Solution for Problem Set 5

Mianzhi Pan, 181240045

October 20, 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:

```
        10
       /  \
      6    11
     / \
  ...   8
       / \
      7   9
```

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).

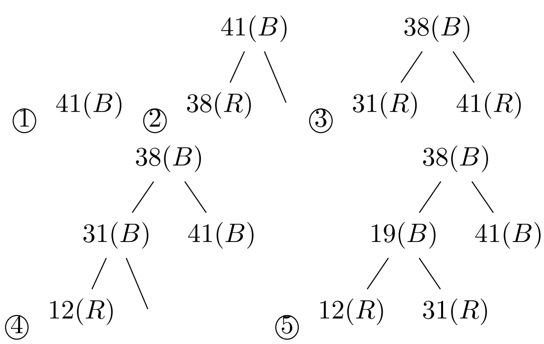
(b) First we prove that at most $n-1$ right rotations are needed to transform the tree into a right-going chain.

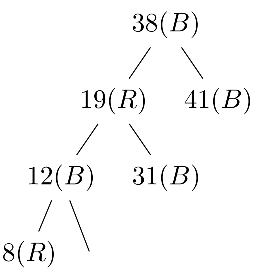
Suppose set R contains all nodes from the tree's root to its right-most children, and L contains the rest nodes. Every time we do right rotation, we will extract a node from L and put it into R. Notice there are at most $n-1$ nodes in L, the statement is proved.

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

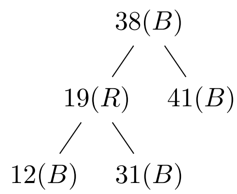
(a)



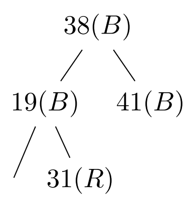


⑥

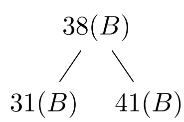
(b)



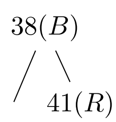
①



②



③



④ ⑤ 41(B) ⑥

6 Problem 6

(a) Assume an AVL tree of height h has $T(h)$ nodes, notice the property of AVL tree, both of the subtrees rooted at *root.left* and *root.right* are also AVL trees, one of which must has height $h-1$ and the other can have height $h-1$ or $h-2$. So we have(notice $T(h-1)$ must be larger than $T(h-2)$)

$$T(h) \geq T(h-1) + T(h-2) + 1$$

Therefore,

$$T(h) > T(h-1) + T(h-2)$$

Obviously, $T(0) = 0$ and $T(1) = 1$, and they are the first two elements of Fibonacci sequence, hence

$$n = T(h) > F_h = \lfloor \frac{\phi^h}{\sqrt{5}} + \frac{1}{2} \rfloor$$

Then

$$n \geq \frac{\phi^h}{\sqrt{5}} + \frac{1}{2}$$

$$h \leq \frac{log(\sqrt{5}(n - \frac{1}{2}))}{log\phi}$$

And we have $h = O(logn)$.

(b) Notice every time we do left-rotation on a node x, the height of its left child $x.left$ will increase and that of right child will decrease. The effect is the opposite when we do right -rotation. So when we do $BALANCE(x)$, if $x.left.h - x.right.h > 1$, we do right-rotation on x. Otherwise we do left-rotation on x. Then we continue this process on $x.left$ and $x.right$ if the subtrees rooted at them do not satisfy the property of AVL tree. At last we check x again(notice $|x.left.h - x.right.h| \leq 1$ may not holds just after one left-rotation or right-rotation on it), if the property does not hold, we repeat the above operation.

(c) First we insert z into the tree just as $INSERT$ operation in BST. Notice only the heights of z's ancestors are changed, we apply $BALANCE$ to them bottom-up.

The heights of z's ancestors will change in the first step, while in the second step, all these nodes are $BALANCE$ and modified to the correct place.

(d) Both the $INSERT$ and the $BALANCE$ process takes $O(h)$ time, so the total time is $O(lgn)$. Notice each of the new inserted node's ancestors will be rotated at most one time, $O(1)$ rotations will be performed.

7 Problem 7

(a) We will extract element from A in order and insert it to the treap. Notice A is sorted, every time we just need to set the inserted node as the right child of the right-most node in the treap. Then we do rotation if it doesn't satisfy heap property.

Consider the worst case, if the priority of the newly inserted node is always maximum(suppose the treap has MinHeap-property), the treap always rotates left and root's right child is always empty, $n - 1$ rotations will be needed and the time complexity is $O(n)$.

(b) Suppose for each key, it stays at the current level with probability p, then the probability it goes to the next level is $1 - p$. Then it reaches height h obeys geometric distribution, i.e. $P(H = h) = (1 - p)^{h-1}p$. Then we obtain

$$P(H \leq h) = 1 - (1 - p)^h$$

by summation. Then

$$P(H > h) = (1-p)^h$$

When $h = O(logn)$, we have $h \leq clogn$. If $p = \frac{1}{2}$,

$$P(H > clogn) = (1 - \frac{1}{2})^{clogn} = \frac{1}{n^c}$$

Notice there are n i.i.d such random variables, use Boole's inequality

$$\begin{aligned}
&P(H_1 > clogn \ or \ H_2 > clogn \ or \cdots or \ H_n > clogn) \\
\leq &P(H_1 > clogn) + P(H_2 > clogn) + \cdots + P(H_n > clogn) \\
= &\frac{1}{n^{c-1}}
\end{aligned}$$

Then

$$\begin{aligned}
&P(H_1 \leq clogn \ and \cdots and \ H_n \leq clogn) \\
= &1 - P(H_1 > clogn \ or \cdots or \ H_n > clogn) \\
\geq &1 - \frac{1}{n^{c-1}} \\
\geq &1 - \frac{1}{n}
\end{aligned}$$

for $c \geq 2$. So the max level is $O(logn)$ with high level.

(c) For a given key, we have argued the the number of its levels is given by geometric distribution, and its expectation value is $\frac{1}{p}$. Notice there are n such i.i.d keys, the total number of nodes is $\frac{n}{p}$. If $p = \frac{1}{2}$, the number is $2n = O(n)$ in expectation.