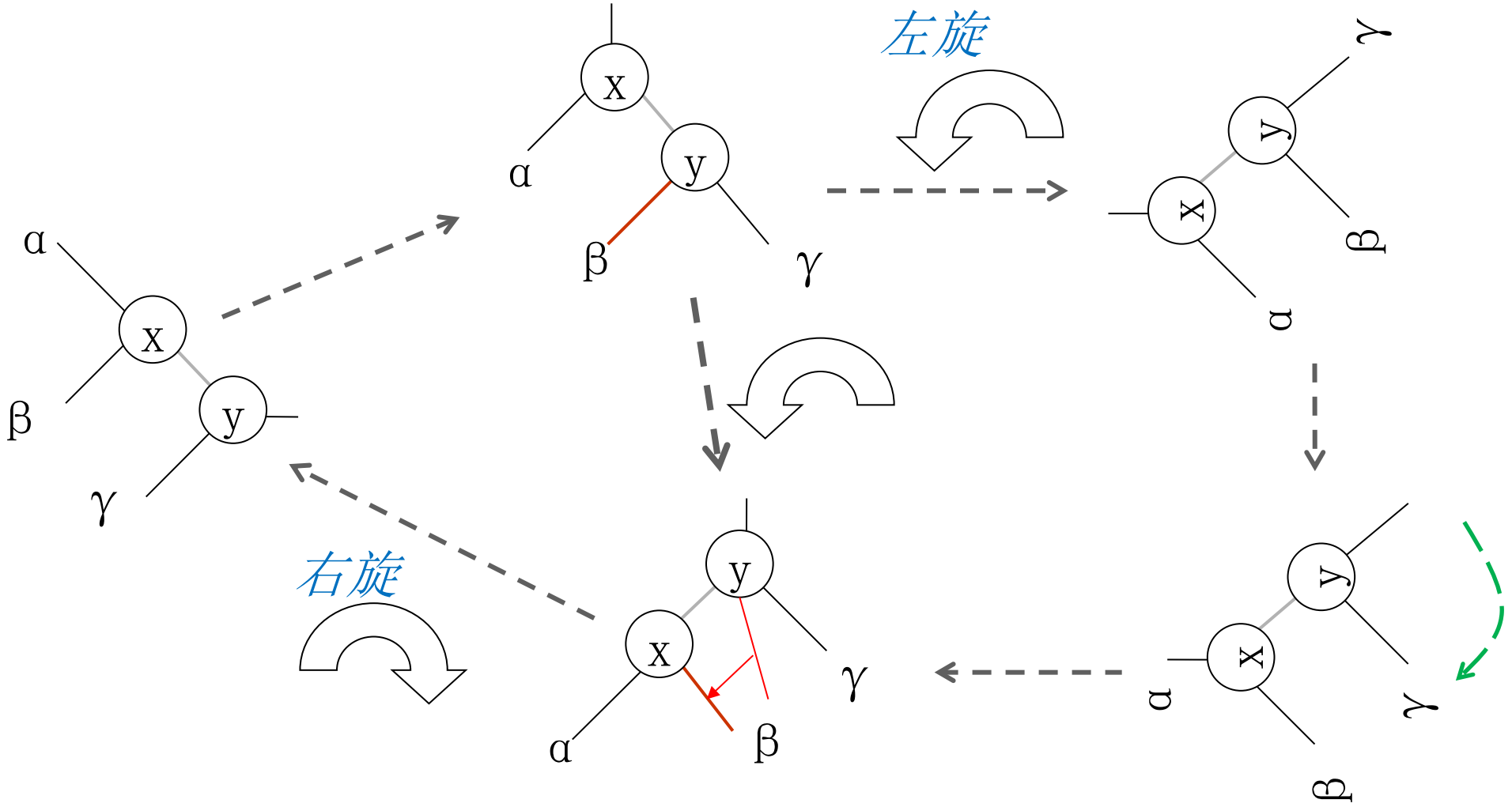
单个操作可以在 $O(\lg n)$ 时间内完成，但不能保证修改后的树仍然是红黑树。

问题是：是否可以在 $O(\lg n)$ 时间内支持插入和删除操作？

红黑树中的旋转

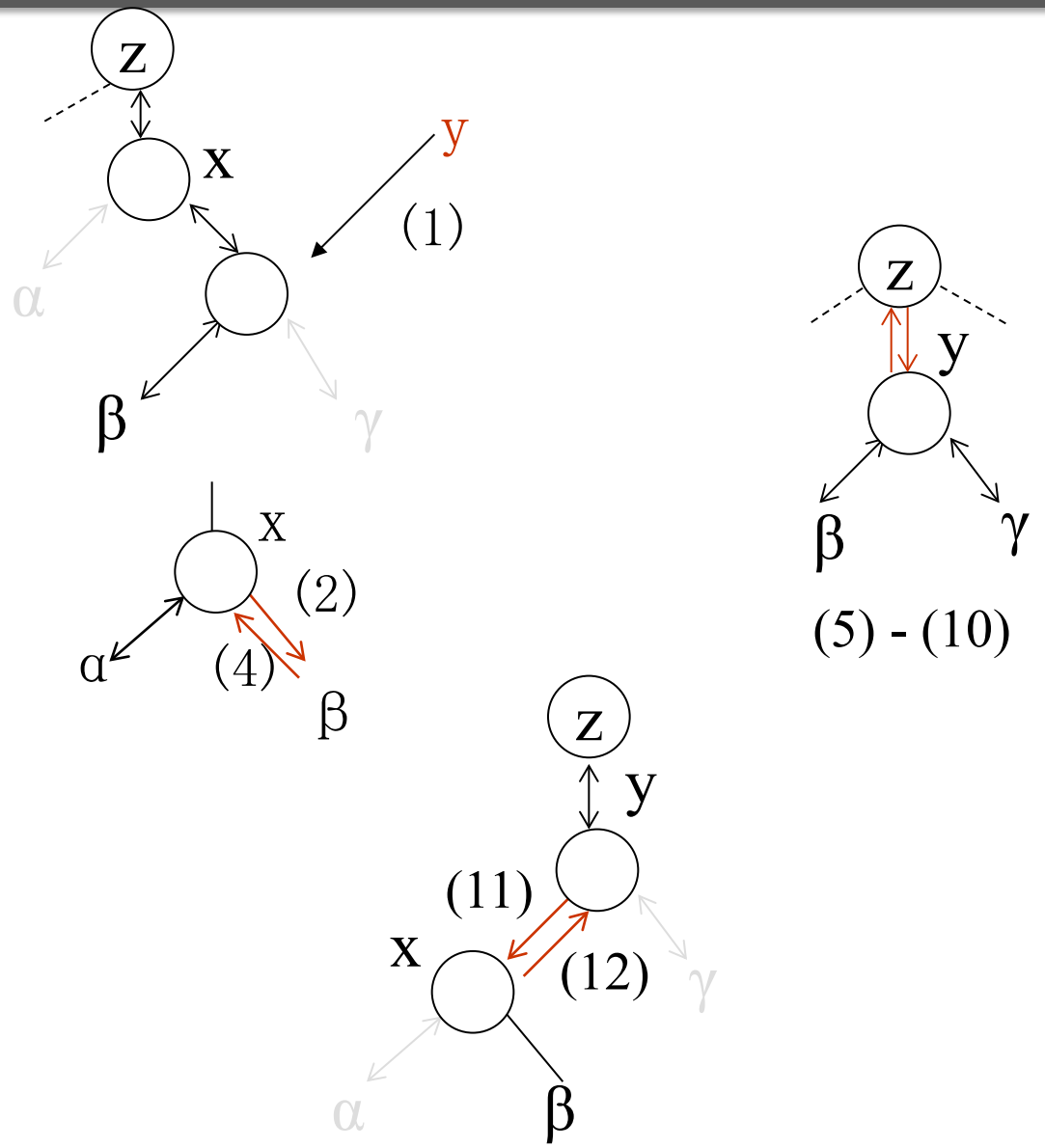
要在插入和删除后保留红黑树的属性，我们需要*向左*或*向右旋转*。

→ CSE551 Splay Tree 伸展树

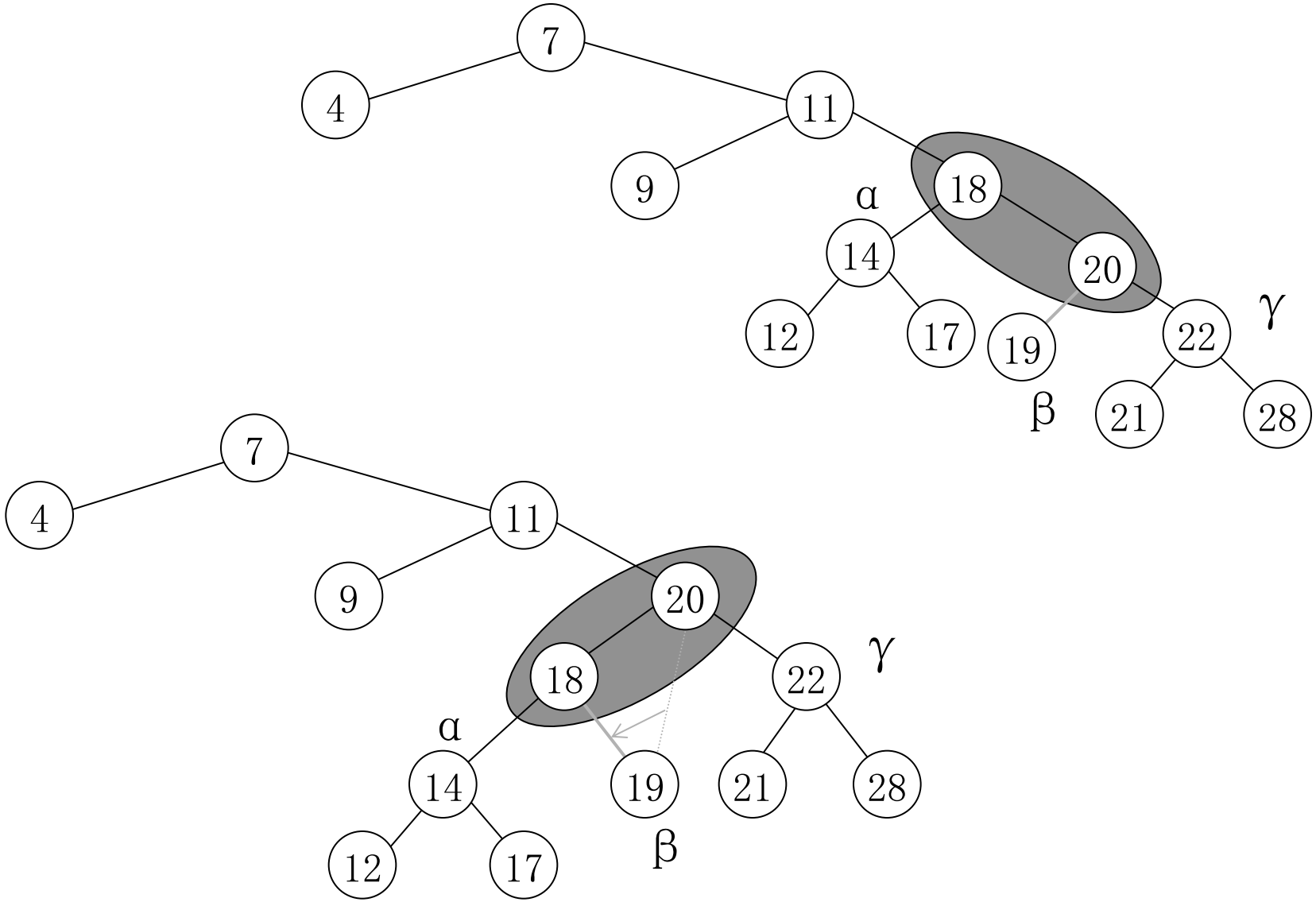


红黑树旋转算法

```
Left-rotate (T, x)
1.    y := x.right
2.    x.right := y.left
3.    if y.left ≠ nil
4.          then y.left.parent := x
5.    y.parent := x.parent
6.    if x.parent == nil
7.          then root[T] := y
8.          else if x == x.parent.left
9.                then x.parent.left := y
10.               else x.parent.right := y
11.   y.left := x
12.   x.parent := y
```



红黑树左旋示例


7
4
11
9
α
18
14
20
12
17
19
γ
22
β
21
28
7
4
11
9
20
18
22
γ
α
14
19
21
28
12
17
β

红黑树插入算法

```
RB-Tree-Insert (T, x)
1.   Tree-Insert(T, x)     // 使用二叉搜索插入，插入一个叶节点
2.   x.color := Red
3.   while x.parent.color == Red do  // 违规：无相邻的红色节点
4.        if x.parent == x.parent.parent.left
5.             then   y := x.parent.parent.right
6.                    if y.color == Red
7.                         then  x.parent.color := Black
8.                               y.color := Black
9.                               x.parent.parent.color :=Red
10.                              x := x.parent.parent
11.                   else   if x == x.parent.right
12.                               then  x := x.parent
13.                                     Left-Rotate(T, x)
14.                              x.parent.color := black
15.                              x.parent.parent.color := Red
16.                              Right-Rotate(T, x.parent.parent)
17.            else
                    (与 then 语句相同，交换“右”和“左”。)
18.  Root[T].color := Black
```

情形 1

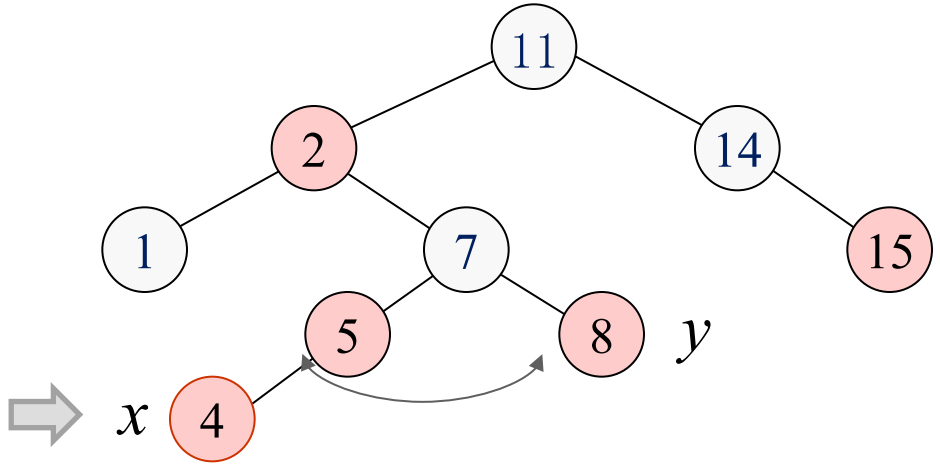
情形 2a

情形 2b

插入算法是如何工作的？

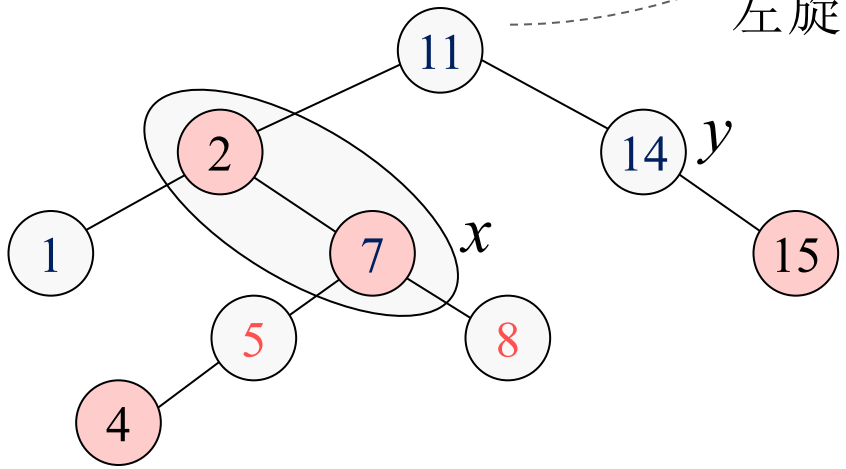
1. 在叶节点之前插入一个红色节点可能只会违反属性 3，即红色节点后面必须跟着**黑色**节点。
   我们假设 x 的母节点确实是红色。
2. while 循环的总体目标是将“违规”向树的上端移动，同时保持每条路径中的黑色节点数恒定（保持不变）。
   1) 如果 x 的叔父节点为红色：我们将 x 的母节点和叔父节点变成**黑色**，并将其祖父节点变成红色。向上移动！
   2) 如果 x 的叔父节点为**黑色**：
      a) 如果 x 是其母节点的右侧子节点，则进行左旋操作，转换为 *b)*
         由于 x 及其母节点均为红色，所以对黑色节点的数量没有影响
      b) 如果 x 是其母节点的左侧子节点，将母节点变黑和祖父节点变红，并向右侧旋转。这样就移除了“违规”，因为叔父节点为黑色节点

红黑树插入示例



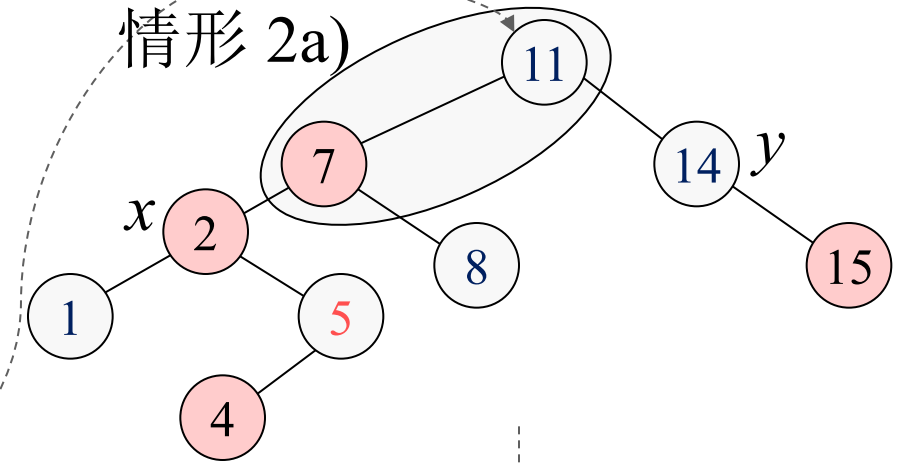
标红
改变母节点
和祖父节点的颜色

情形 1)

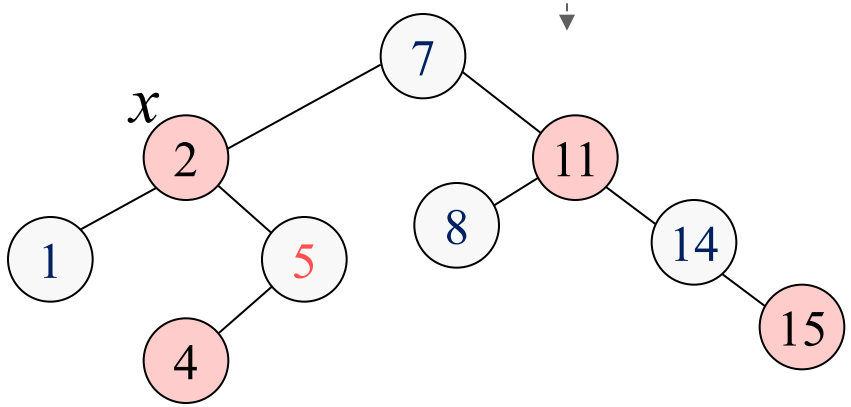


左旋

情形 2a)



改变母节点
和祖父节点的颜色 情形 2b)
右旋



红黑树插入和删除的运行时间

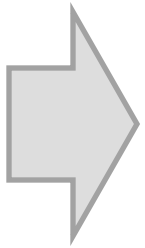
```
RB-Tree-Insert (T, x)
    Binary-Tree-Insert(T, x)                                  O(h)
    x.color := Red
    while x ≠ root[T] and x.parent.color = Red do    O(h)
        loop-body                                              move up
    Root[T].color := Black
```

由于 $h = O(\lg n)$，则红黑树插入的运行时间为 $O(\lg n)$

对于红黑树删除操作，运行时间也为 $O(\lg n)$

红黑树的总结

- 红黑树
- 是一种具有额外属性的二叉搜索树
- 复杂度为 $h = O(\lg n)$ 的平衡树
- 红黑树的所有动态操作都可以在 $O(\lg n)$ 时间内完成，包括：
  - 搜索　　最小值、最大值
  - 插入　　后继节点
  - 删除　　前置节点
- 红黑树中的旋转
- 红黑树中的插入算法



B树和B+树