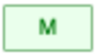
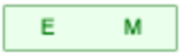
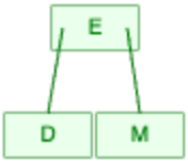
COMP582 Problem Set 8

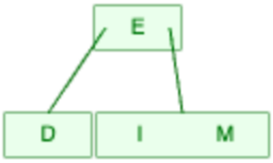
Name: Ting Yen
netID: ty36

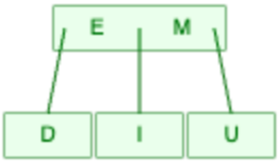
1. Trace of Insertions

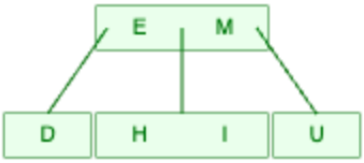


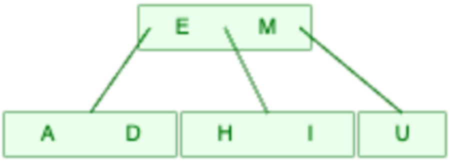


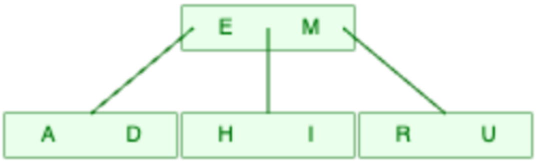


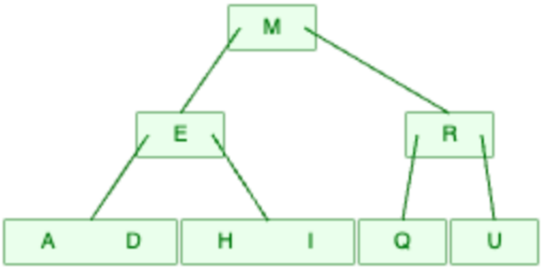


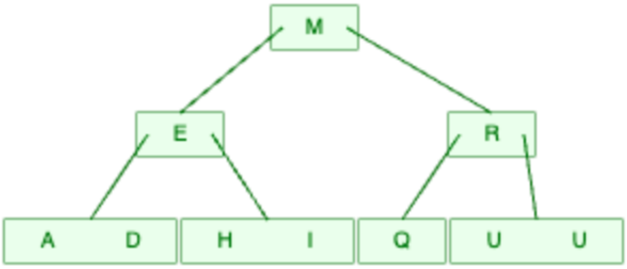


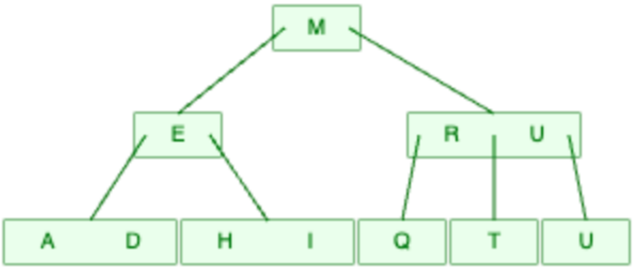


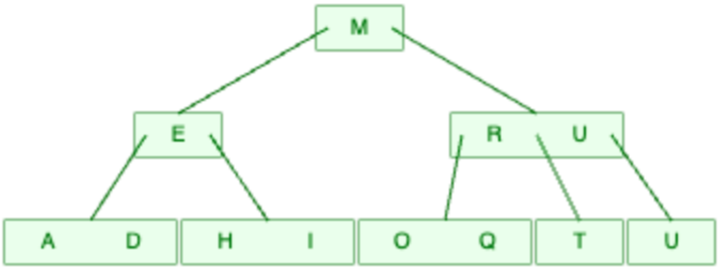

E M
A D
H I
U

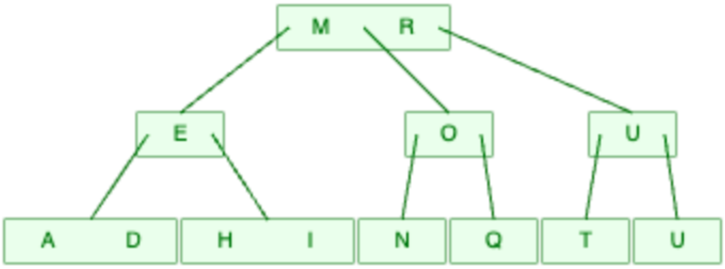

E M
A D
H I
R U


M
E
R
A D
H I
Q
U

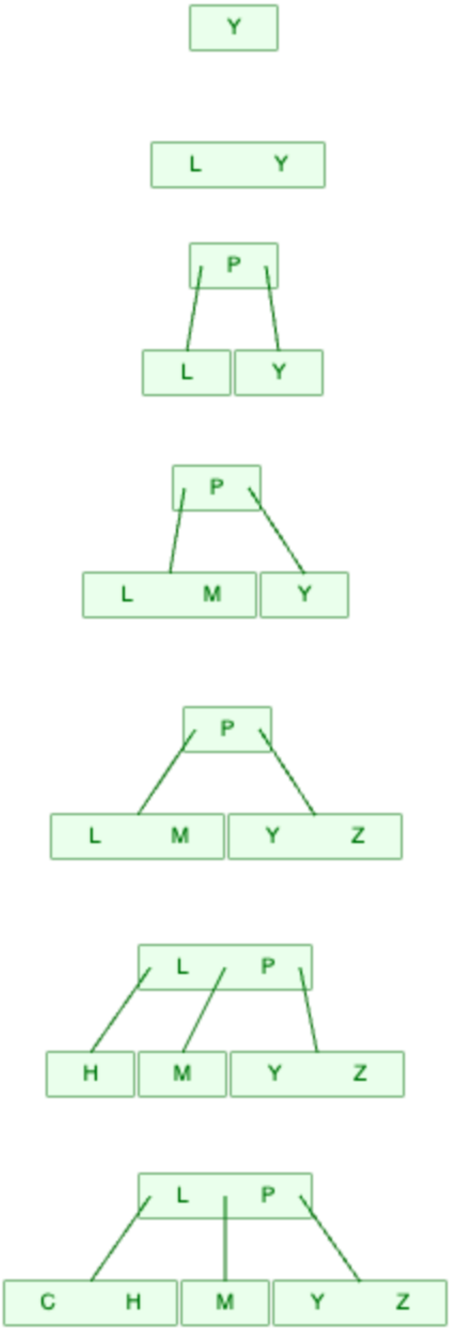

M
E
R
A D
H I
Q
U U

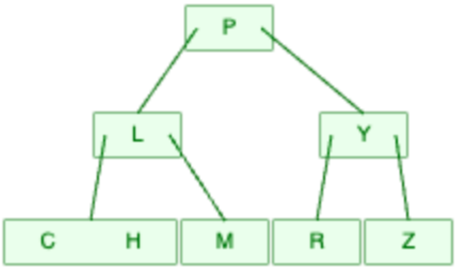

M
E
R U
A D
H I
Q
T
U

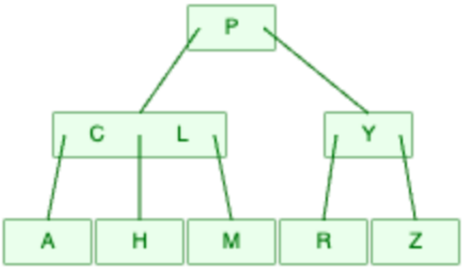

M
E
R U
A D
H I
O Q
T
U

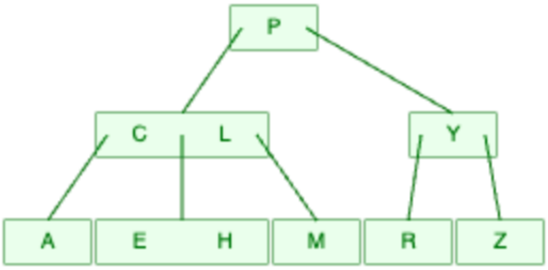

M R
E
O
U
A D
H I
N
Q
T
U

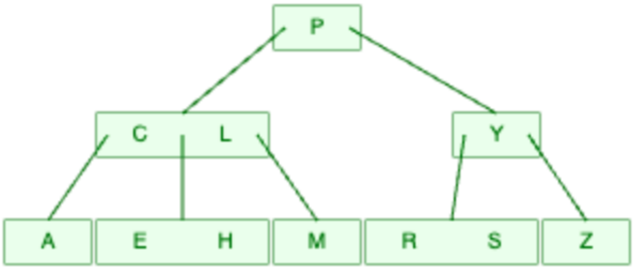
2. Trace of Insertions



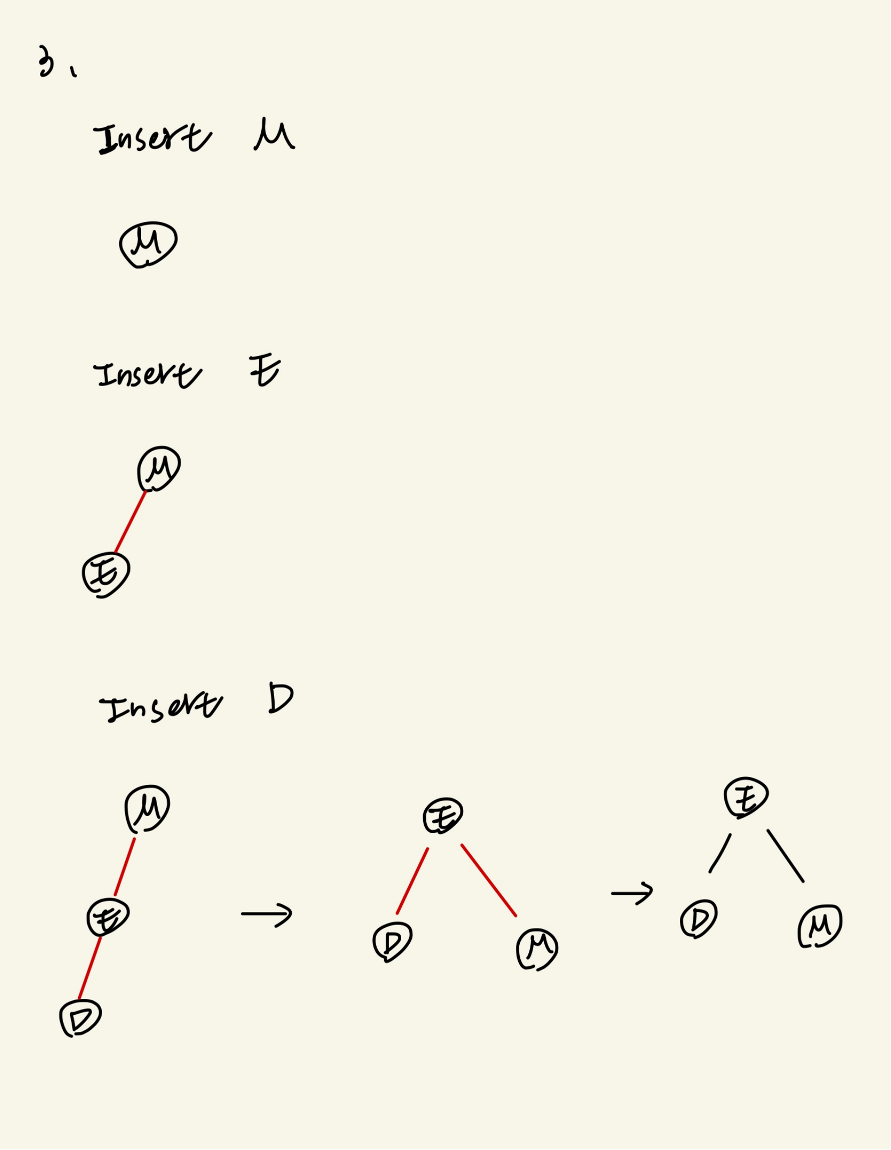

P
L
Y
C H
M
R
Z


P
C L
Y
A
H
M
R
Z


P
C L
Y
A
E H
M
R
Z


P
C L
Y
A
E H
M
R S
Z

3. Trace of Insertions



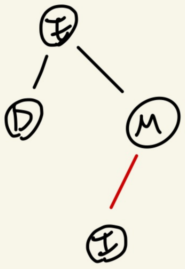
3、

Insert M

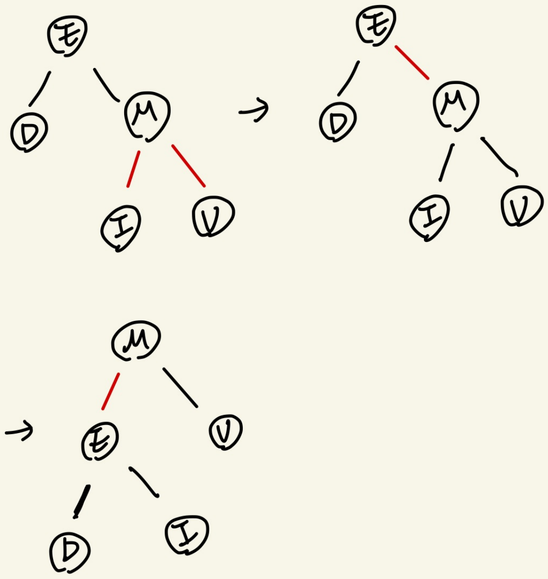
Insert E

Insert D

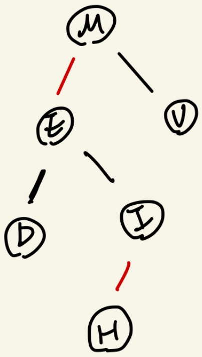
Insert I



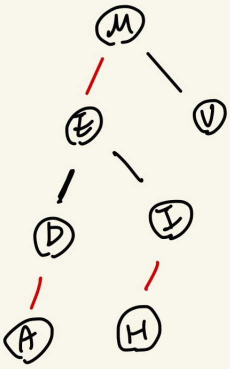
Insert U



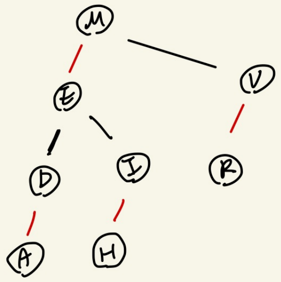
Insert H



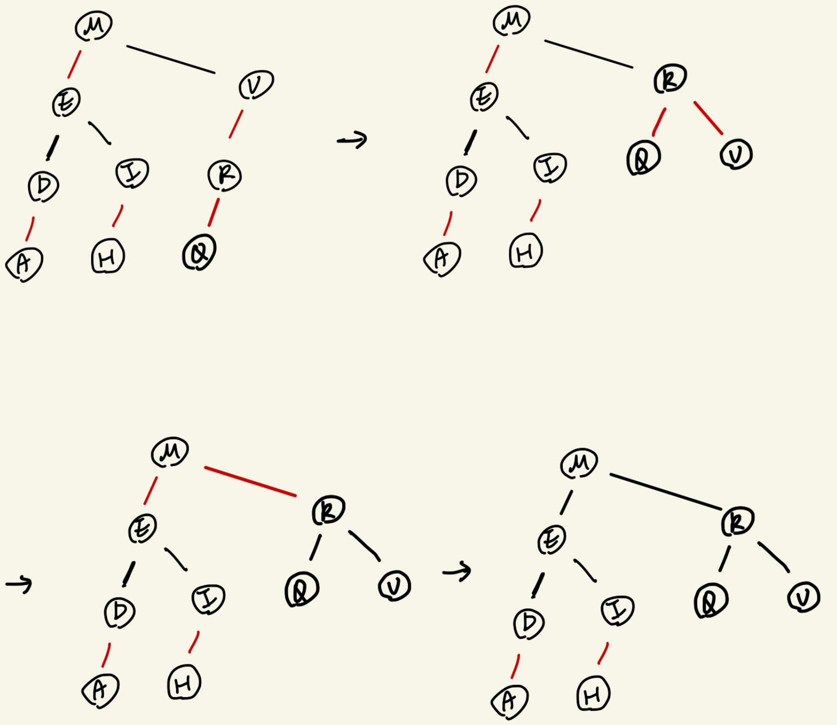
Insert A



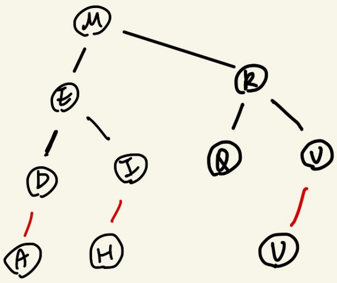
Insert R



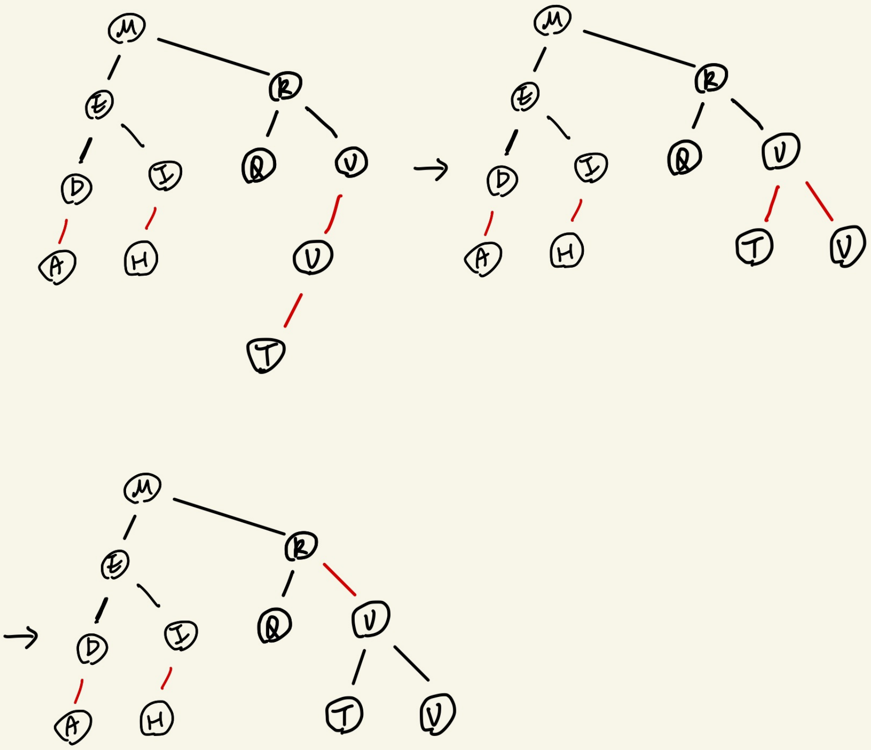
Insert Q

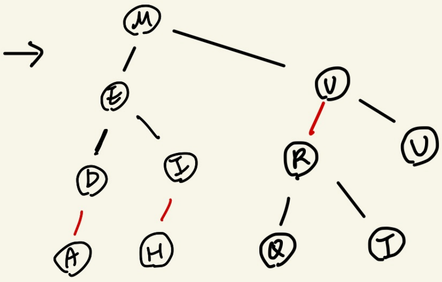


Insert U

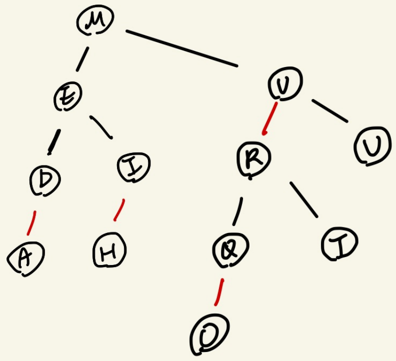


Insert T

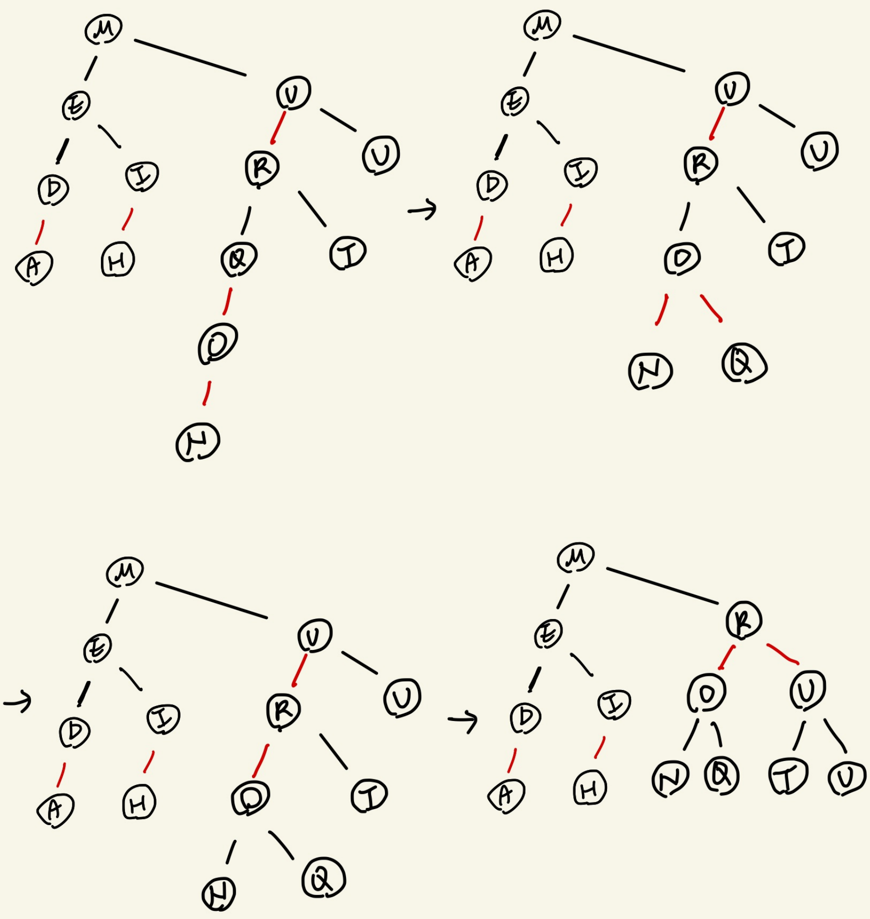


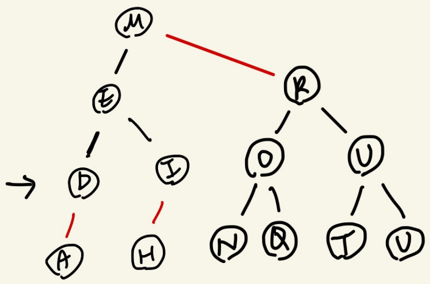

M
E
V
D
I
R
U
A
H
Q
T

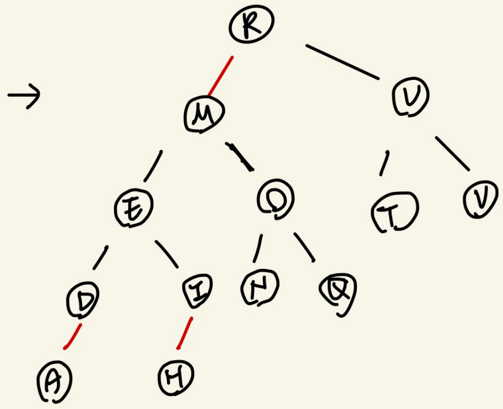
Insert O



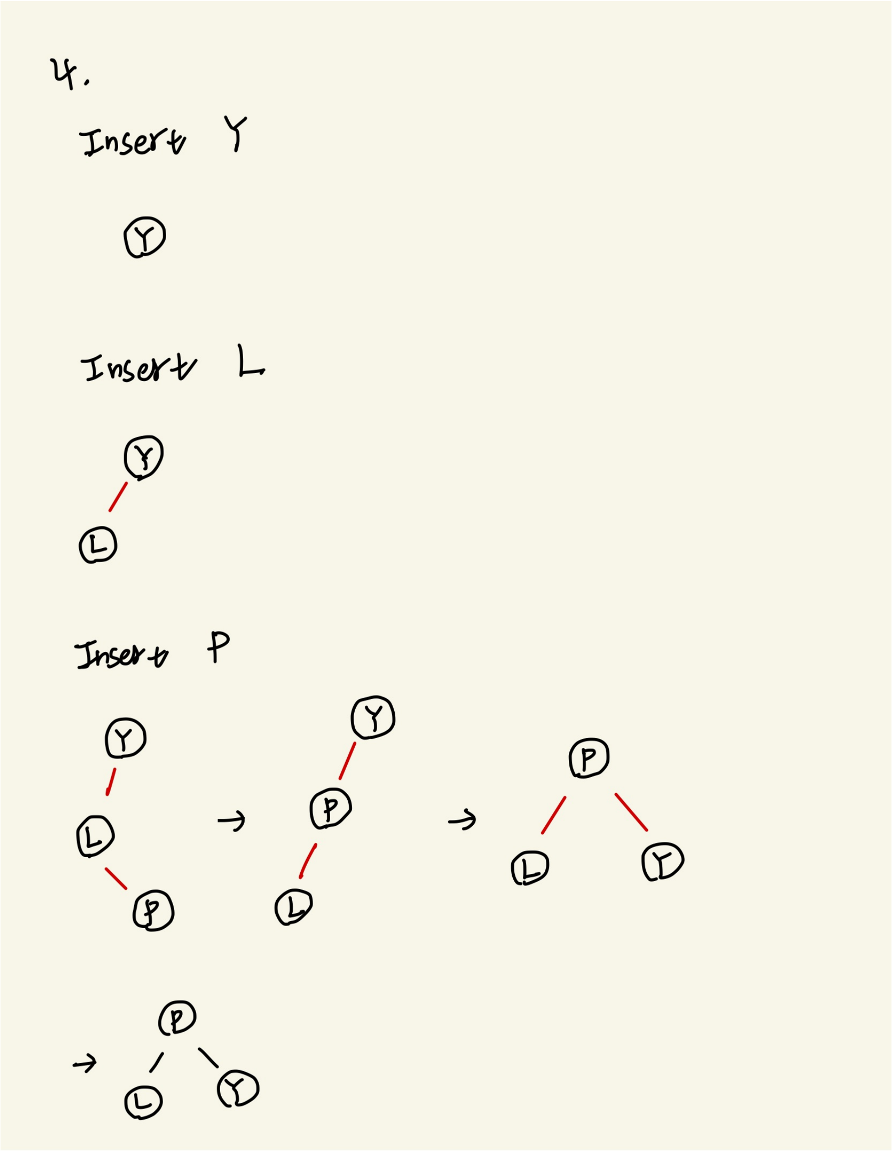
Insert N


M
E
U
D
I
R
V
A
H
Q
T
O
N
→
M
E
U
D
I
R
V
A
H
O
T
N
Q
→
M
E
U
D
I
R
V
A
H
O
T
N
Q
→
M
E
R
D
I
O
U
A
H
N
Q
T
V


→
M
E
R
D
I
O
U
A
H
N
Q
T
V


→
R
M
U
E
O
T
V
D
I
N
Q
A
H

4. Trace of Insertions



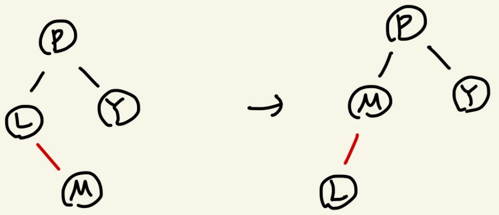
4.

Insert Y

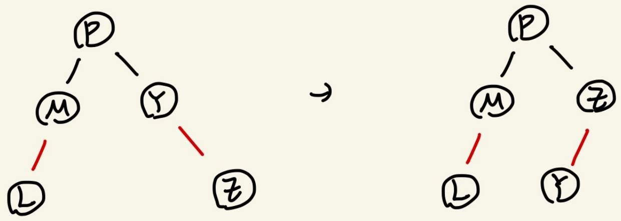
Insert L

Insert P

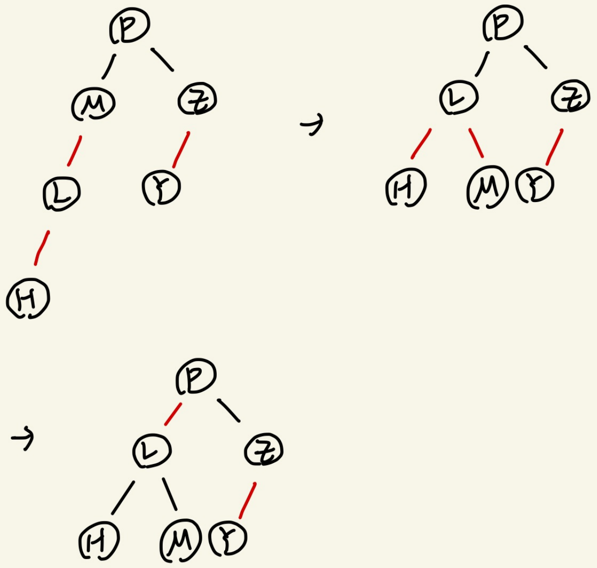
Insert M



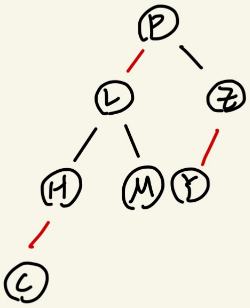
Insert Z



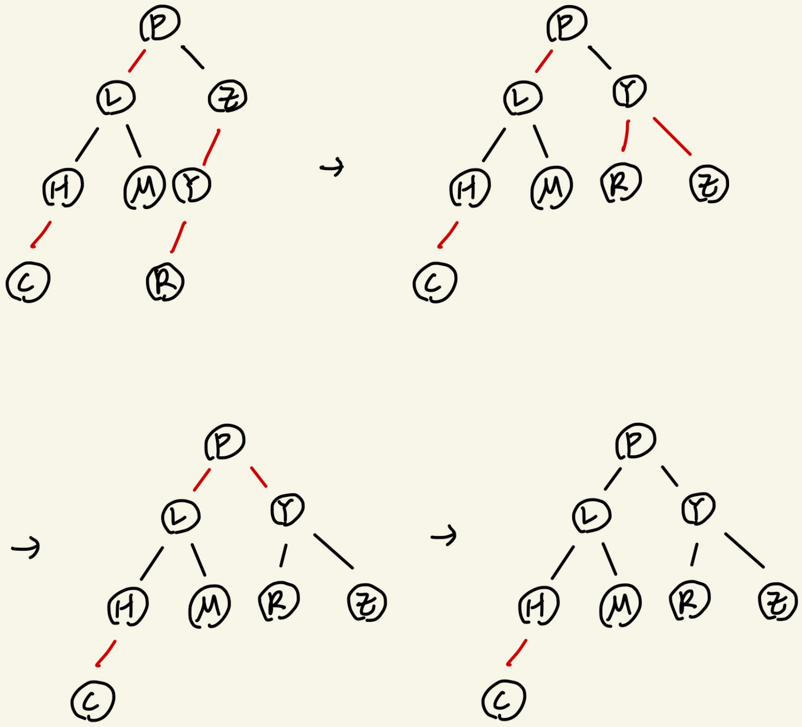
Insert H



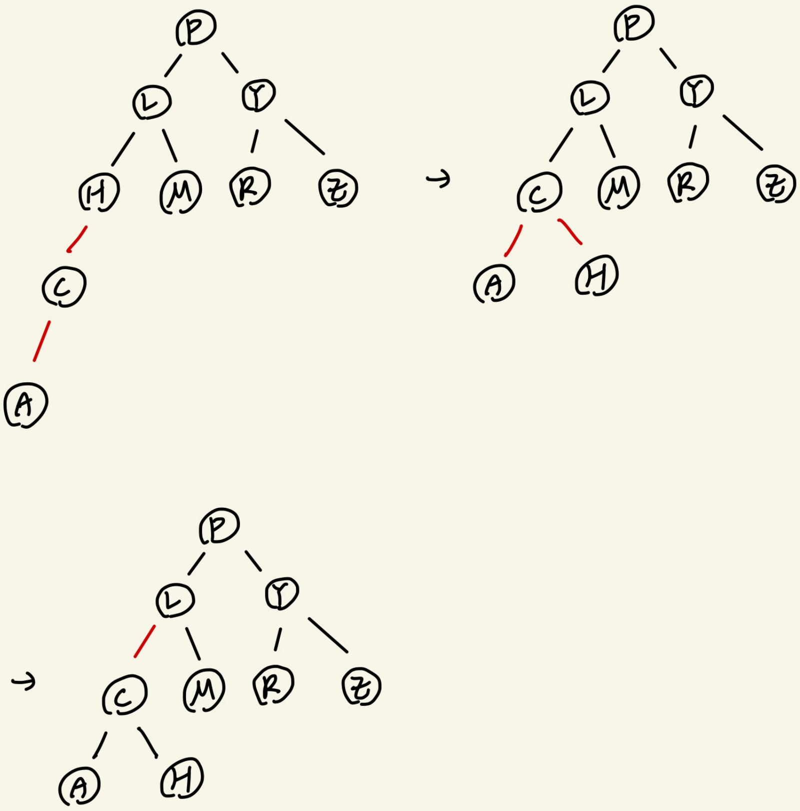
Insert C



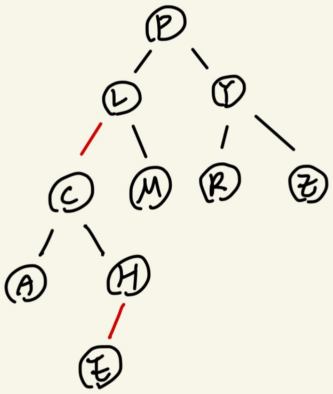
Insert R



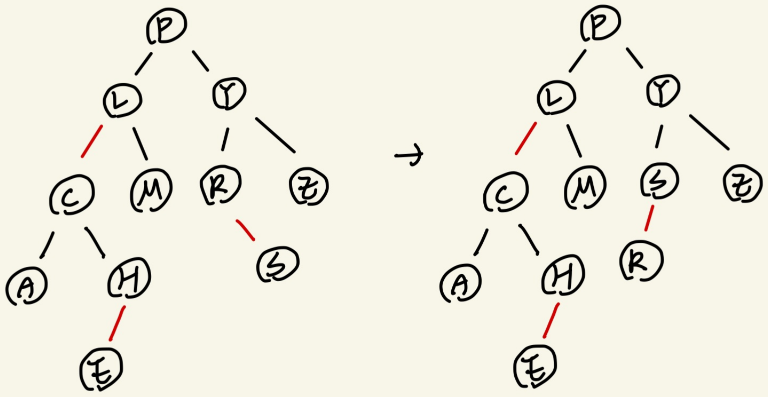
Insert A



Insert E



Insert S



5.

```
private Node moveRedLeft(Node h)
{
    colorFlip(h);
    if (isRed(h.right.left))
    {
        h.right = rotateRight(h.right);
        h = rotateLeft(h);
        colorFlip(h);
    }
    return h;
}

private Node moveRedRight(Node h)
{
    colorFlip(h);
    if (isRed(h.left.left))
    {
        h = rotateRight(h);
        colorFlip(h);
    }
    return h;
}

public void delete(Key key)
{
    root = delete(root, key);
    root.color = BLACK;
}

private Node delete(Node h, Key key)
{
    if (key.compareTo(h.key) < 0)
    {
        if (!isRed(h.left) && !isRed(h.left.left))
           h = moveRedLeft(h);
        h.left = delete(h.left, key);
    }
    else
    {
        if (isRed(h.left))
           h = rotateRight(h);

        if (key.compareTo(h.key) == 0 && (h.right == null))
           return null;

        if (!isRed(h.right) && !isRed(h.right.left))
           h = moveRedRight(h);
        if (key.compareTo(h.key) == 0)
        {
            h.val = get(h.right, min(h.right).key);
            h.key = min(h.right).key;
            h.right = deleteMin(h.right);
        }
        else
           h.right = delete(h.right, key);
    }

    return fixUp(h);
}
```

The code on top is provided by the inventor of the left-leaning red-black tree – Robert Sedgewick.

6.

```
class TreeNode:
    def __init__(self, val = None, left = None, right = None):
        self.val = val
        self.left = left
        self.right = right


def find_min(root):
    if not root: return None
    if root and not root.left: return root.val

    if root and root.left:
        return find_min(root.left)
```

Since a left-leaning red black tree is a BST, the minimum val of the tree is the most left one. Thus, my algorithm basically recursively traverses to the left node if there exists one and return the value of the most left node. If the tree doesn't have any node, then it will return None.

A left-leaning red black tree is a balanced BST, so the time complexity of searching any node is O(logN).

7.

```
class TreeNode:
    def __init__(self, val = None, left = None, right = None):
        self.val = val
        self.left = left
        self.right = right

class LLRB:
    def __init__(self):
        self.size = 0
        self.root = None

    def insert(self, val):
        ...
        self.size += 1

    def delete(self, val):
        ...
        self.size -= 1

    ...

def find_median(root, count, llrb_size, res):
    # inorder traversal


    if root.left: find_median(root.left, count, llrb_size, res)

    count[0] += 1

    # llrb has odd elements
    if llrb_size % 2 != 0 and count[0] == (llrb_size // 2) + 1:
        res.append(root.val)

    # llrb has even elements
    if llrb_size % 2 == 0 and count[0] in (llrb_size/2, llrb_size/2 + 1):
        res.append(root.val)


    if root.right: find_median(root.right, count, llrb_size, res)


llrb = LLRB()

"""
insertions and deletions
"""
count = [0]
res = []
find_median(llrb.root, count, llrb.size, res)
print(sum(res) / len(res)) if res else print(None)
```

My algorithm records the size of the llrb tree. When ever we want to find the median value of the llrb tree, it does an inorder traversal with a count incrementing in every iteration, and appends the value to the result list when the count reaches the target number. Finally, it returns the average of the numbers in the result list to get the median. Since the algorithm is based on a inorder traversal, which traverses all the elements once, so the time complexity is O(n).

8. (a)

```
hashMap = [[] for _ in range(11)]
nums = [12, 14, 34, 88, 23, 94, 11, 39, 20, 16, 5]

for num in nums:
    key = (2*num + 5) % 11
    hashMap[key].append(num)

print(hashMap)

"""
Output

[[14], [20], [], [], [16, 5], [88, 11], [94, 39], [12, 34, 23], [], [], []]
"""
```

(b)

```
hashMap = [None for _ in range(11)]
nums = [12, 14, 34, 88, 23, 94, 11, 39, 20, 16, 5]

for num in nums:
    key = (2*num + 5) % 11

    while True:
        key = (key + 1) % 11

        if hashMap[key] == None:
            hashMap[key] = num
            break

print(hashMap)


"""
Output

[11, 14, 39, 20, 5, 16, 88, 94, 12, 34, 23]
"""
```

9.

```python
def two_sum(H, J, K):
    H_hash_map = set(H)

    for num in J:
        if K - num in H_hash_map:
            return (True, num, K - num)

    return (False, None, None)


H = [4, 5, 7, -1, -9, 22]
J = [0, -5, 10, 44, 100, -12]
print(two_sum(H, J, 91))
print(two_sum(H, J, 92))

"""
Output

(True, 100, -9)
(False, None, None)
"""
```

My algorithm first turn array H into a set, then uses one for-loop to traverse all the elements in array J, which takes O(n). During each iteration, it checks whether there is an element in H that can be added up with the current J element to form K, which takes O(1). Thus, my algorithm has a time complexity of O(n), which meets the requirement.

10. (a) The worst time complexity of my algorithm is O(n).

    (b) The worst case will happen when the algorithm traverses all the elements in array J without an early termination.

11.

```python
def two_sum(H, K):
    Set = set()

    for num in H:
        if K - num in Set:
            return (True, num, K - num)

        Set.add(num)

    return (False, None, None)


H = [4, 5, 7, -1, -9, 22]
print(two_sum(H, -5))
print(two_sum(H, 7))

"""
Output

(True, -9, 4)
(False, None, None)
"""
```

My algorithm first creates an empty set, then uses one for-loop to traverse all the elements in array H, which takes O(n). During each iteration, it checks whether there is an previous element in H (which is stored in the set) that can be added up with the current element to form K, which takes O(1). Thus, my algorithm has a time complexity of O(n), which meets the requirement.

1 Reference

https://sedgewick.io/wp-content/themes/sedgewick/papers/2008LLRB.pdf