COL 106 Lecture 17

Topic : Binary Search Trees :

Min / Max, Predecessor / Successor, Deletion

Binary Search Trees

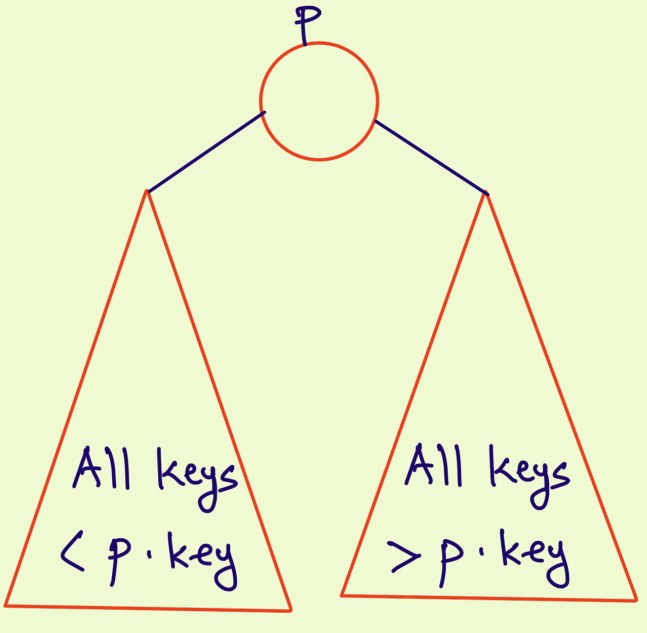
Definition : Let $(U, <)$ be an ordered set. A binary tree with keys from U is said to be a binary search tree if for every node p the following are true.

1. For every node q in the left subtree of x:

   $q.key < p.key$

2. For every node q in the right subtree of x:

   $p.key < q.key$



Binary Search Tree Operations

☑ List in sorted order	(Accessor)	$O(n)$
☑ Search	(Accessor)	$O(h)$
☑ Add element (Insertion)	(Modifier)	$O(h)$
☑ Remove element (Deletion)	(Modifier)	$O(h)$
☑ Min, Max,	(Accessor)	$O(h)$
☑ Predecessor, Successor	(Accessor)	$O(h)$.

Minimum

Define First(q) = first node in the inorder traversal of subtree rooted at q.

Claim:

$$First(q) = \begin{cases} q & \text{if } q.left \text{ is None} \\ First(q.left) & \text{otherwise} \end{cases}$$

Min :

Return First(root).key

Time: $O(h)$.

Maximum

Define Last(q) = last node in the inorder traversal of subtree rooted at q.

Claim:

$$Last(q) = \begin{cases} q & \text{if } q.right \text{ is None} \\ Last(q.right) & \text{otherwise} \end{cases}$$

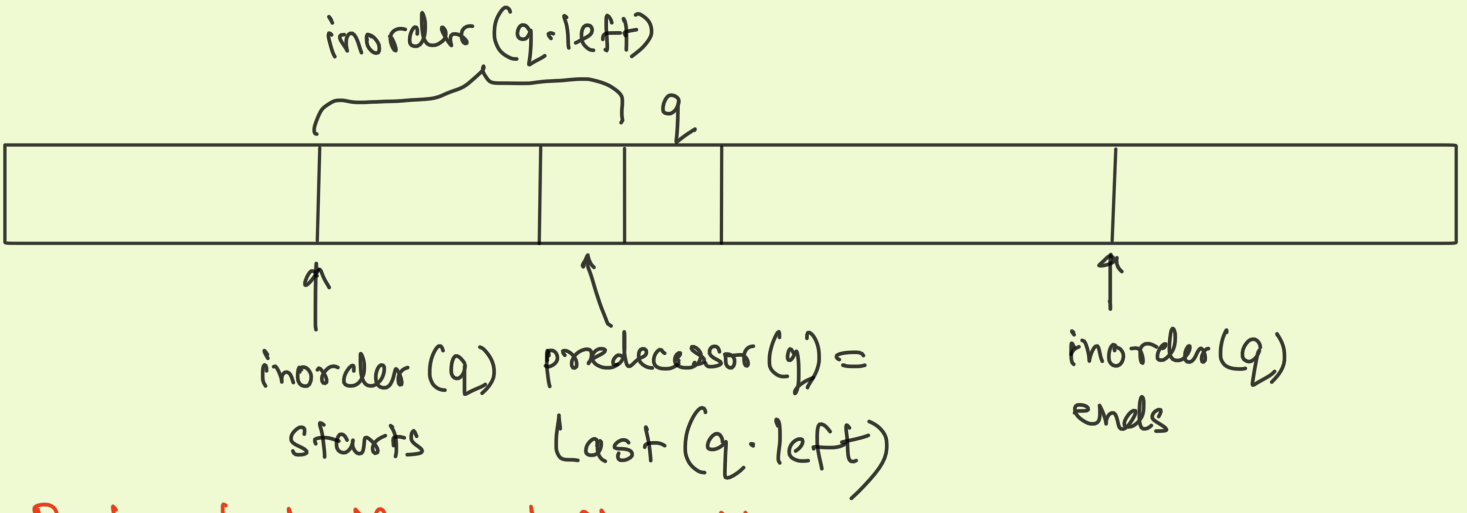
Max :

Return Last(root).key

Time: $O(h)$.

Predecessor and Successor

Definition: { predecessor (q) / successor (q) } is the node just { before / after } q in the inorder traversal.



But what if q.left = None, ie. inorder (q.left) is empty?

Predecessor

predecessor (q):

If q.left is not None

Return Last (q.left)

Else

Find the node r whose successor is q and return it — but how?

Successor

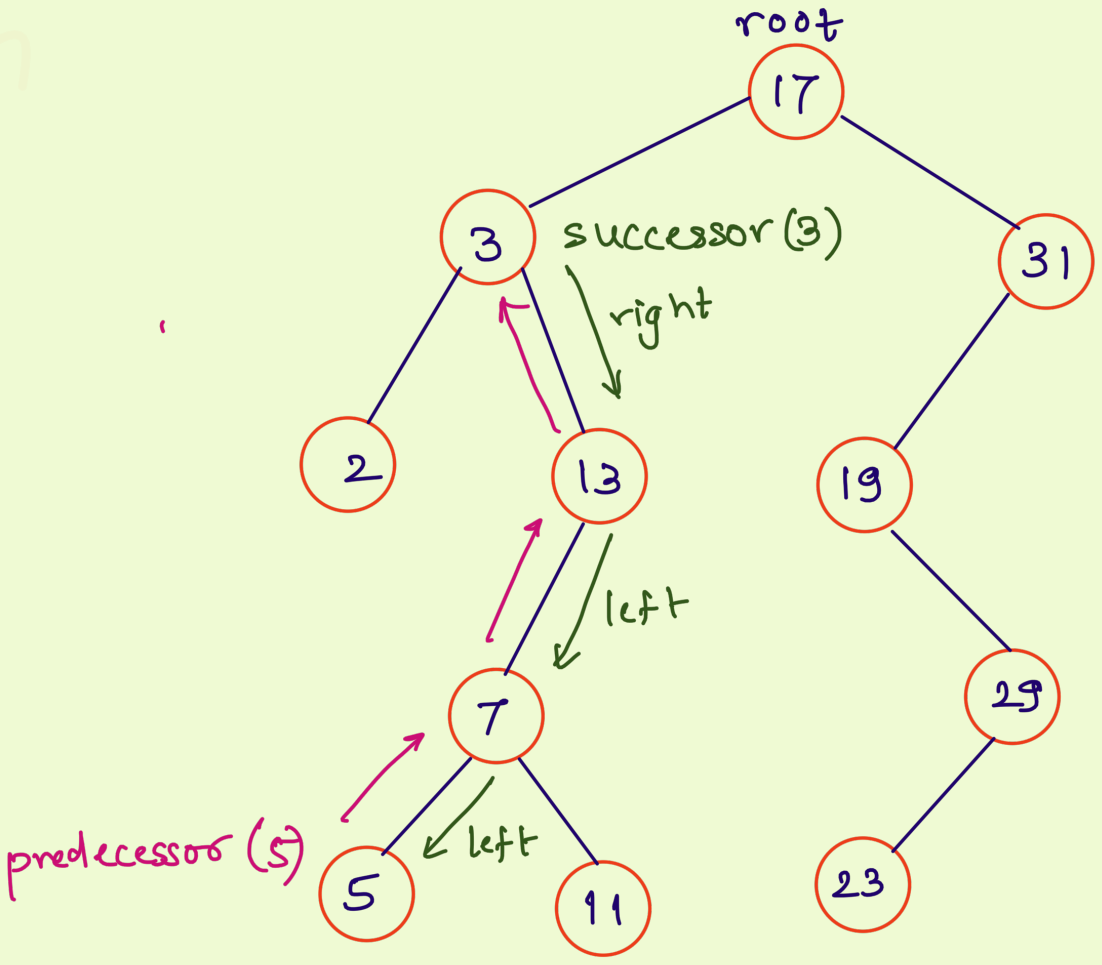
successor (q):

If q.right is not None

Return First (q.right)

Else

Find the node r whose predecessor is q and return it — but how?



Predecessor

```
predecessor (q):
    If q.left is not None
        Return Max (q.left)
    Else
        p ← q
        while (p = p.parent.left):
            p ← p.parent
        Return p.parent
```

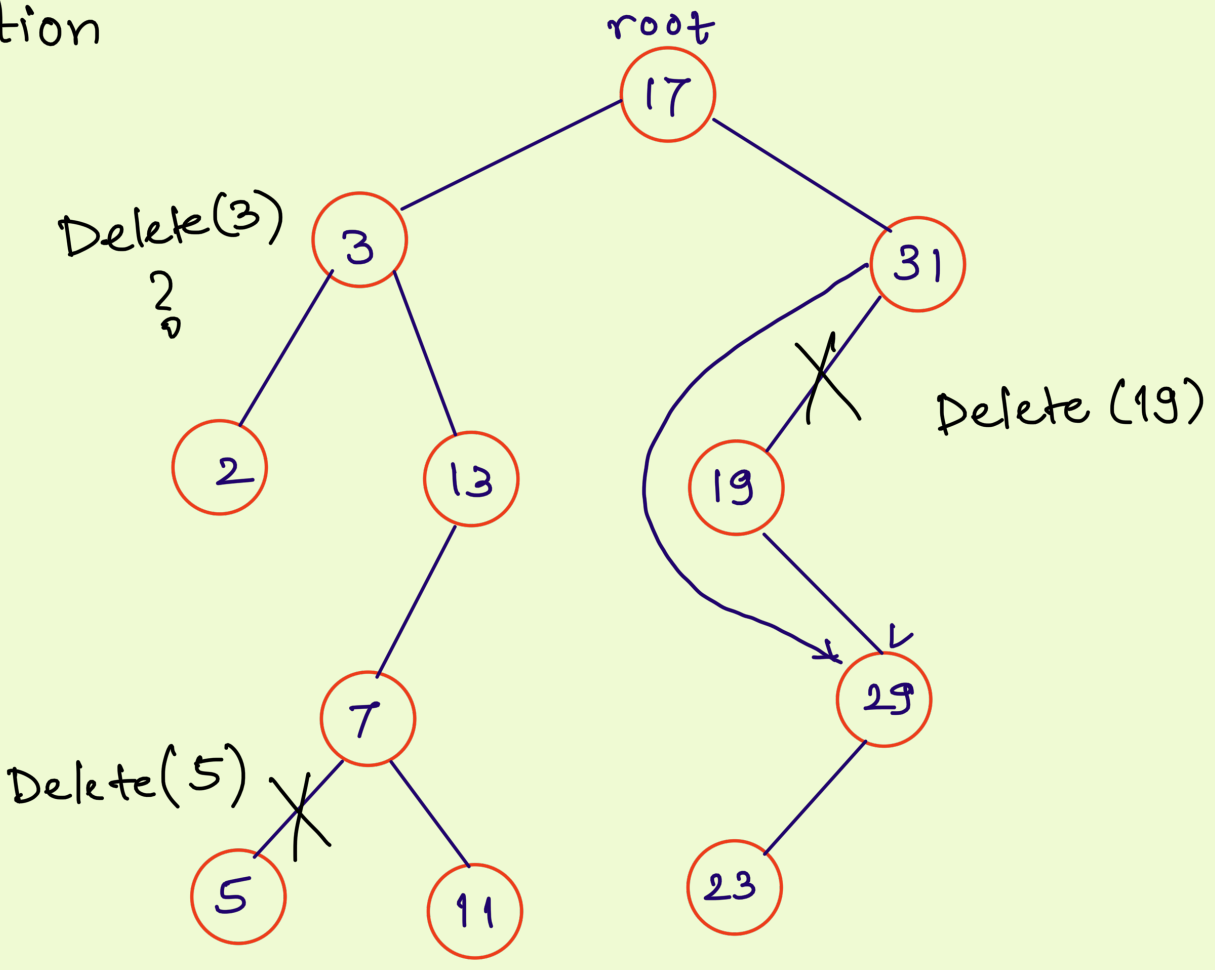
Time : O(h)

Successor

```
successor (q):
    If q.right is not None
        Return Min (q.right)
    Else
        p ← q
        while (p=p.parent.right):
            p ← p.parent
        Return p.parent
```

Time : O(h)

Deletion



Deletion

Delete (x):

Search for the node q such that q.key = x. — O(h)

If q has no child:

Disconnect q from q.parent. — O(1)

Else if q has (exactly) one child: — O(1)

Make q's child a child of q.parent instead of q.

Else: — O(h)

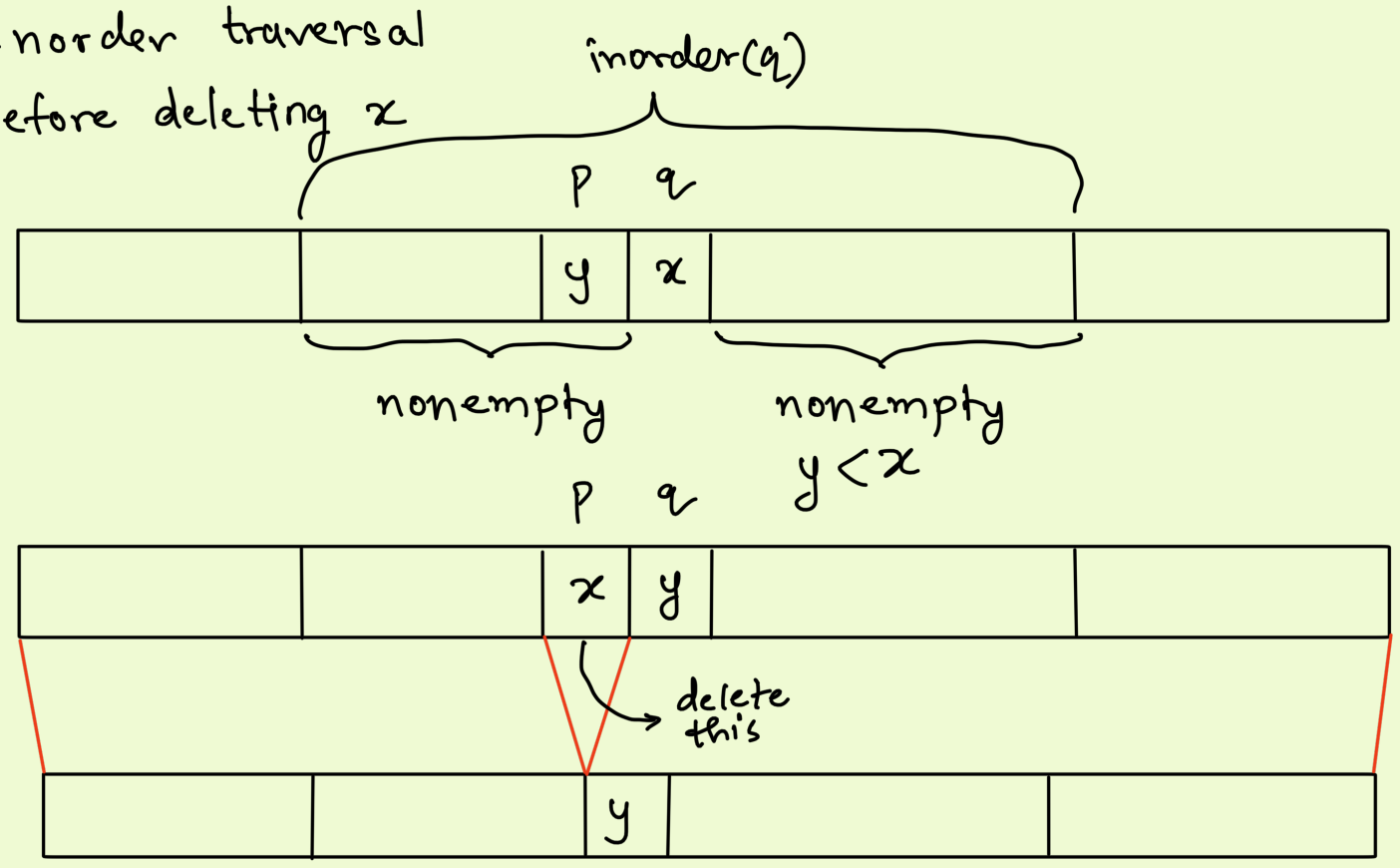
Exchange q.key with key of predecessor of q

Delete the predecessor of q. (It has ≤ 1 child.)

Time: O(h).

Delete (x) — the "else" case

Inorder traversal
before deleting x



Inorder traversal after deleting x.

The predecessor or successor of a node in case 3 can have at most one child, so we can manage it by case 1 or 2