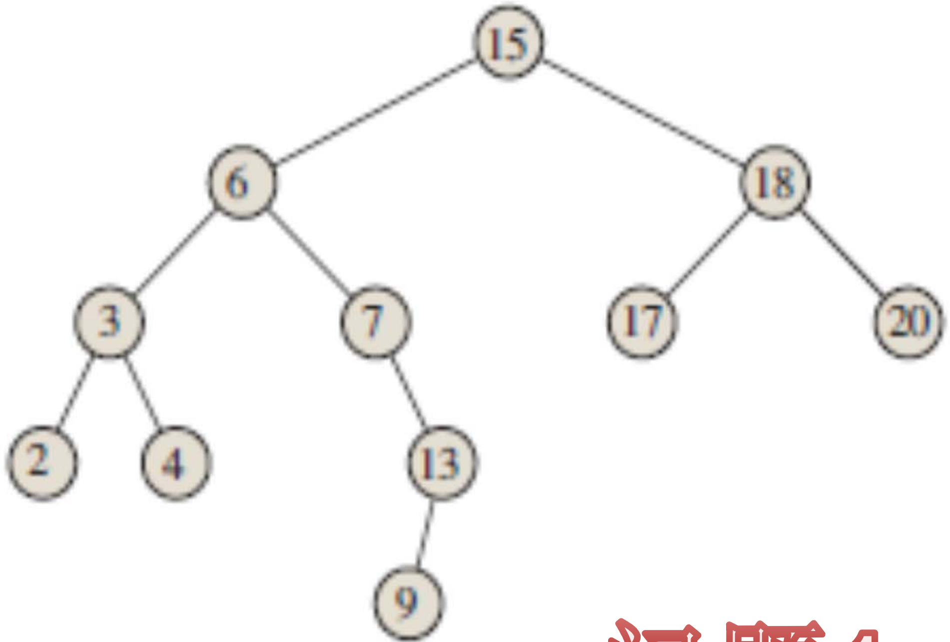
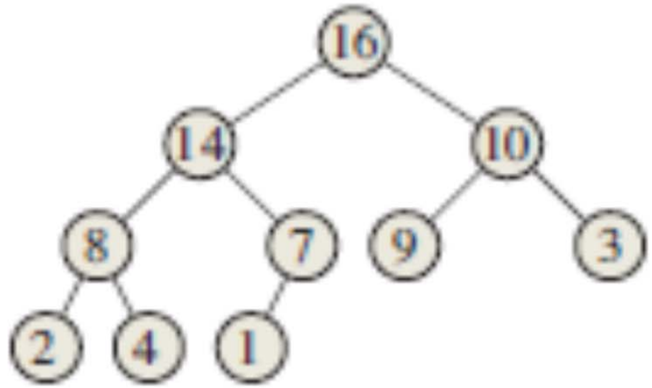
计算机问题求解 – 论题2-11

- 搜索树

2019年05月06日

Part I
搜索效率与平衡





问题1:

这各是什么结构？他们有什么相同与不同之处？

Binary-Search-Tree Property

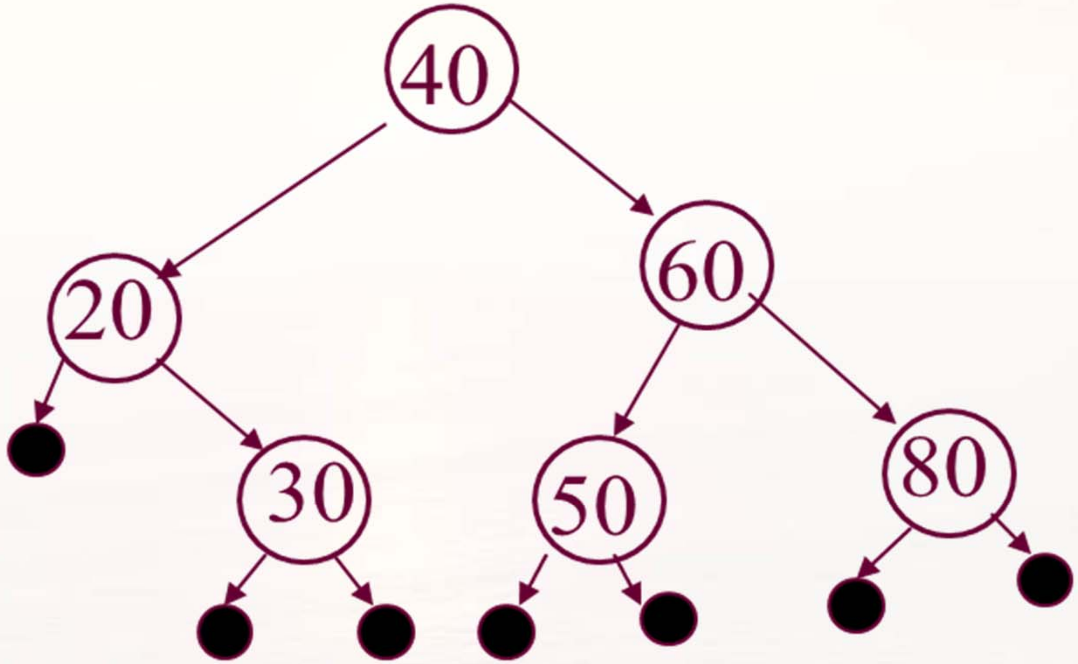
Let x be a node in a binary search tree. If y is a node in the left subtree of x, then $y.key \le x.key$. If y is a node in the right subtree of x, then $y.key \ge x.key$.

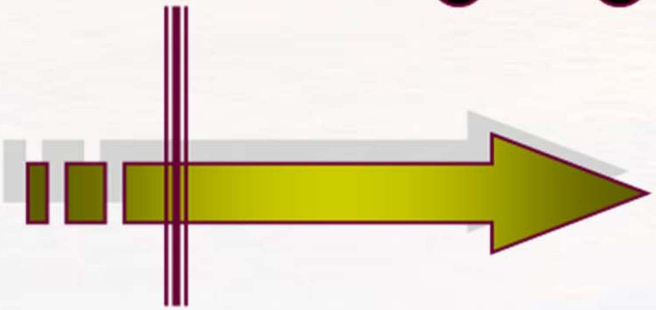
```
INORDER-TREE-WALK(x)
if x ≠ NIL
    INORDER-TREE-WALK(x.left)
    print x.key
    INORDER-TREE-WALK(x.right)
```

问题2:

你能解释这个过程对一个binary search tree执行的结果吗?你能否从BST的性质来说明为什么是这样的结果?

Properly Drawn Tree





In a properly drawn tree, pushing forward to get the ordered list.

“扫描”BST的代价是线性的

If x is the root of an n-node subtree, then the call INORDER-TREE-WALK(x) takes $\Theta(n)$ time.

We use the substitution method to show that $T(n) = O(n)$ by proving that $T(n) \le (c+d)n + c$. For $n = 0$, we have $(c+d)\cdot 0 + c = c = T(0)$. For $n > 0$, we have

$$
\begin{aligned}
T(n) &\le T(k) + T(n-k-1) + d \\
&= ((c+d)k + c) + ((c+d)(n-k-1) + c) + d \\
&= (c+d)n + c - (c+d) + c + d \\
&= (c+d)n + c ,
\end{aligned}
$$

其实是数学归纳法

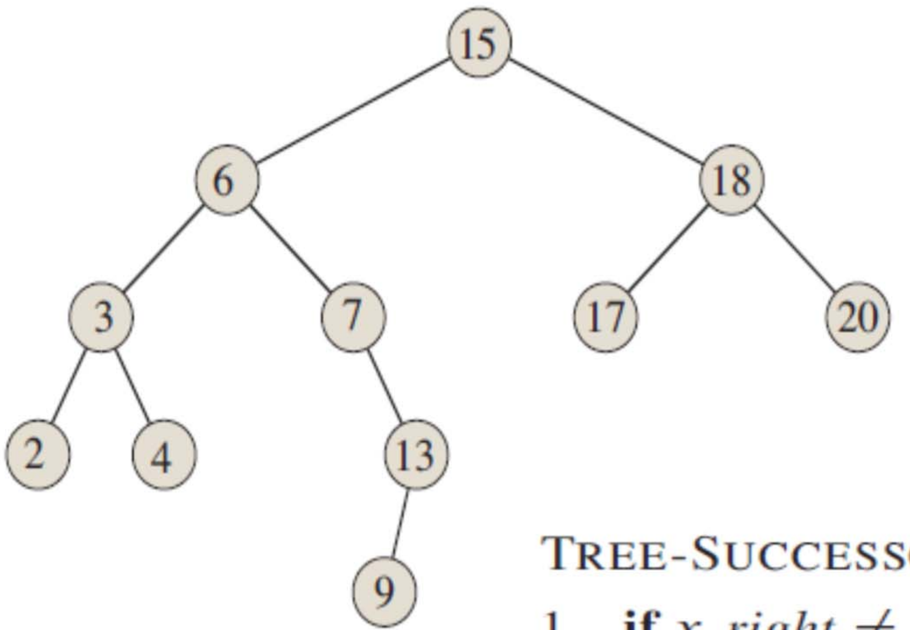
问题3：

为什么二分搜索树性质使得搜索很方便？效率是否也很高呢？

```
TREE-SEARCH(x, k)
if x == NIL or k == x.key
    return x
if k < x.key
    return TREE-SEARCH(x.left, k)
else return TREE-SEARCH(x.right, k)
```

```
ITERATIVE-TREE-SEARCH(x, k)
while x ≠ NIL and k ≠ x.key
    if k < x.key
        x = x.left
    else x = x.right
return x
```

BST中结点的“后继”



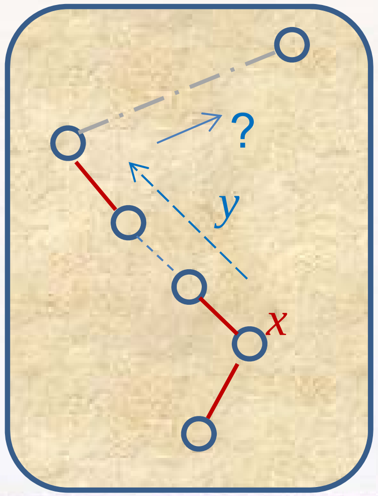
问题4：

你能否结合左图，解释下面的过程，特别是红框中的部分？

什么情况下，*y*是NIL?

```
TREE-SUCCESSOR(x)
if x.right ≠ NIL
        return TREE-MINIMUM(x.right)
y = x.p
while y ≠ NIL and x == y.right
        x = y
        y = y.p
return y
```

后继是右子树中最小元，或者…



问题5：

用BST实现动态集合，为什么需要过程Tree-Successor?还需要其他什么辅助过程吗？

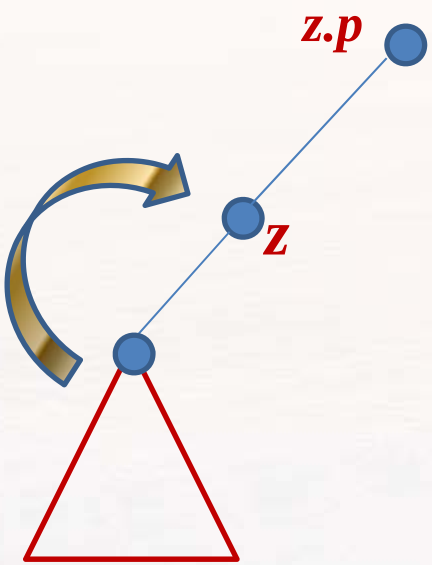
问题6：为什么在搜索树中删除比插入复杂？

```
TREE-INSERT(T, z)
y = NIL
x = T.root
while x ≠ NIL
    y = x
    if z.key < x.key
        x = x.left
    else x = x.right
z.p = y
if y == NIL
    T.root = z          // tree T was empty
elseif z.key < y.key
    y.left = z
else y.right = z
```

插入的位置一定是叶子

问题7：

什么情况下待删除结点的后继即其父结点，这对删除操作带来什么方便？



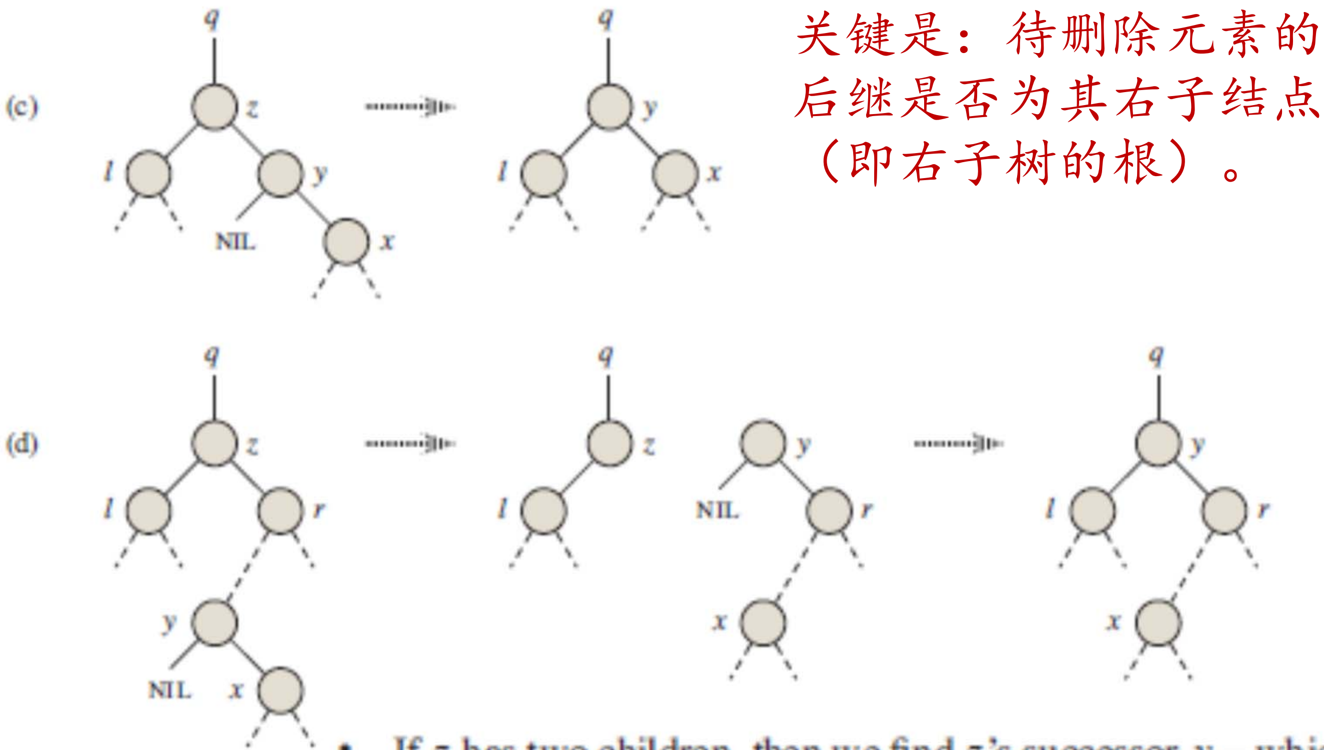

```
TRANSPLANT(T, u, v)
if u.p == NIL
    T.root = v
elseif u == u.p.left
    u.p.left = v
else u.p.right = v
if v ≠ NIL
    v.p = u.p
```

问题8：

待删结点的左子树为空怎么样呢？

从BST中删除

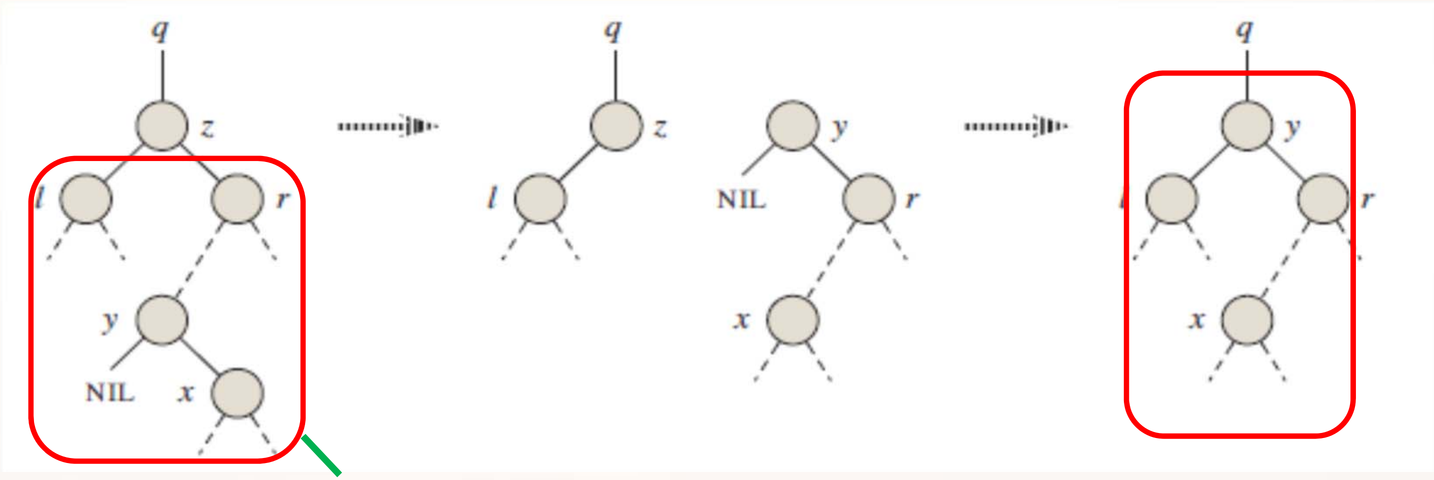
假设待删除元素所在结点左右子树皆非空:



关键是：待删除元素的后继是否为其右子结点（即右子树的根）。

- If z has two children, then we find z's successor y—which must be in z's right subtree—and have y take z's position in the tree. The rest of z's original right subtree becomes y's new right subtree, and z's left subtree becomes y's new left subtree. This case is the tricky one because, as we shall see, it matters whether y is z's right child.

看得更仔细一点




```
else y = TREE-MINIMUM(z.right)
    if y.p ≠ z
        TRANSPLANT(T, y, y.right)
        y.right = z.right
        y.right.p = y
    TRANSPLANT(T, z, y)
    y.left = z.left
    y.left.p = y
```

删除z后，*x*,*y*,*r* 这三个结点“父子”关系被改变了！

问题9:

BST是否能够有效地实现动态集合,为什么?

平衡程度是关键!

Part II
红黑树

Red-Black Property

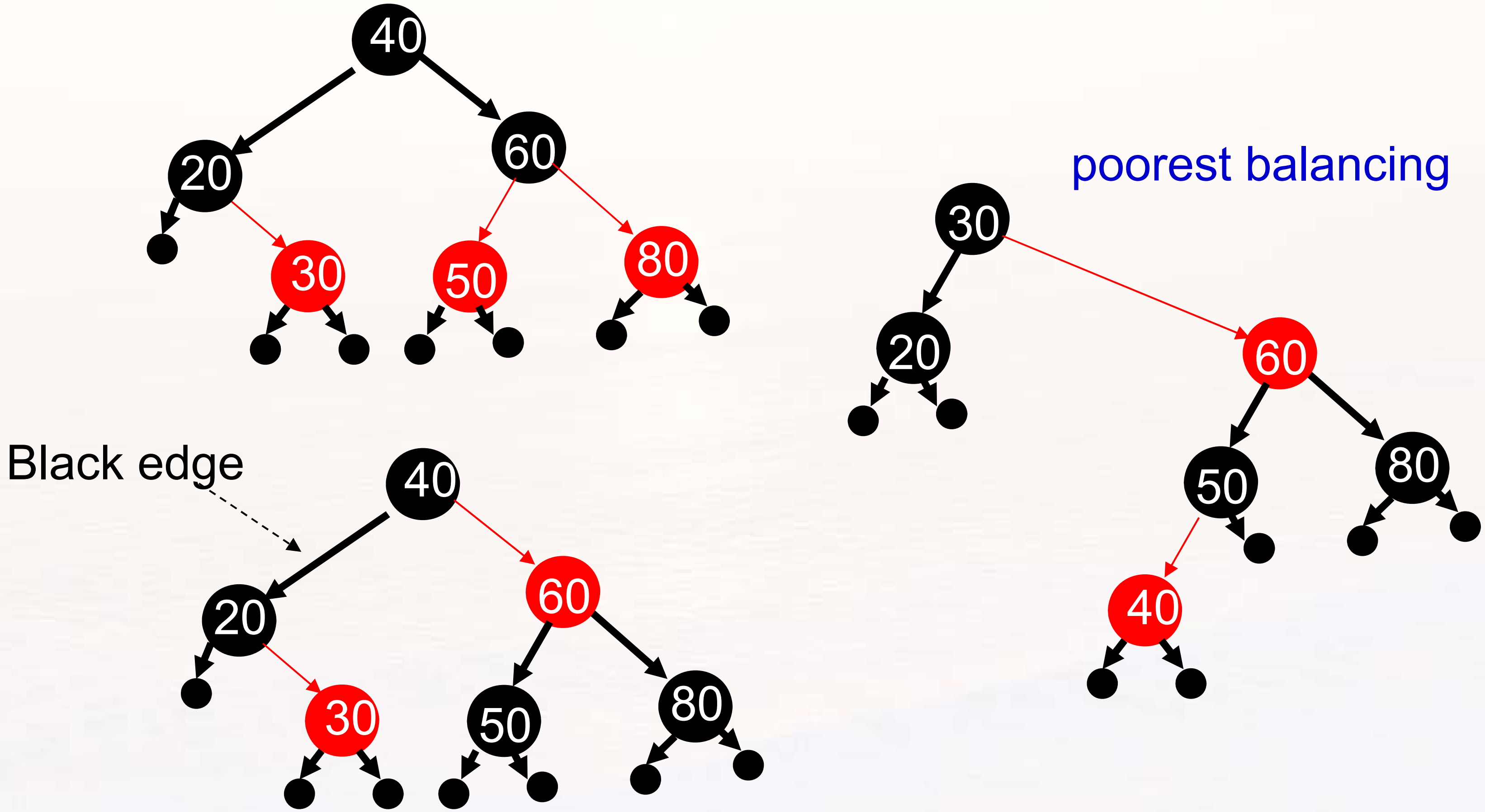
A red-black tree is a binary tree that satisfies the following ***red-black properties***:

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both its children are black.
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

问题10：

你能描述一下在这样的树中任一从根到叶的通路有什么特征吗？为什么说红黑树是approximately balanced?

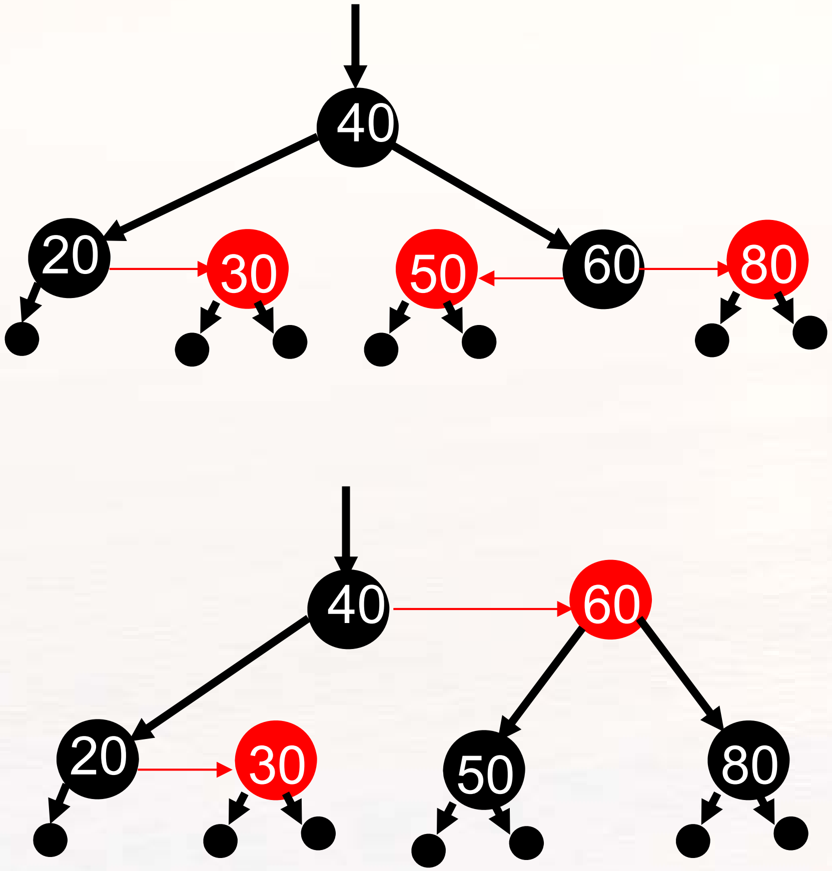
6个结点的红黑树

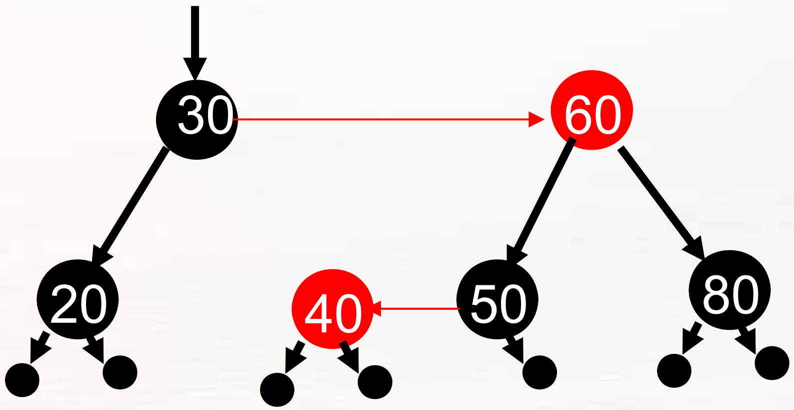


40
20
60
30
50
80
poorest balancing
30
20
60
50
80
40
Black edge
40
20
60
30
50
80

Black-Depth Convention

All with the same largest black depth: 2





红黑树高度的上限

- Let T be a red-black tree with n internal nodes，the height of T in the usual sense is at most $2\lg(n+1)$.
  - 引理：以x为根的子树至少包含$2^{bh(x)}-1$个内部结点。（这个引理很容易用数学归纳法证明）

To complete the proof of the lemma, let h be the height of the tree. According to property 4, at least half the nodes on any simple path from the root to a leaf, not including the root, must be black. Consequently, the black-height of the root must be at least $h/2$; thus,

$$n \geq 2^{h/2} - 1 .$$

Moving the 1 to the left-hand side and taking logarithms on both sides yields $\lg(n+1) \geq h/2$, or $h \leq 2\lg(n+1)$. ■

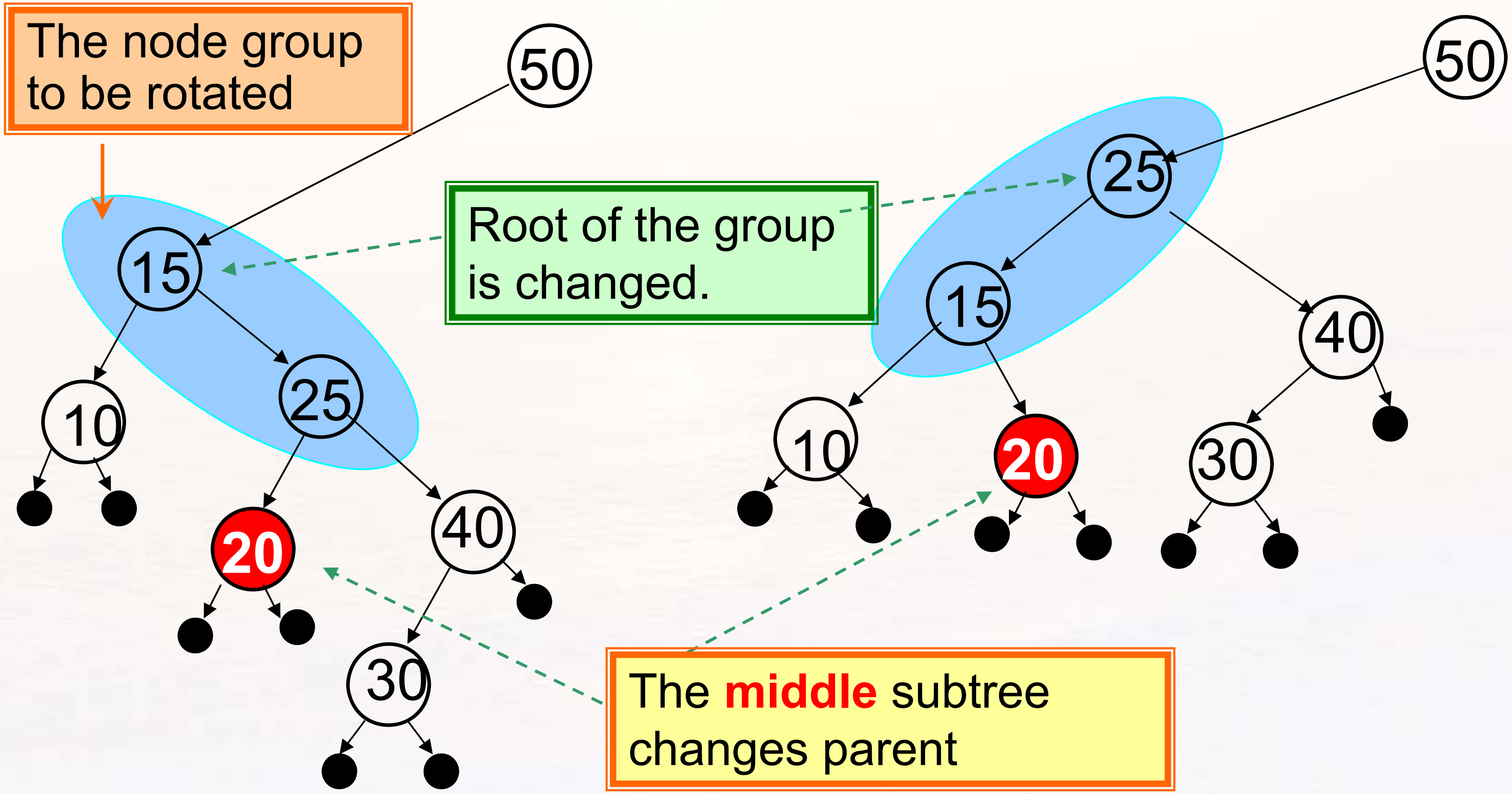
问题11：

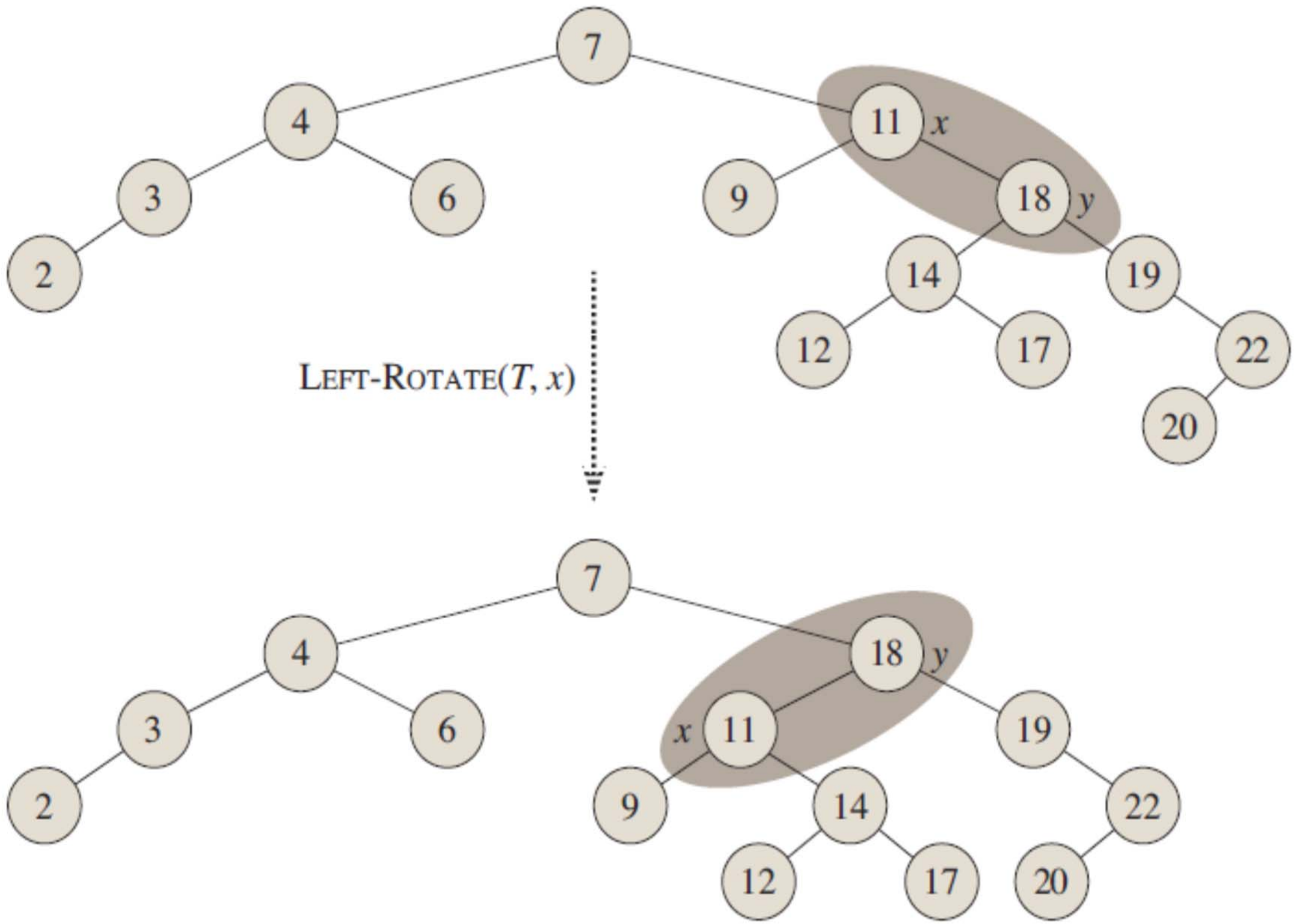
Red-Black树的Dynamic Set Operation与一般BST的有什么相同与不同之处？$2\lg(n+1)$: 这个结论有什么意义？

问题12：

如果我们想提高树的平衡度，又不破坏搜索性质，有什么办法？为什么可以这样做？

Improving the Balancing by Rotation




7
4
11
x
3
6
9
18
y
2
14
19
12
17
22
20
LEFT-ROTATE(T, x)
7
4
18
y
3
6
x
11
19
2
9
14
22
12
17
20

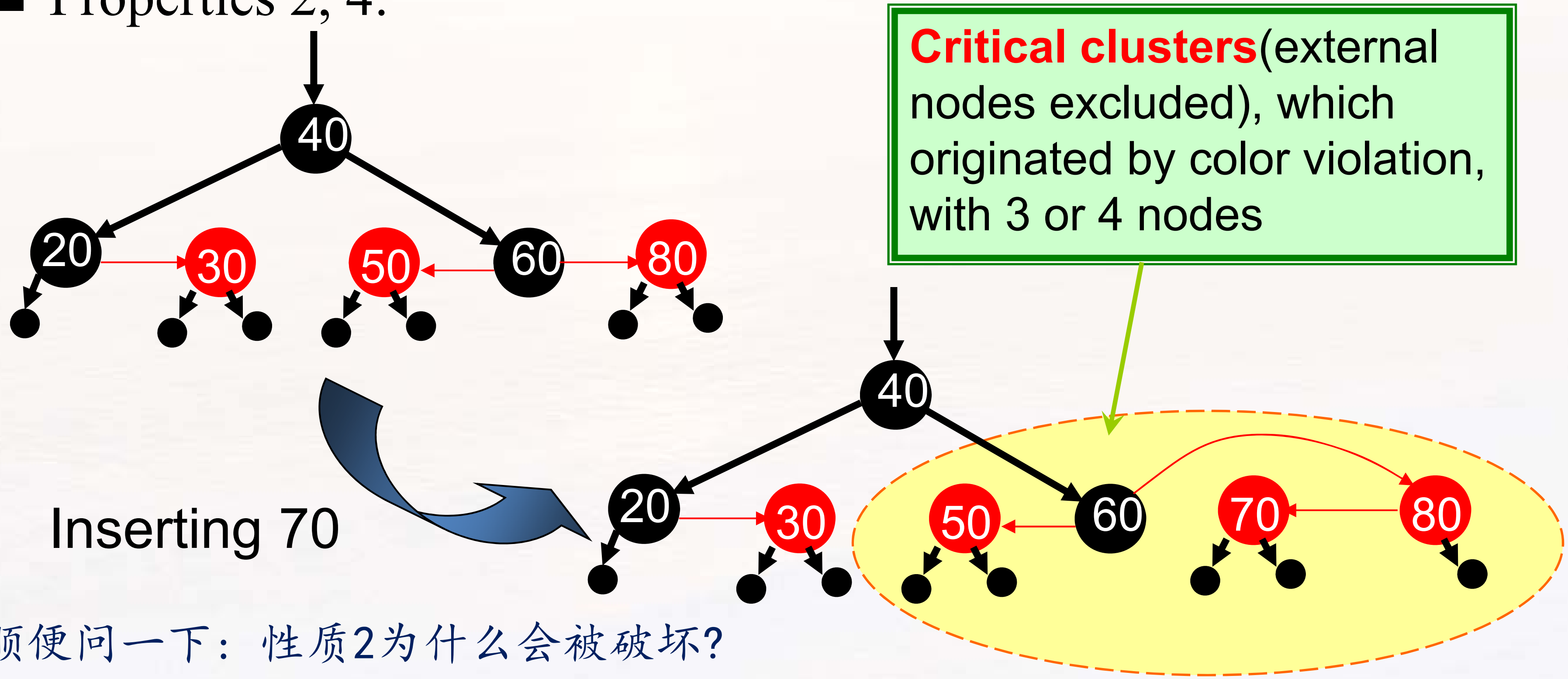
问题13：

在红黑树中插入元素与在一般BST中插入有什么不同？

关键是颜色的处理。

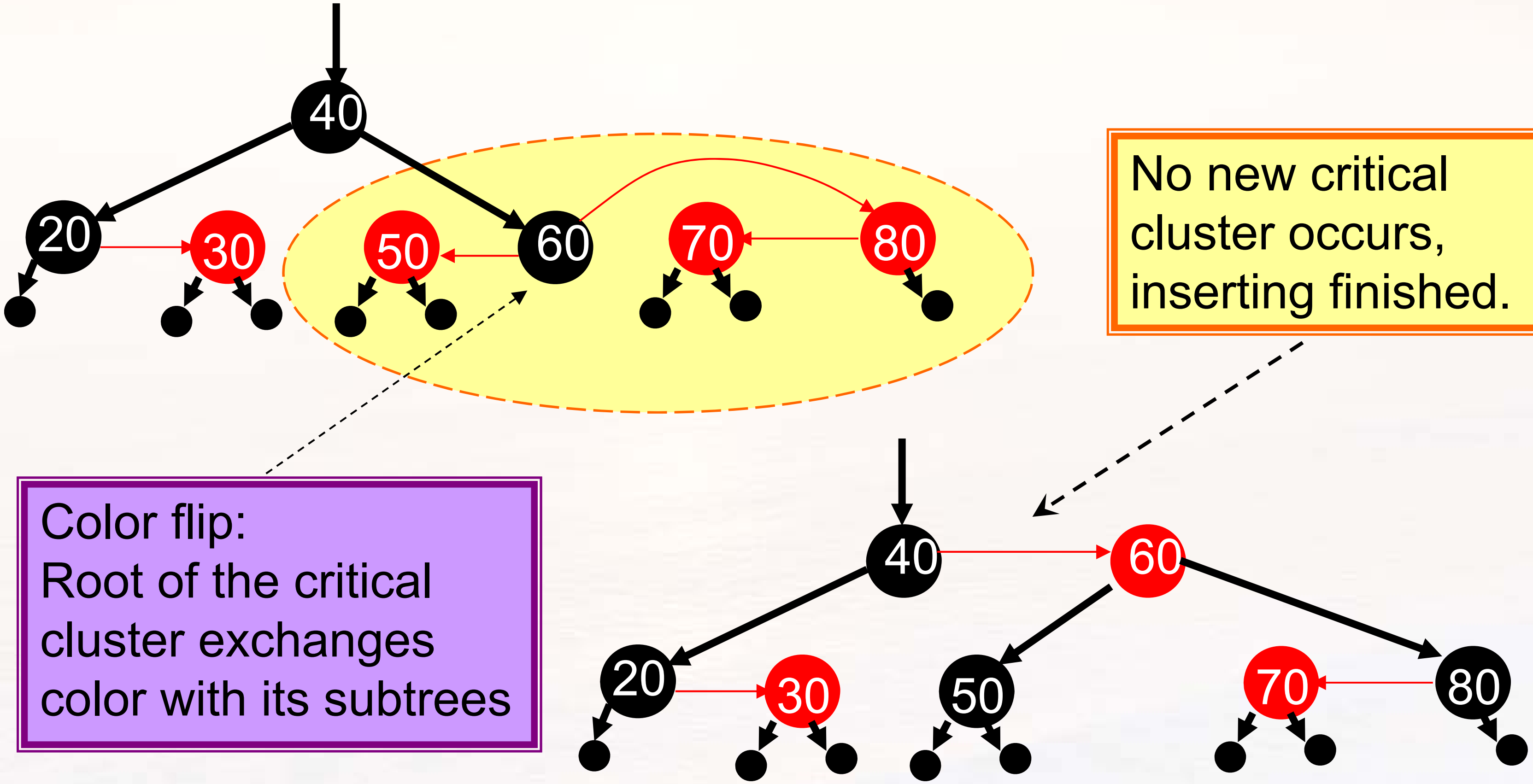
Influences of Insertion into an RB Tree

- Properties 1, 3, 5:
  - No violation ***if*** inserting a red node.
- Properties 2, 4:



顺便问一下：性质2为什么会被破坏?

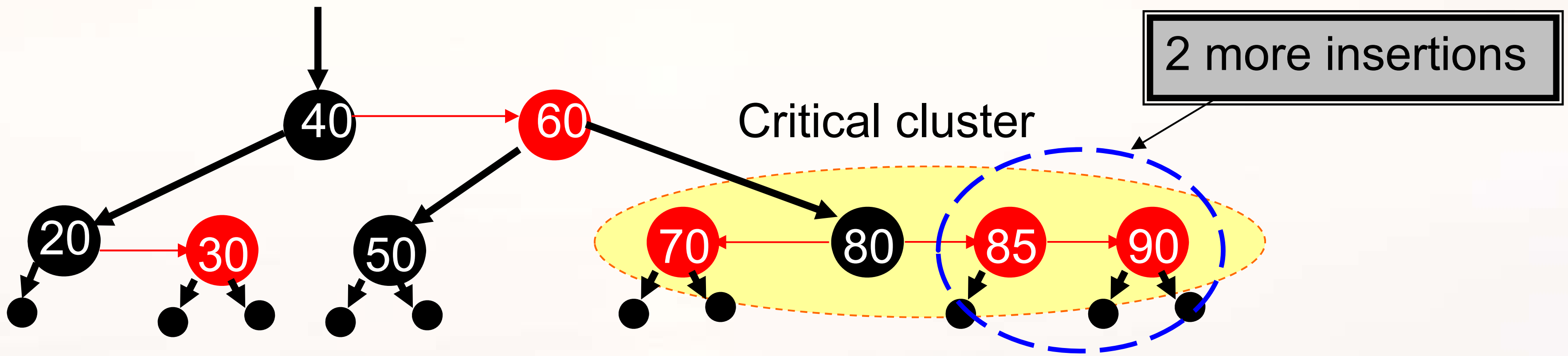
Repairing 4-node Critical Cluster



No new critical cluster occurs, inserting finished.

Color flip:
Root of the critical cluster exchanges color with its subtrees

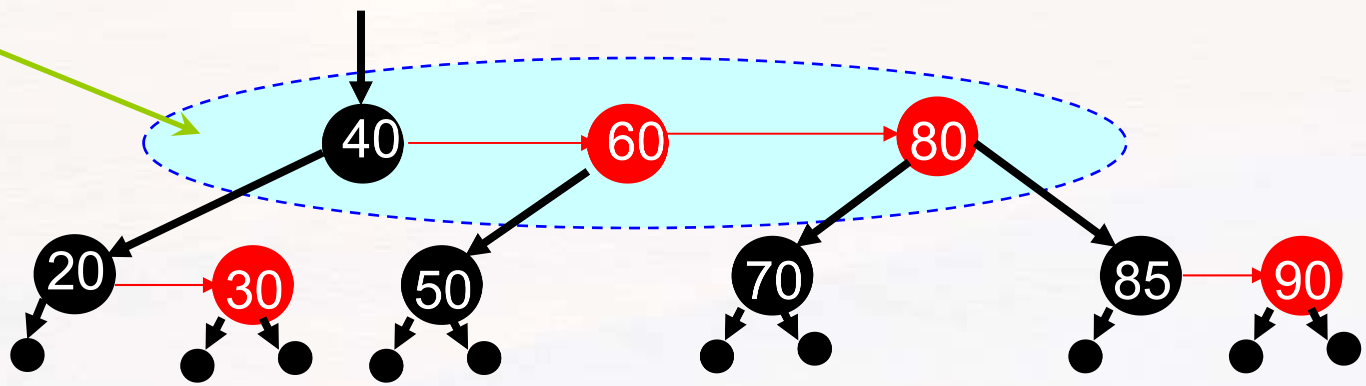
Repairing 4-node Critical Cluster



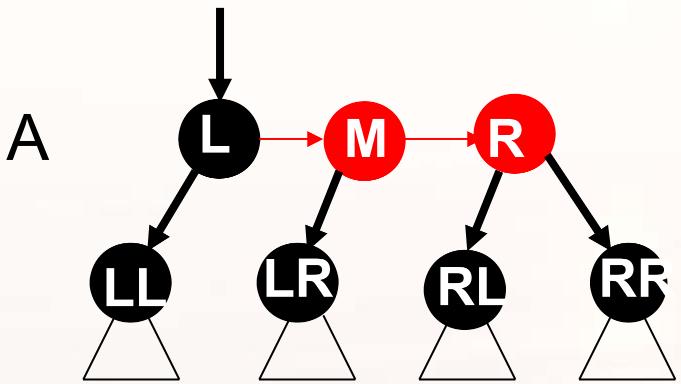
New critical cluster with 3 nodes.

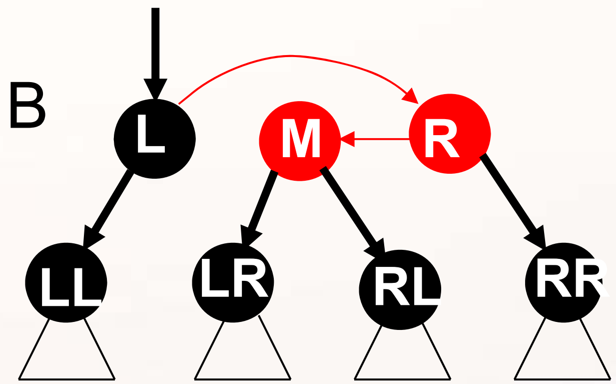
Color flip doesn't work,

Why?

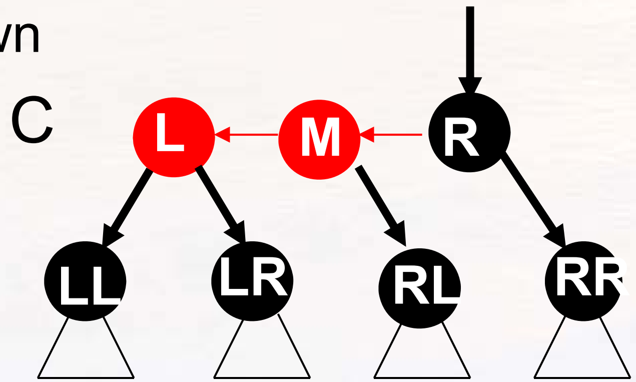


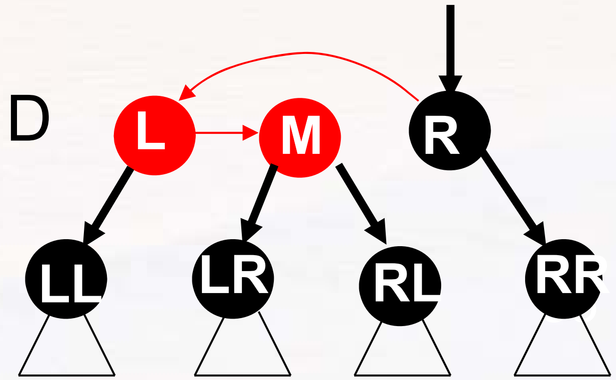
Patterns of 3-Node Critical Cluster





Shown as properly drawn

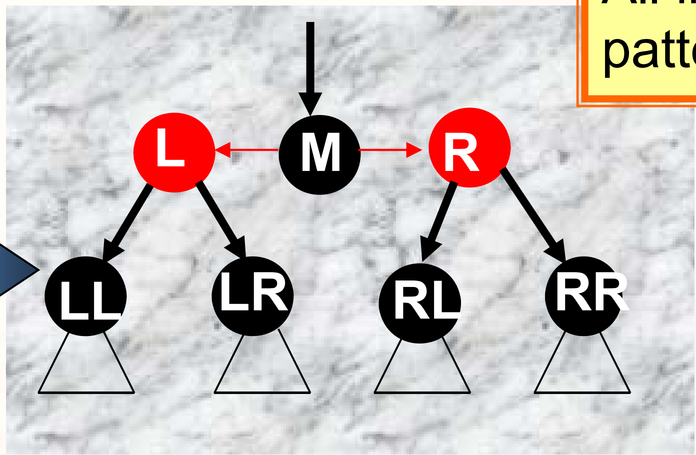




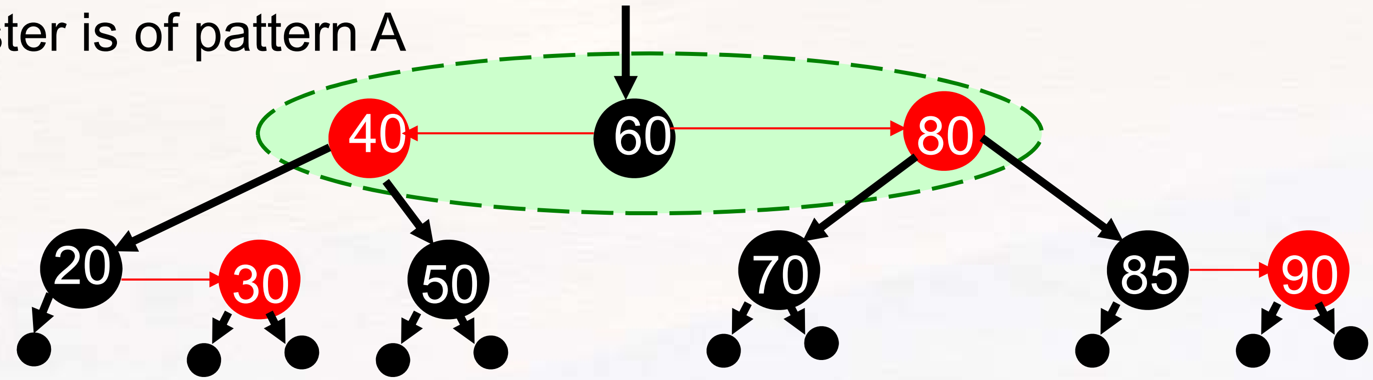
Repairing 3-Node Critical Cluster

All into one pattern

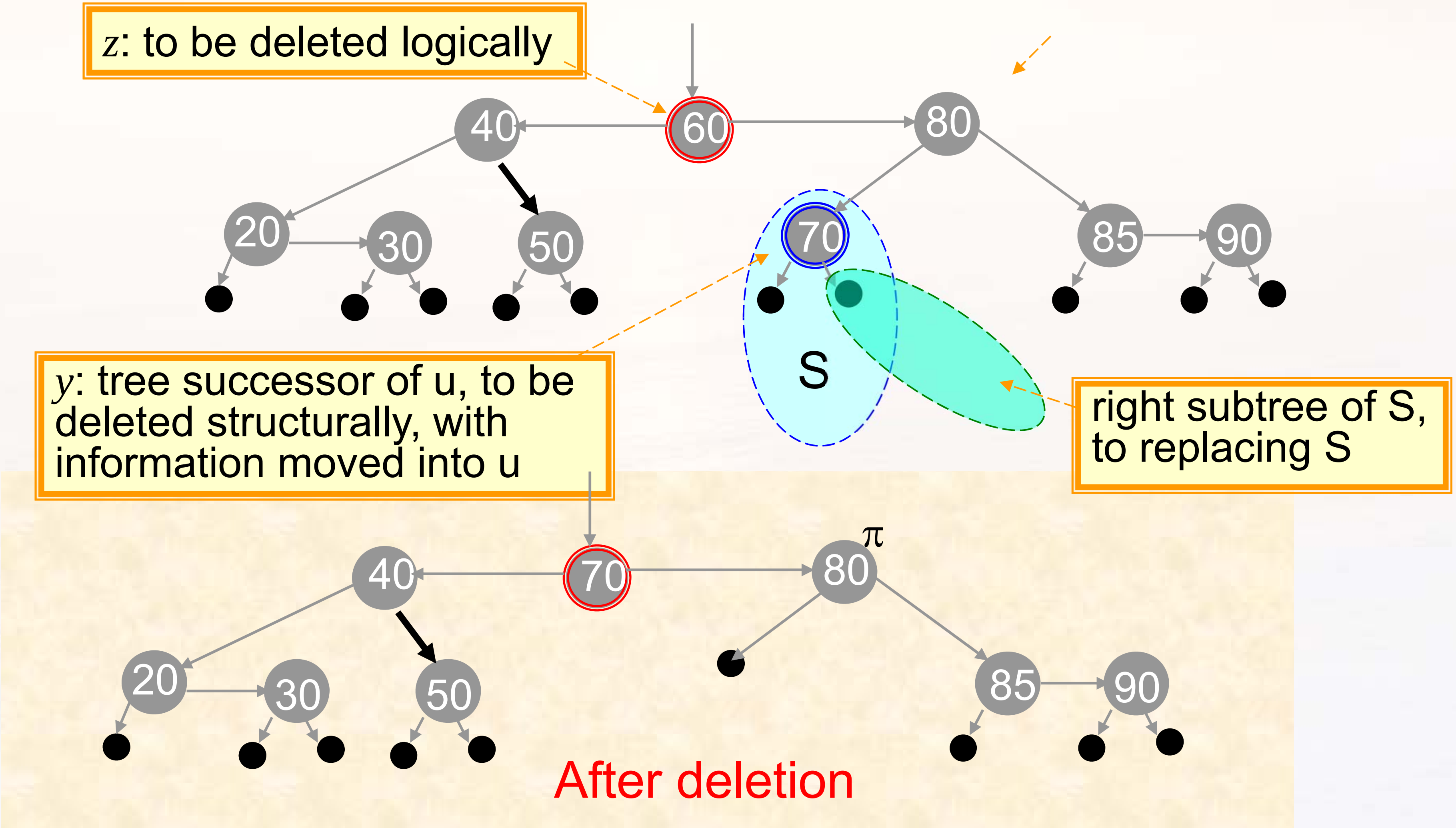
Root of the critical cluster is changed to **M**, and the parentship is adjusted accordingly



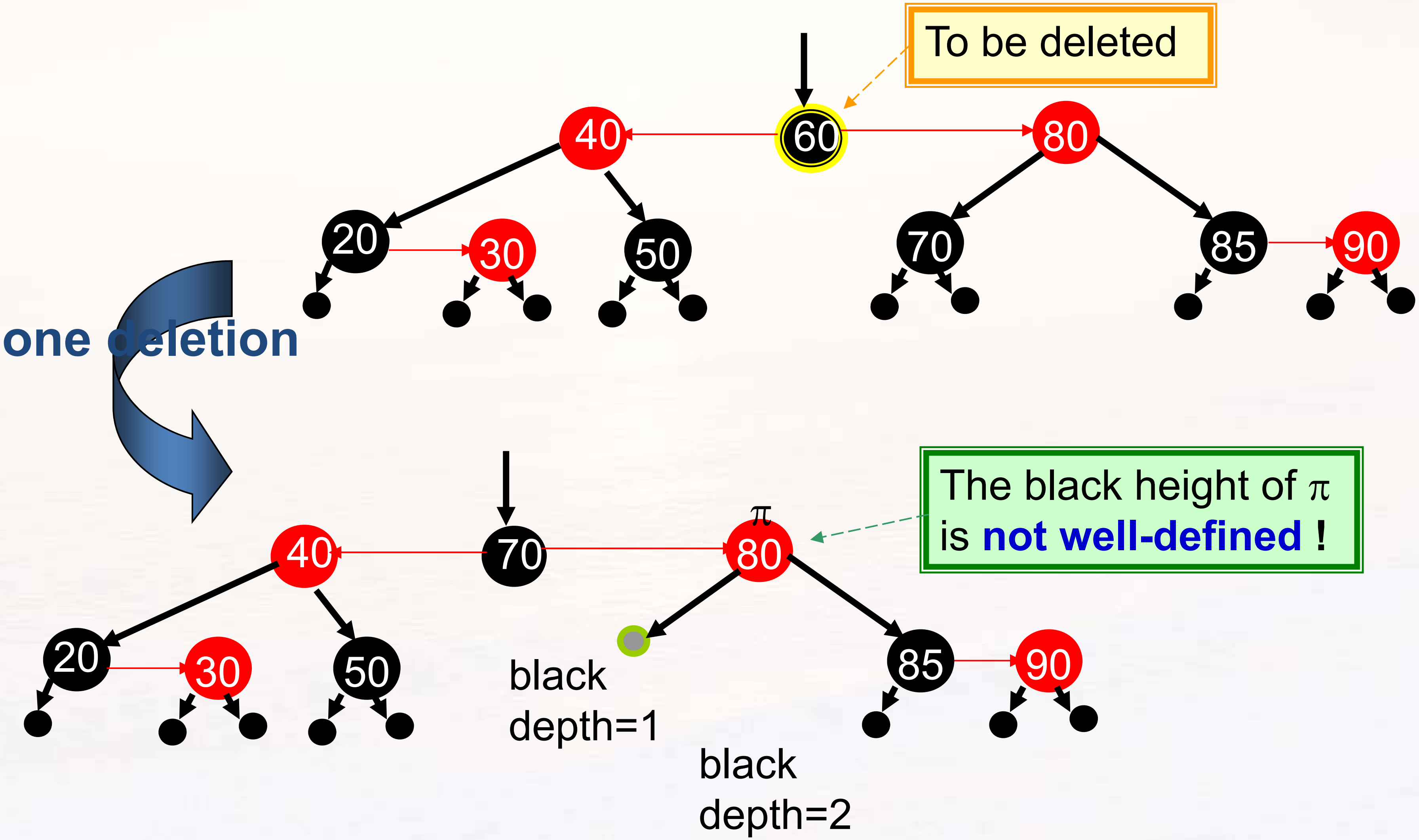
The incurred critical cluster is of pattern A



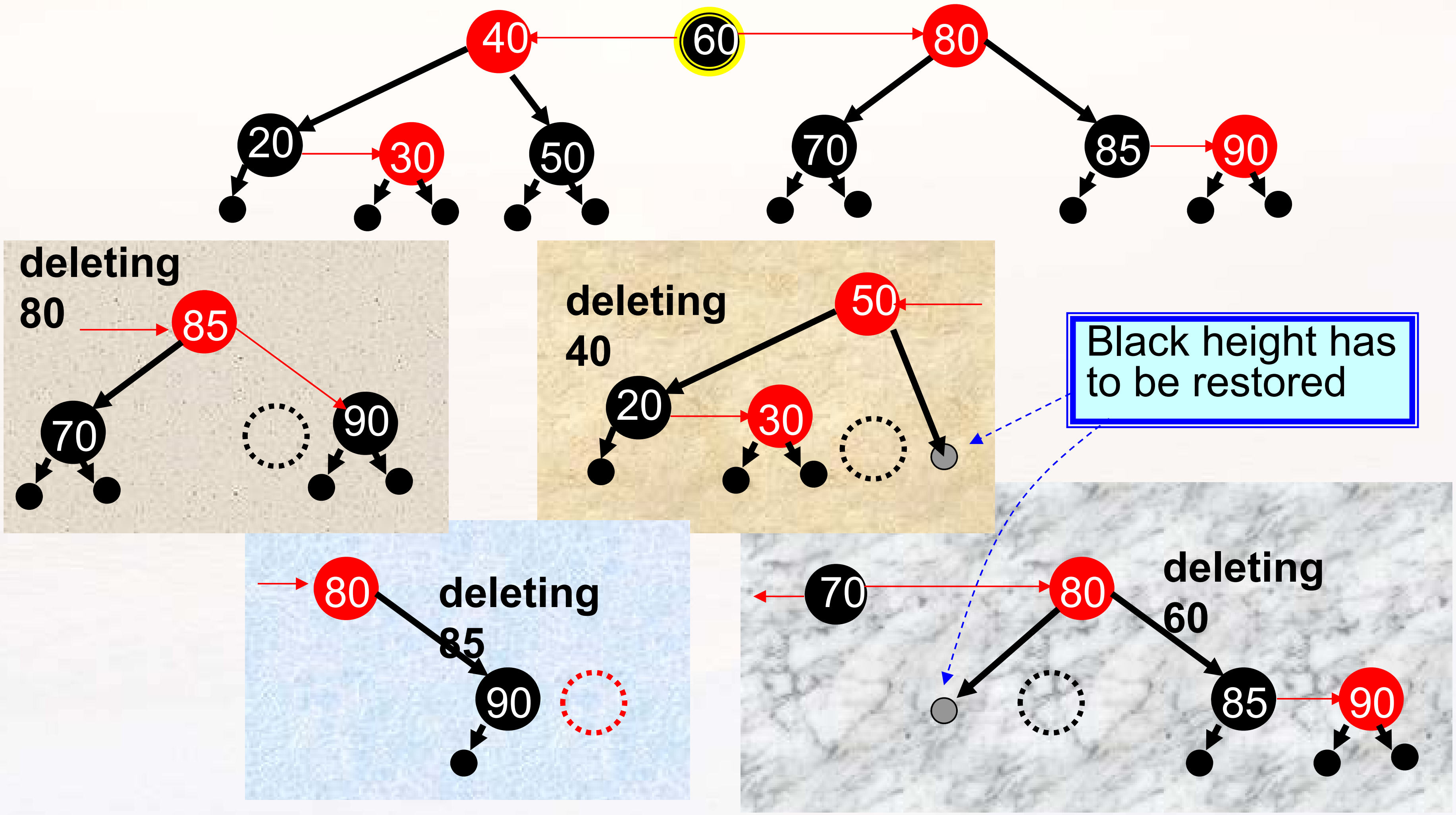
Deletion: Logical and Structral



Deletion in a Red-Black Tree



Imbalance of Black Height

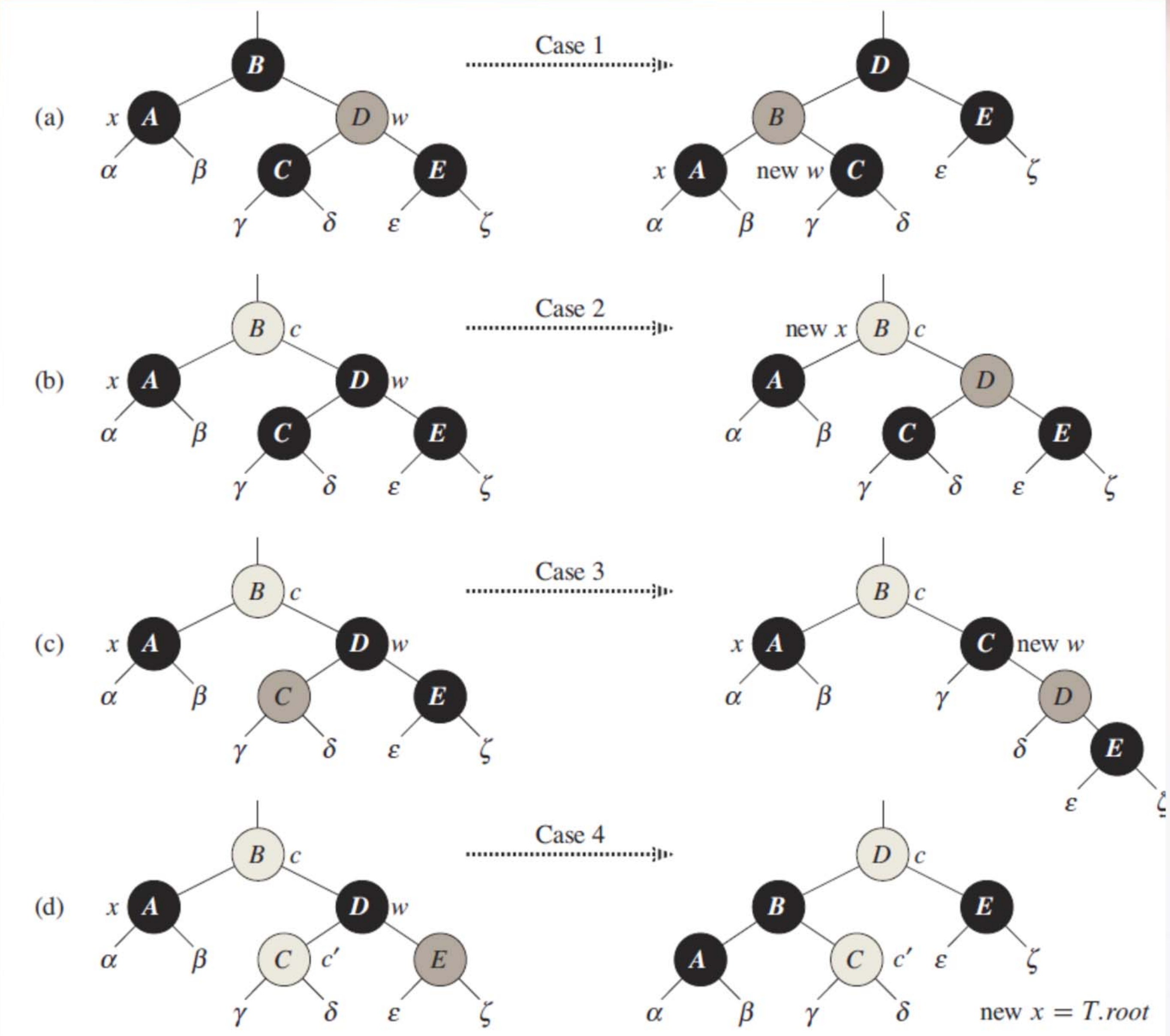


问题14:

在红黑树中删除时，什么情况会破坏红黑性质？

If node y was black, three problems may arise, which the call of RB-DELETE-FIXUP will remedy. First, if y had been the root and a red child of y becomes the new root, we have violated property 2. Second, if both x and $x.p$ are red, then we have violated property 4. Third, moving y within the tree causes any simple path that previously contained y to have one fewer black node. Thus, property 5 is now violated by any ancestor of y in the tree.

“双重”颜色的修复。



Case 1
Case 2
Case 3
Case 4
(a)
(b)
(c)
(d)
x
w
new w
new x
c
c′
new x = T.root
A
B
C
D
E
α
β
γ
δ
ε
ζ

问题15：

为什么不能将堆和搜索树各自的优点结合起来？

课外作业

- TC pp.289-: ex.12.1-2, 12.1-5
- TC pp.293-: ex.12.2-5, 12.2-8, 12.2-9
- TC pp.299-: ex.12.3-5
- TC pp.303-: prob.12-1
- TC pp.311-: ex.13.1-5, 13.1-6, 13.1-7
- TC pp.313-: ex.13.2-2
- TC pp.322-: ex.13.3-1, 13.3-5
- TC pp.330-: ex. 13.4-1, 13.4-2, 13.4-7