Homework Solutions – Unit 3: Chapter 13

CMPSC 465

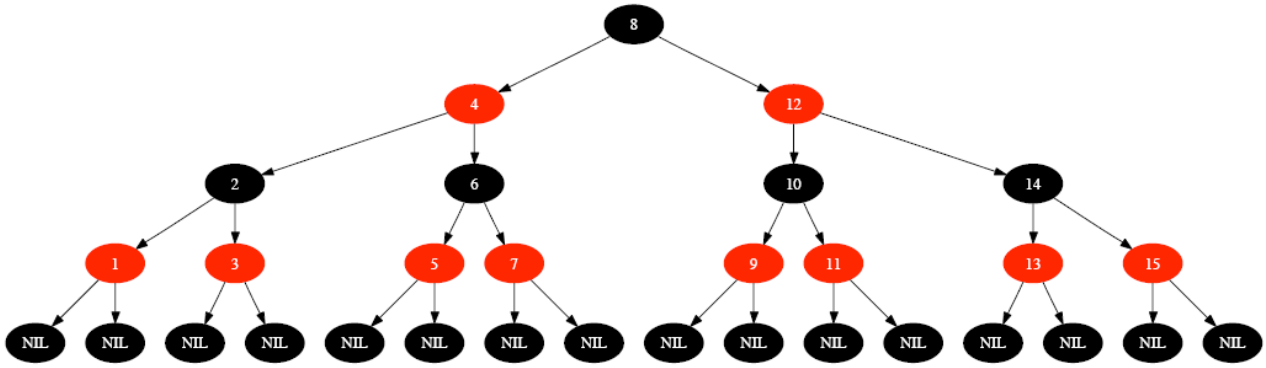
Disclaimer: This is a draft of solutions that has been prepared by the TAs and the instructor makes no guarantees that solutions presented here contain the level of detail that would be expected on an exam. Any errors or explanations you find unclear should be reported to either of the TAs for corrections first.

Exercise 13.1-1

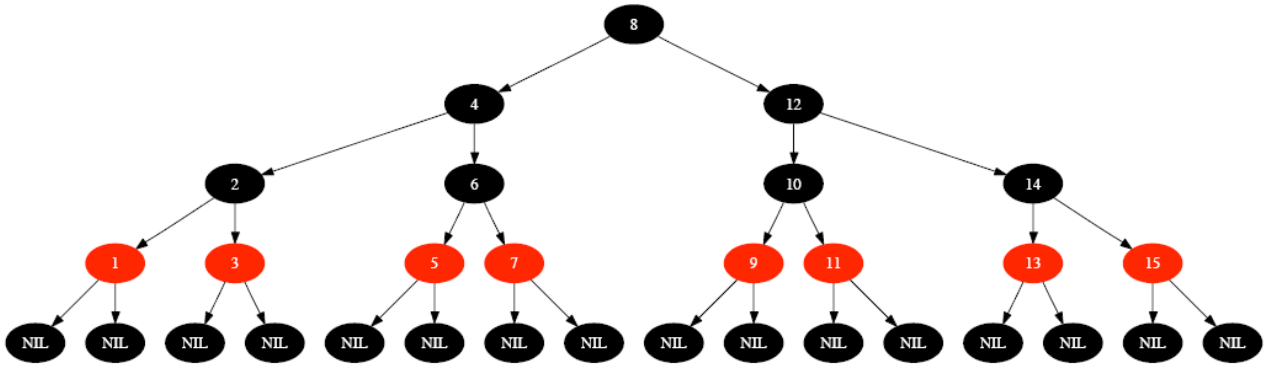
In the style of Figure 13.1(a), draw the complete binary search tree of height 3 on the keys {1, 2, …, 15}. Add the NIL leaves and color the nodes in three different ways such that the black-heights of the resulting red-black trees are 2, 3, and 4.

Solution:

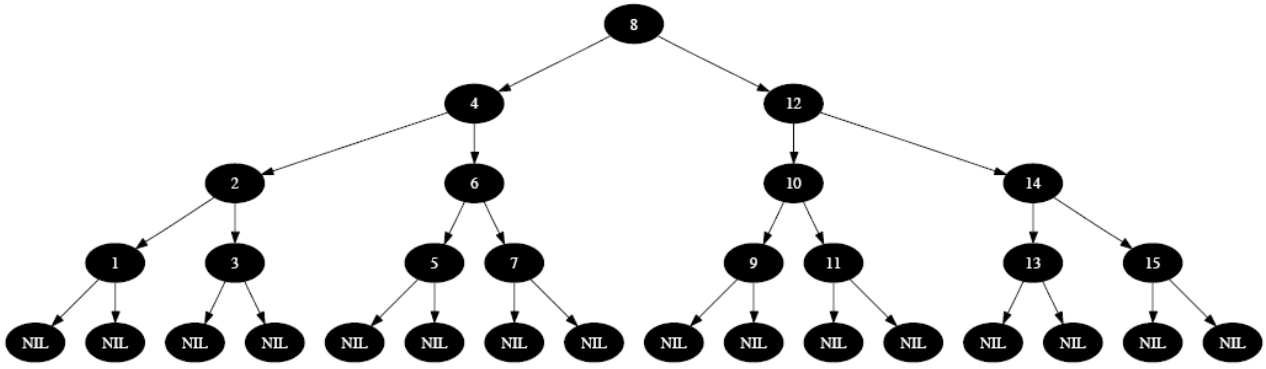
Black-height 2:



Black-height 3:

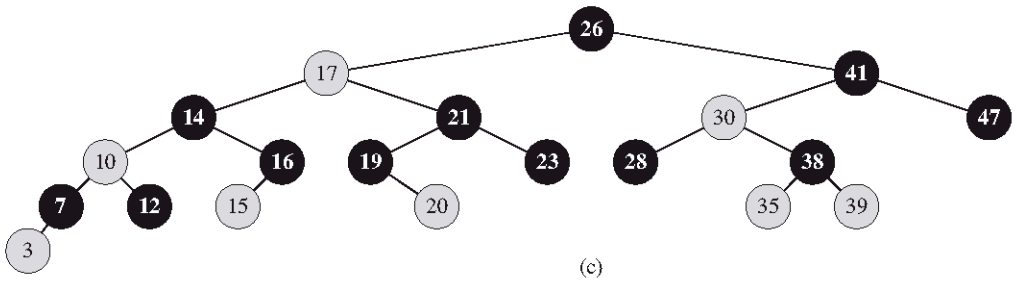


Black-height 4:



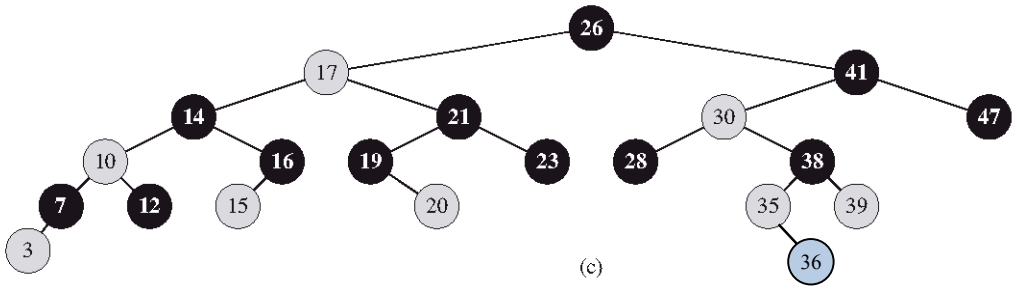
Exercise 13.1-2

Draw the red-black tree that results after TREE-INSERT is called on the tree shown in the figure below with key 36. If the inserted node is colored red, is the resulting tree a red-black tree? What if it is colored black?



Solution:

If the node with key 36 is inserted and colored red, the red-black tree becomes:

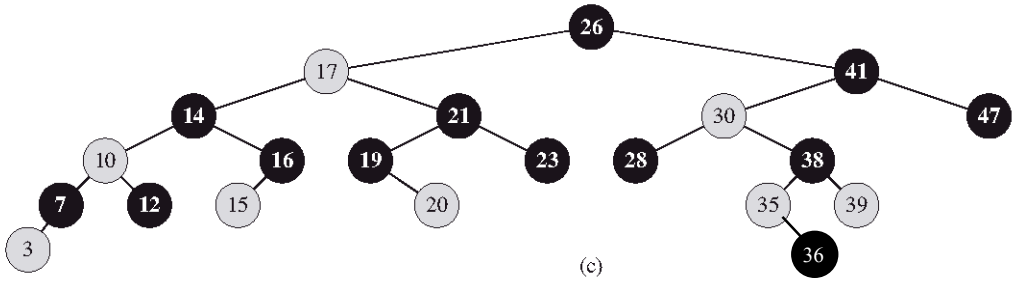


We can see that it violates following red-black tree property:

A red node in the red-black tree cannot have a red node as its child.

So the resulting tree is not a red-black tree.

If the node with key 36 is inserted and colored black, the red-black tree becomes:



We can see that it violates following red-black tree property:

For each node, all paths from the node to descendent leaves contain the same number of black nodes (e.g. consider node with key 30).

So the resulting tree is not a red-black tree either.

Exercise 13.1-5

Show that the longest simple path from a node x in a red-black tree to a descendant leaf has length at most twice that of the shortest simple path from node x to a descendant leaf.

Proof:

In the longest path, at least every other node is black. In the shortest path, at most every node is black. Since the two paths contain equal numbers of black nodes, the length of the longest path is at most twice the length of the shortest path.

We can say this more precisely, as follows:

Since every path contains bh(x) black nodes, even the shortest path from x to a descendant leaf has length at least bh(x). By definition, the longest path from x to a descendant leaf has length height(x). Since the longest path has bh(x) black nodes and at least half the nodes on the longest path are black (by property 4 in CLRS), bh(x) ≥ height(x)/2, so

length of longest path = height(x) ≤ 2×bh(x) ≤ twice length of shortest path.

Exercise 13.2-1

Write pseudocode for RIGHT-ROTATE.

Solution:

The pseudocode for RIGHT-ROTATE is shown below:

```
RIGHT-ROTATE (T, x)
    y = x.left                  //set y
    x.left = y.right            // turn y's right subtree into x's left subtree
    if y.right ≠ NIL
        y.right.p = x
    y.p = x:p                   // link x's parent to y
    if x.p == NIL
        T.root = y
    else if x == x.p.right
        x.p.right = y
    else
        x.p.left = y
    y.right = x                 //put x on y's right
    x.p = y
```

Exercise 13.2-4

Show that any arbitrary n-node binary search tree can be transformed into any other arbitrary n-node binary search tree using $O(n)$ rotations. (Hint: First show that at most $n-1$ right rotations suffice to transform the tree into a right-going chain.)

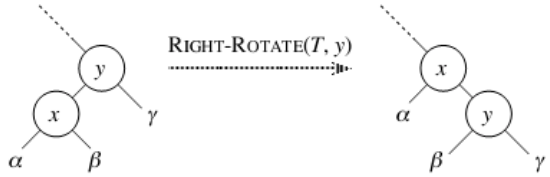
Solution:

Since the exercise asks about binary search trees rather than the more specific red-black trees, we assume here that leaves are full-fledged nodes, and we ignore the sentinels.

Taking the book's hint, we start by showing that with at most $n-1$ right rotations, we can convert any binary search tree into one that is just a right-going chain.

The idea is: Let us define the ***right spine*** as the root and all descendants of the root that are reachable by following only right pointers from the root. A binary search tree that is just a right-going chain has all n nodes in the right spine.

As long as the tree is not just a right spine, repeatedly find some node y on the right spine that has a non-leaf left child x and then perform a right rotation on y:



(In the above figure, note that any of α, β, and γ can be an empty subtree.)

Observe that this right rotation adds x to the right spine, and no other nodes leave the right spine. Thus, this right rotation increases the number of nodes in the right spine by 1. Any binary search tree starts out with at least one node — the root — in the right spine. Moreover, if there are any nodes not on the right spine, then at least one such node has a parent on the right spine. Thus, at most $n-1$ right rotations are needed to put all nodes in the right spine, so that the tree consists of a single right-going chain.

If we knew the sequence of right rotations that transforms an arbitrary binary search tree T to a single right-going chain T', then we could perform this sequence in reverse — turning each right rotation into its inverse left rotation — to transform T' back into T.

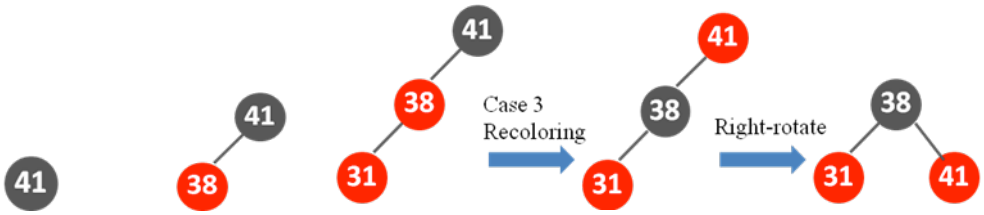
Therefore, here is how we can transform any binary search tree T_1 into any other binary search tree T_2. Let T' be the unique right-going chain consisting of the nodes of T_1 (which is the same as the nodes of T_2). Let $r = \langle r_1, r_2, ..., r_k \rangle$ be a sequence of right rotations that transforms T_1 to T', and let $r' = \langle r'_1, r'_2, ..., r'_{k'} \rangle$ be a sequence of right rotations that transforms T_2 to T'. We know that there exist sequences r and r' with $k, k' \le n-1$. For each right rotation r'_i, let l'_i be the corresponding inverse left rotation. Then the sequence $\langle r_1, r_2, ..., r_k, l'_{k'}, l'_{k'-1}, \ldots, l'_2, l'_1 \rangle$ transforms T_1 to T_2 in at most $2n-2$ rotations.

Exercise 13.3-2

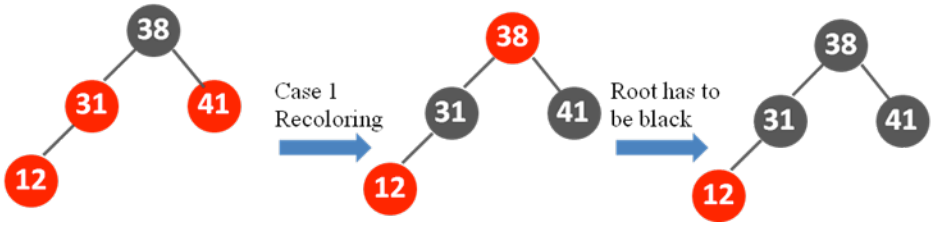
Show the red-black trees that result after successively inserting the keys 41, 38, 31, 12, 19, 8 into an initially empty red-black tree.

Solution:

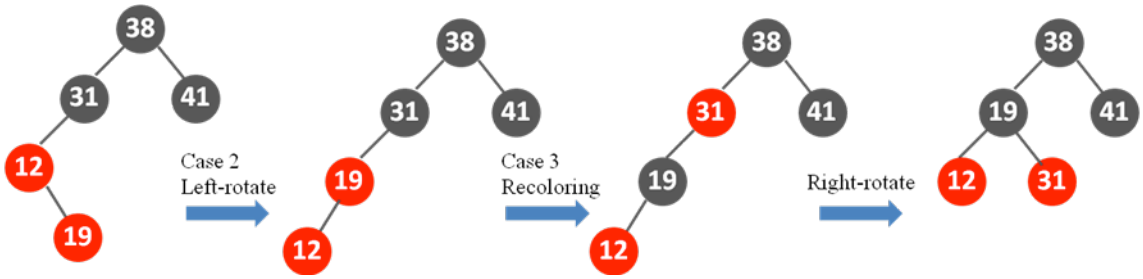
Insert 41 Insert 38 Insert 31



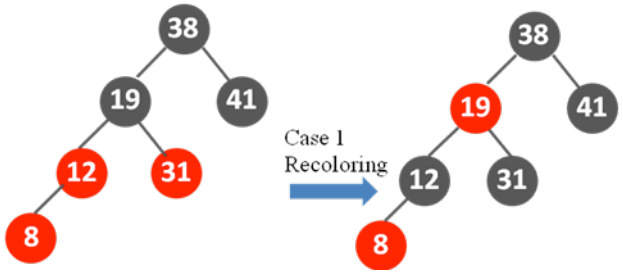
Insert 12



Insert 19



Insert 8



Exercise 13.3-3

Suppose that the black-height of each of the subtrees $\alpha, \beta, \gamma, \delta, \varepsilon$ in Figures 13.5 and 13.6 is k. Label each node in each figure with its black-height to verify that the indicated transformation preserves property 5.

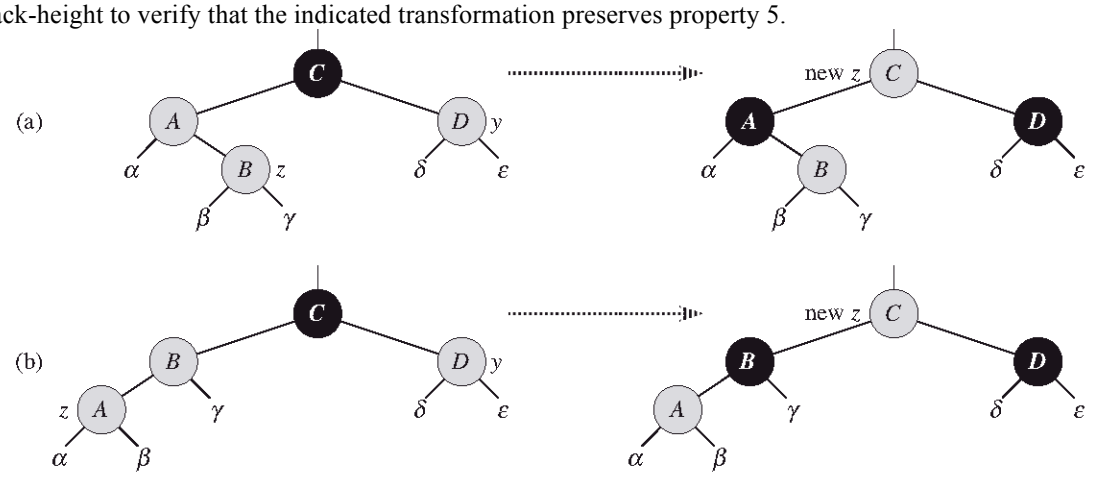


Figure 13.5

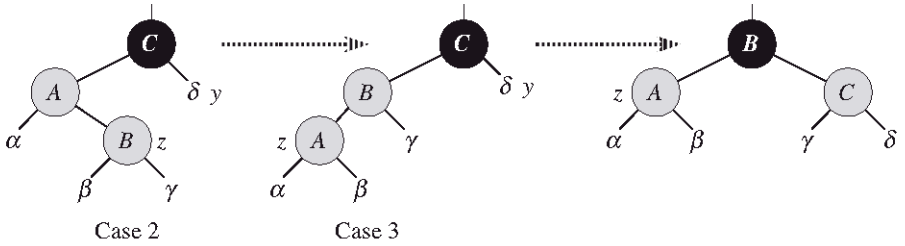
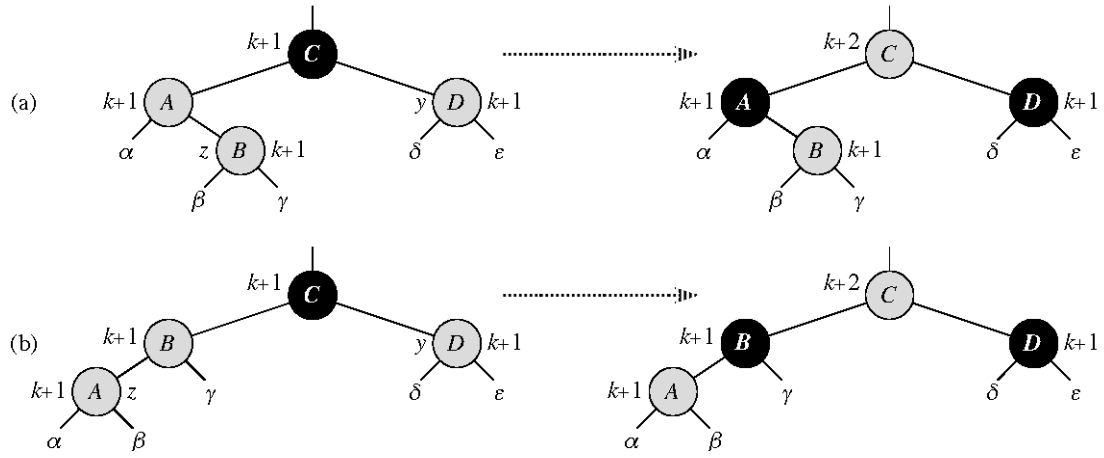


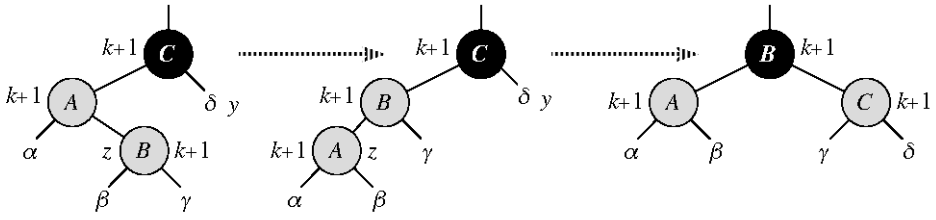
Figure 13.6

Solution:

In Figure 13.5, nodes A, B, and D have black-height $k + 1$ in all cases, because each of their subtrees has black-height k and a black root. Node C has black-height $k + 1$ on the left (because its red children have black-height $k + 1$) and black-height $k + 2$ on the right (because its black children have black-height $k + 1$).



In Figure 13.6, nodes A, B, and C have black-height $k + 1$ in all cases. At left and in the middle, each of A's and B's subtrees has black-height k and a black root, while C has one such subtree and a red child with black-height $k + 1$. At the right, each of A's and C's subtrees has black-height k and a black root, while B's red children each have black-height $k + 1$.



Property 5 is preserved by the transformations. We have shown above that the black-height is well-defined within the subtrees pictured, so property 5 is preserved within those subtrees. Property 5 is preserved for the tree containing the subtrees pictured, because every path through these subtrees to a leaf contributes $k + 2$ black nodes.