CSE 2320 Notes 12: Red-Black Trees

(Last updated 10/21/18 10:42 AM)

CLRS 13.1, 13.3

12.A. STRUCTURAL PROPERTIES

A *red-black tree* is a binary search tree whose height is $O(\log n)$ in the number of keys (n) stored.

1. Every node is colored red or black. (Colors are only examined during insertion and deletion)

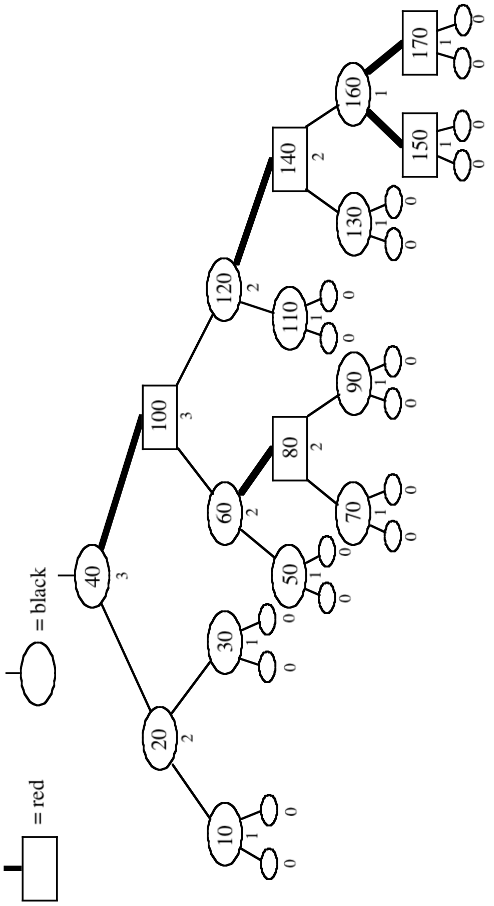
2. Every "leaf" (the sentinel) is colored black.

3. Both children of a red node are black.

4. Every simple path from a child of node X to a leaf has the same number of black nodes.

   This number is known as the *black-height* of X (bh(X)). These are not stored, but appear below nodes in some diagrams.

Example (`http://ranger.uta.edu/~weems/NOTES2320/REDBLACKC/notes12.page1.dat`):



```
                    [170 1 1]
                (160 3 1)
                    [150 1 1]
            [140 5 2]
                (130 1 1)
        (120 7 2)
            (110 1 1)
    [100 13 3]
                (90 1 1)
            [80 3 2]
                (70 1 1)
        (60 5 2)
            (50 1 1)
(40 17 3)
        (30 1 1)
    (20 3 2)
        (10 1 1)
```

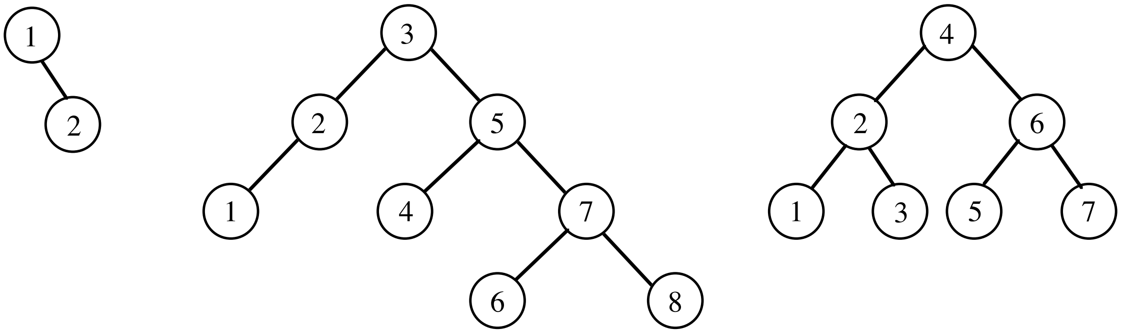
Observations:

1. A red-black tree with *n* internal nodes ("keys") has height at most 2 lg(*n*+1).

2. If a node X is not a leaf and its sibling is a leaf, then X must be red.

3. There may be many ways to color a binary search tree to make it a red-black tree.

4. If the root is colored red, then it may be switched to black without violating structural properties. (Implementations usually color root as black.)



12.B. INSERTION

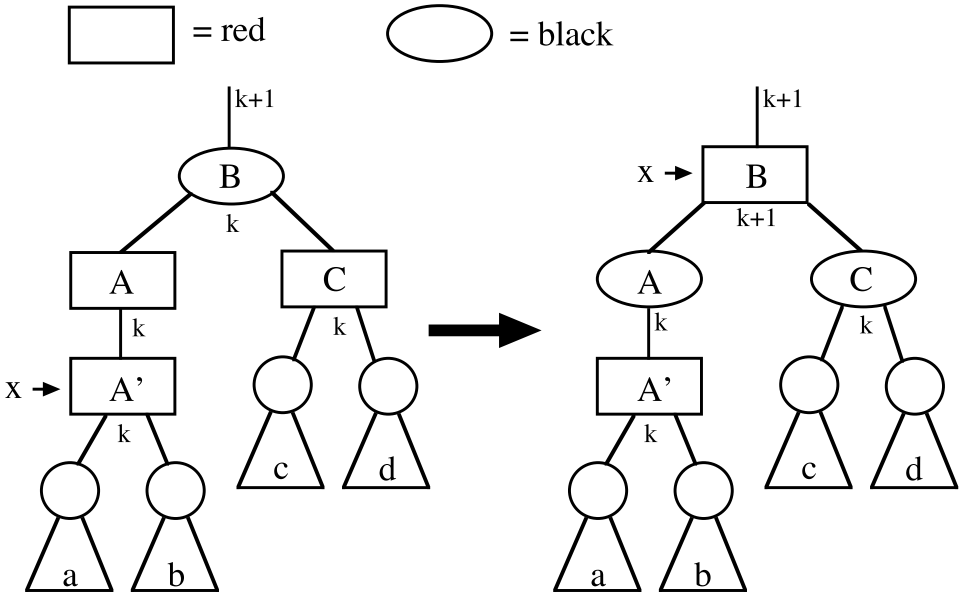
1. Start with unbalanced insert of a "data leaf" (both children are the sentinel).

2. Color of new node is _________.

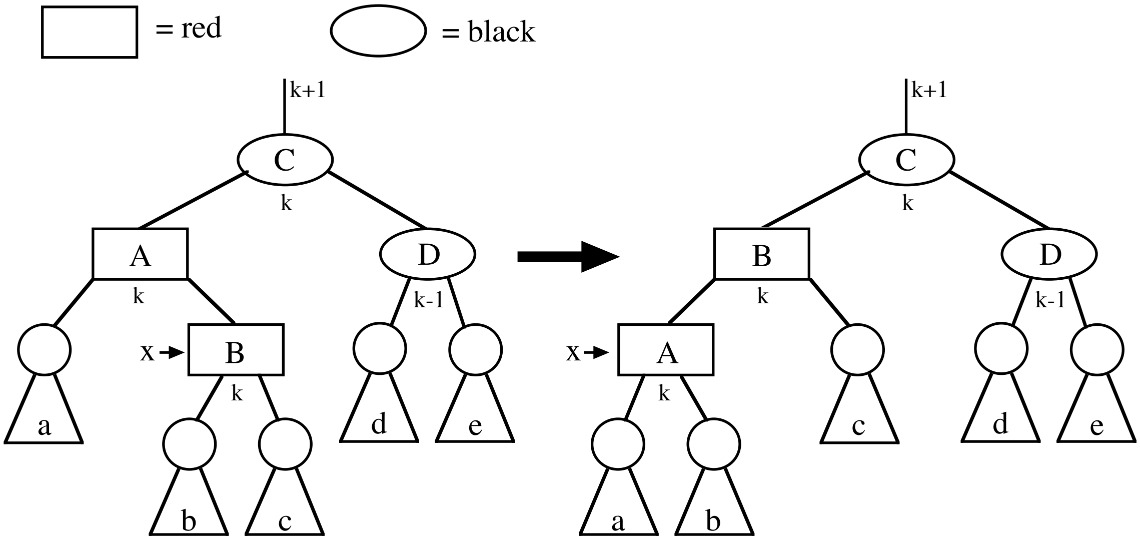
3. May violate structural property 3. Leads to three cases, along with symmetric versions.

   The x pointer points at a red node whose parent might also be red.

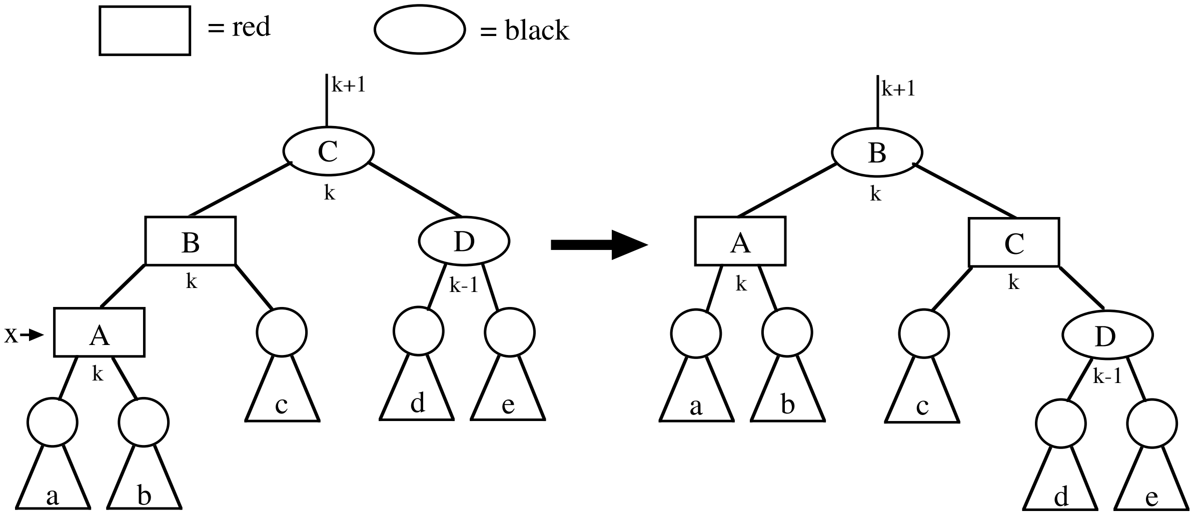
Case 1:



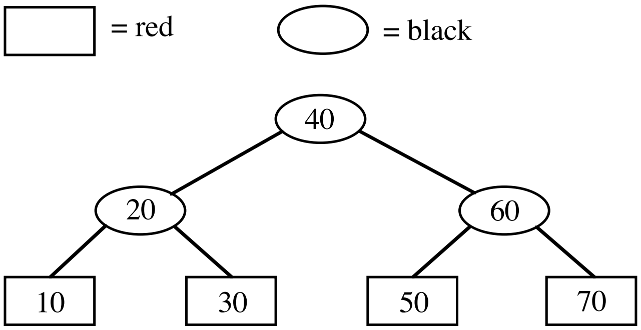
Case 2:



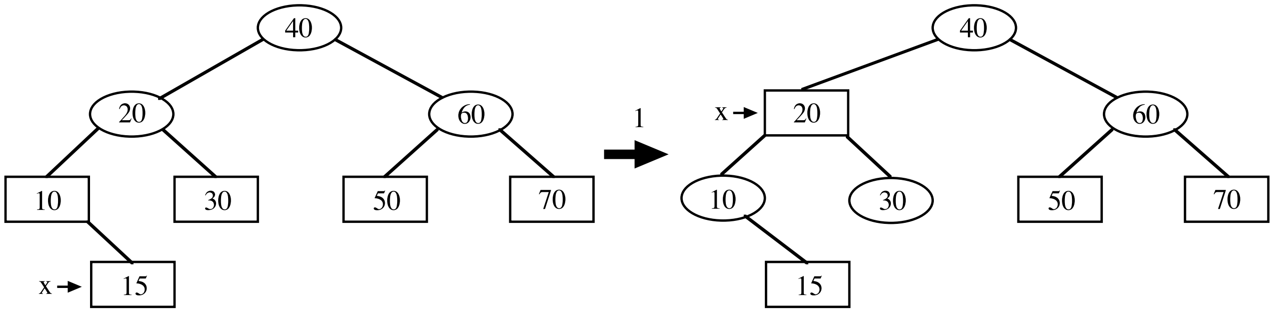
Case 3:



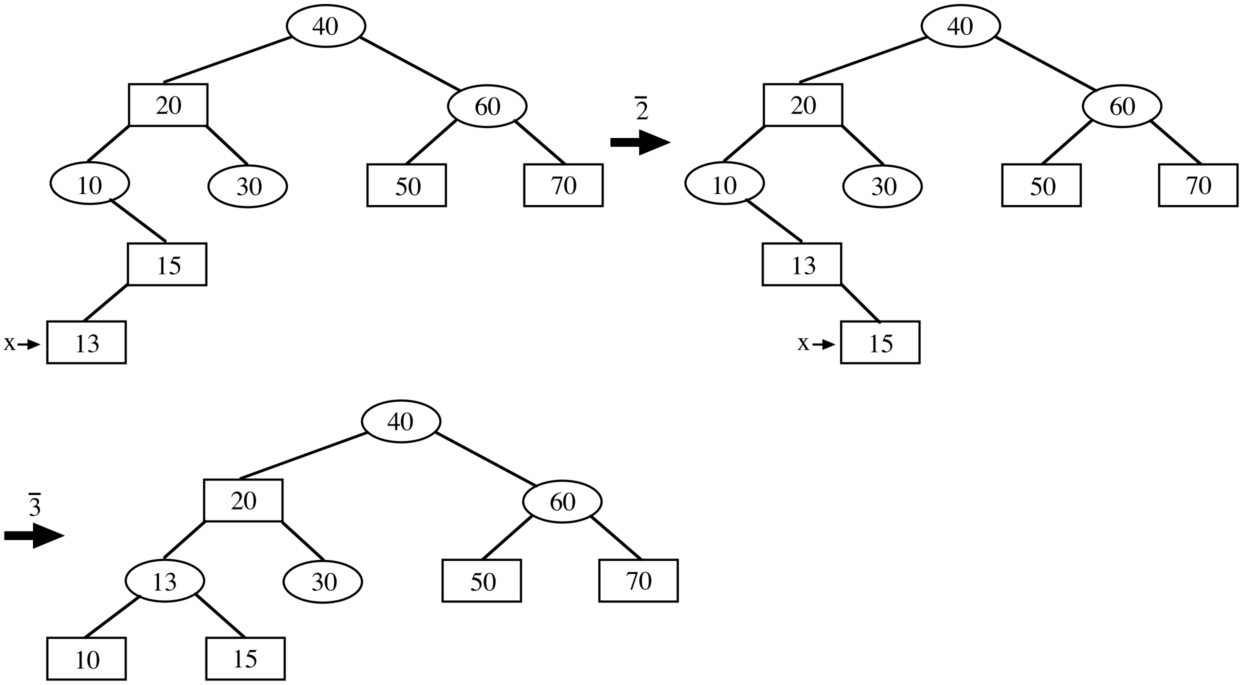
Example 1 (`http://ranger.uta.edu/~weems/NOTES2320/REDBLACKC/notes12.ex1.dat`):



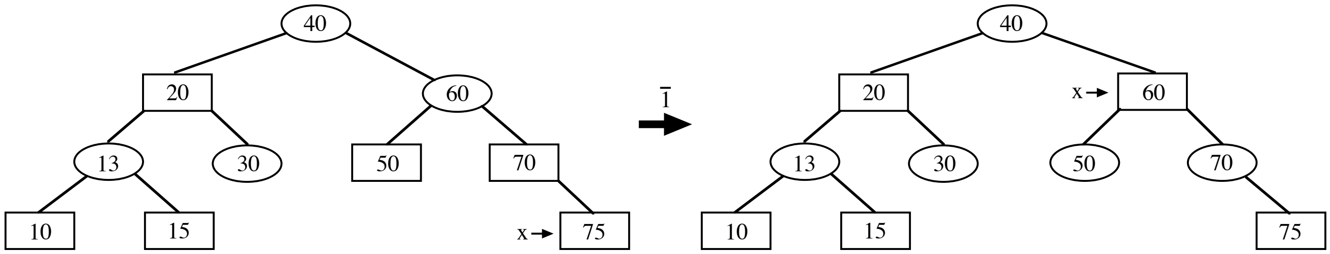
Insert 15



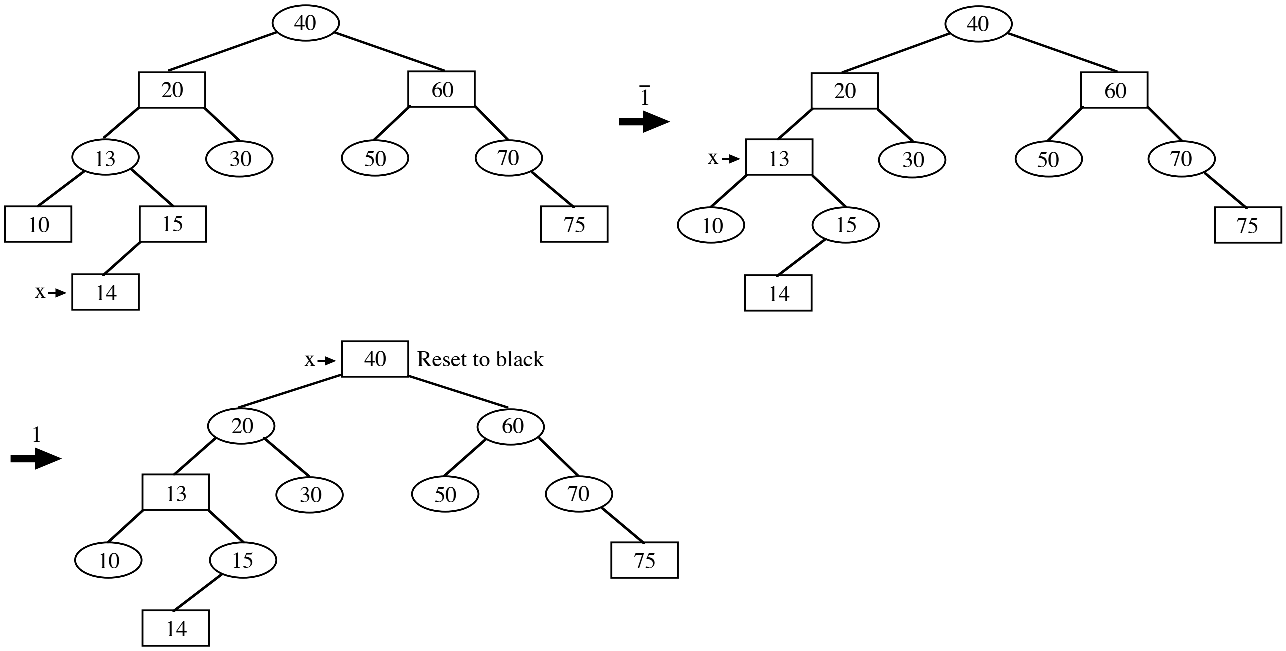
Insert 13



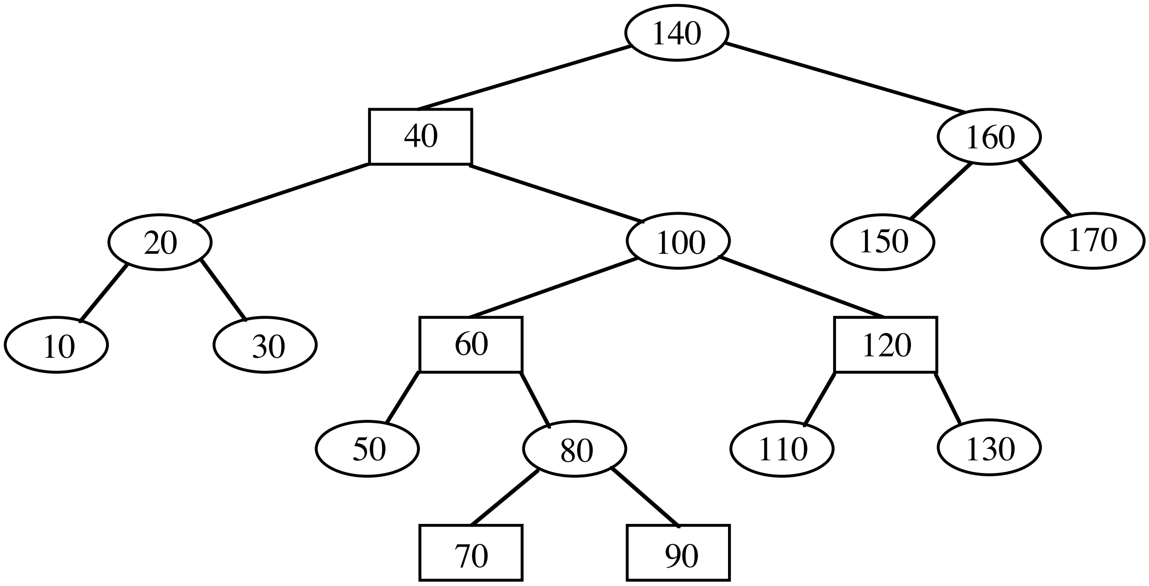
Insert 75



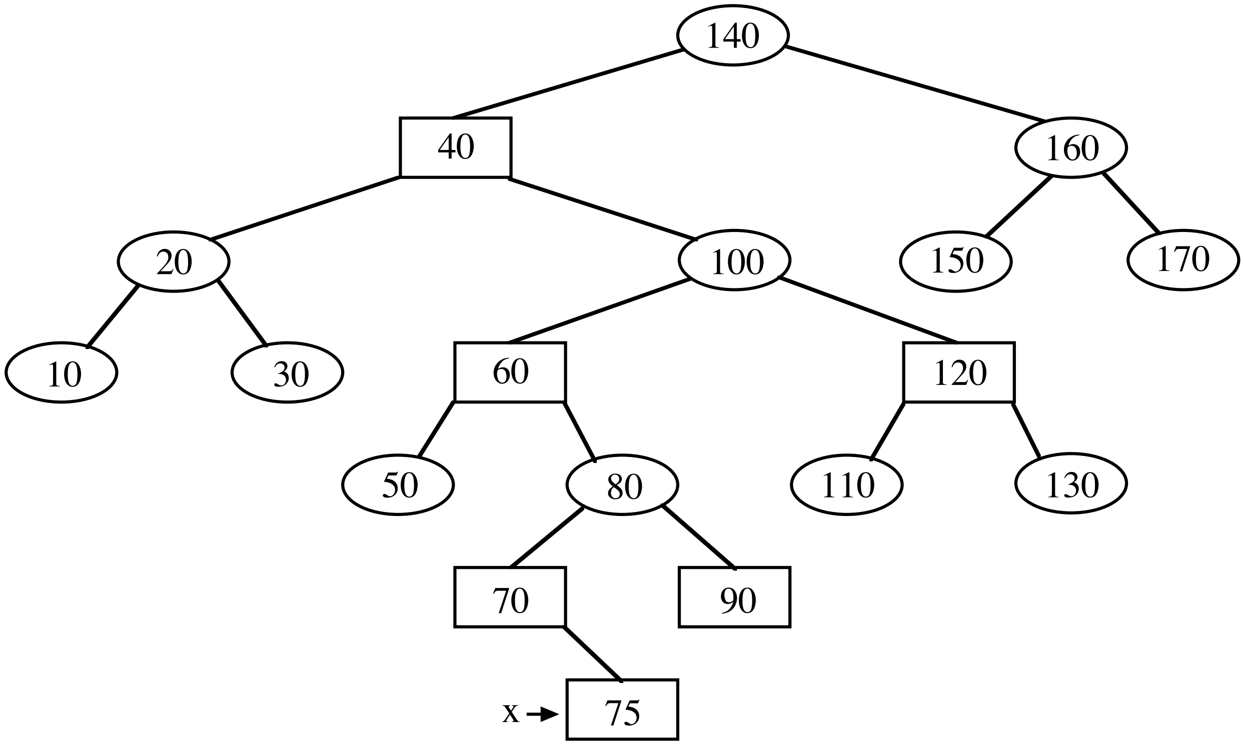
Insert 14

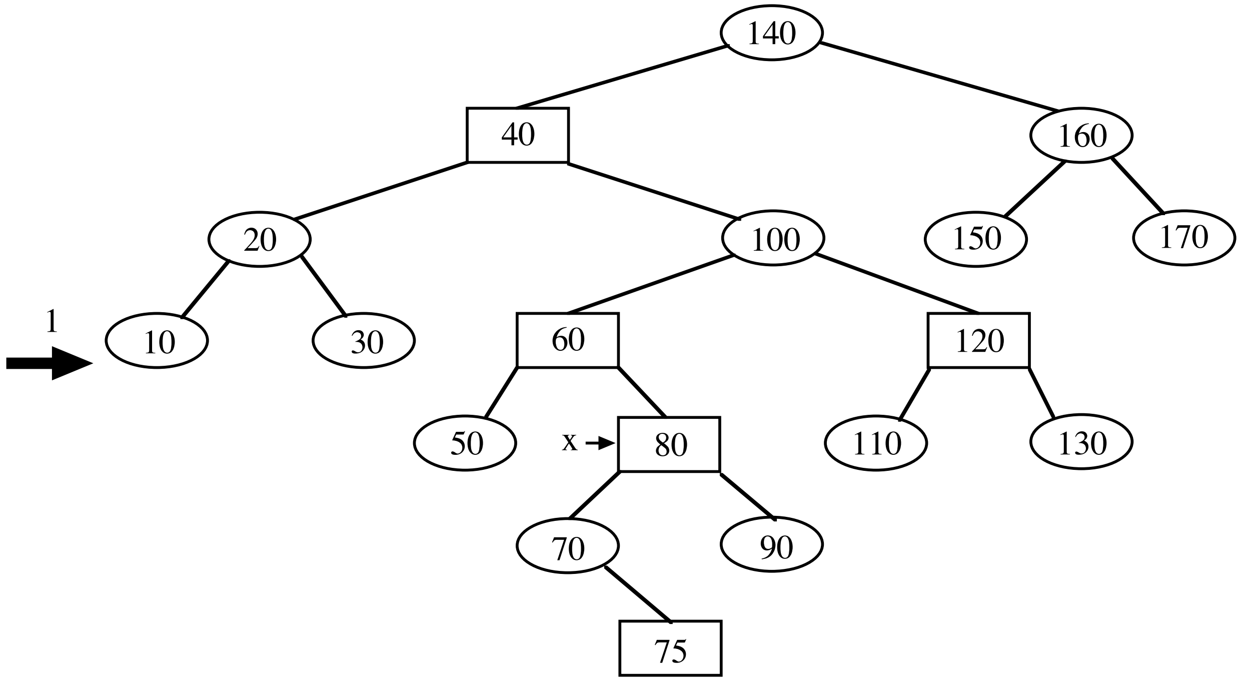


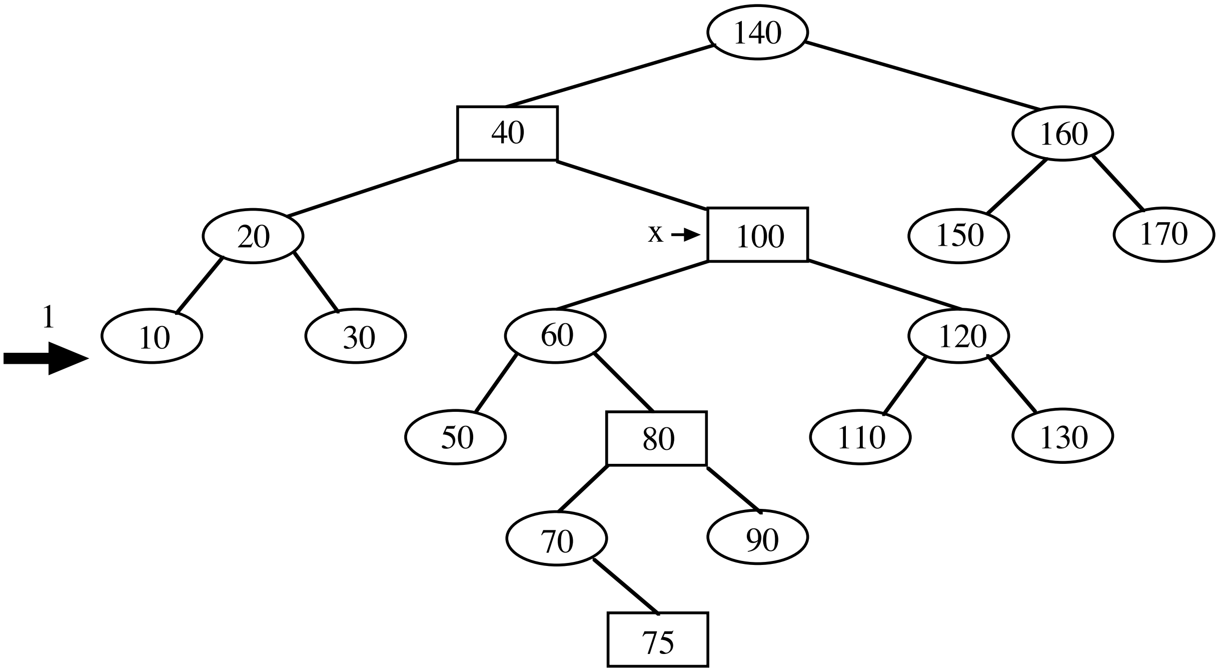
Example 2 (`http://ranger.uta.edu/~weems/NOTES2320/REDBLACKC/notes12.ex2.dat`):

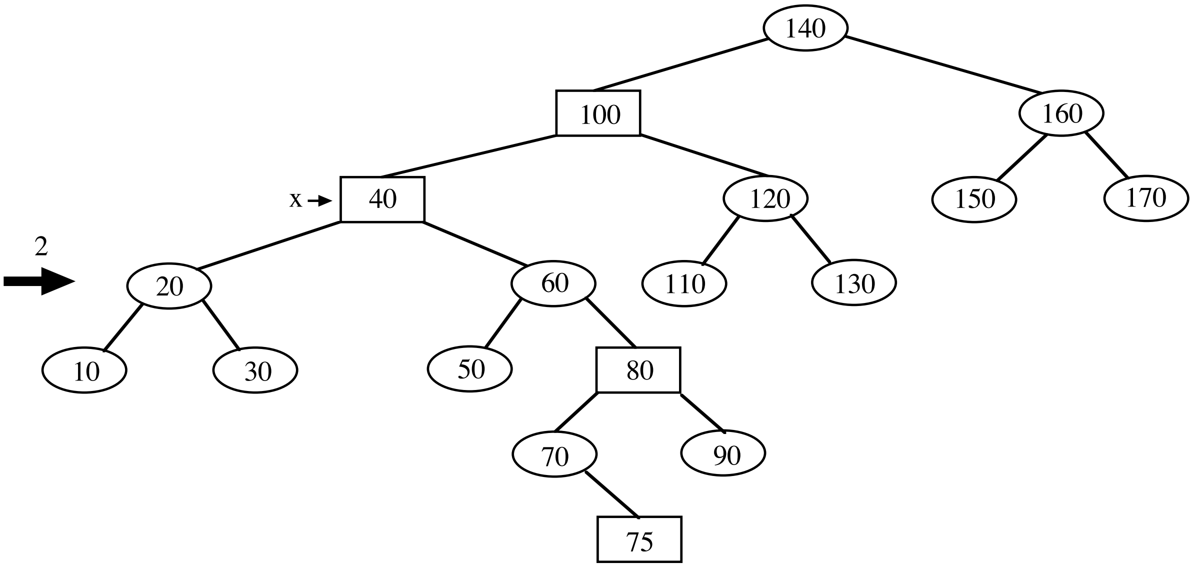


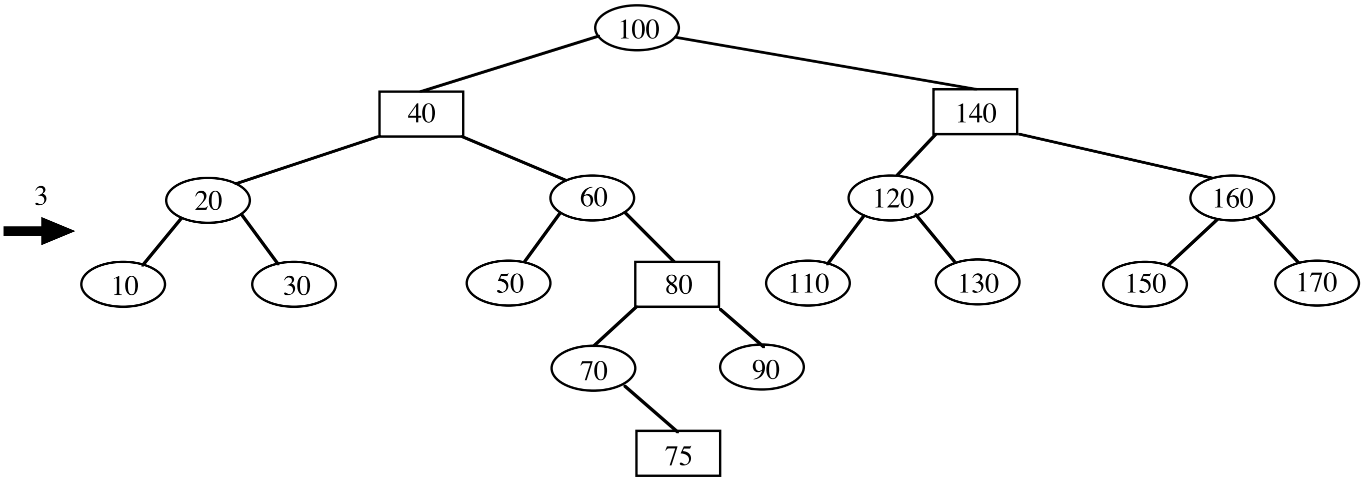
Insert 75


140
40
160
20
100
150
170
10
30
60
120
50
80
110
130
70
90
x
75


140
40
160
20
100
150
170
1
10
30
60
120
50
x
80
110
130
70
90
75


140
40
160
x
100
20
150
170
1
10
30
60
120
50
80
110
130
70
90
75


140
100
160
x
40
120
150
170
2
20
60
110
130
10
30
50
80
70
90
75


100
40
140
3
20
60
120
160
10
30
50
80
110
130
150
170
70
90
75

```
link RBinsert(link h, Item item, int sw, int siblingRed, link hParent)
// CLRS, 3rd ed., RB tree insertion done recursively without parent pointers.
// Also includes tracing.  See 2320 notes.  BPW
// h is present node in search down tree.
// Returns root of modified subtree.
// item is the Item to be inserted.
// sw == 1 <=> h is to the right of its parent.
// siblingRed has color of h's sibling.
// hParent has h's parent to facilitate case 1 color flips.
{
Key v = key(item);
link before;  // Used to trigger printing of an intermediate tree

tracePrint("Down",h);
if (h == z)
  return NEW(item, z, z, 1);  // Attach red leaf

if (eq(v, h->item))
  return h;
else if (less(v, h->item)) {
  tracePrint("Insert left",h);
  before=h->l;
  h->l = RBinsert(h->l, item, 0, h->r->red, h); // Insert in left subtree
  if (trace==2 && before!=h->l)  // Has a rotation occurred?
    STprintTree();
  if (h->l->red) {
    if (h->red)
      if (sw)
        if (siblingRed) {
          tracePrint("Case ~1l",hParent);
          hParent->red = 1;
          hParent->l->red = 0;
          hParent->r->red = 0;
          if (trace==2)
            STprintTree();
          }
        else {
          tracePrint("Case ~2",h);
          h = rotR(h);  // Set up case ~3 after return
          }
      else if (siblingRed) {
        tracePrint("Case 1l",hParent);
        hParent->red = 1;
        hParent->l->red = 0;
        hParent->r->red = 0;
        if (trace==2)
          STprintTree();
        }
      else
        ;  // Future case 3
    else if (!h->r->red && h->l->l->red) {
      tracePrint("Case 3",h);
      h = rotR(h);
      h->red = 0;
      h->r->red = 1;
      }
    }
  }
else {
  tracePrint("Insert right",h);
  before=h->r;
  h->r = RBinsert(h->r, item, 1, h->l->red, h); // Insert in right subtree
  if (trace==2 && before!=h->r)  // Has a rotation occurred?
    STprintTree();
  if (h->r->red) {
    if (h->red)
      if (!sw)
        if (siblingRed) {
          tracePrint("Case 1r",hParent);
          hParent->red = 1;
          hParent->l->red = 0;
          hParent->r->red = 0;
          if (trace==2)
            STprintTree();
          }
        else {
          tracePrint("Case 2",h);
          h = rotL(h);  // Set up case 3 after return
          }
      else if (siblingRed) {
        tracePrint("Case ~1r",hParent);
        hParent->red = 1;
        hParent->l->red = 0;
        hParent->r->red = 0;
        if (trace==2)
          STprintTree();
        }
      else
        ;  // Future case ~3
    else if (!h->l->red && h->r->r->red) {
      tracePrint("Case ~3",h);
      h = rotL(h);
      h->red = 0;
      h->l->red = 1;
      }
    }
  }

fixN(h);
tracePrint("Up",h);
return h;
}
```