# Binary Search Tree – II

Querying a BST : finding min, max, predecessor, successor, or search

**Search :** returns a pointer to a node with key k if one exists in the subtree, else returns NIL.

```
Tree-Search (x, k)        // x → root of a subtree in T ; k → the value to search for
if x == NIL or k == x.key
    return x
if k < x.key
    return Tree-Search (x.left, k)
else
    return Tree-Search (x.right, k)
```

Running time of Tree-Search is $O(h)$

where $h$ is the height of the tree

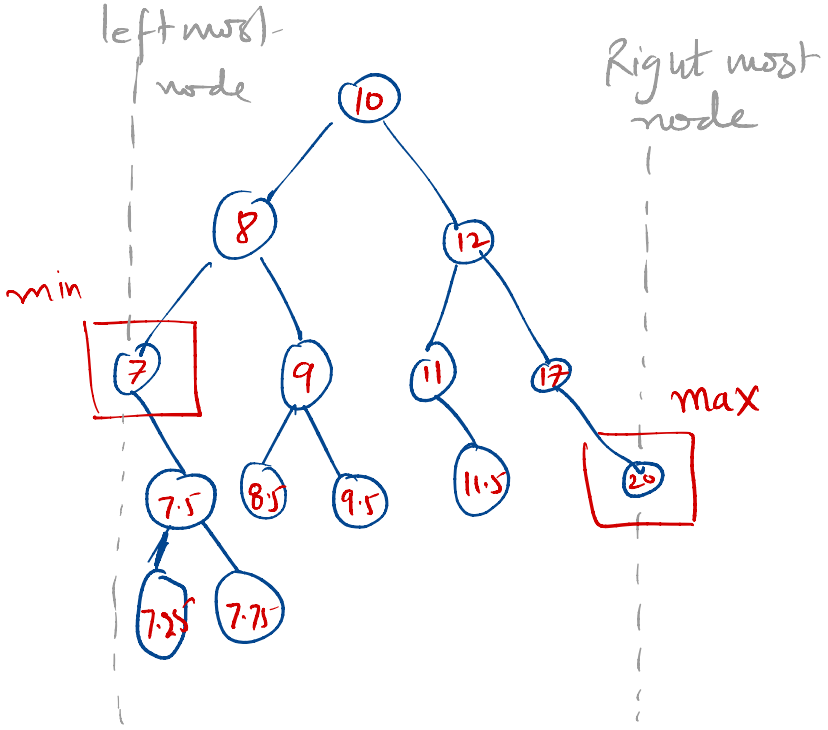
## Iterative-Tree-Search $(x, k)$

```
while  x ≠ NIL  and  k ≠ x.key
    if  k < x.key
        x = x.left
    else
        x = x.right
return x
```

→ $O(h)$

## – Minimum / Maximum

↳ returns a pointer to min/max element in a subtree rooted at node $x$ (assumed to be not NIL)



**Tree - Minimum (x)**

```
while  x.left ≠ NIL
|      x = x.left
return x
```

**Tree - Maximum (x)**

```
while  x.right ≠ NIL
|      x = x.right
return x
```

- Successor and predecessor

↳ Find successor of a given node in the sorted order determined by the inorder walk.

⇓

Define successor as the next node visited in an inorder tree walk.

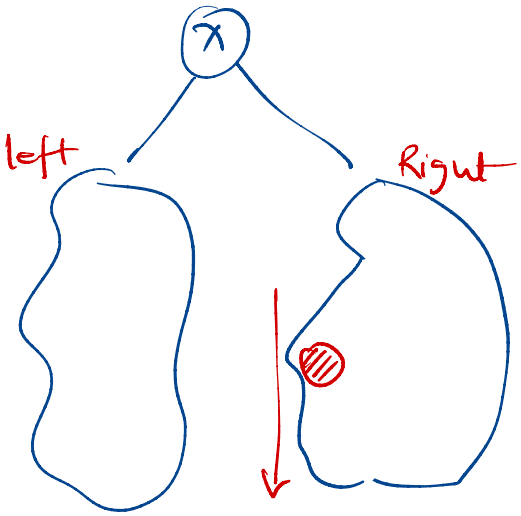
Two cases : (I) If the right subtree of x is non empty

↳ successor = leftmost node in x's right subtree

↓

Call

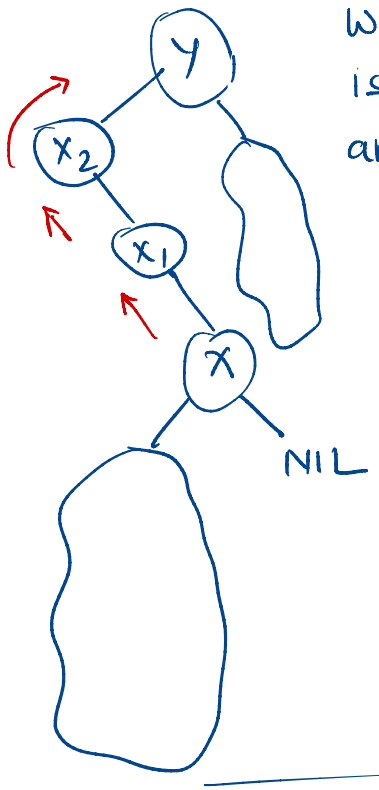
Tree - Minimum (x.right)




succ = leftmost node

(11) If the right subtree of x is empty

succ = lowest ancestor whose left child is also an ancestor of x.




Then y is a successor, if y is the lowest ancestor of x whose left child is also an an ancestor of x.

---

Tree - Successor (x)

---

```
if x.right ≠ NIL
    return Tree-Minimum(x.right)
else
    y = x.p
    While y ≠ NIL and x == y.right
        x = y
        y = y.p
    return y
```

---