Solution for Problem Set 5

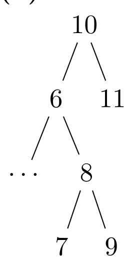
Mianzhi Pan, 181240045

October 18, 2020

1 Problem 1

(a) The third and the last could not be the sequence because they violate the *BST property*.(In the third sequence $912 > 911$ and in the last one $299 < 347$)

(b) This claim is false apparently, we just need to show a counter-example:



Suppose key k is 9, then $B = \{10, 6, 8, 9\}$, $A = \{7\}$, $C = \{11\}$. However, $7 > 6$.

2 Problem 2

$SEARCH$ operation doesn't change

Algorithm 1 SEARCH(T,k)

$x = T.root$
while $x! = NIL$ and $x.key! = k$ **do**
    if $x.key > k$ **then**
        $x = x.left$
    else
        $x = x.right$
    end if
end while
return x

We need to record the successor of the new inserted node compared with the origin $TREE - INSERT$. Besides the inserted node, the only node needing update *succ* is its predecessor.

Algorithm 2 INSERT(T,z)

```
x = T.root
y = NIL, s = NIL, pred = NIL
while x != NIL do
    y = x
    if x.key > z.key then
        s = x
        x = x.left
    else
        pred = x
        x = x.right
    end if
end while
z.succ = s
if y == NIL then
    T.root = z
else if z.key < y.key then
    y.left = z
    pred.succ = z
else
    y.right = z
    y.succ = z
end if
```

To implement $DELETE$, we should implement $PARENT(T, z)$ to find the parent of a given node z:

Algorithm 3 PARENT(T,z)

```
x = T.root
y = NIL
while x != NIL and x.key != z.key do
    y = x
    if x.key > z.key then
        x = x.left
    else
        x = x.right
    end if
end while
return y
```

Then we modify $TRANSPLANT$

Algorithm 4 TRANSPLANT(T, u, v)

$p = PARENT(T, u)$
if $p == NIL$ **then**
    $T.root = v$
else if $u == p.left$ **then**
    $p.left = v$
else
    $p.right = v$
end if

Notice the only change $DELETE$ do to the in-order sequence is removing the target number, node which need update *succ* is just the predecessor of the target node. So we first implement $TREE - PREDECESSOR$.

Algorithm 5 TREE-PREDECESSOR(x)

if $x.left! = NIL$ **then**
    return $TREE - MAXIMUM(x.left)$
else
    $y = PARENT(x)$
    while $y! = NIL$ and $x == y.left$ **do**
        $x = y$
        $y = PARENT(y)$
    end while
end if
return y

Algorithm 6 DELETE(T,z)

$pred = TREE - PREDECESSOR(z)$
$pred.succ = z.succ$
if $z.left == NIL$ **then**
    $TRANSPLANT(T, z, z.right)$
else if $z.right == NIL$ **then**
    $TRANSPLANT(T, z, z.left)$
else
    $y = TREE - MINIMUM(z.right)$
    if $PARENT(T, y)! = z$ **then**
        $TRANSPLANT(T, y, y.right)$
        $y.right = z.right$
    end if
    $TRANSPLANT(T, z, y)$
    $y.left = z.left$
end if

3 Problem 3

(a) $O(n^2)$

(b) Notice all n nodes with same value will fill the binary search tree level by level. Total runtime is $T = 0 + 1 \times 2^1 + 2 \times 2^2 + \cdots + lgn \times 2^{lgn}$. We can easily find time complexity is $O(nlgn)$.

(c) For n nodes with identical keys, we just need to insert the node into a list every time except the first time, so the time complexity is $O(n)$.

4 Problem 4

(a) The largest ratio is 2.(Every black node has two red nodes).

The smallest is 0.(All nodes are black and the tree is perfect).

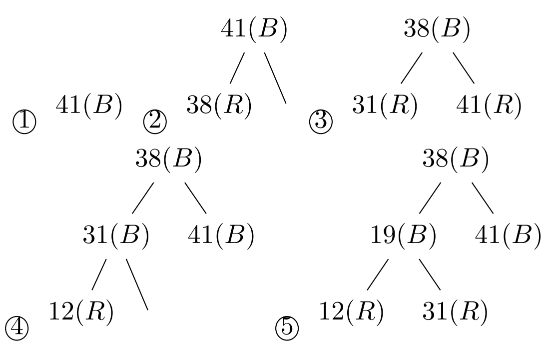
(b) First we prove that at most $n-1$ right rotations are needed to transform the tree into a right-going chain.

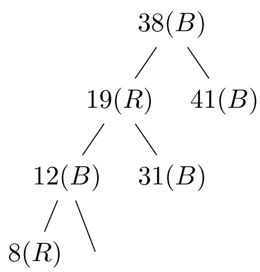
Suppose set R contains all nodes from the tree's root to its right-most children, and L contains the rest nodes. Every time we do right rotation, we will extract a node from L and put it into R. Notice there are at most $n-1$ nodes in L, the statement is proved.

Then we can do left rotation on an arbitary node from R, which consists of all n nodes. Therefore we can construct any other arbitary n-node binary search tree with a particular rotation operation sequence. Notice this process needs at most $n-1$ rotations as well, the total time complexity is $O(n)$.

5 Problem 5

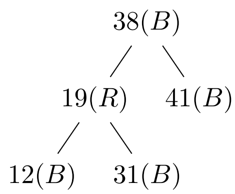
(a)



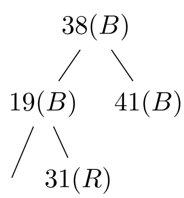


⑥

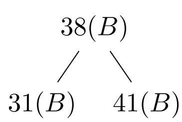
(b)



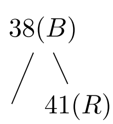
①



②



③



④ 41(B) ⑤ ⑥