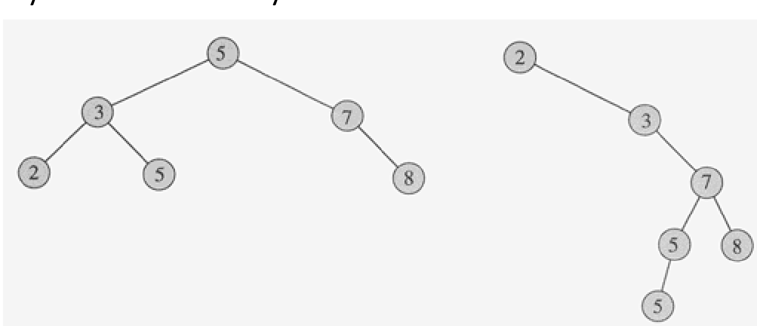
Binary Search Trees

Chapter 12

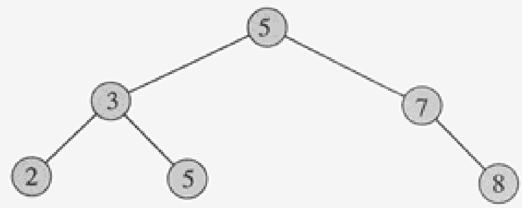
What is a Binary Search Tree?

- A binary tree where each node is an object
  - Each node has a key value, left child, and right child (might be empty)
- Each node satisfies the binary search tree property
  - Let x be a node in the BST. The left child's key must be <= x's key. The right child's key must be >= x's key



Traversing the BST

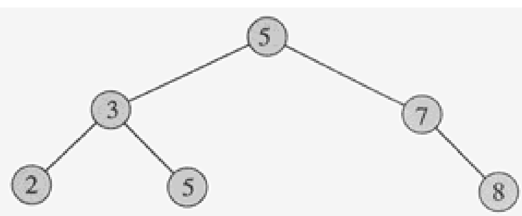
```
INORDER-TREE-WALK(x)
if x ≠ NIL
    then INORDER-TREE-WALK(left[x])
         print key[x]
         INORDER-TREE-WALK(right[x])
```



O(n) time

Searching a BST

```
TREE-SEARCH(x, k)
if x = NIL or k = key[x]
    then return x
if k < key[x]
    then return TREE-SEARCH(left[x], k)
    else  return TREE-SEARCH(right[x], k)
```

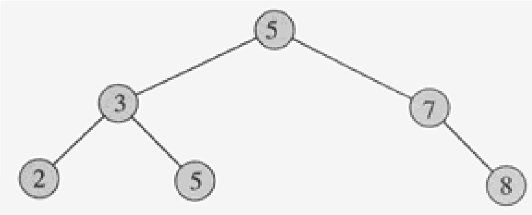


Runs in O(h) time but this could be O(n) in the worst case!
O(lgn) if the tree is balanced!

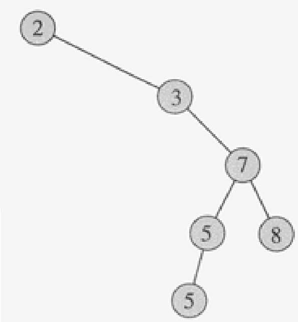
Finding min and max?

Successor

- Finding the node with the next largest (or equal) value



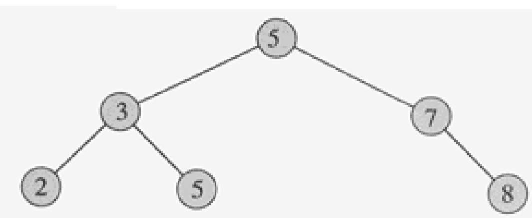

```
TREE-SUCCESSOR(x)
if right[x] ≠ NIL
   then return TREE-MINIMUM(right[x])
y ← p[x]
while y ≠ NIL and x = right[y]
     do x ← y
        y ← p[y]
return y
```



O(h) runtime

Insertion

```
TREE-INSERT(T, z)
y ← NIL
x ← root[T]
while x ≠ NIL
    do y ← x
       if key[z] < key[x]
          then x ← left[x]
          else x ← right[x]
p[z] ← y
if y = NIL
   then root[T] ← z                    ▷ Tree T was empty
   else if key[z] < key[y]
           then left[y] ← z
           else right[y] ← z
```



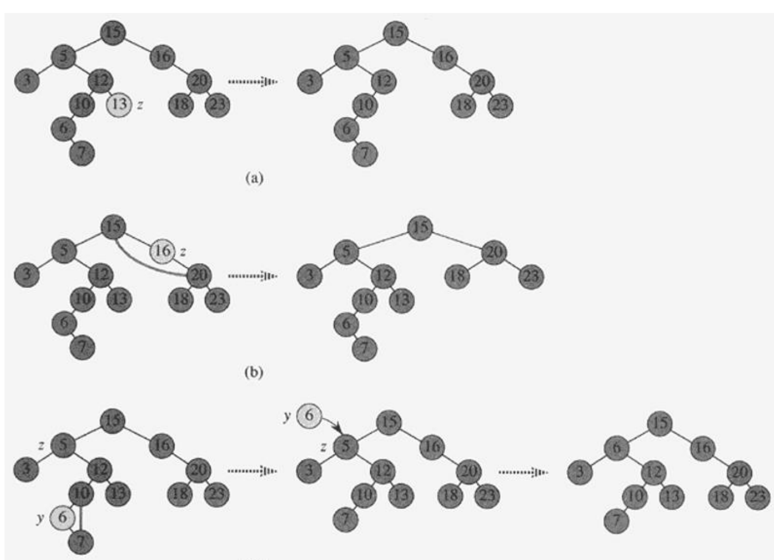
O(h) runtime

Deletion

- Deleting a node z from a BST T

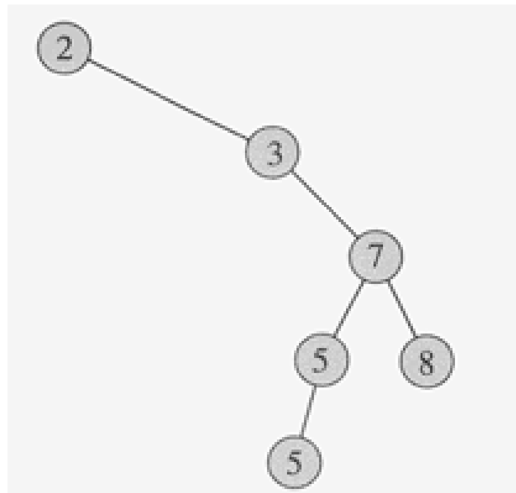
1. If z has no children the simply remove it by modifying its parent to replace z with nil as its child
2. If z has just one child then we elevate that child to take z's position in the tree by modifying z's parent to replace z by z's child
3. If z has two children then:
   - Find z's successor y – which must be in z's right subtree – and have y take z's position in the tree
   - As a successor y in the right subtree, y has at most one child. Remove y using rule 2
   - The rest of z's original right subtree becomes y's right subtree and z's left subtree becomes y's left subtree

Delete Examples



Deletion

```
Tree-Delete(T,z)
if z.left == NIL
        Transplant(T, z, z.right)
elseif z.right == NIL
        Transplant(T, z, z.left)
else
        y = Tree-Minimum(z.right)
        if y.p != z
                Transplant(T,y,y,right)
                y.right = z.right
                y.right.p = y
        Transplant(T, z, y)
        y.left = z.left
        y.left.p = y
```



BST

- Worst case?
- Best case?
- Expectation for randomly built BST?