CSE 3318 Notes 11: Rooted Trees

(Last updated 8/2/22 9:34 AM)

CLRS 10.3, 12.1-12.3, 17.1, 13.2

11.A. TREES

Representing Trees (main memory, disk devices in CSE 3330)

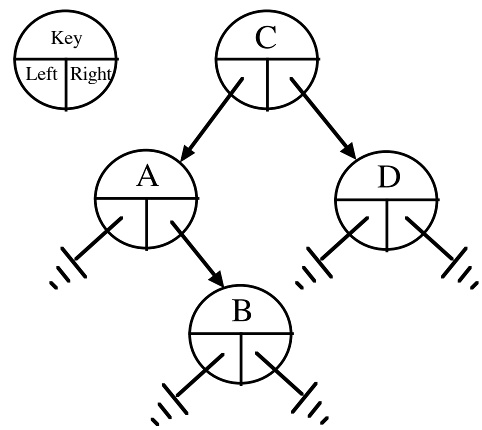
Binary tree

Mandatory

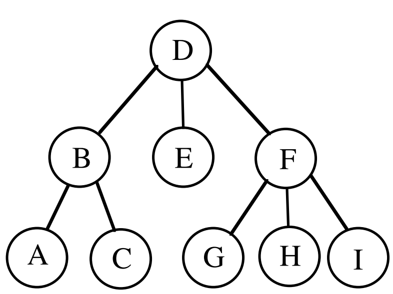
Left
Right

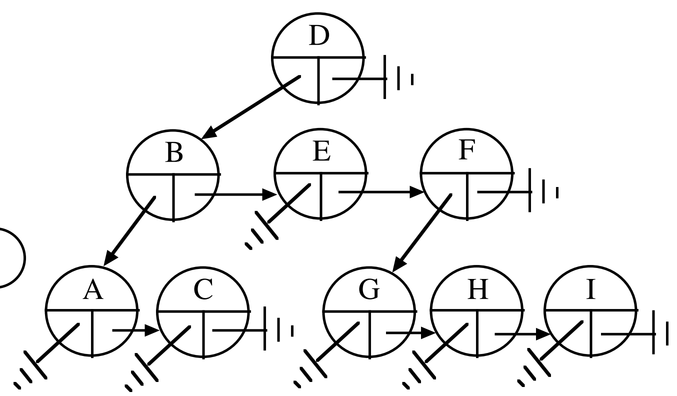
Optional

Parent
Key
Data
Subtree Size



Rooted tree with linked siblings





Mandatory

First Child
Right Sibling

Optional

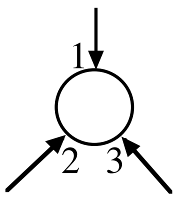
Last Child
Left Sibling
Parent
Key
Data
Subtree Size

11.B. Binary Tree Traversals (review)

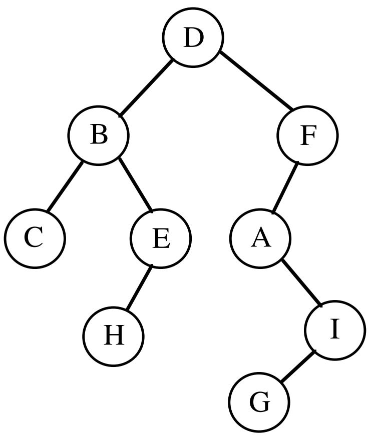
1st Visit – Preorder

2nd Visit – Inorder

3rd Visit – Postorder




```
recTraversal(Node h)
{
  if (h!=null)
  {

    recTraversal(h.l);

    recTraversal(h.r);

  }
}
```



Preorder

D B C E H F A I G

Inorder

C B H E D A G I F

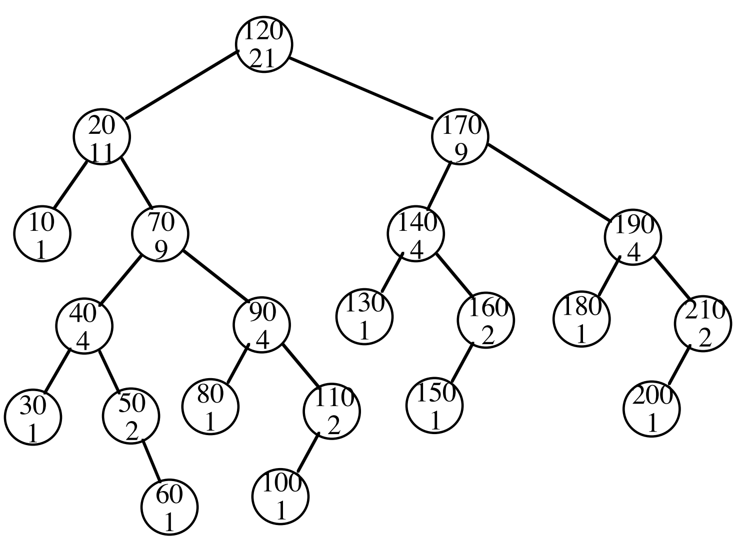
Postorder

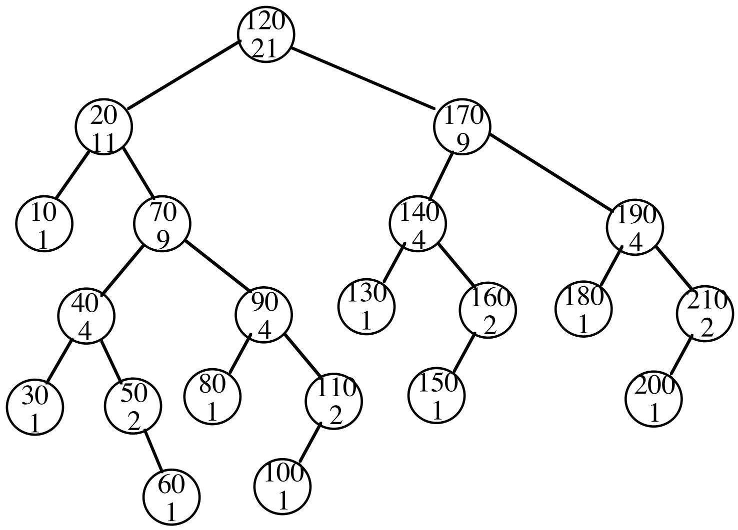
C H E B G I A F D

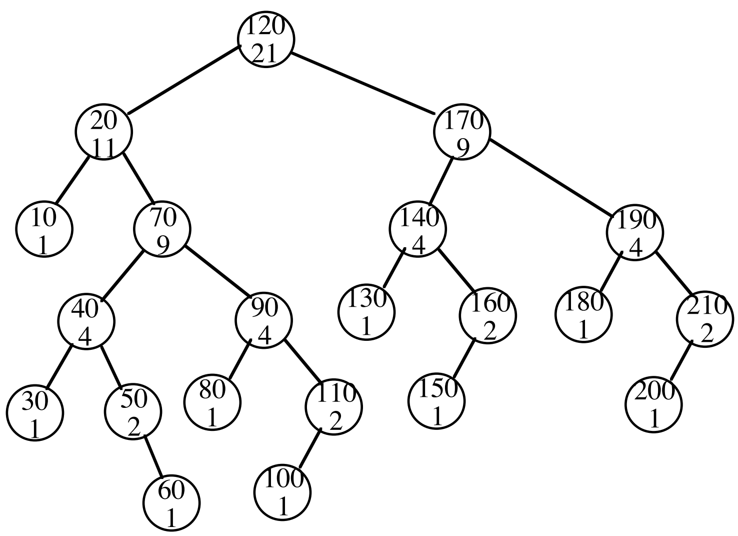
11.C. BINARY SEARCH TREES

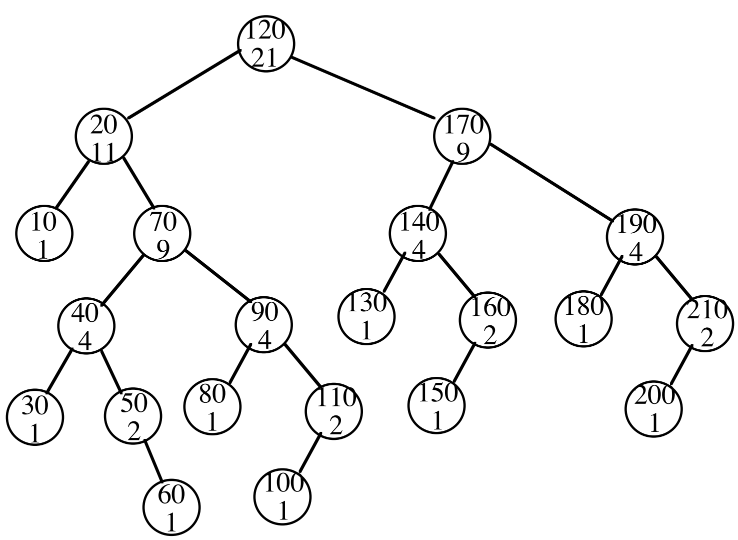
Basic property – Go left for smaller keys. Go right for larger keys. (Use of sentinel)

Which traversal lists the keys in ascending order?




120
21
20
11
170
9
10
1
70
9
140
4
190
4
40
4
90
4
130
1
160
2
180
1
210
2
30
1
50
2
80
1
110
2
150
1
200
1
60
1
100
1


120
21
20
11
170
9
10
1
70
9
140
4
190
4
40
4
90
4
130
1
160
2
180
1
210
2
30
1
50
2
80
1
110
2
150
1
200
1
60
1
100
1


120
21
20
11
170
9
10
1
70
9
140
4
190
4
40
4
90
4
130
1
160
2
180
1
210
2
30
1
50
2
80
1
110
2
150
1
200
1
60
1
100
1

Operations: (see `https://ranger.uta.edu/~weems/NOTES3318/REDBLACKC/RB.c`)

1. Search (`searchR`)

2. Minimum / maximum in tree

3. Successor/predecessor of a node

4. Insert (unbalanced in `https://ranger.uta.edu/~weems/NOTES3318/REDBLACKC/RB.loadAndGo.c`)

5. Deletion of key and associated data is contained in:

   a. Leaf

   b. Node with one child

   c. Node with two children

      1. Find node's successor (convention)

      2. Move key and data (but not pointer values) from successor node to node of deletion.

      3. Successor has either

         a. Zero children – leaf is removed (5.a)

         b. One child (right) – point around successor node to remove (5.b)

   May also use *tombstones* and periodically recycle dead nodes.

Implementing operations 6. and 7. efficiently requires maintaining subtree sizes "incrementally".

Rank of a key X that appears in tree = number of nodes with keys ≤ X.

Number of nodes on search path to X with keys ≤ key in given node

+

Sizes of their left subtrees

6. Rank of a key (`invSelectR`).

7. Finds key with a given rank (`selectR`) - This is the same as flattening tree into an ordered array and then subscripting (or using inorder traversal).

Time for operations?

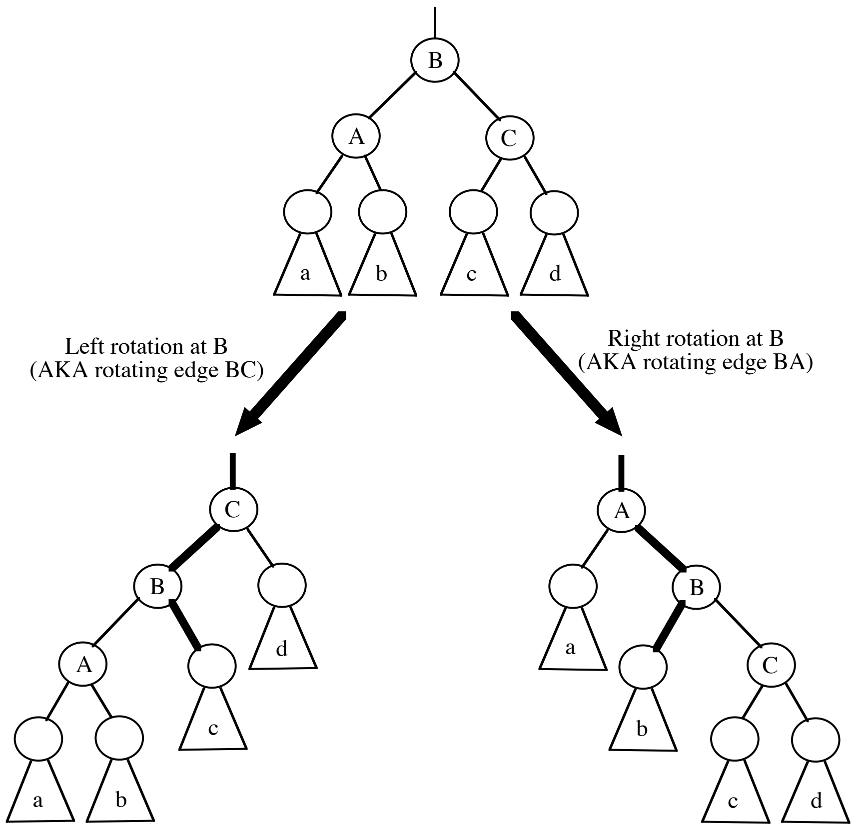
From `https://ranger.uta.edu/~weems/NOTES3318/REDBLACKC/RB.c`

(`z` points to the sentinel)

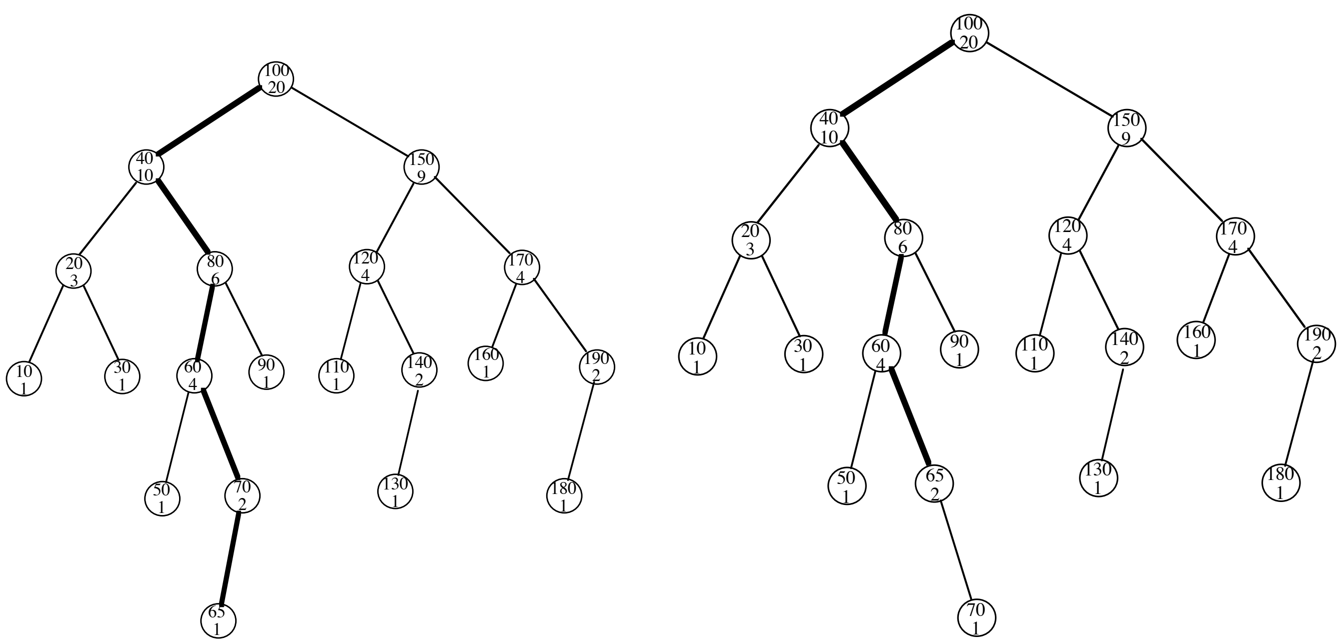
```
int invSelectR(link h, Key v)
// Inverse of selectR
{
Key t = key(h->item);
int work;

if (h==z)
  return -1;  // v doesn't appear as a key
if (eq(v, t))
  return h->l->N+1;
if (less(v, t))
  return invSelectR(h->l,v);
work=invSelectR(h->r,v);
if (work==(-1))
  return -1;  // v doesn't appear as a key
return 1 + h->l->N + work;
}

int STinvSelect(Key v)
{
return invSelectR(head,v);
}


Item selectR(link h, int i)
// Returns the ith smallest key where i=1 returns the smallest
// key.  Thus, this is like flattening the tree inorder into an array
// and applying i as a subscript.
{
int r = h->l->N+1;

if (h == z)
{
  printf("Impossible situation in selectR\n");
  STprintTree();
  exit(0);
}
if (i==r)
  return h->item;
if (i<r)
  return selectR(h->l, i);
return selectR(h->r, i-r);
}

Item STselect(int k)
{
if (k<1 || k>head->N)
{
  printf("Range error in STselect() k %d N %d\n",k,head->N);
  exit(0);
}
return selectR(head, k);
}
```

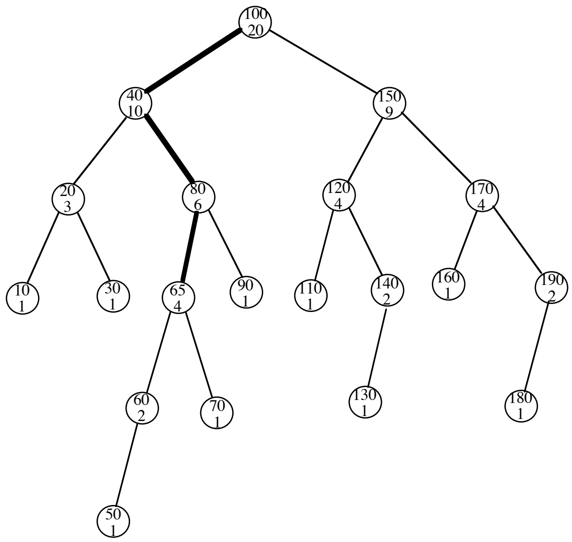
11.D. ROTATIONS

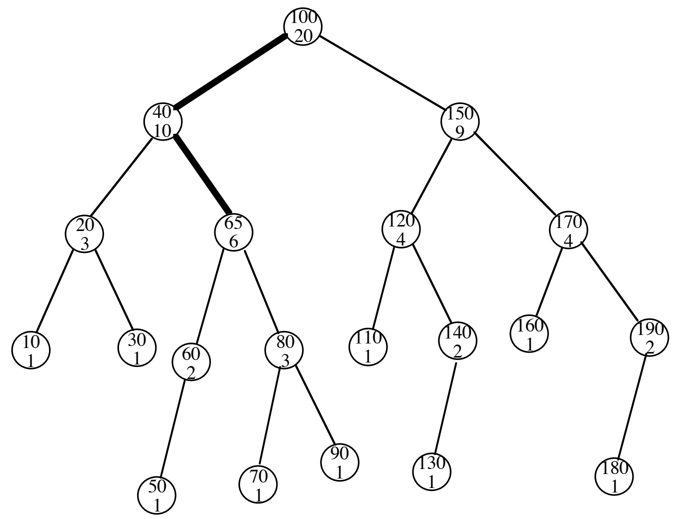
Technique for rebalancing in balanced binary search tree schemes. Takes Θ(1) time.

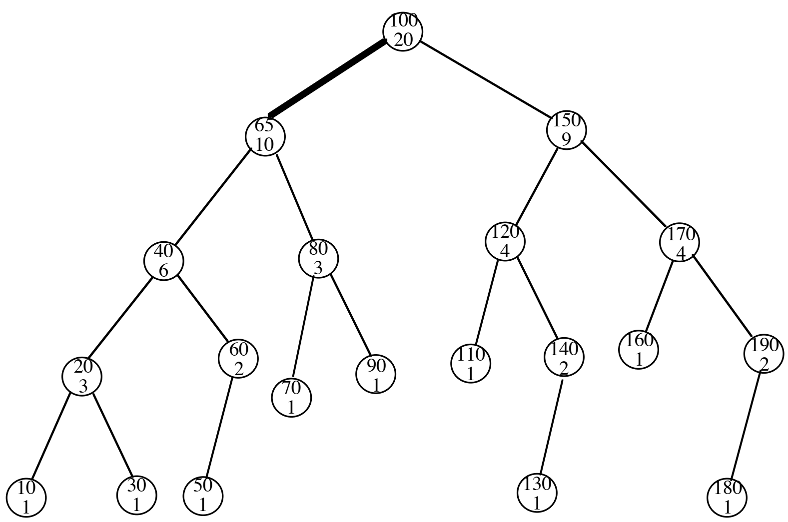


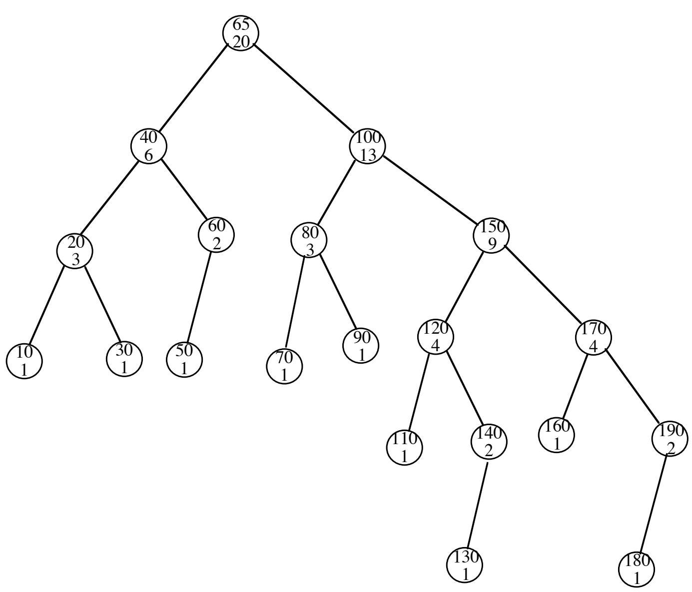
11.E. INSERTION AT ROOT: rotates all edges on the insertion path:











https://ranger.uta.edu/~weems/NOTES3318/bst.step.html