Binary Search Trees

Ira A. Fulton Schools of Engineering
Arizona State University

Binary Search Trees, Part 1

- Binary Search Trees, Representation
- Tree Walks
- Search, Min, Max, Successor
- Insertion
- Deletion

Dynamic Data Structure, Dictionary

- A dynamic data structure manipulates sets that can
  - grow, shrink, or change (over time)
- A dictionary is a dynamic data structure that supports:
  - Insertion, Deletion, and Search (verification of membership)

Search Tree

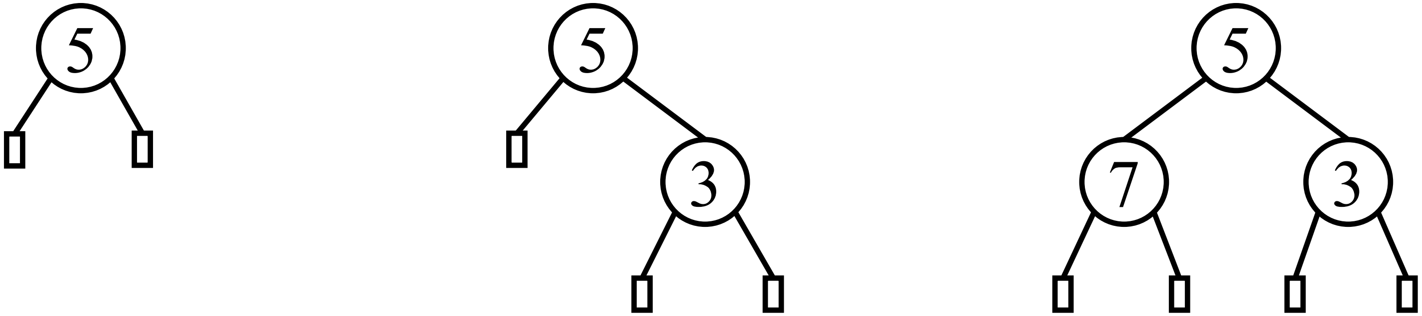
- **A search tree is a data structure that supports both dictionary and priority queue operations:**
  - Search, insert, delete;
  - Minimum, maximum;
  - Predecessor, successor.

Binary Trees (not BST, yet)

A binary tree is a data structure defined on a finite set of nodes that either

- contains no nodes (called a null tree, denoted by null), or
- is composed of three disjoint sets of nodes: a root node, a binary tree called its left subtree, and a binary tree called its right subtree.

The definition of binary tree is recursive.



Binary Search Trees

- **Let x be a node in a binary tree. If its left subtree is not null, we call the root of its left subtree the left child of x. Otherwise we say the left child of x is null or missing.**

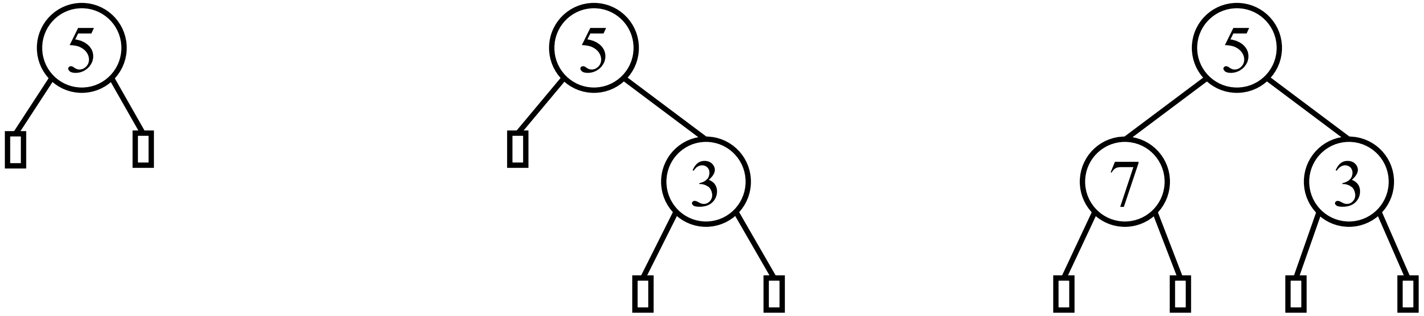
- **The right child of a node is defined similarly.**

- **We assume that each tree node has a key field.**

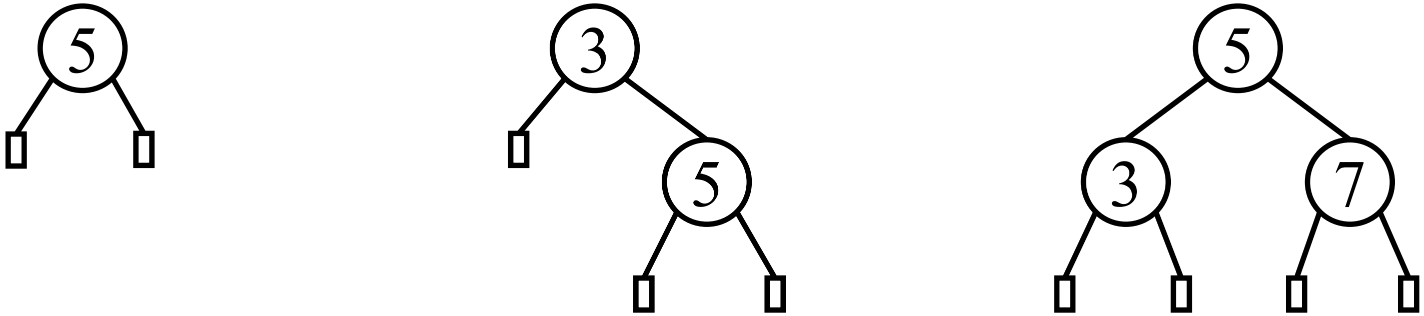
- **A binary search tree is a binary tree with the following property:**
  - For each node i in the tree, if x is any node in its left sub-tree and y is any node in its right sub-tree, then x.key ≤ i.key ≤ y.key

Examples of Binary Trees and BSTs

Examples of binary trees

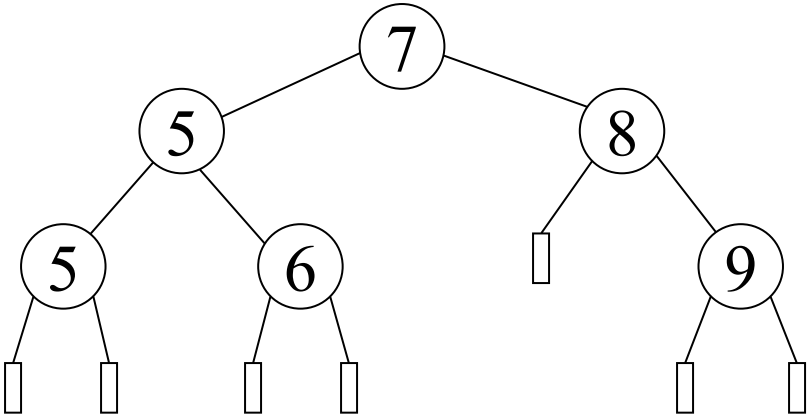


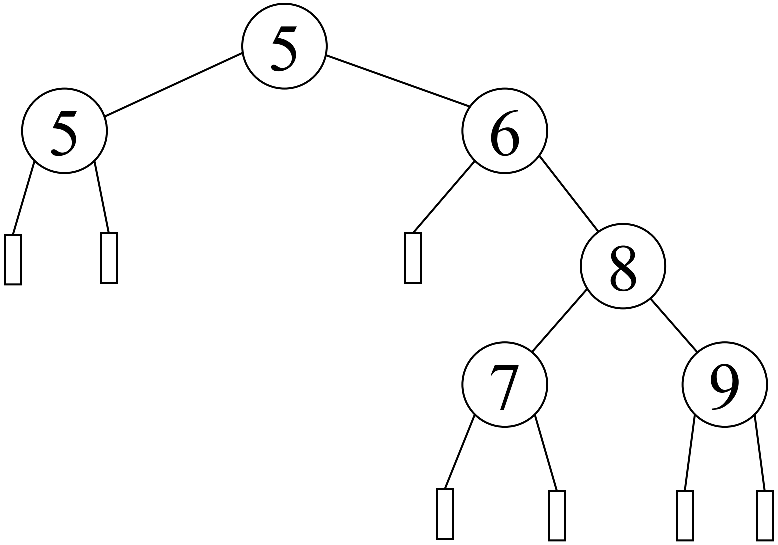
Examples of binary search trees



Examples of BST

Two BSTs for the same set

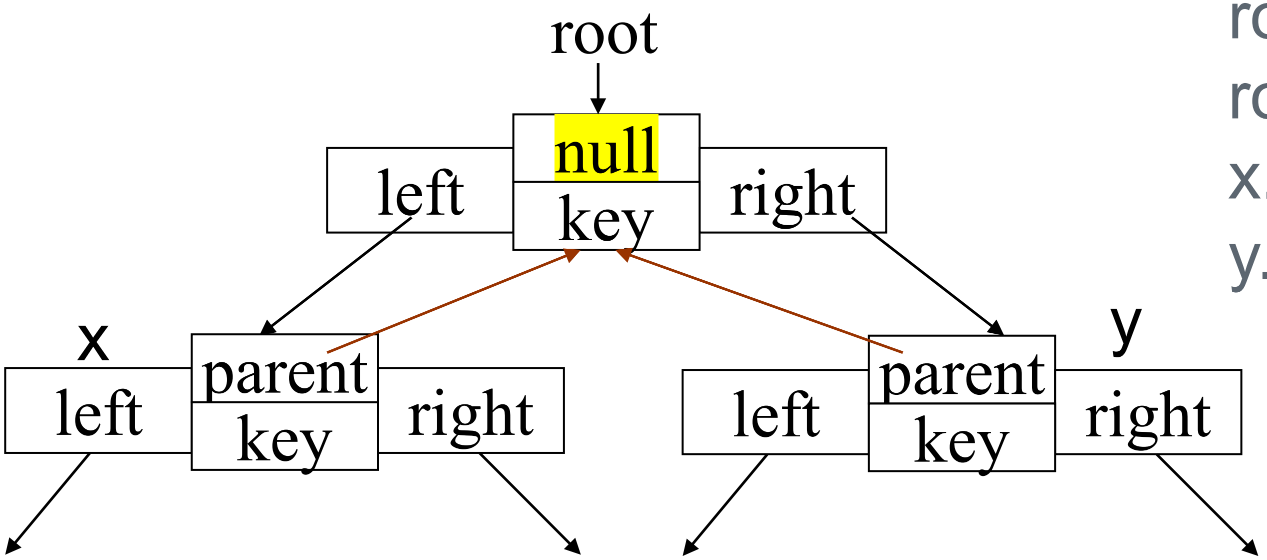




Representation of Binary Search Trees

- A tree node is defined by an object of at least four fields:
- Tree-node

  key: int // also called data…can be of other types parent, left, right: pointer to a tree node



root.left = x
root.right = y
x.parent = root
y.parent = root

ASU®

Ira A. Fulton Schools of
Engineering

Arizona State University

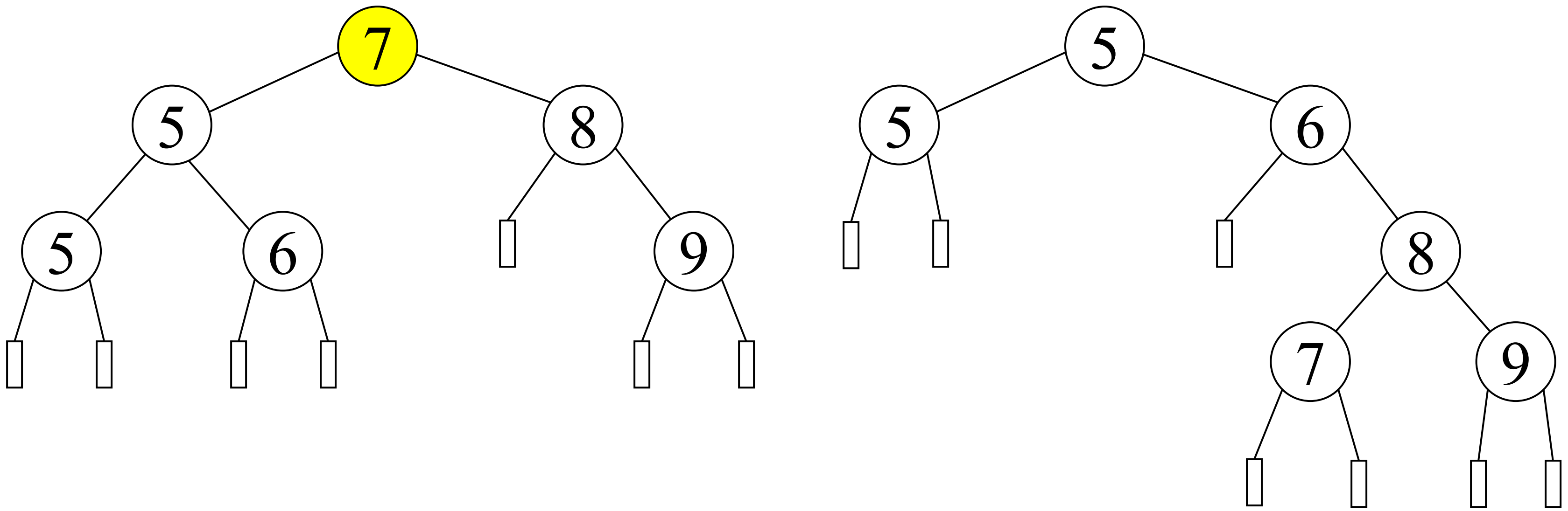
Binary Search Trees, Part 2

- Binary Search Trees, Representation
- Tree Walks
- Search, Min, Max, Successor
- Insertion
- Deletion

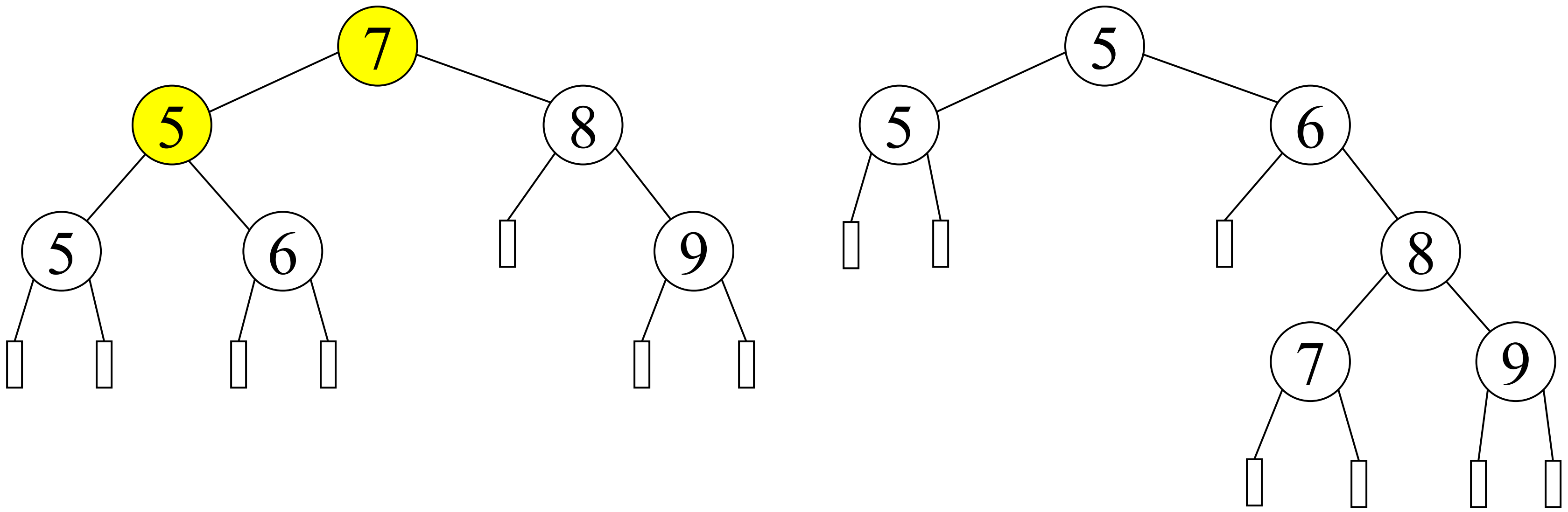
Tree Walks (Inorder Walk)

```
Inorder-Tree-Walk(x)
if $x \neq$ null then
    Inorder-Tree-Walk(x.left)       //left[x]
    print(x.key)                    //key[x]
    Inorder-Tree-Walk(x.right)      //right[x]
```

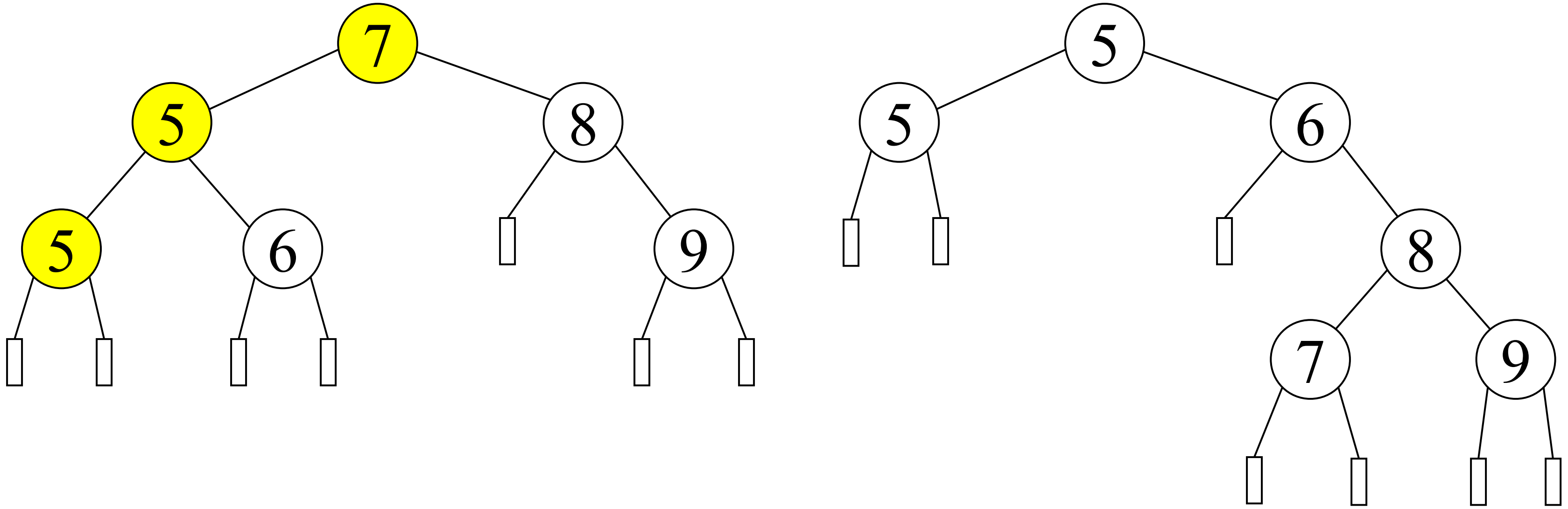
Inorder Tree Walk (example)



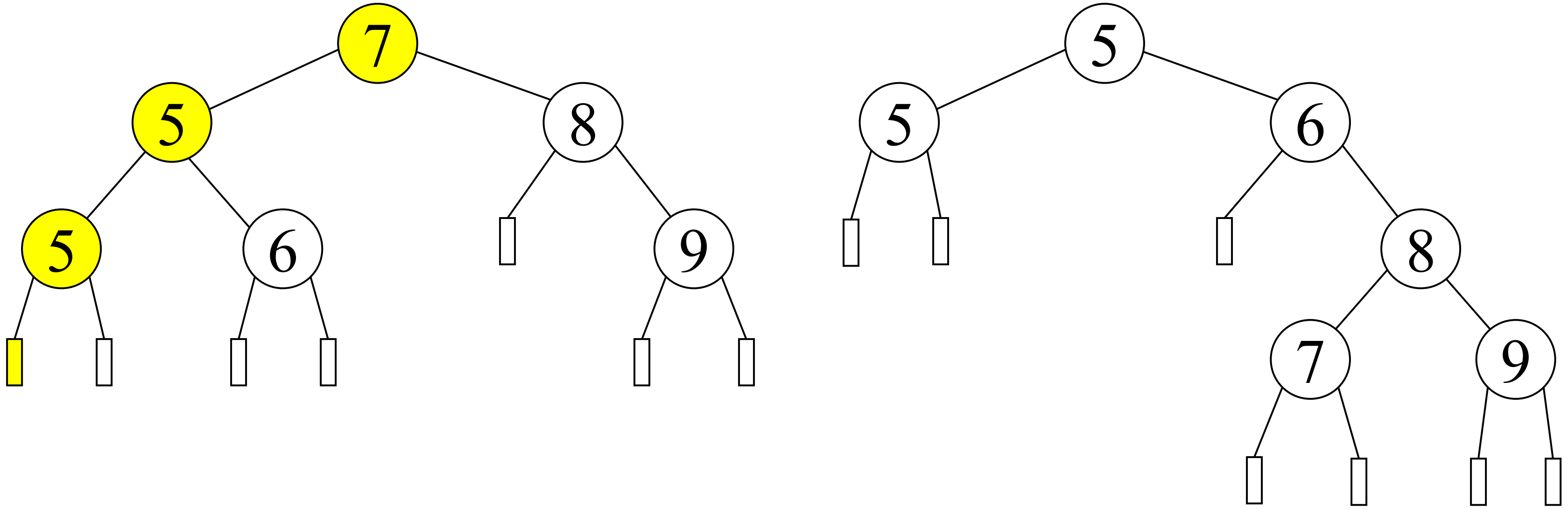
Inorder Tree Walk (example)



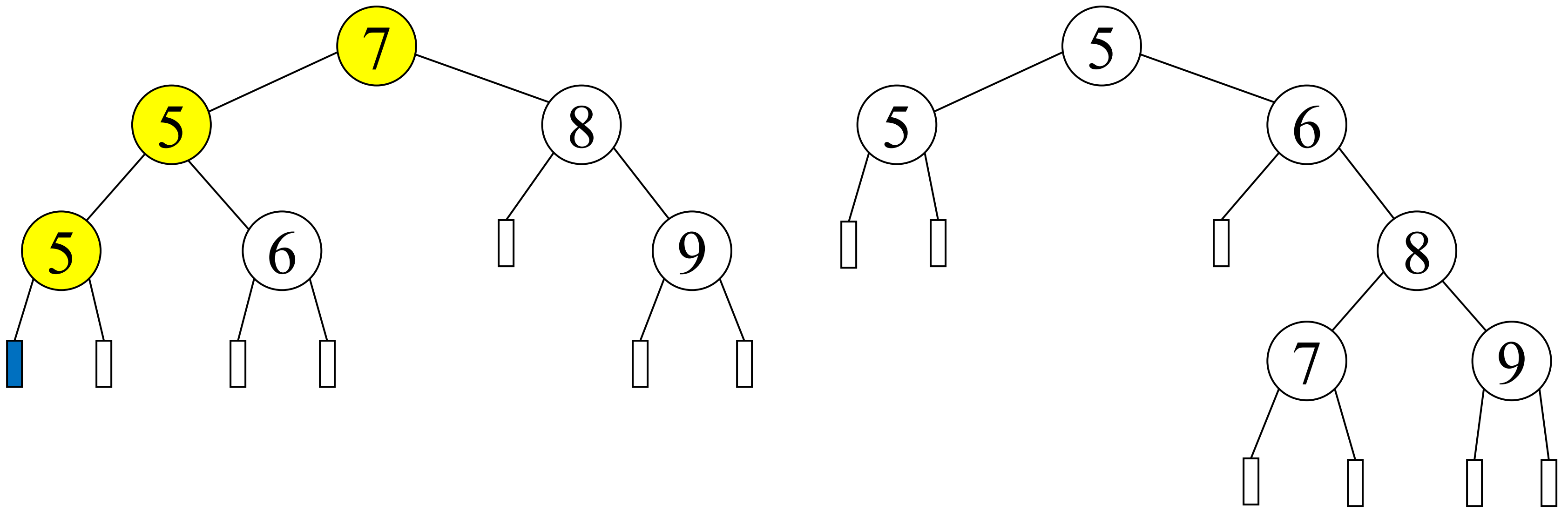
Inorder Tree Walk (example)



Inorder Tree Walk (example)

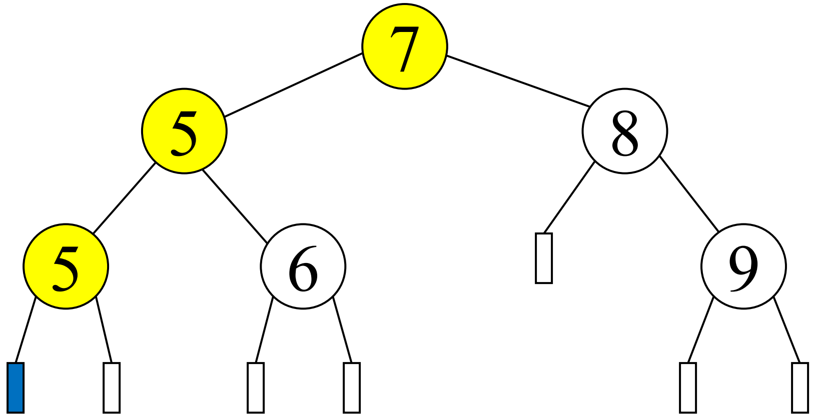


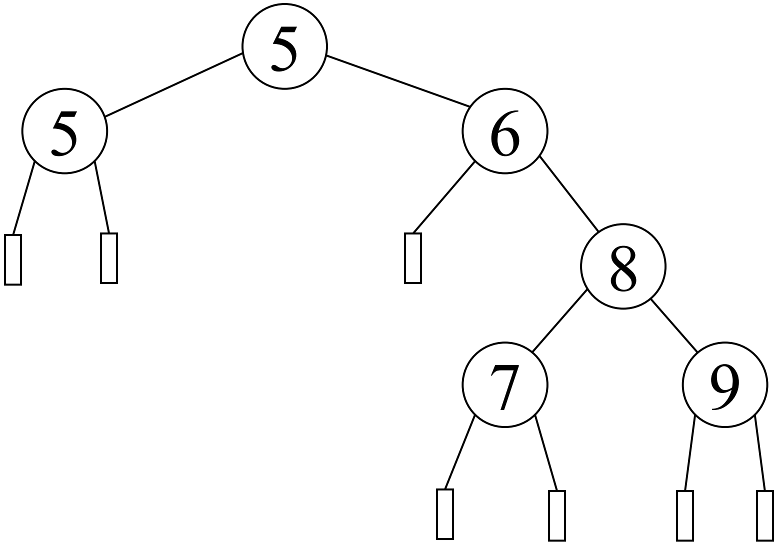
Inorder Tree Walk (example)



Inorder Tree Walk (example)

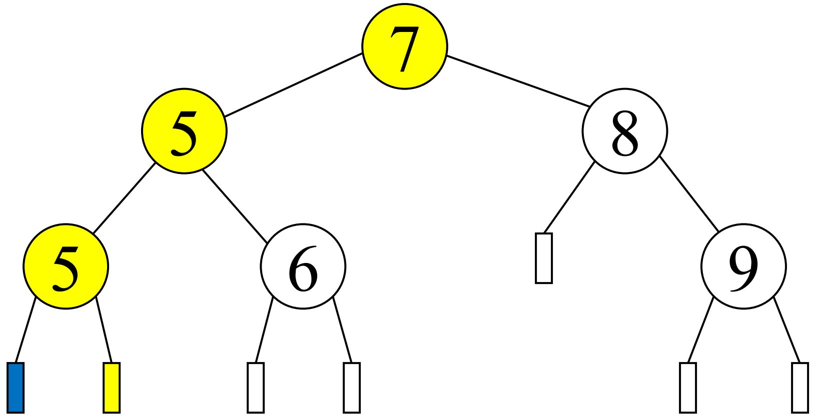
5

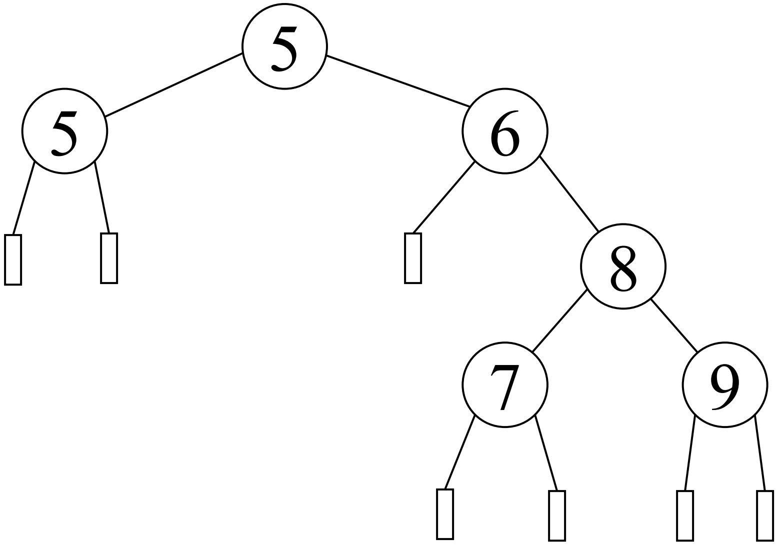




Inorder Tree Walk (example)

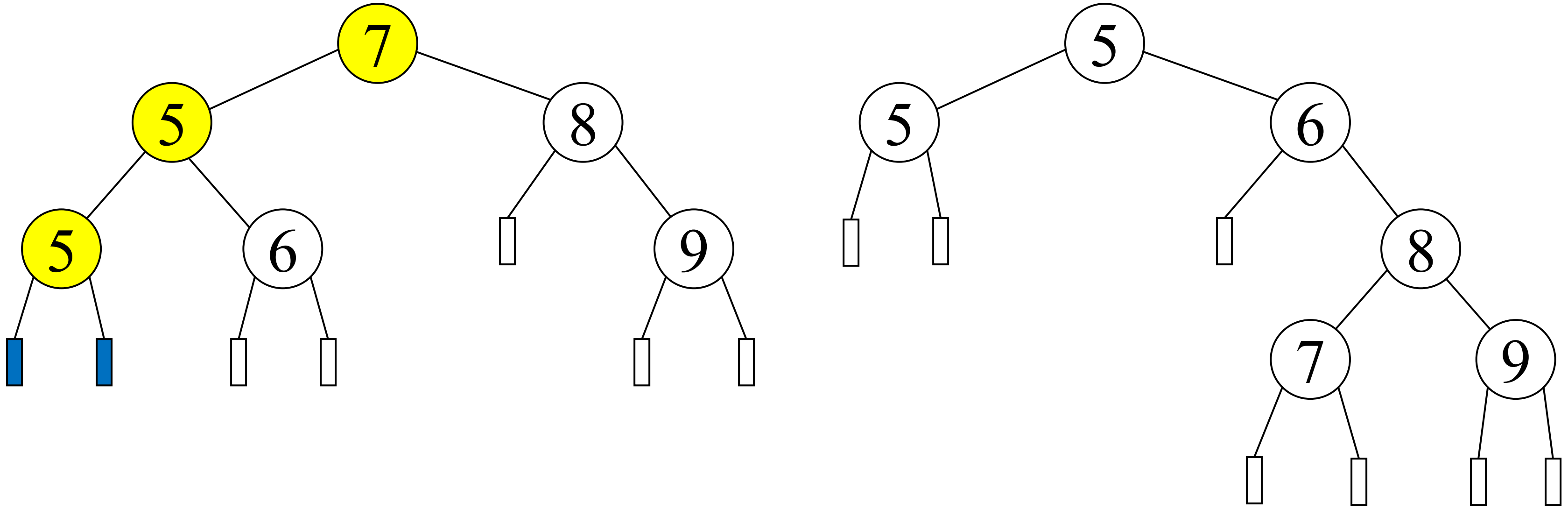
5





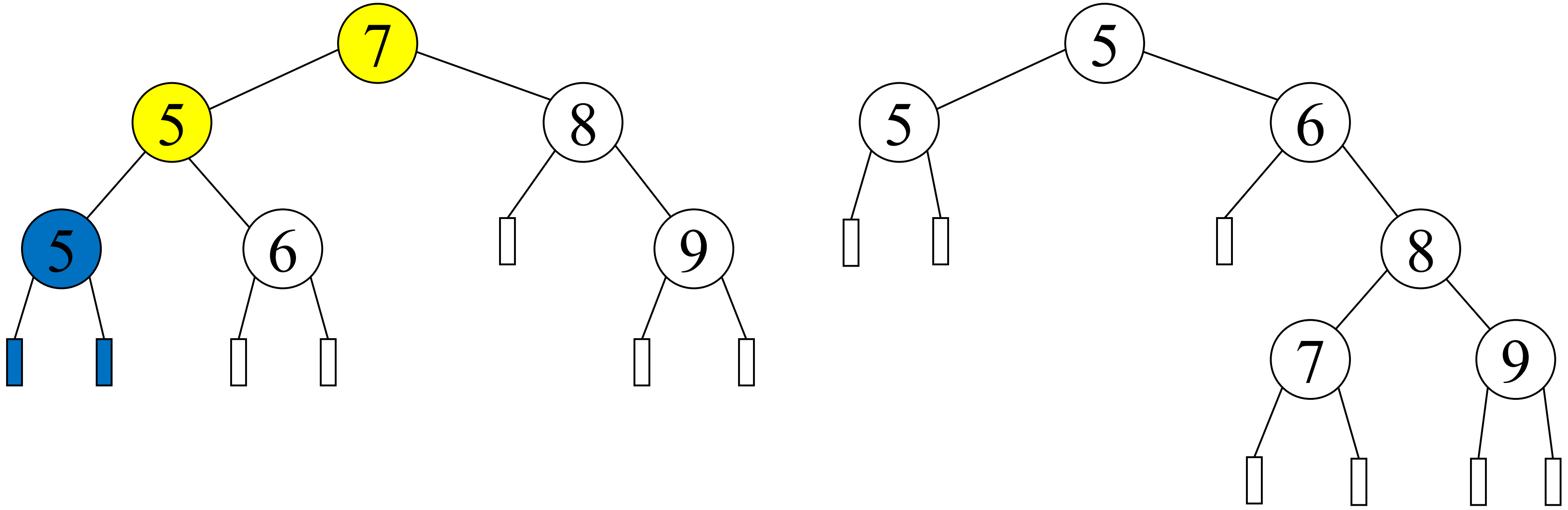
Inorder Tree Walk (example)

5



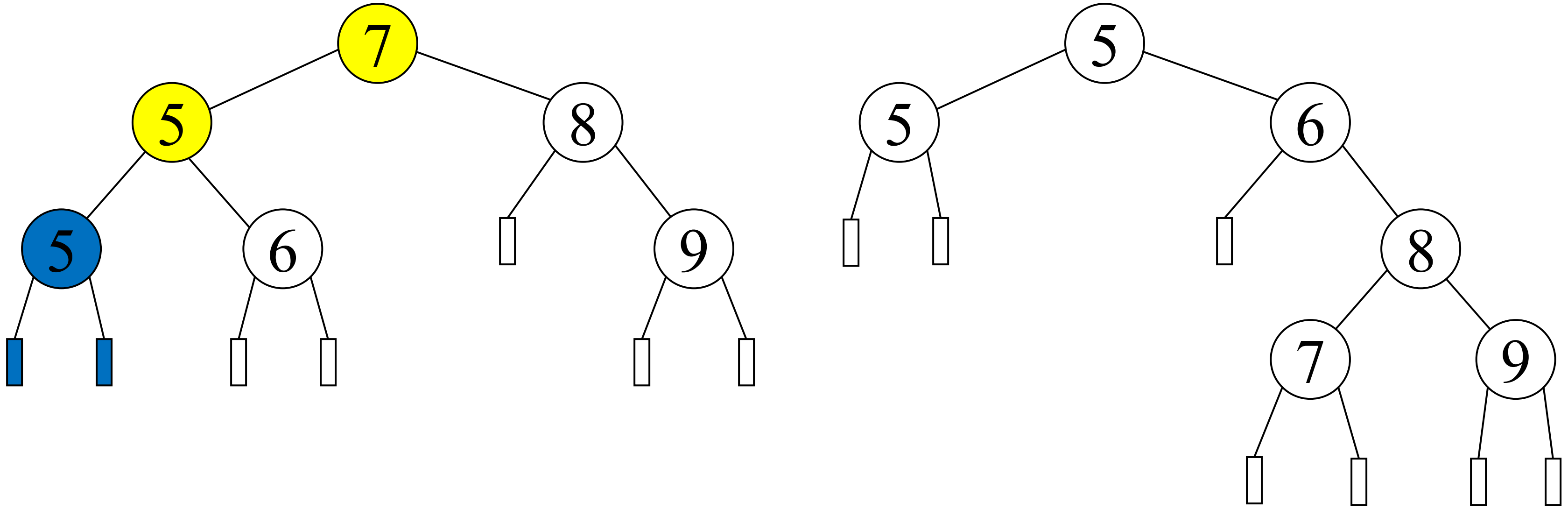
Inorder Tree Walk (example)

5



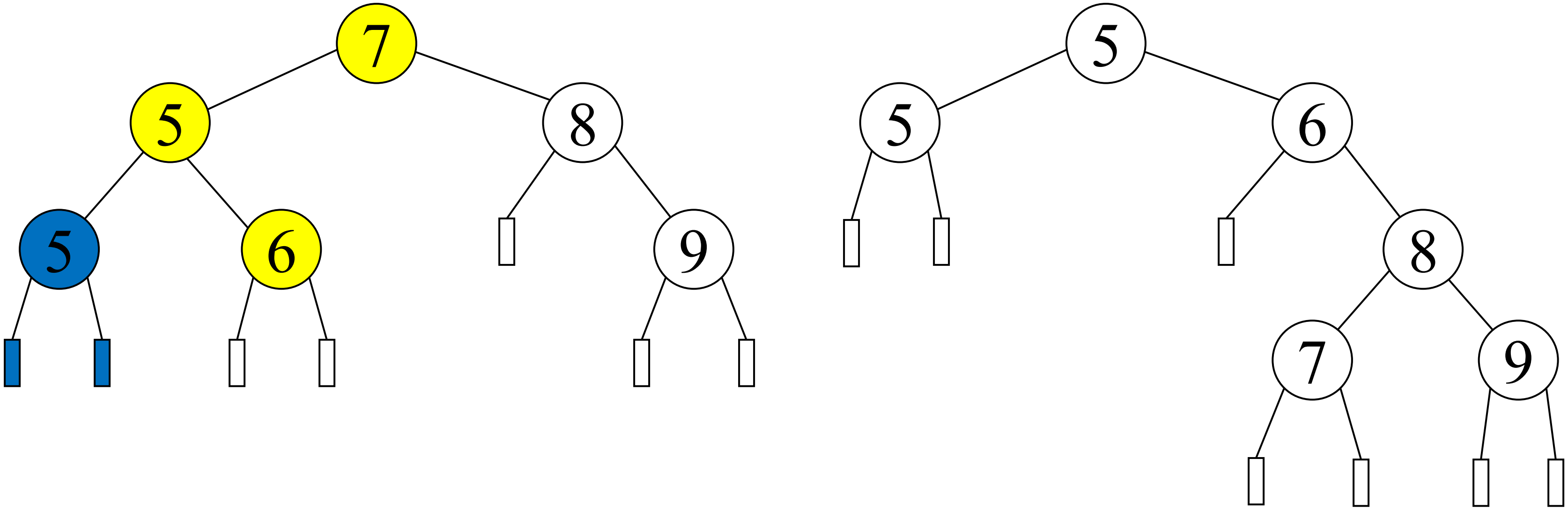
Inorder Tree Walk (example)

5, 5



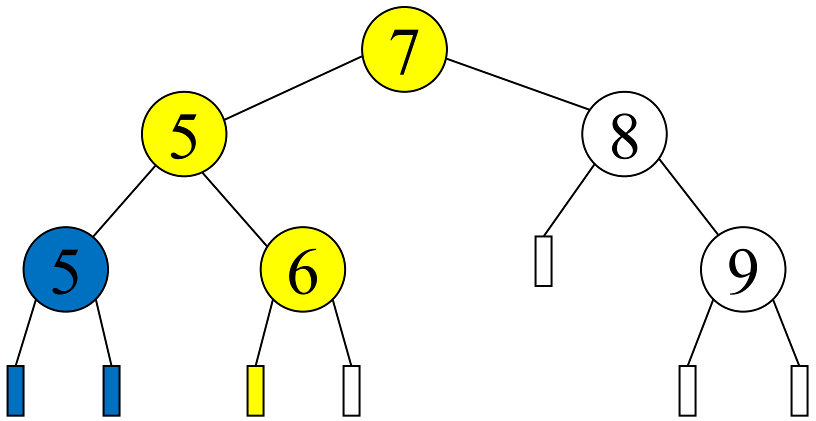
Inorder Tree Walk (example)

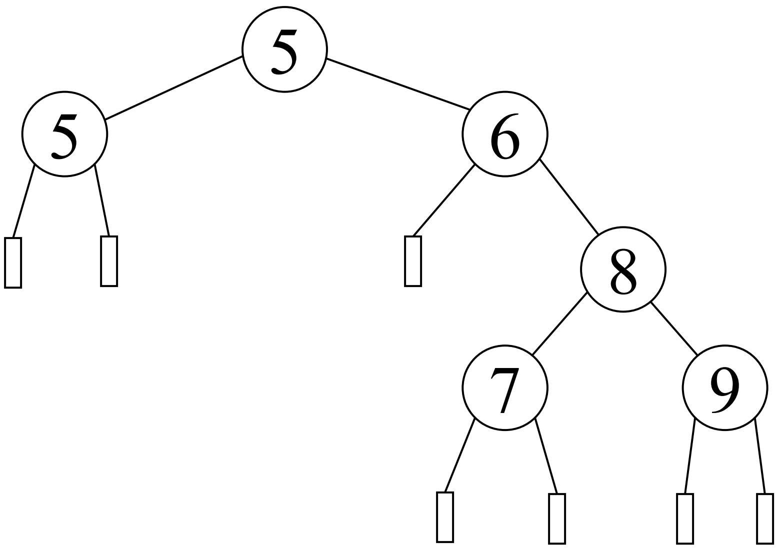
5, 5



Inorder Tree Walk (example)

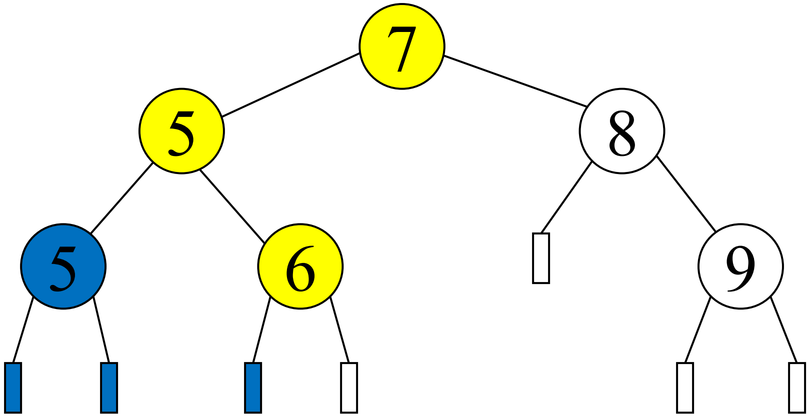
5, 5

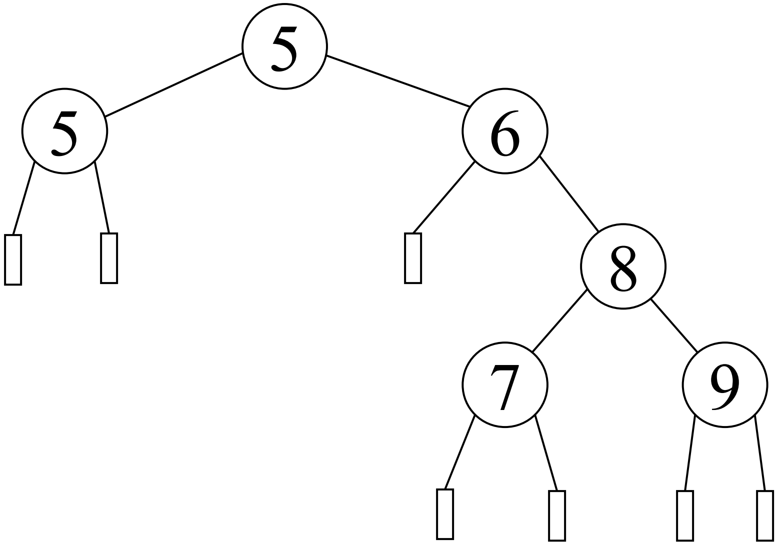




Inorder Tree Walk (example)

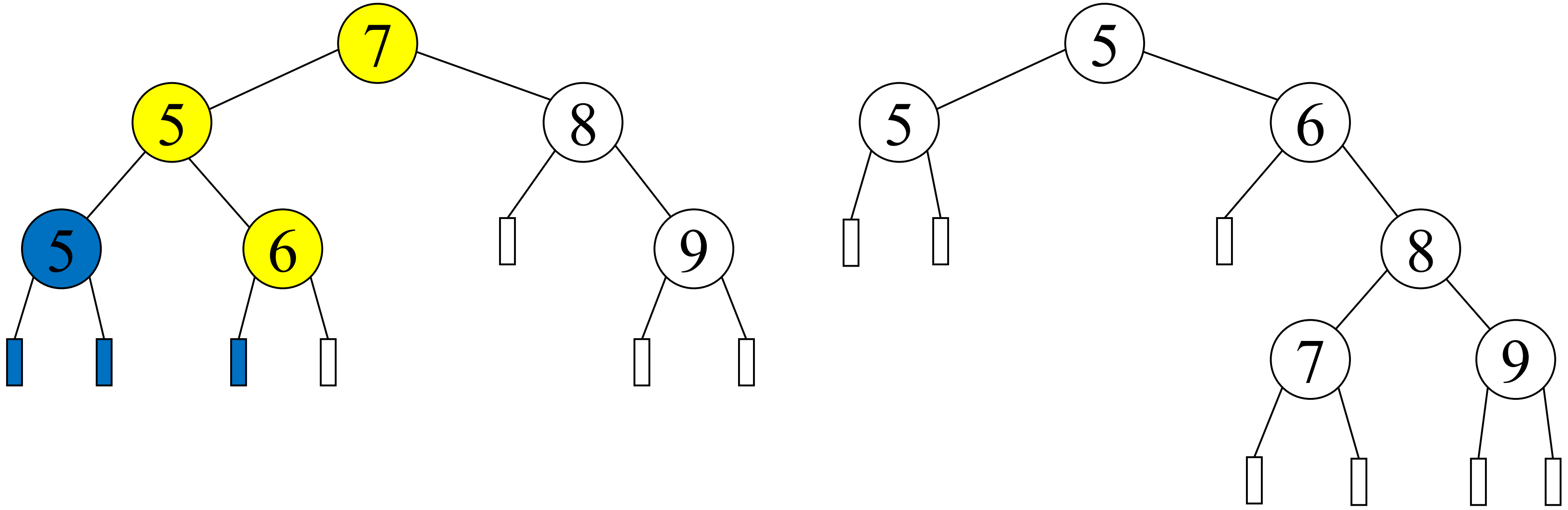
5, 5





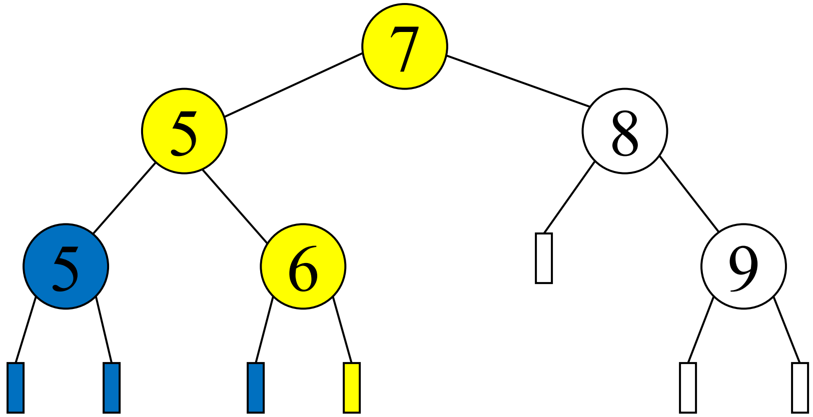
Inorder Tree Walk (example)

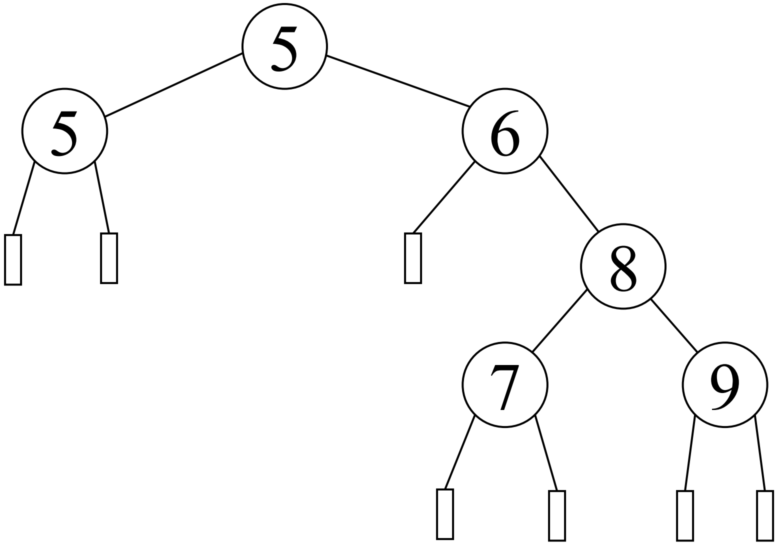
5, 5, 6



Inorder Tree Walk (example)

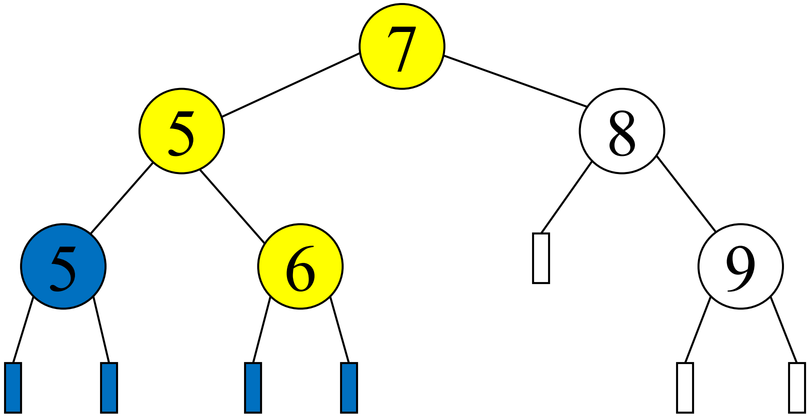
5, 5, 6

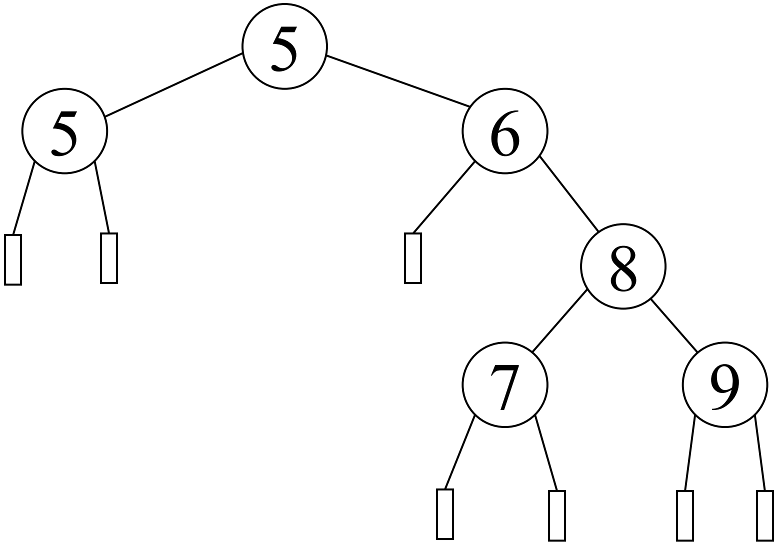




Inorder Tree Walk (example)

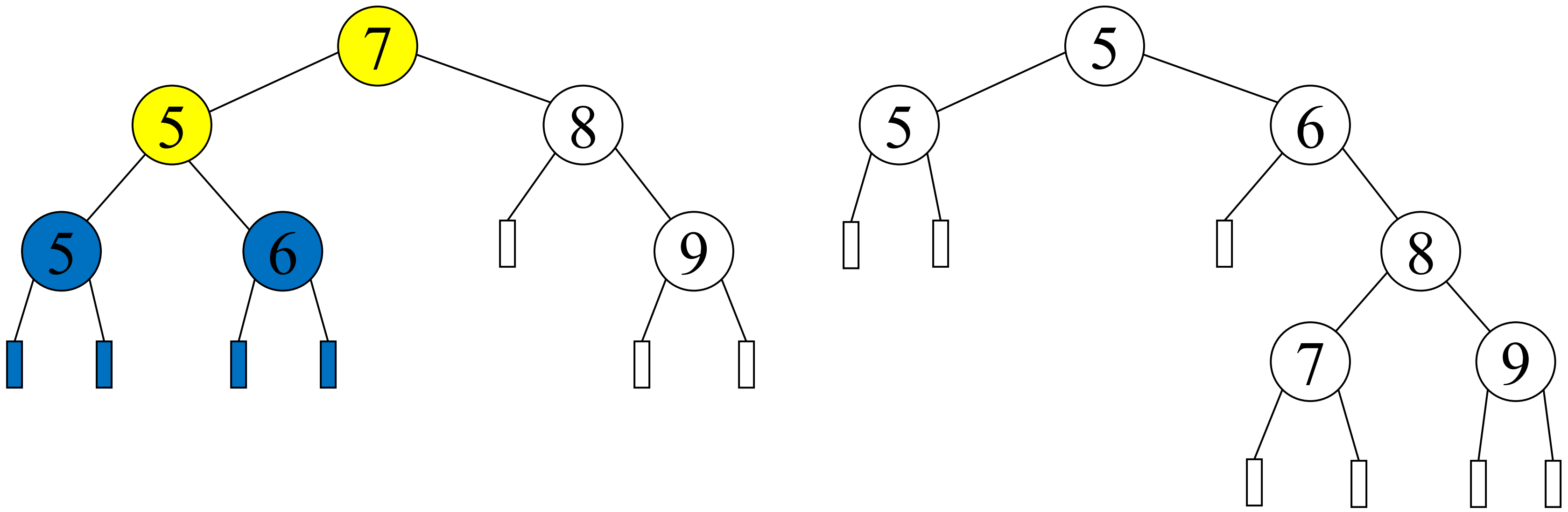
5, 5, 6





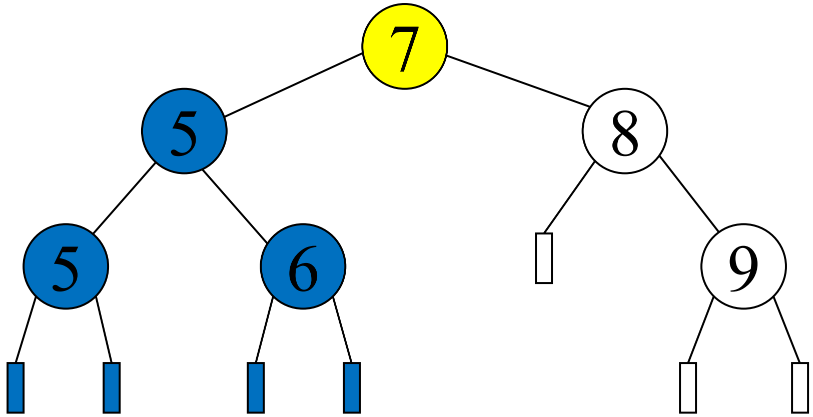
Inorder Tree Walk (example)

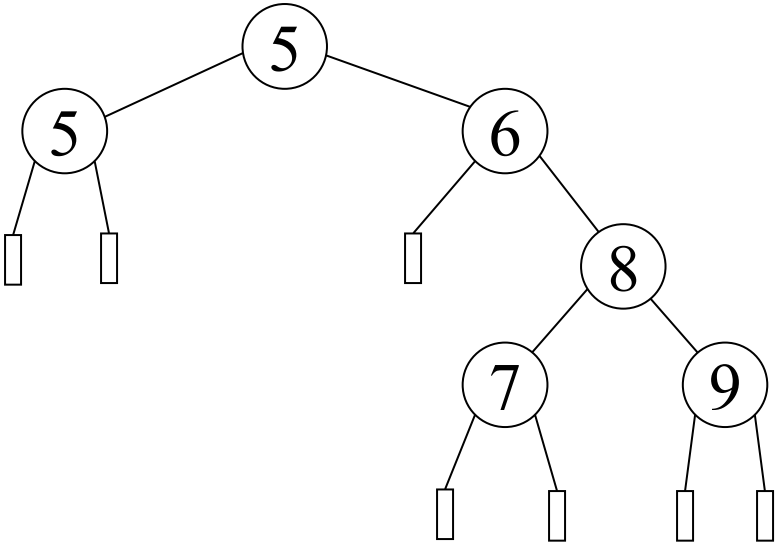
5, 5, 6



Inorder Tree Walk (example)

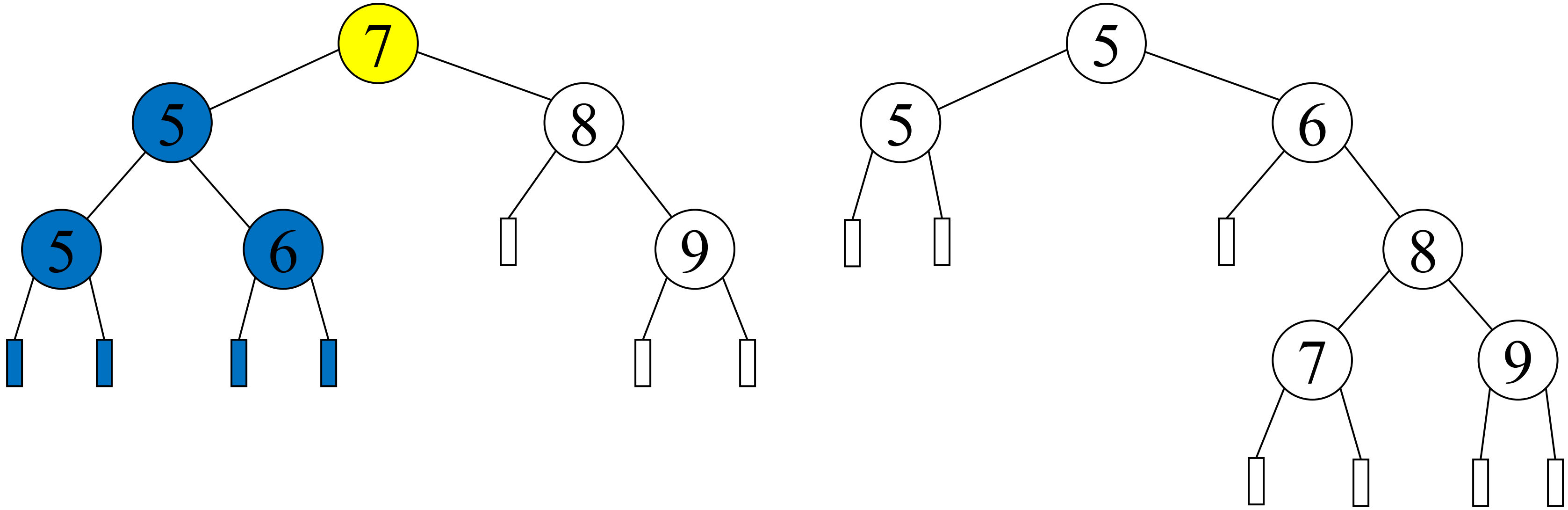
5, 5, 6





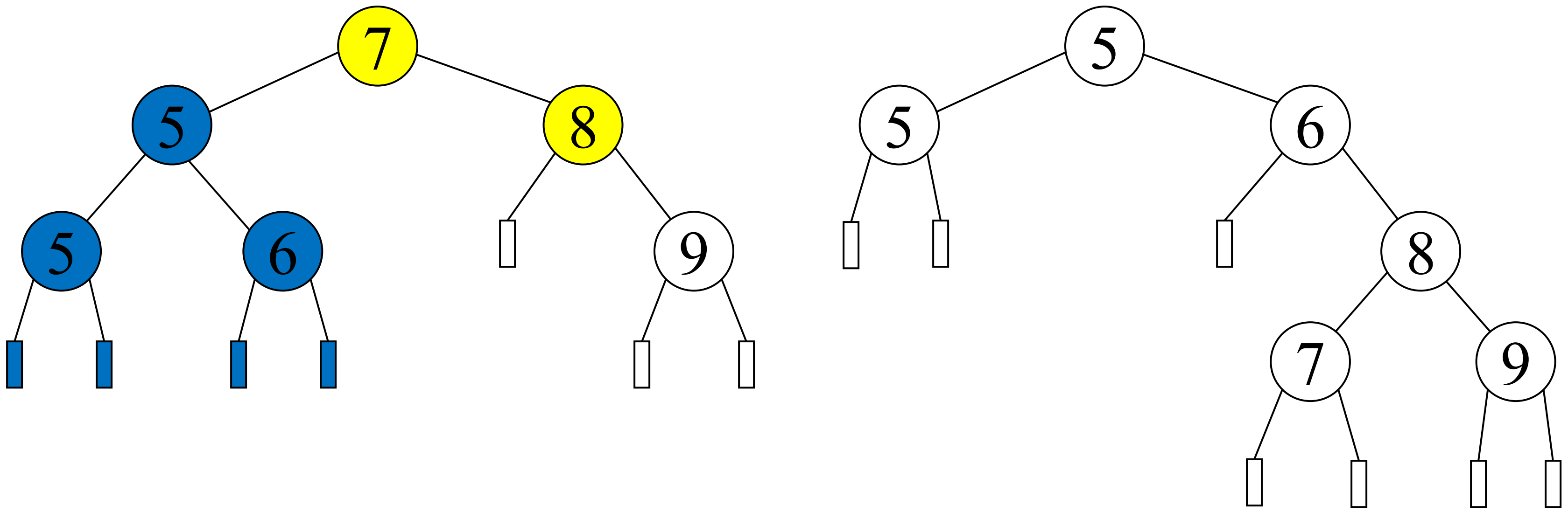
Inorder Tree Walk (example)

5, 5, 6, 7



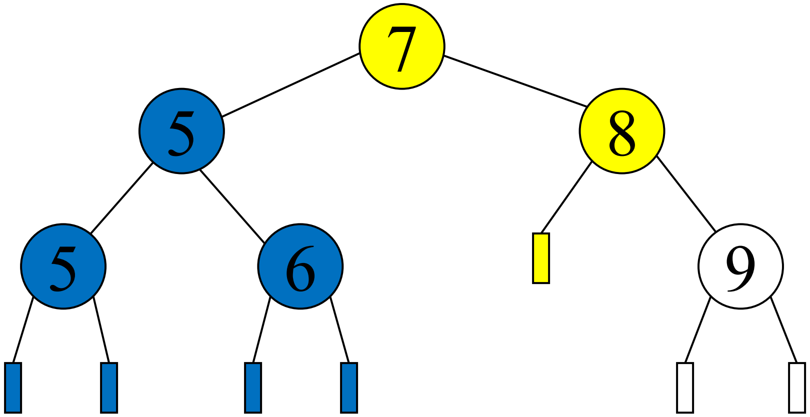
Inorder Tree Walk (example)

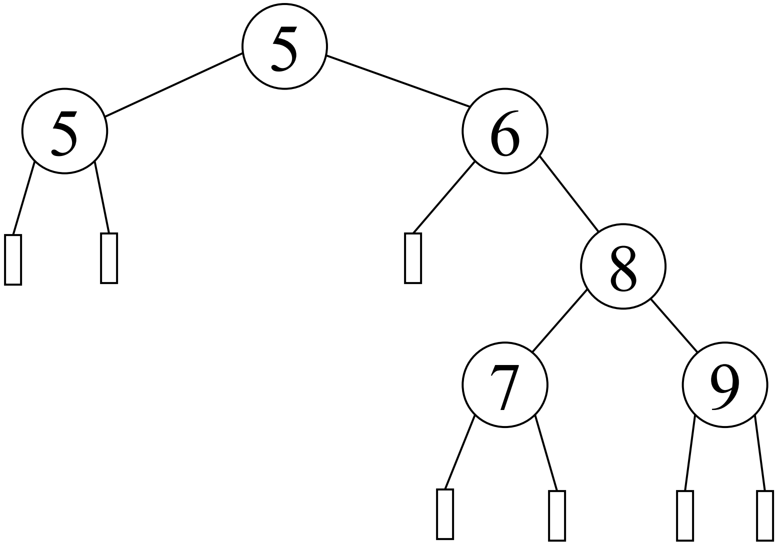
5, 5, 6, 7



Inorder Tree Walk (example)

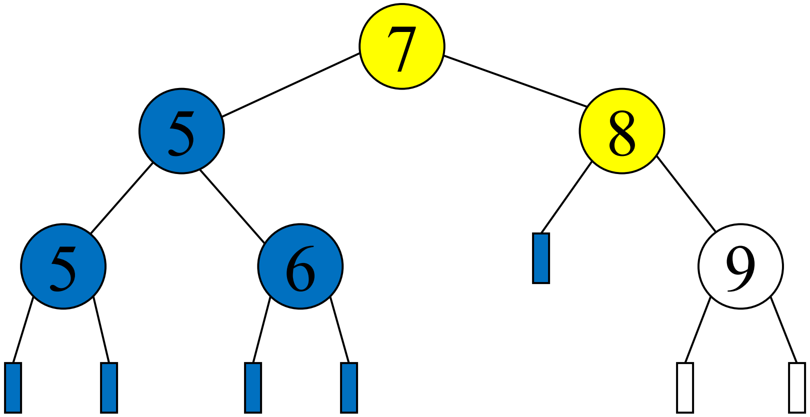
5, 5, 6, 7

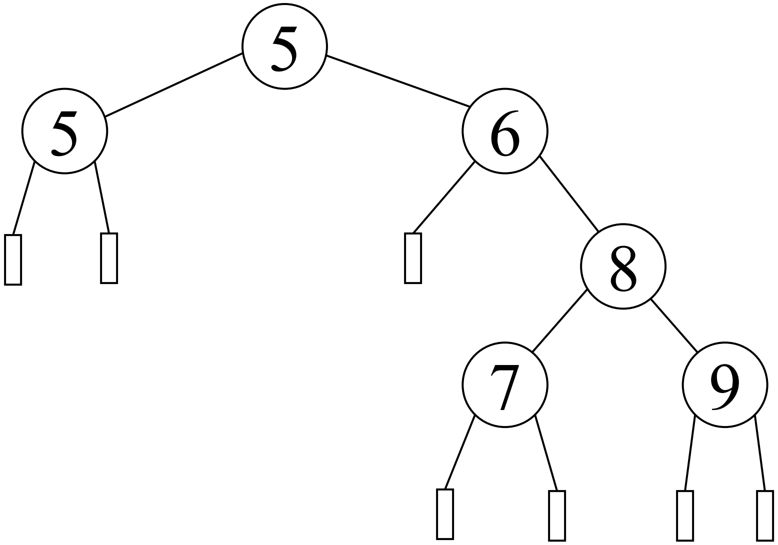




Inorder Tree Walk (example)

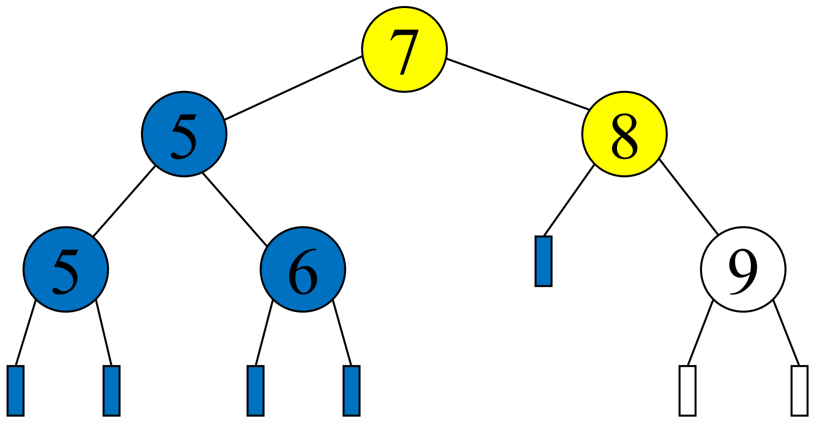
5, 5, 6, 7

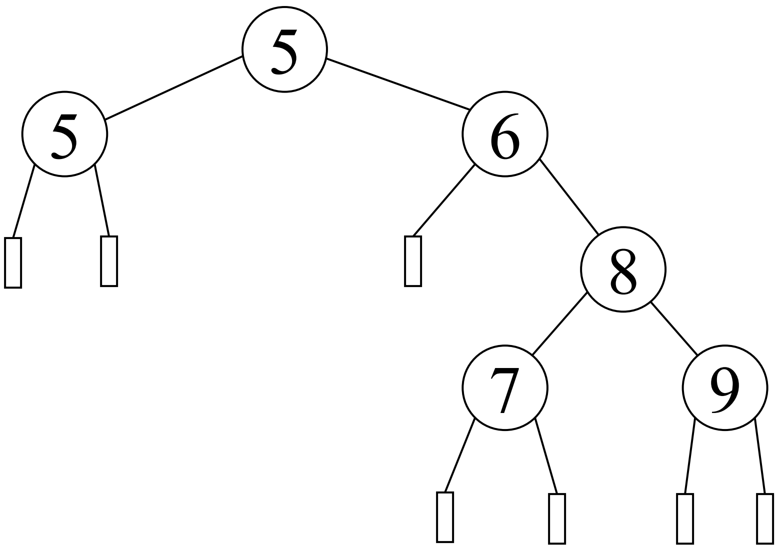




Inorder Tree Walk (example)

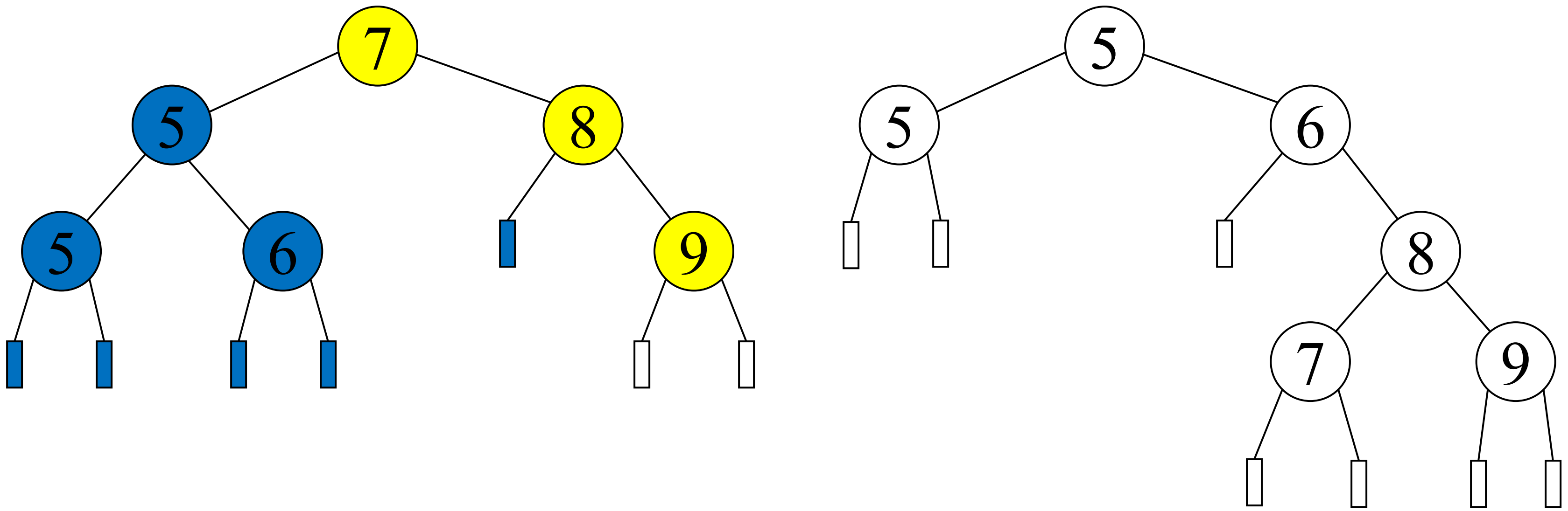
5, 5, 6, 7, 8





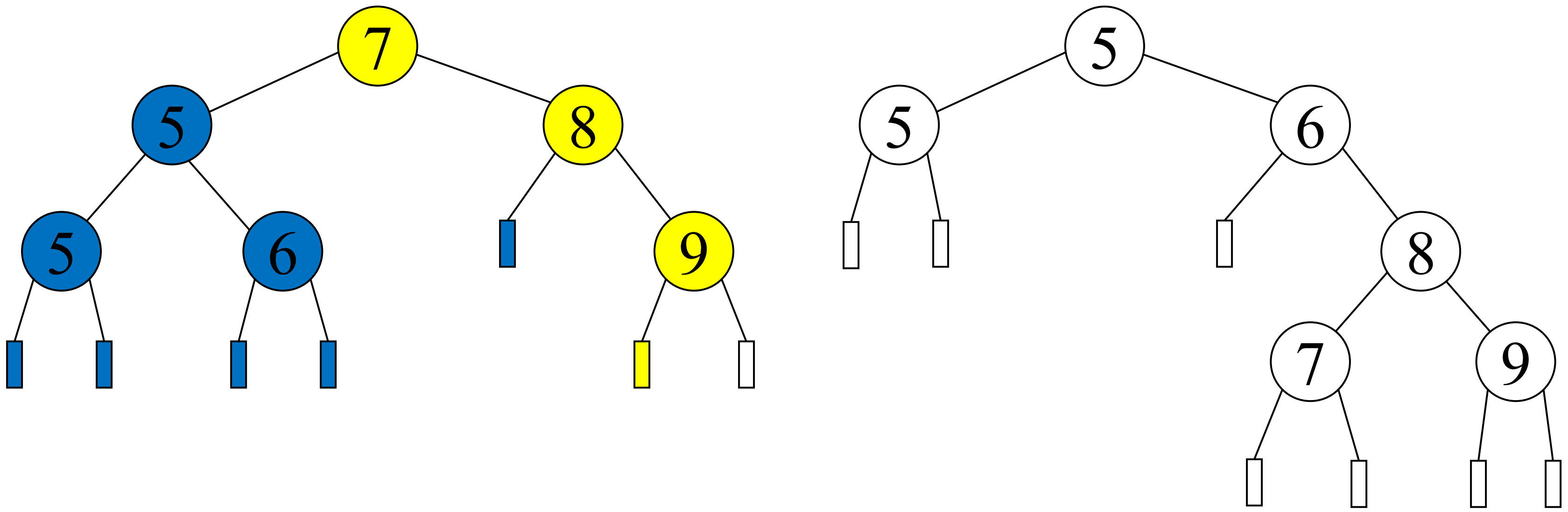
Inorder Tree Walk (example)

5, 5, 6, 7, 8



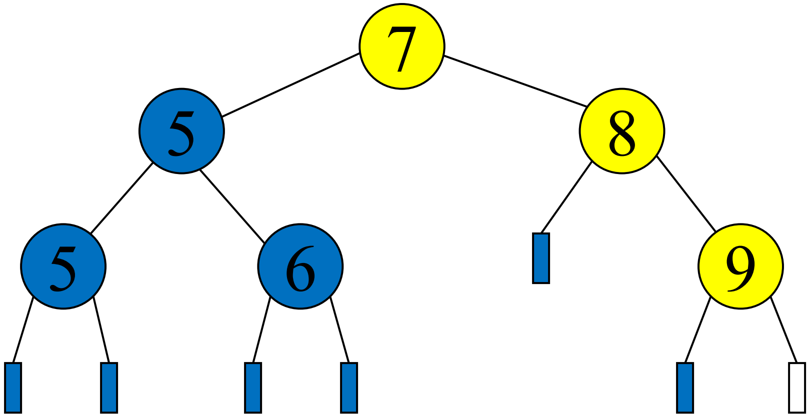
Inorder Tree Walk (example)

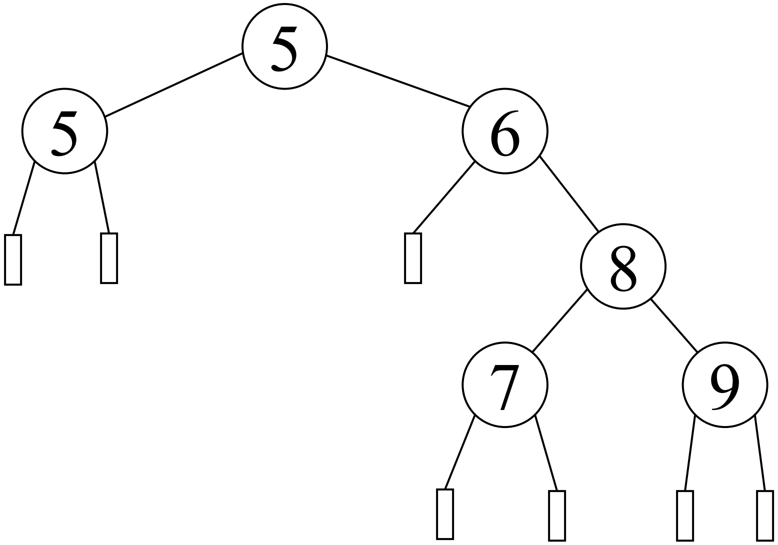
5, 5, 6, 7, 8



Inorder Tree Walk (example)

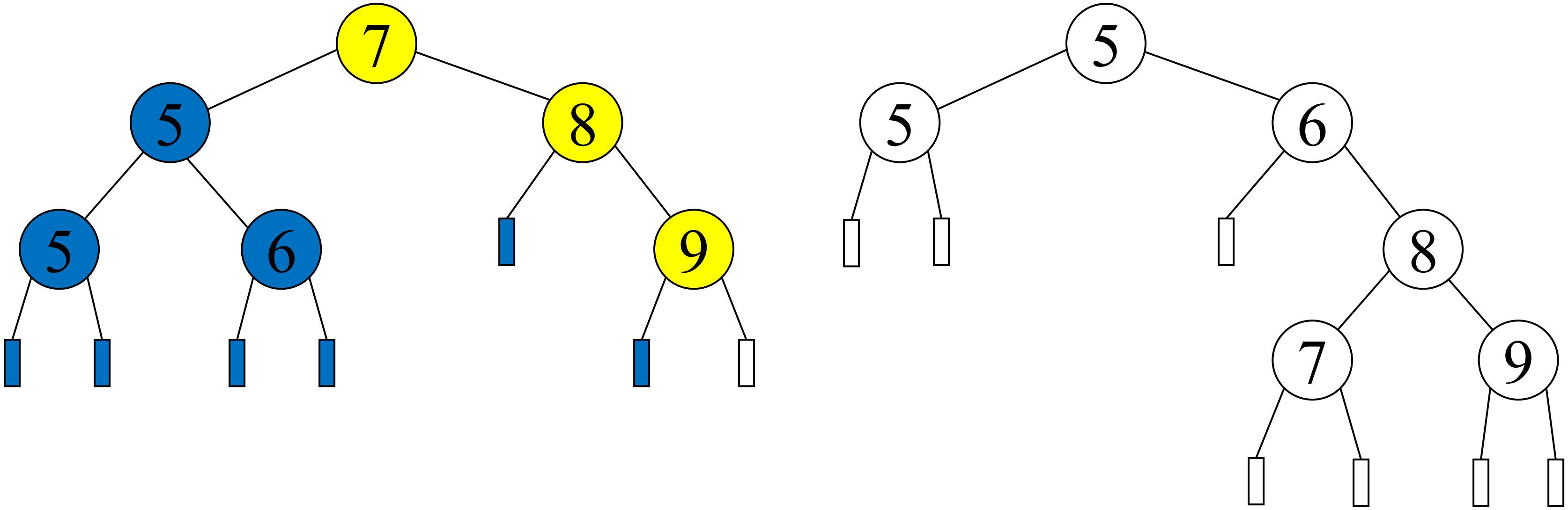
5, 5, 6, 7, 8





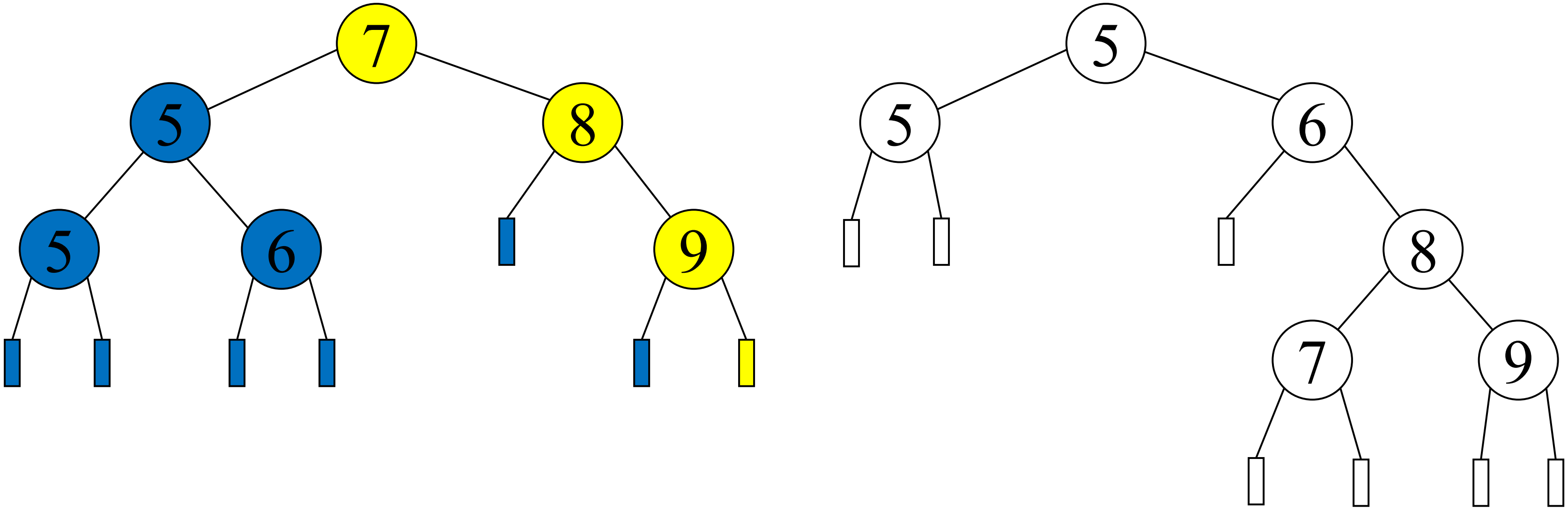
Inorder Tree Walk (example)

5, 5, 6, 7, 8, 9



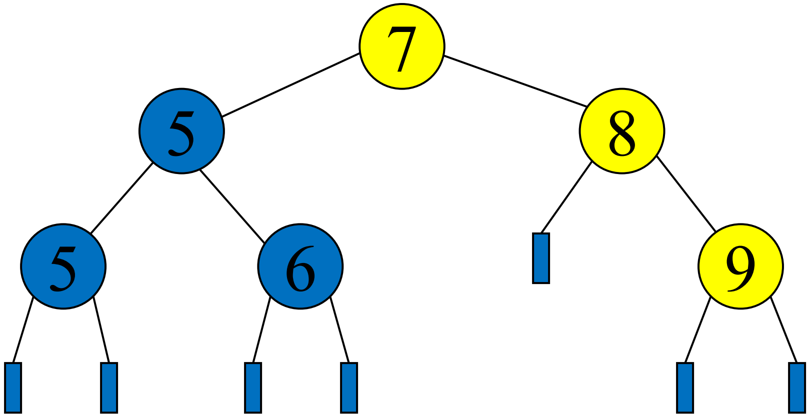
Inorder Tree Walk (example)

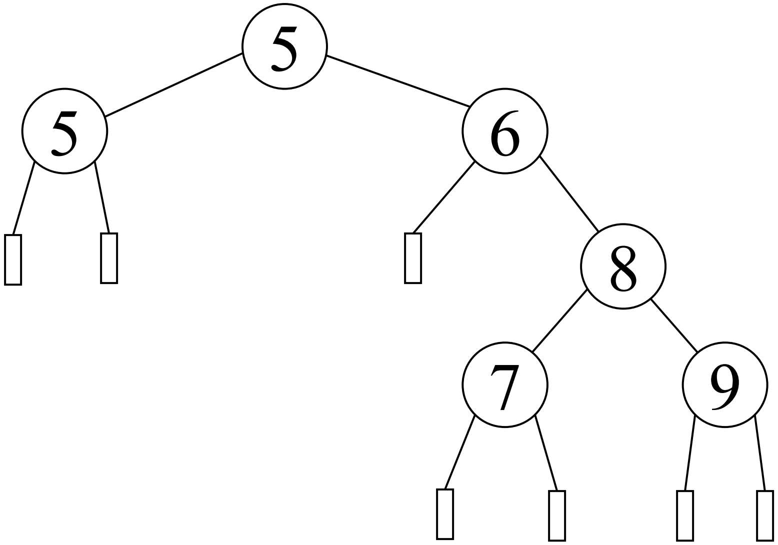
5, 5, 6, 7, 8, 9



Inorder Tree Walk (example)

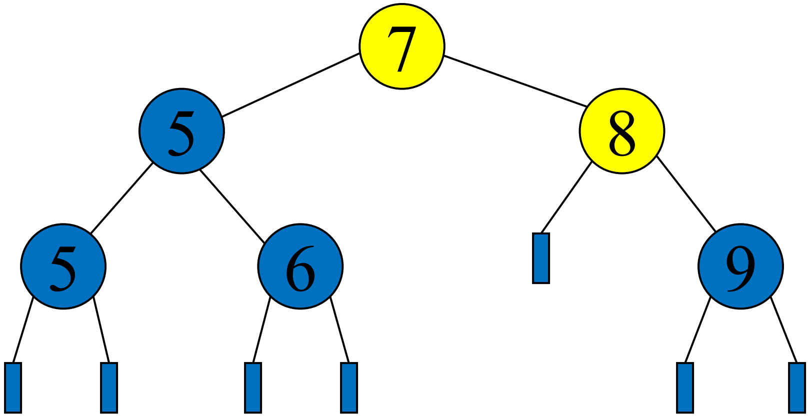
5, 5, 6, 7, 8, 9

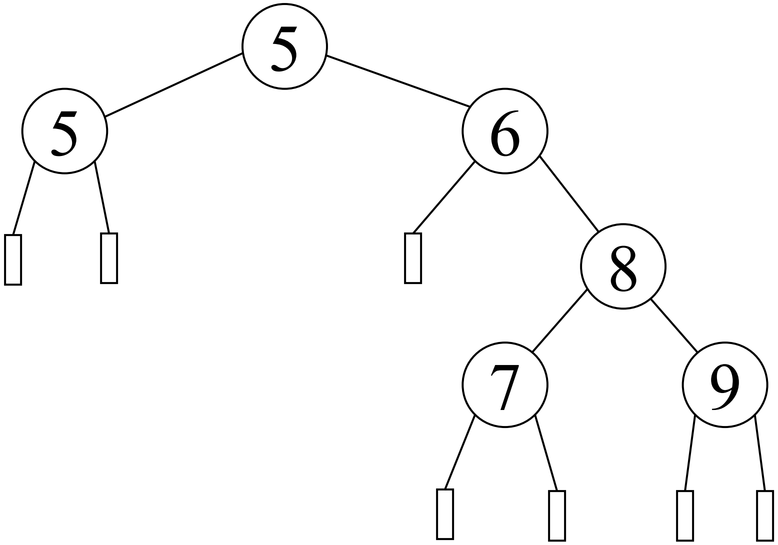




Inorder Tree Walk (example)

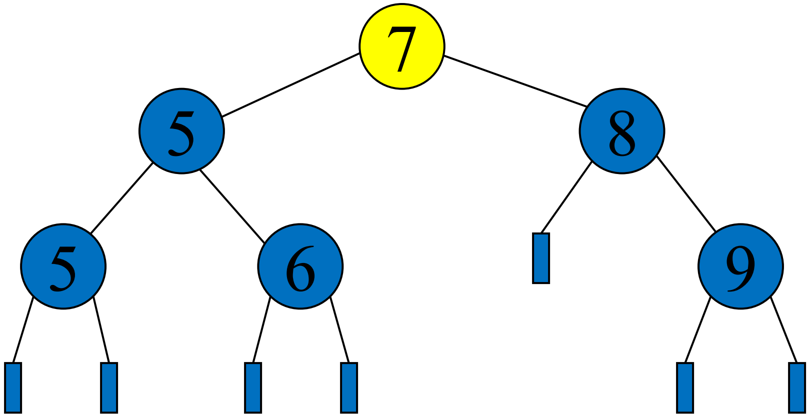
5, 5, 6, 7, 8, 9

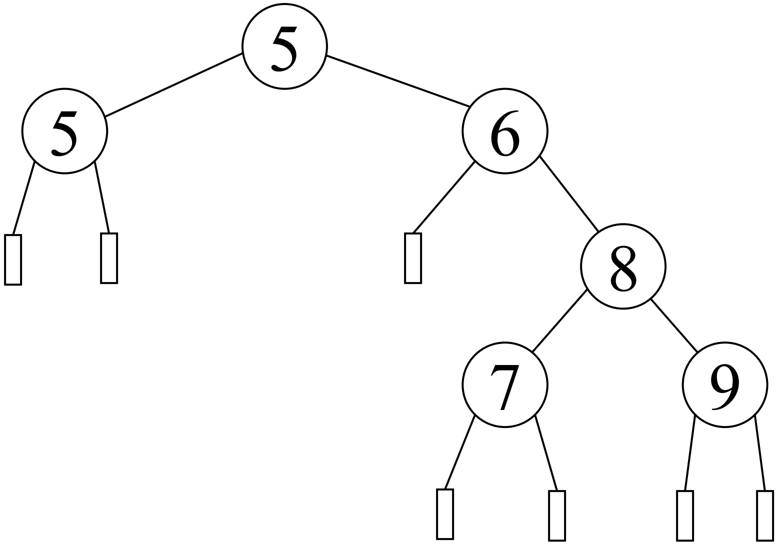




Inorder Tree Walk (example)

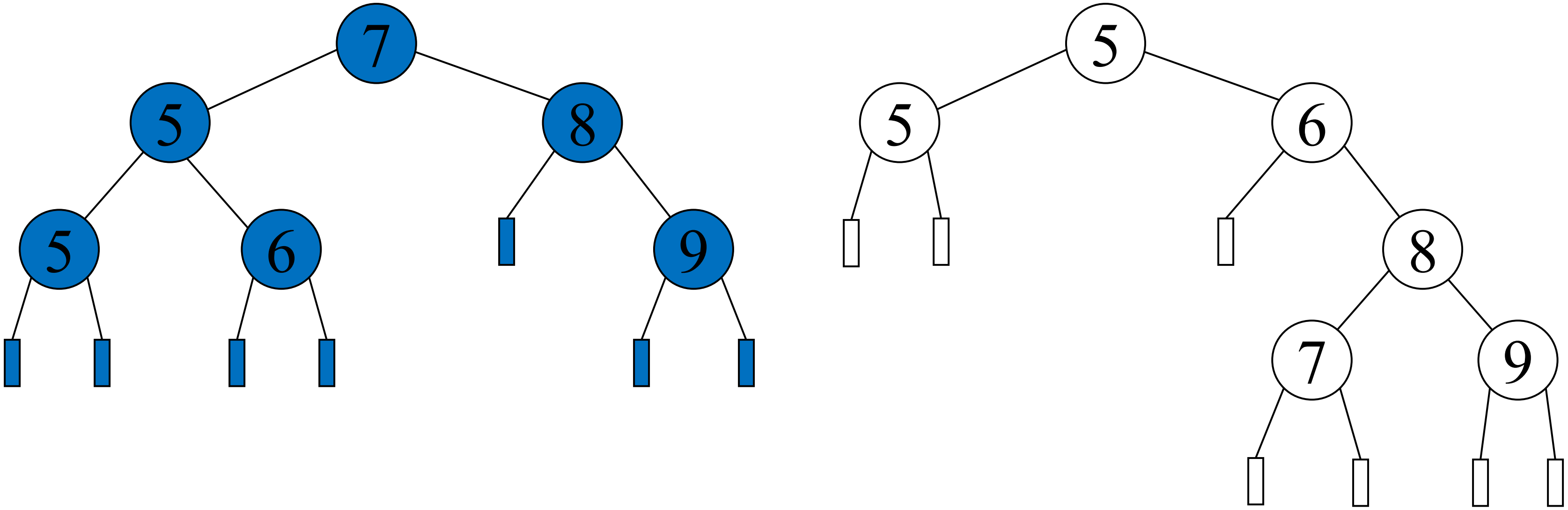
5, 5, 6, 7, 8, 9





Inorder Tree Walk (example)

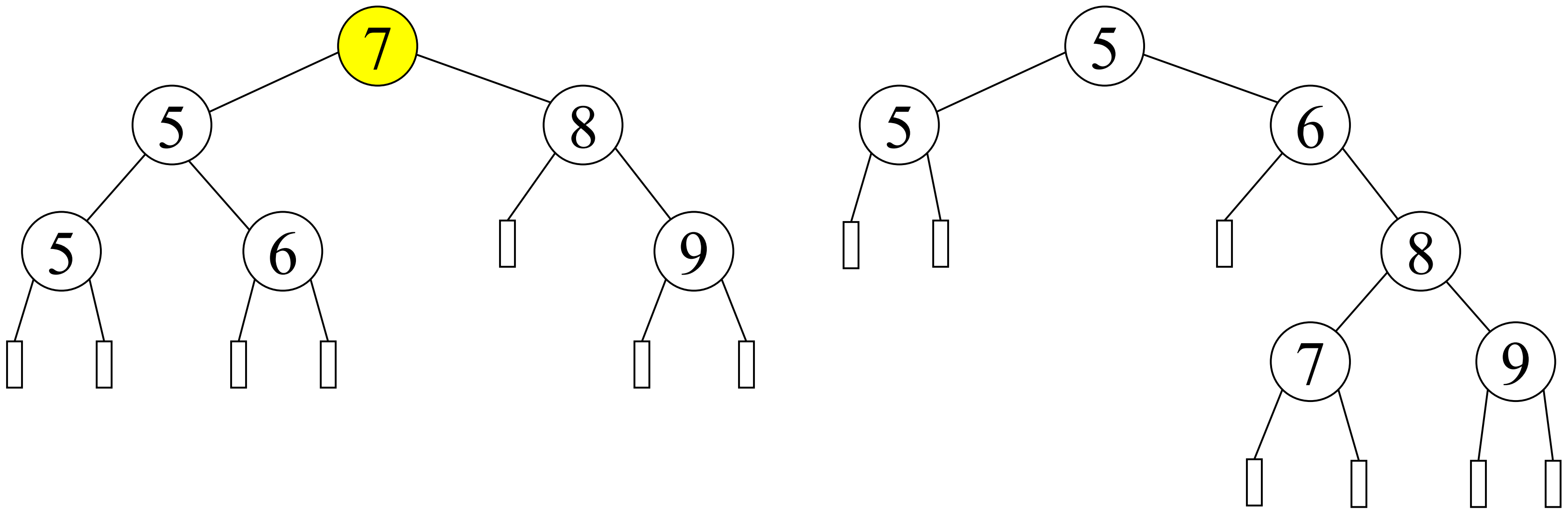
5, 5, 6, 7, 8, 9



Tree Walks (Preorder Walk)

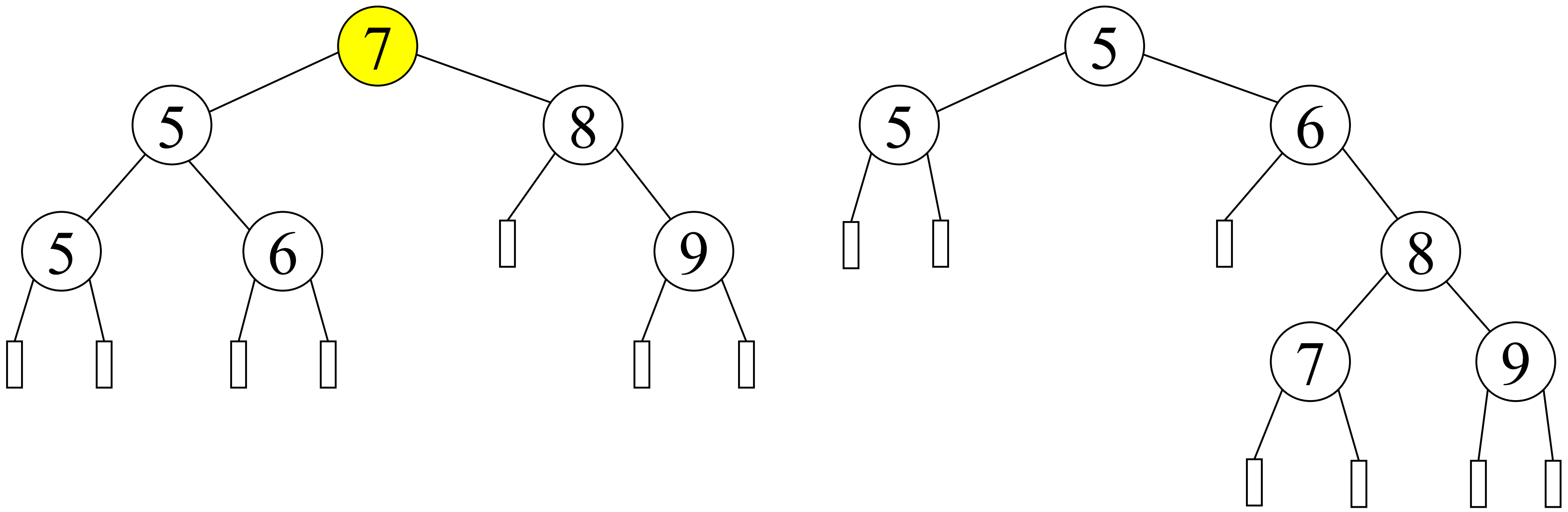
```
Preorder-Tree-Walk(x)
if $x \neq$ null then
    print(x.key)                          //key[x]
    Preorder-Tree-Walk(x.left)            //left[x]
    Preorder-Tree-Walk(x.right)           //right[x]
```

Preorder Tree Walk (example)



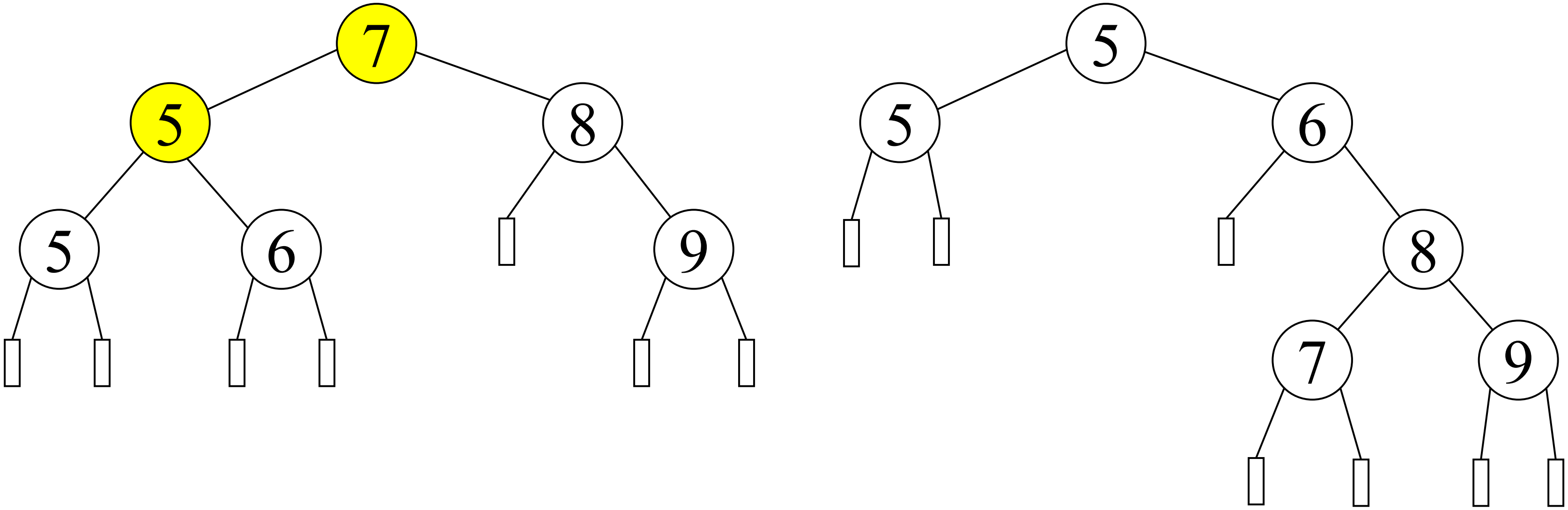
Preorder Tree Walk (example)

7



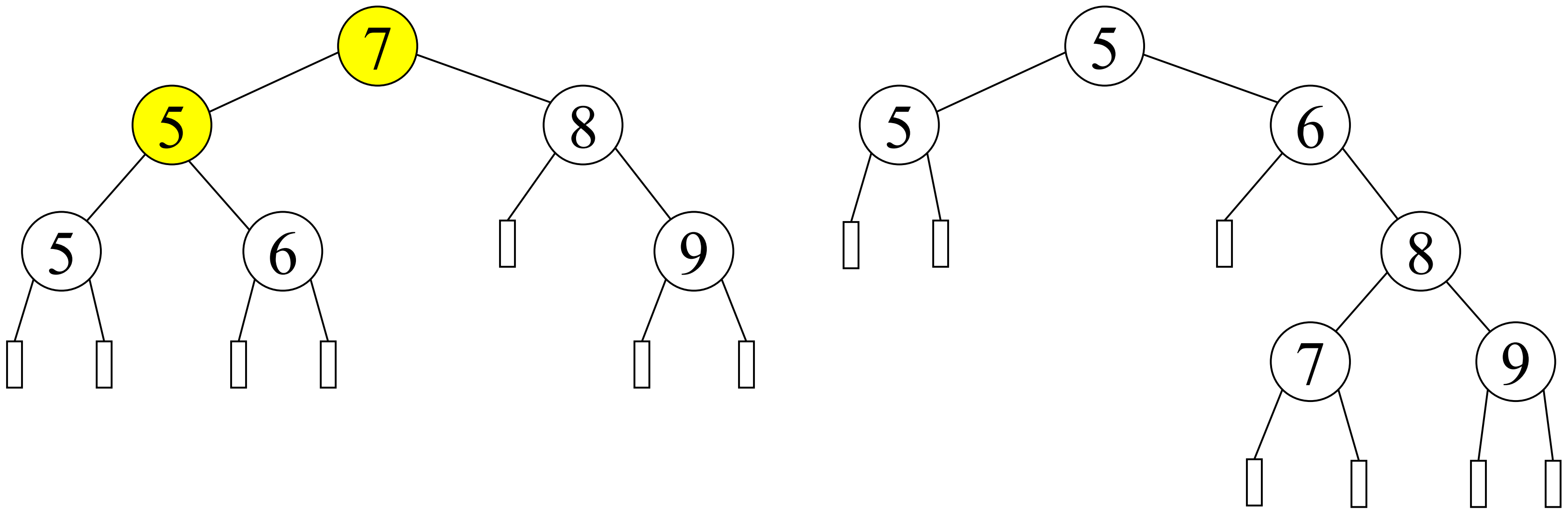
Preorder Tree Walk (example)

7



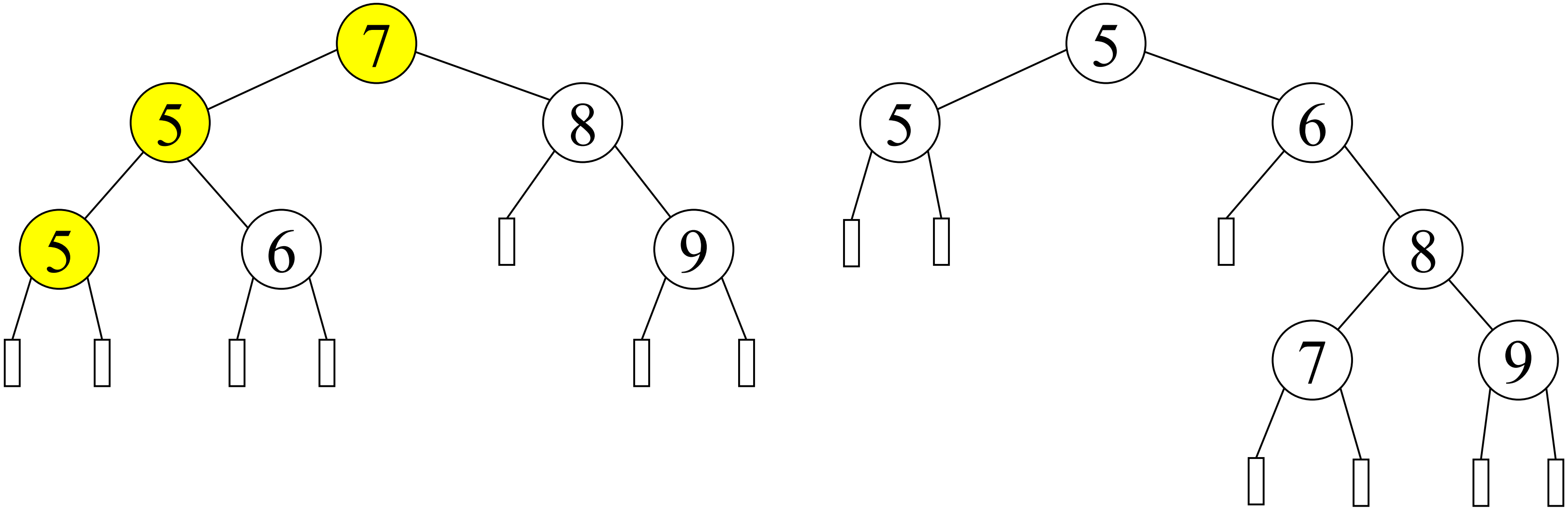
Preorder Tree Walk (example)

7, 5



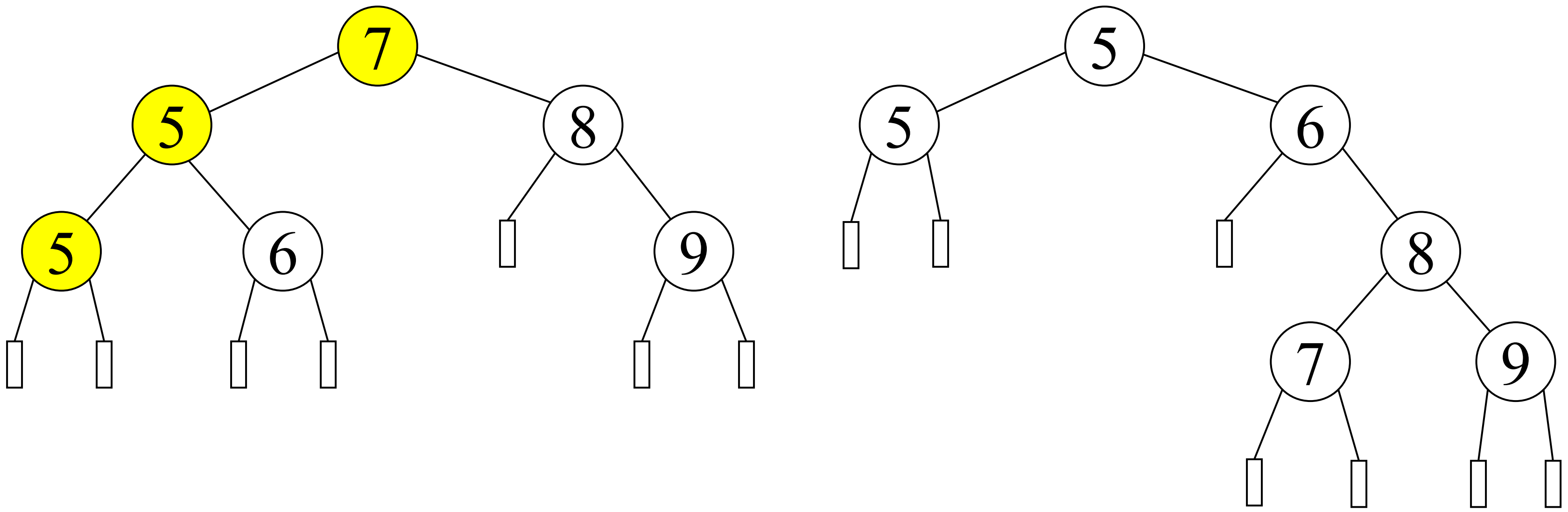
Preorder Tree Walk (example)

7, 5



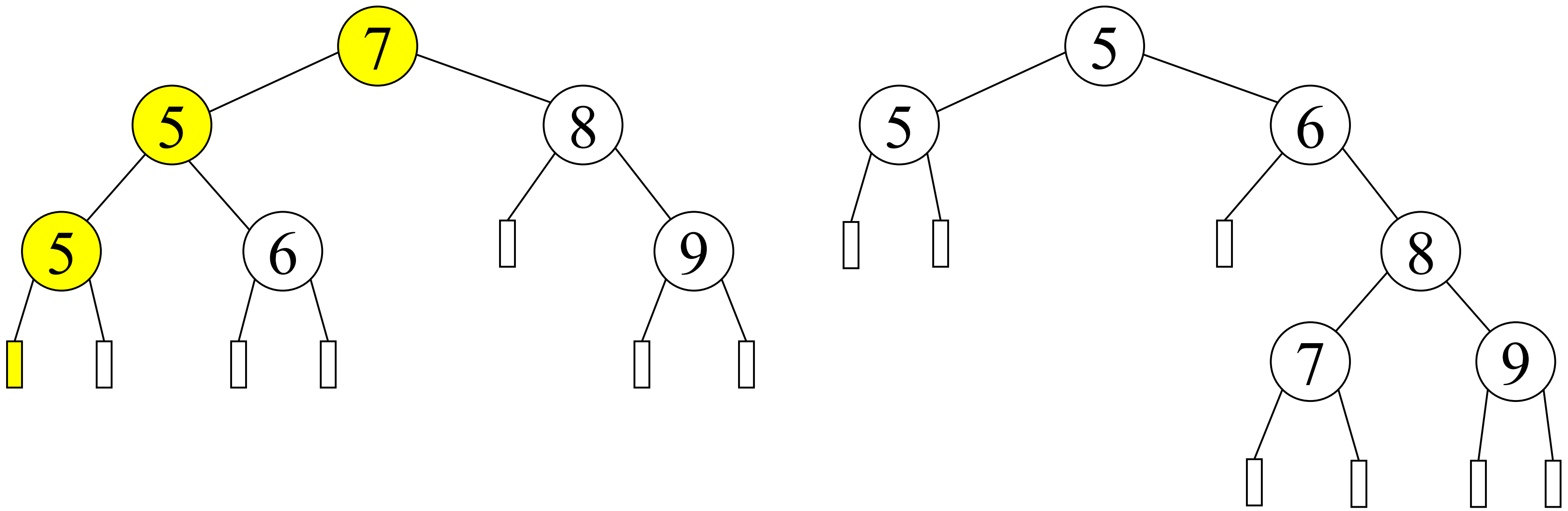
Preorder Tree Walk (example)

7, 5, 5



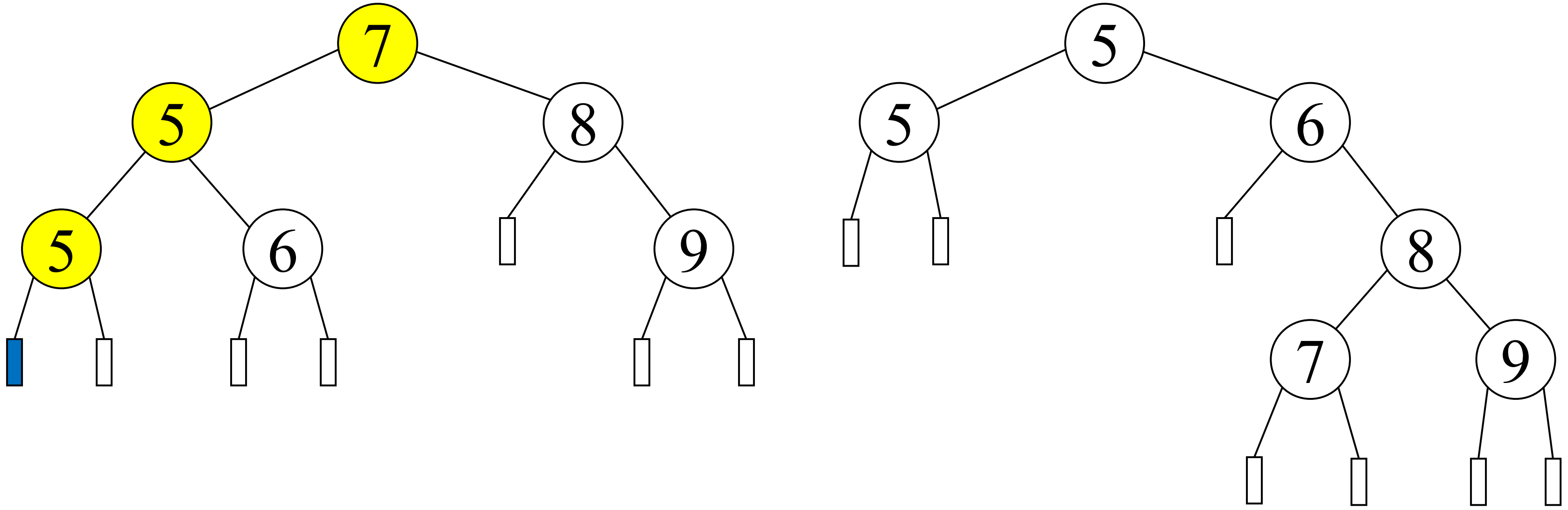
Preorder Tree Walk (example)

7, 5, 5



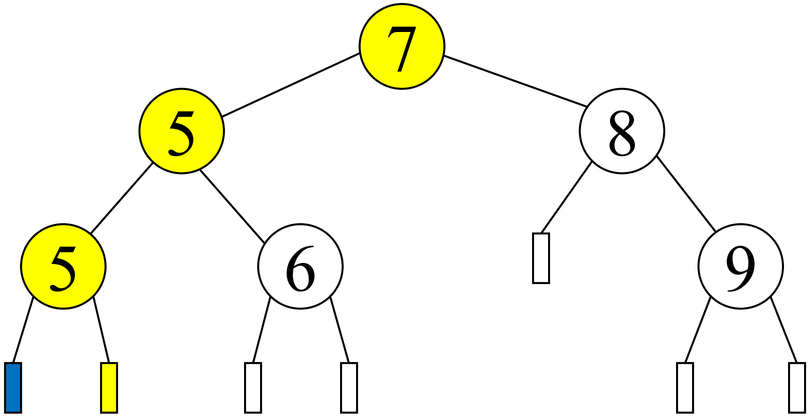
Preorder Tree Walk (example)

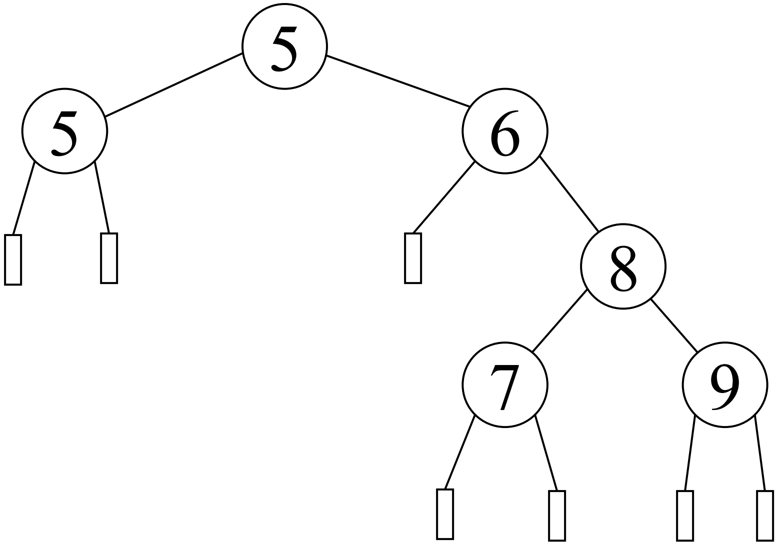
7, 5, 5



Preorder Tree Walk (example)

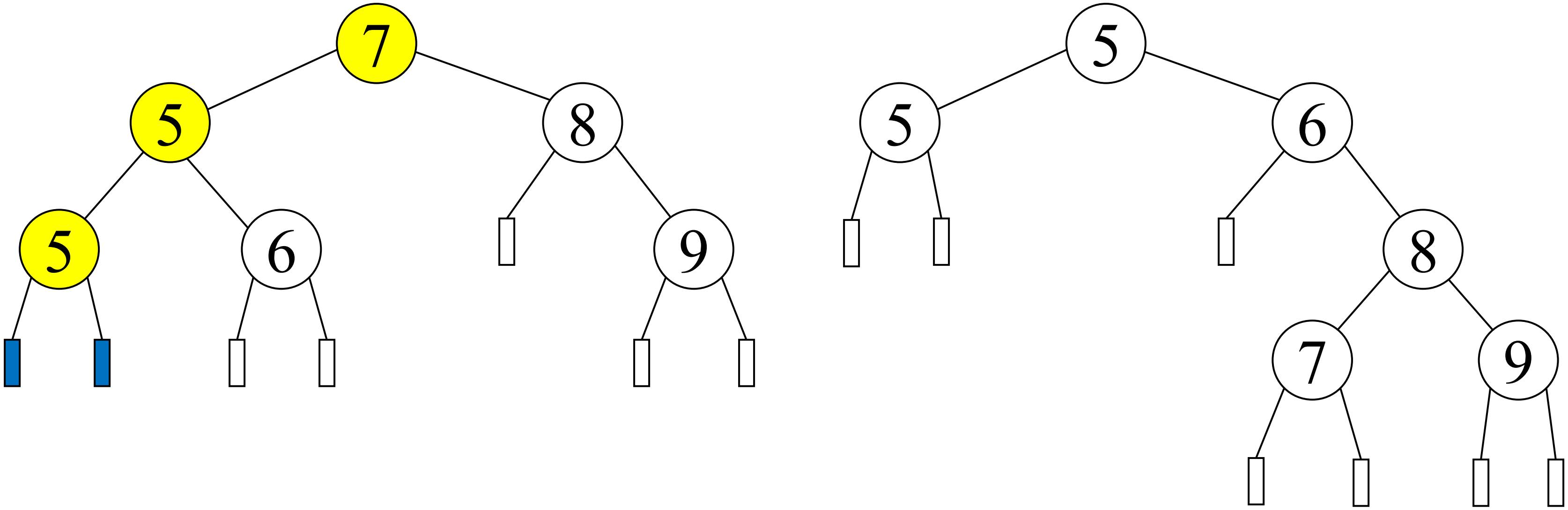
7, 5, 5





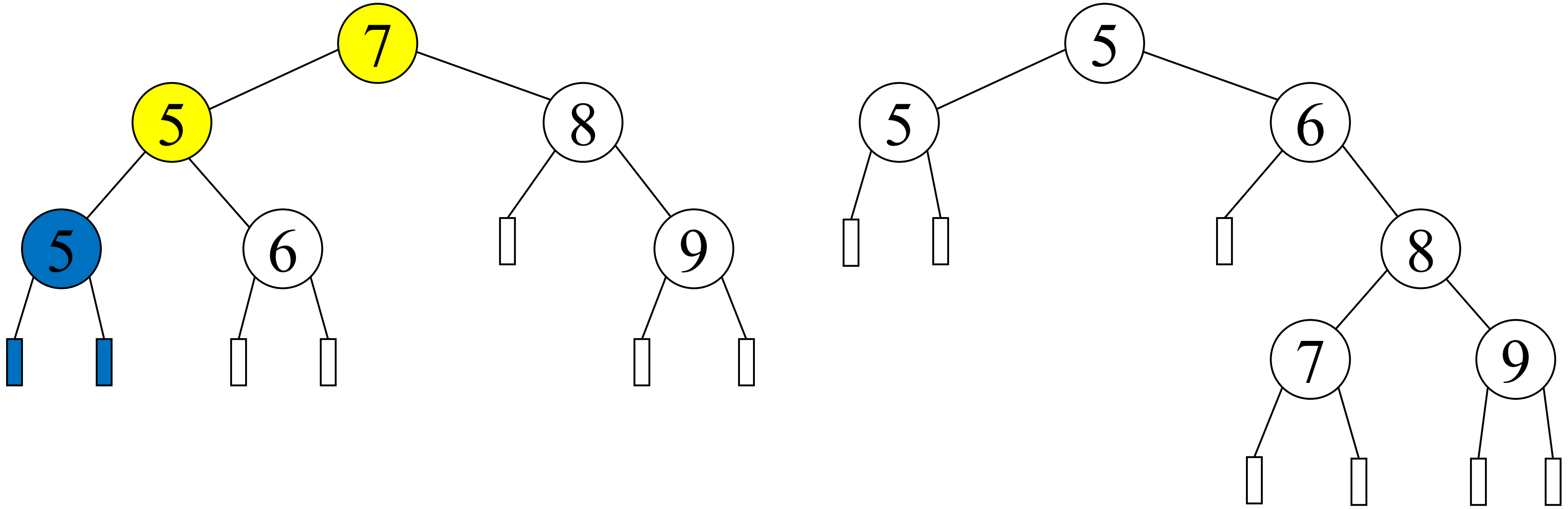
Preorder Tree Walk (example)

7, 5, 5



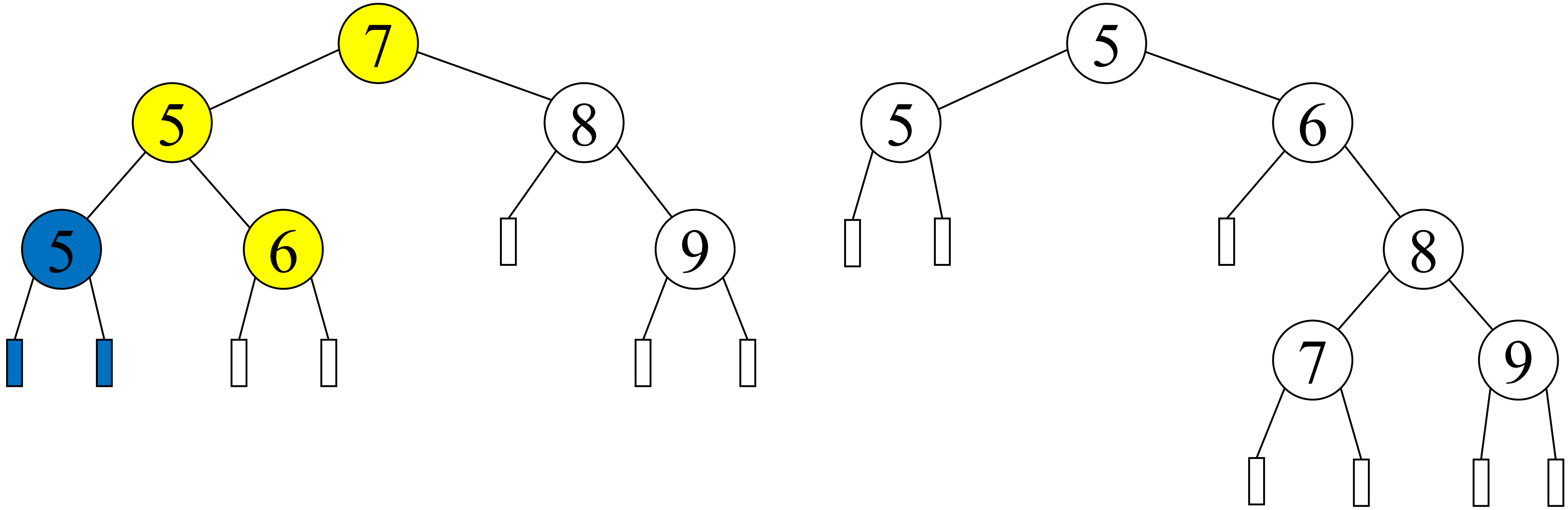
Preorder Tree Walk (example)

7, 5, 5



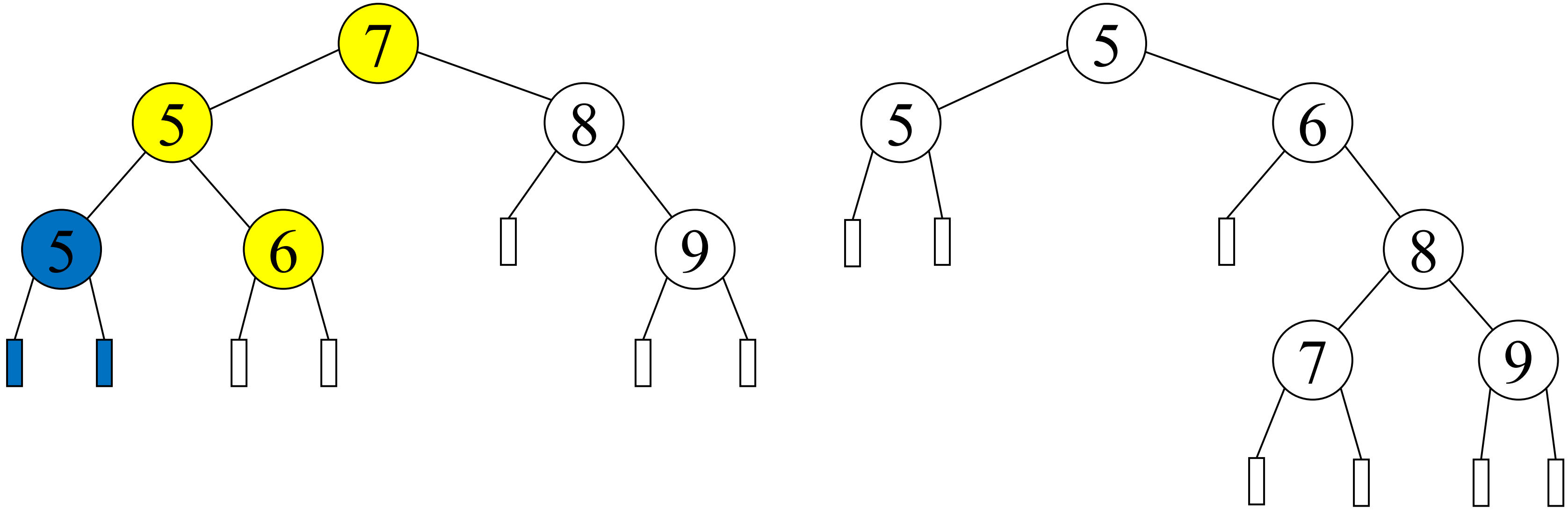
Preorder Tree Walk (example)

7, 5, 5



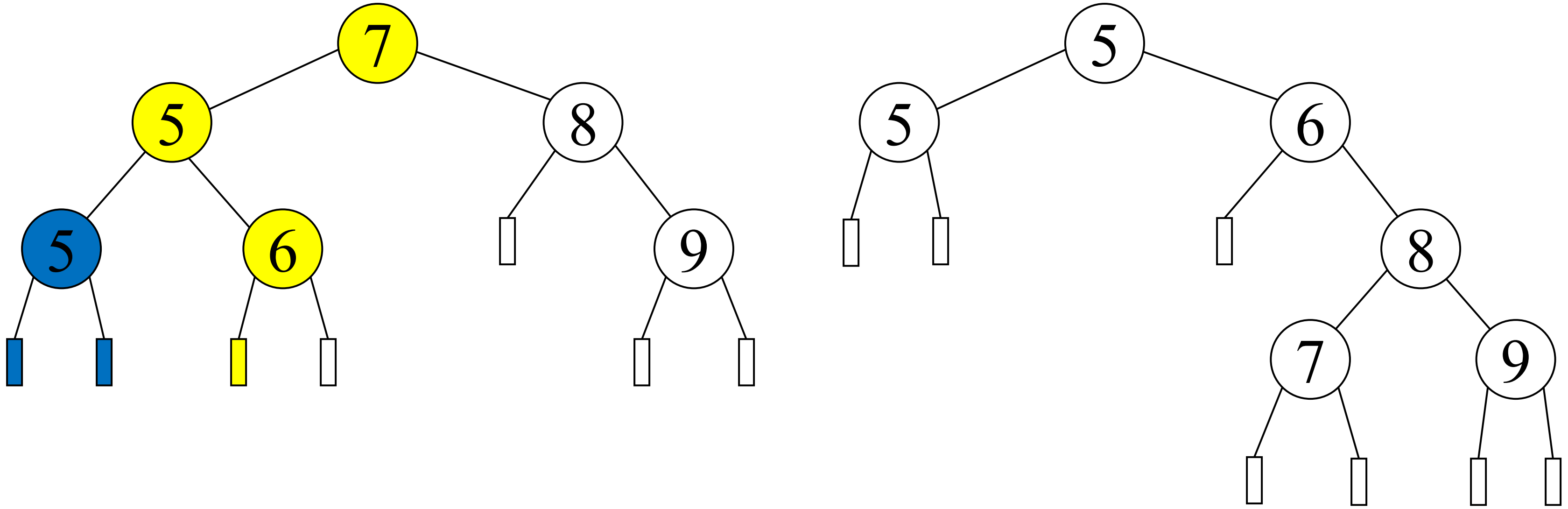
Preorder Tree Walk (example)

7, 5, 5, 6



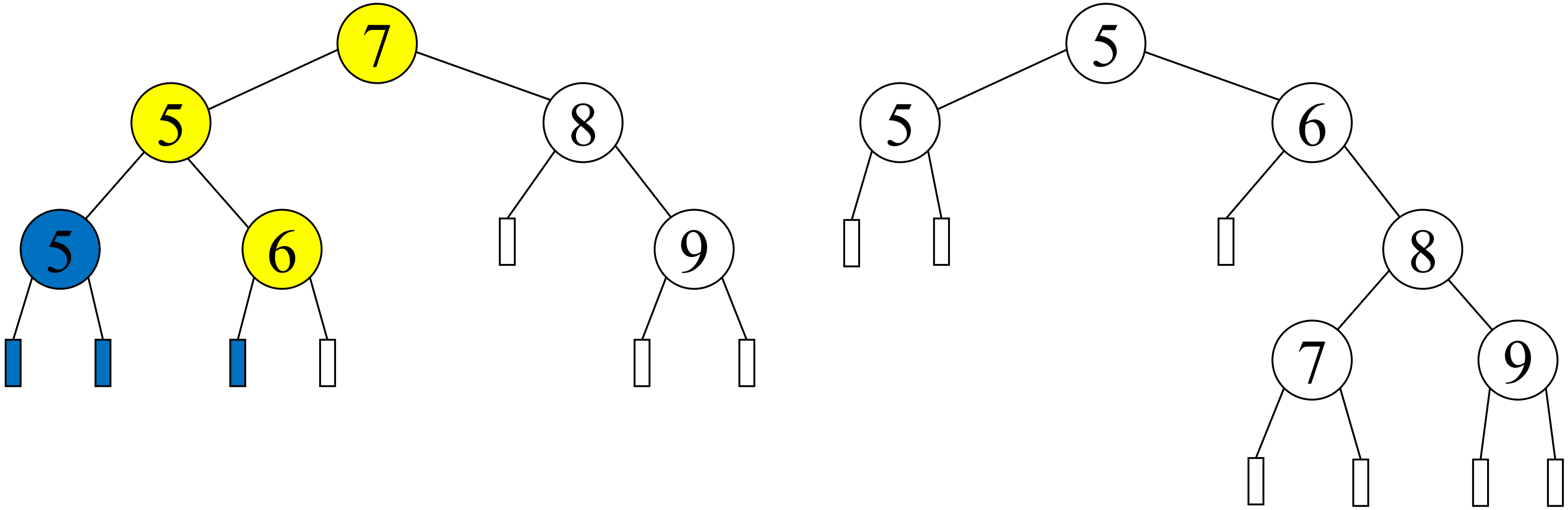
Preorder Tree Walk (example)

7, 5, 5, 6



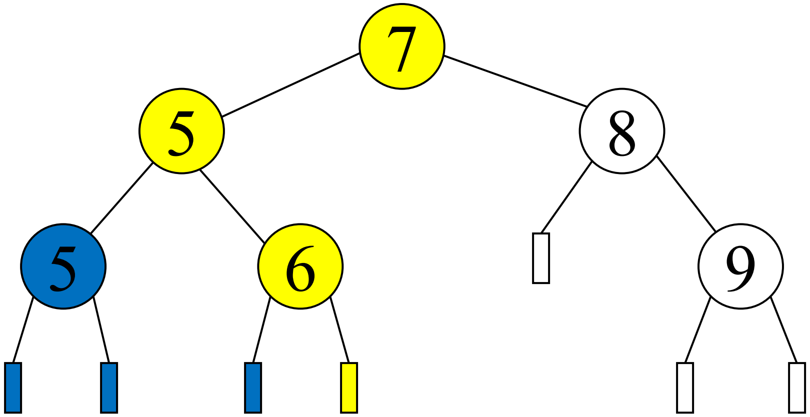
Preorder Tree Walk (example)

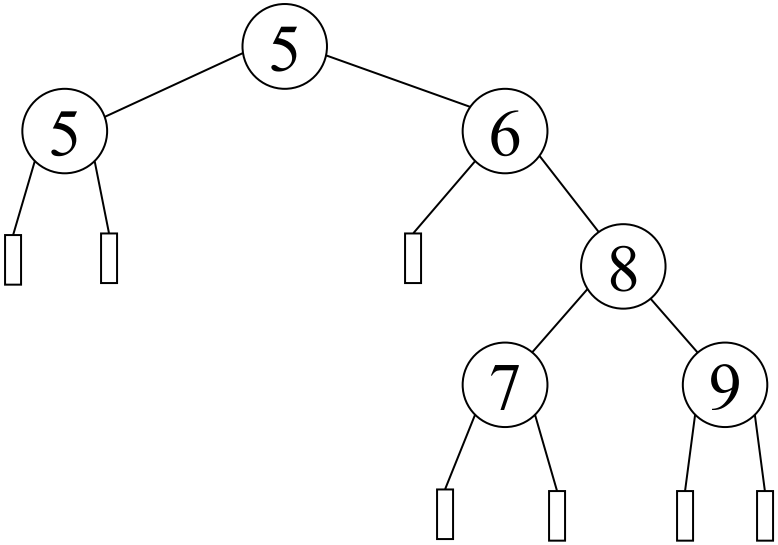
7, 5, 5, 6



Preorder Tree Walk (example)

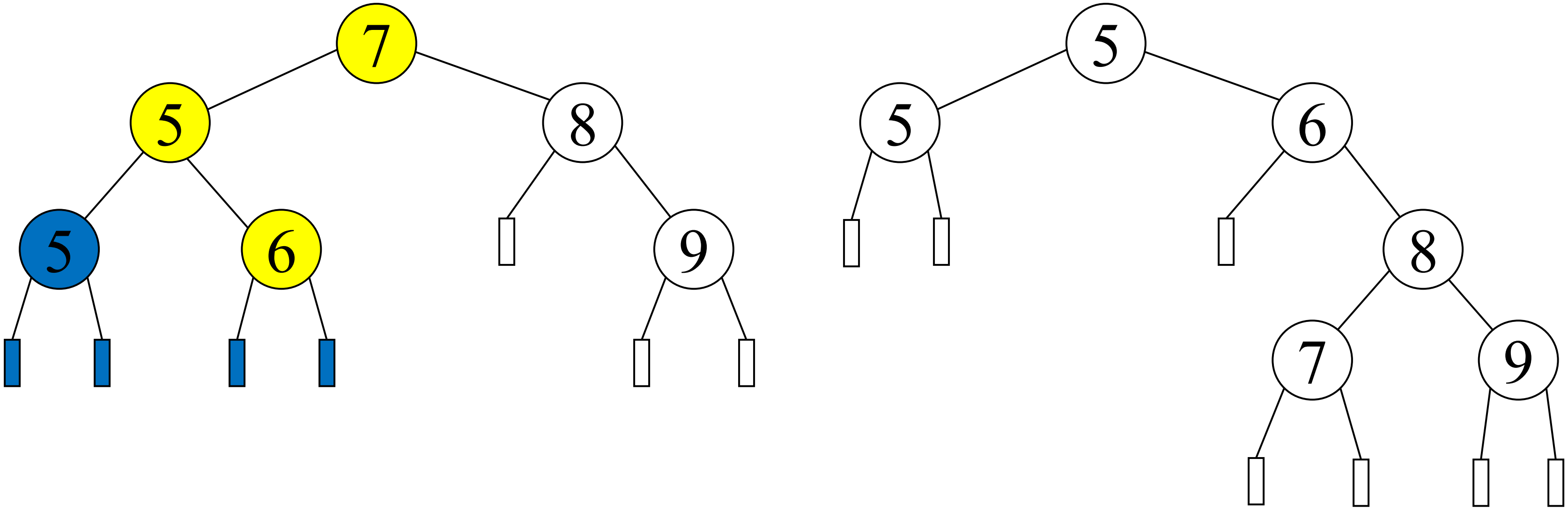
7, 5, 5, 6





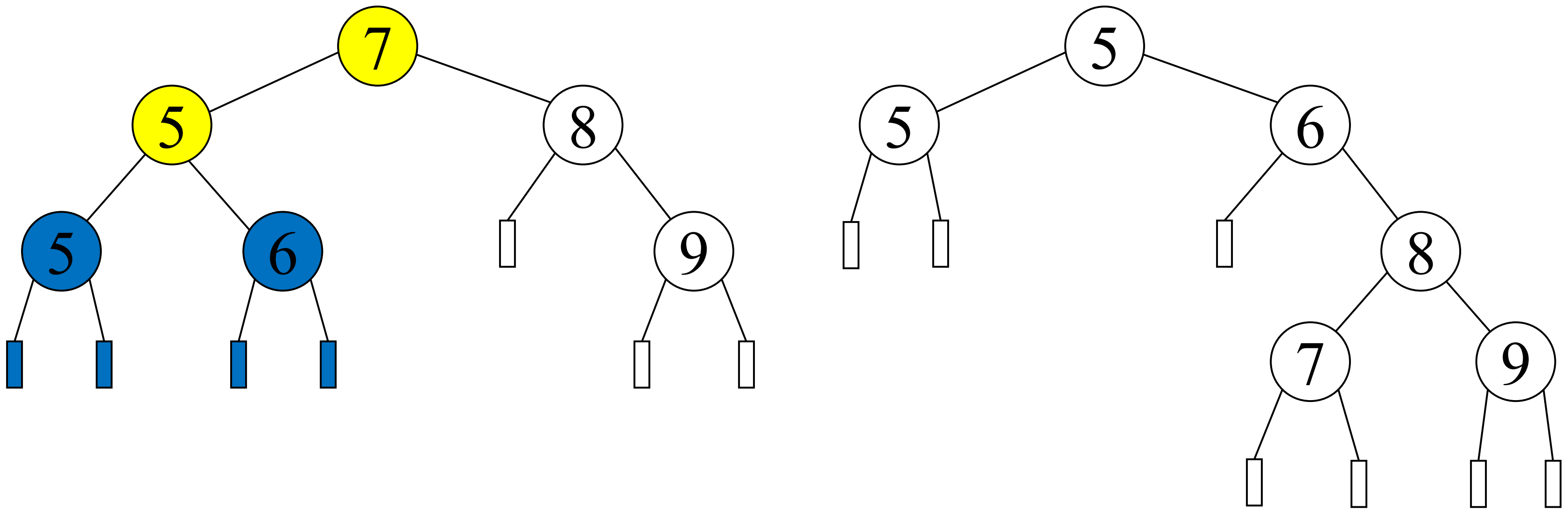
Preorder Tree Walk (example)

7, 5, 5, 6



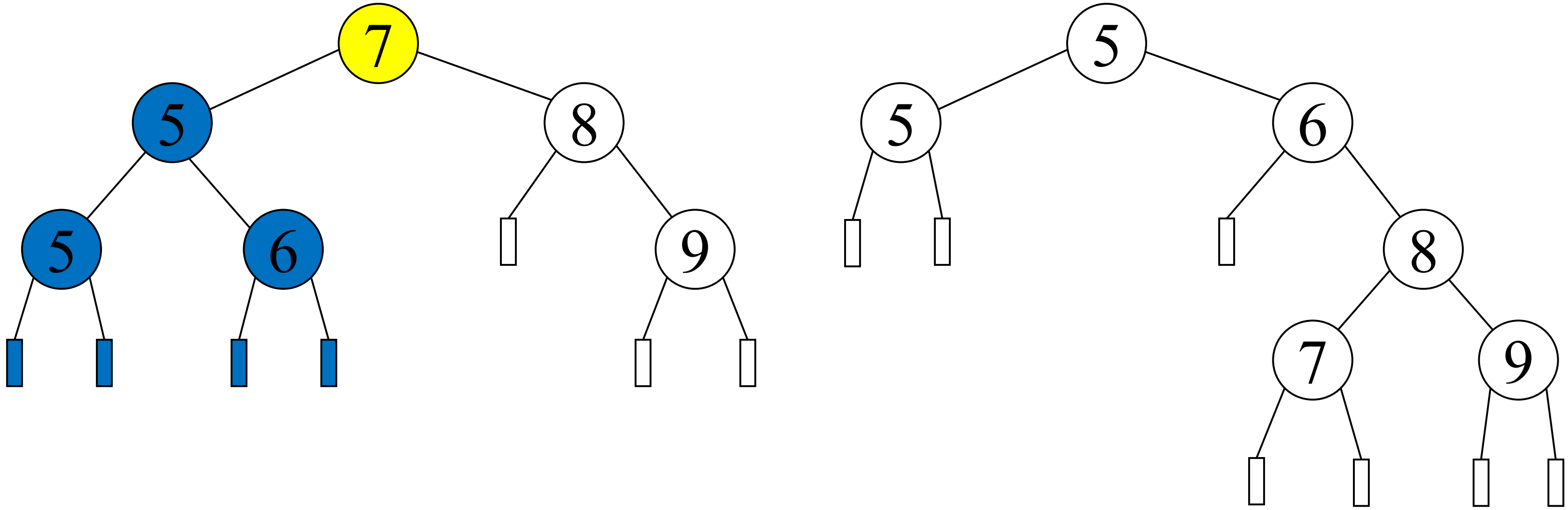
Preorder Tree Walk (example)

7, 5, 5, 6



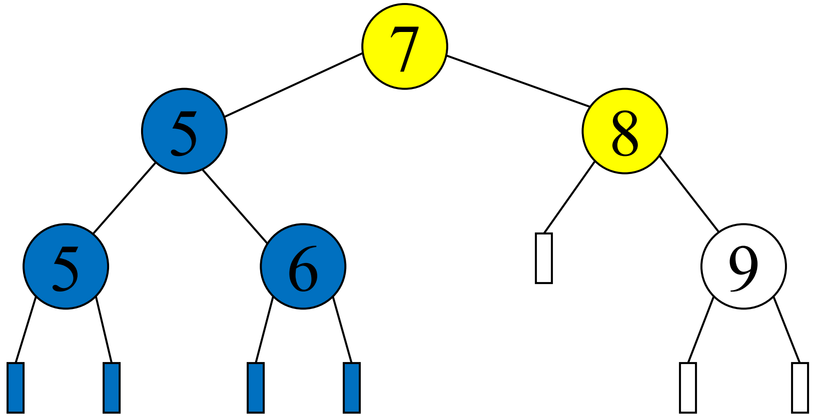
Preorder Tree Walk (example)

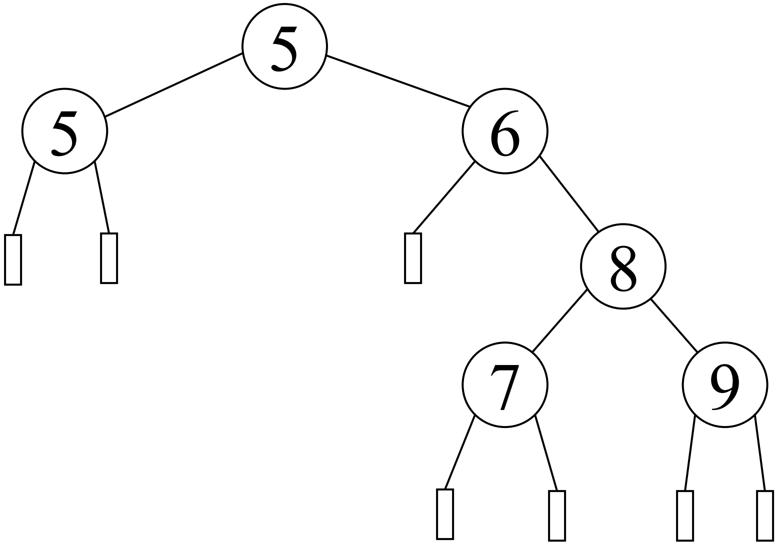
7, 5, 5, 6



Preorder Tree Walk (example)

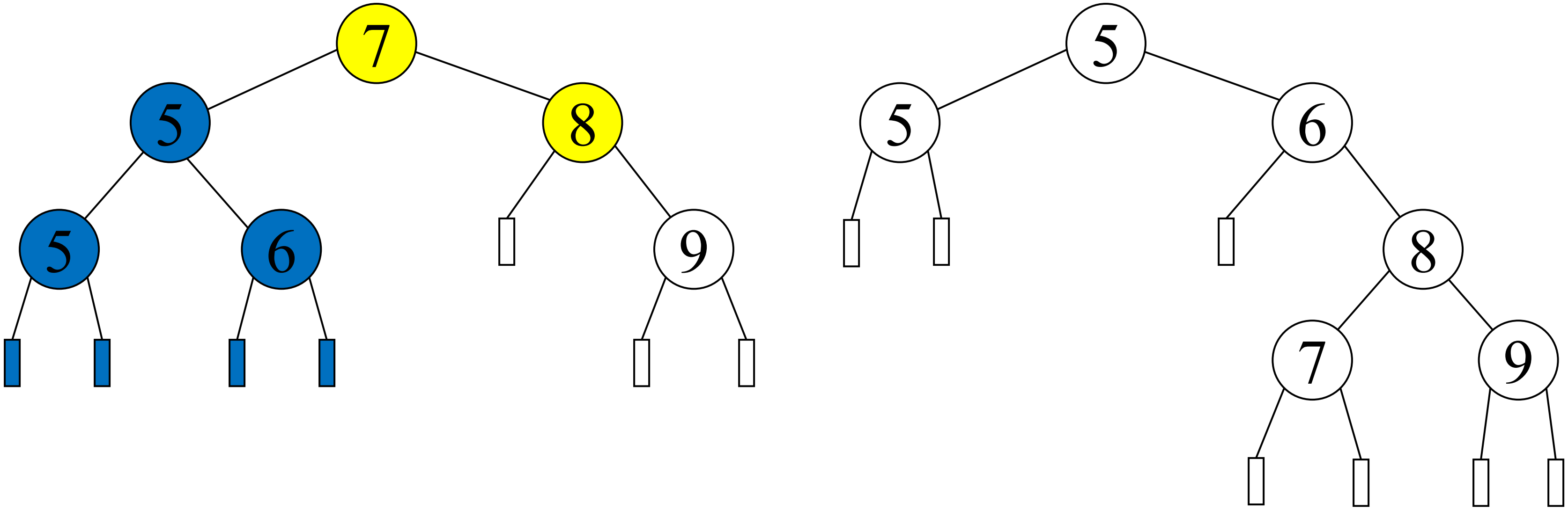
7, 5, 5, 6





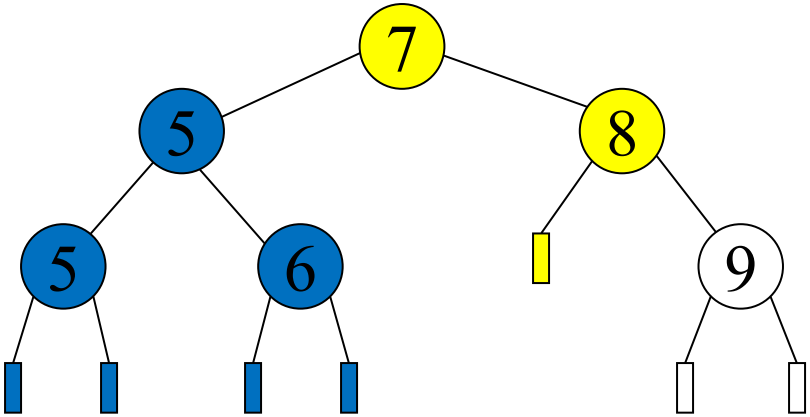
Preorder Tree Walk (example)

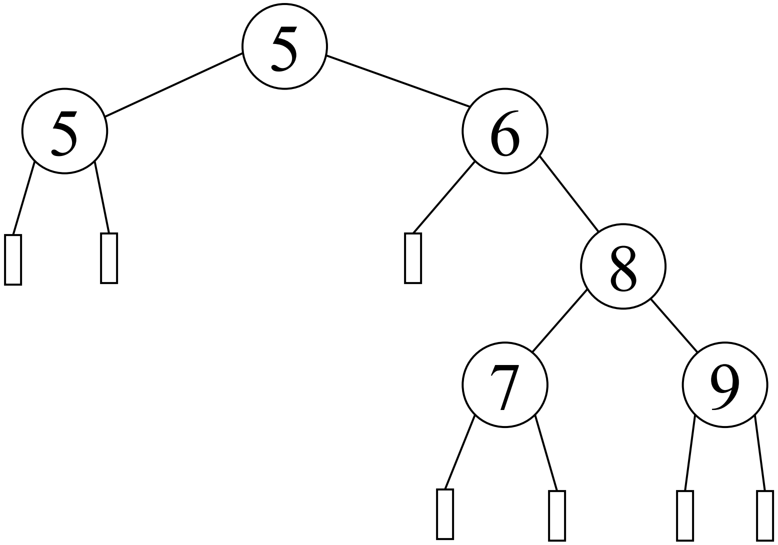
7, 5, 5, 6, 8



Preorder Tree Walk (example)

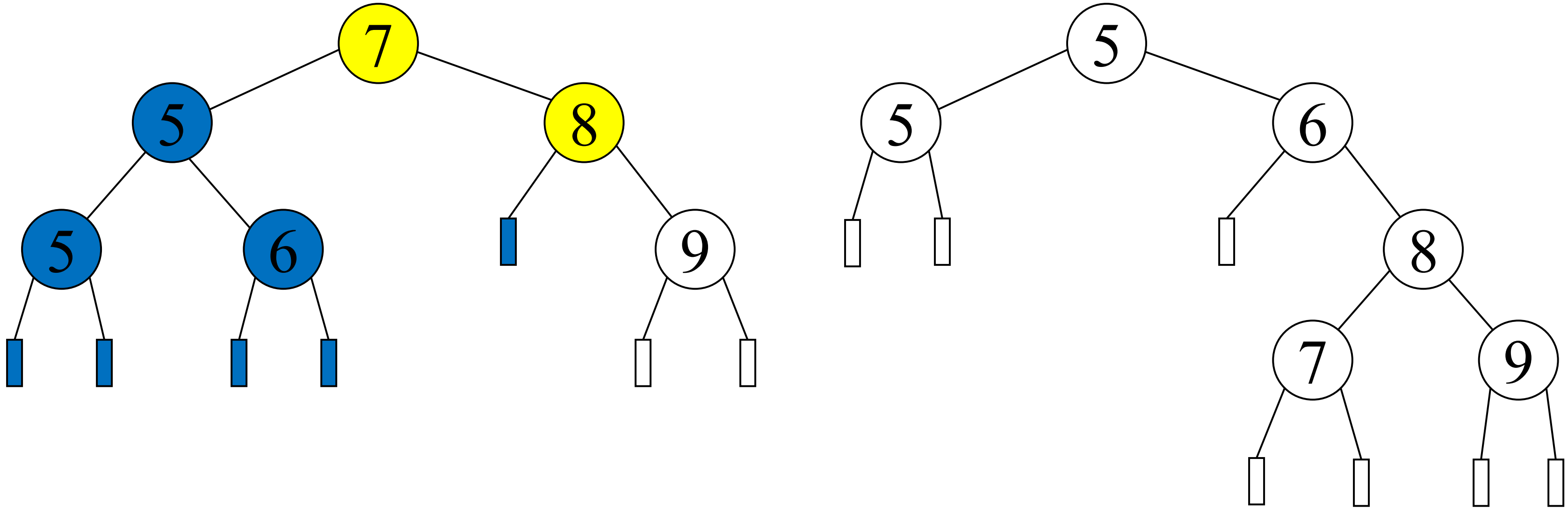
7, 5, 5, 6, 8





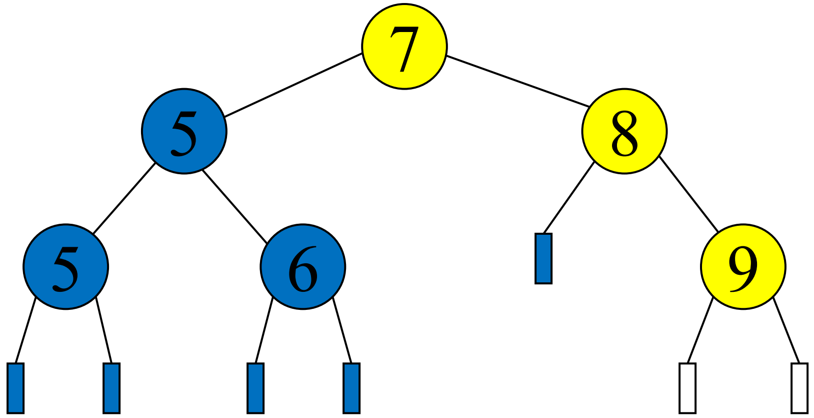
Preorder Tree Walk (example)

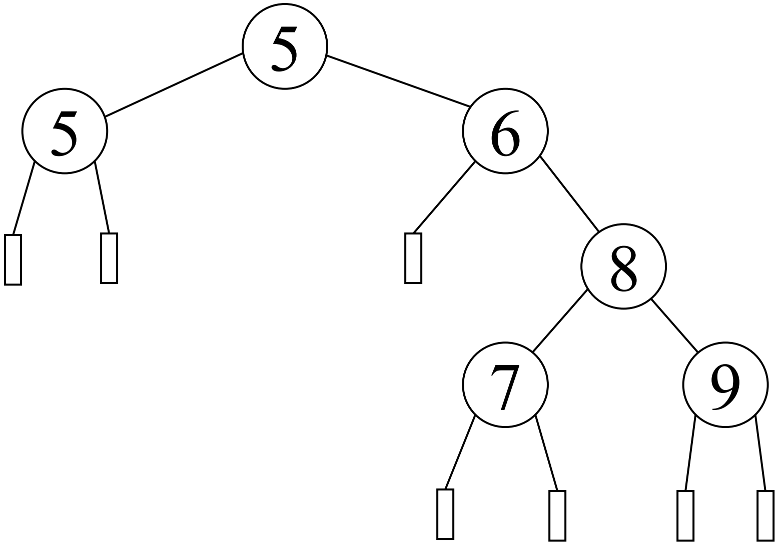
7, 5, 5, 6, 8



Preorder Tree Walk (example)

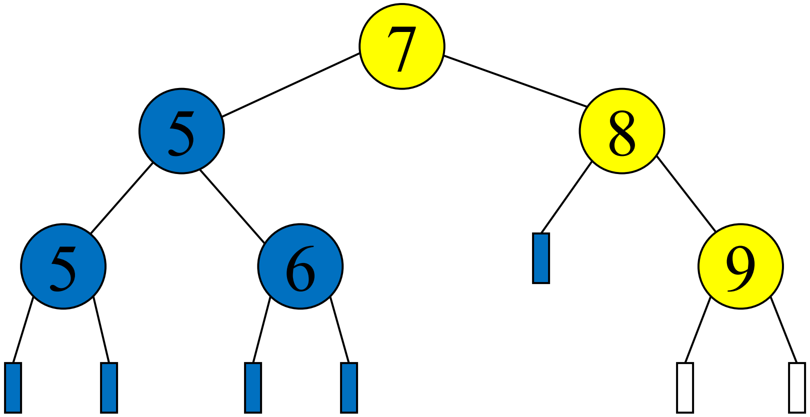
7, 5, 5, 6, 8

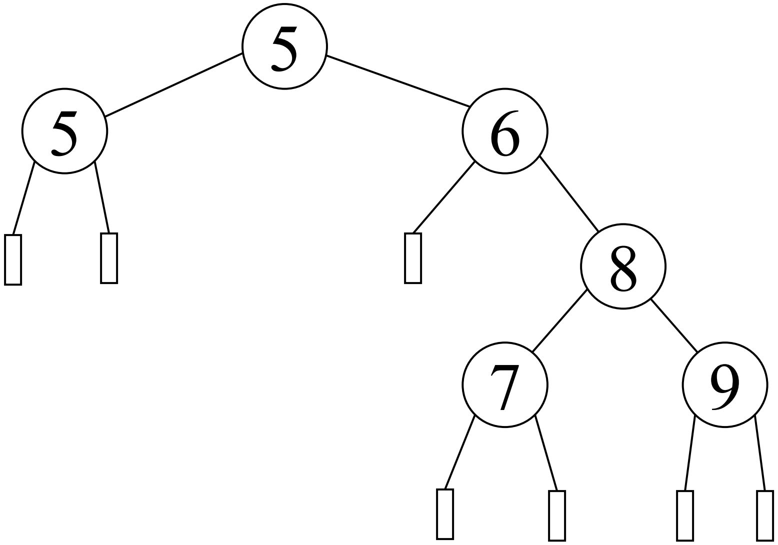




Preorder Tree Walk (example)

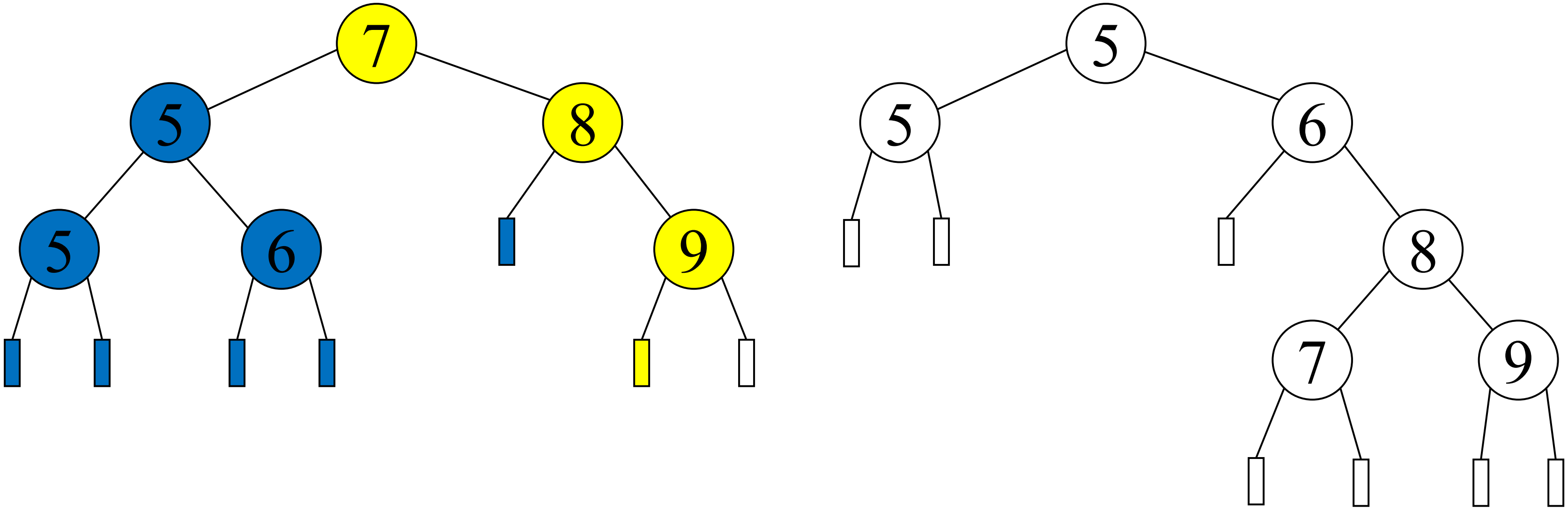
7, 5, 5, 6, 8, 9





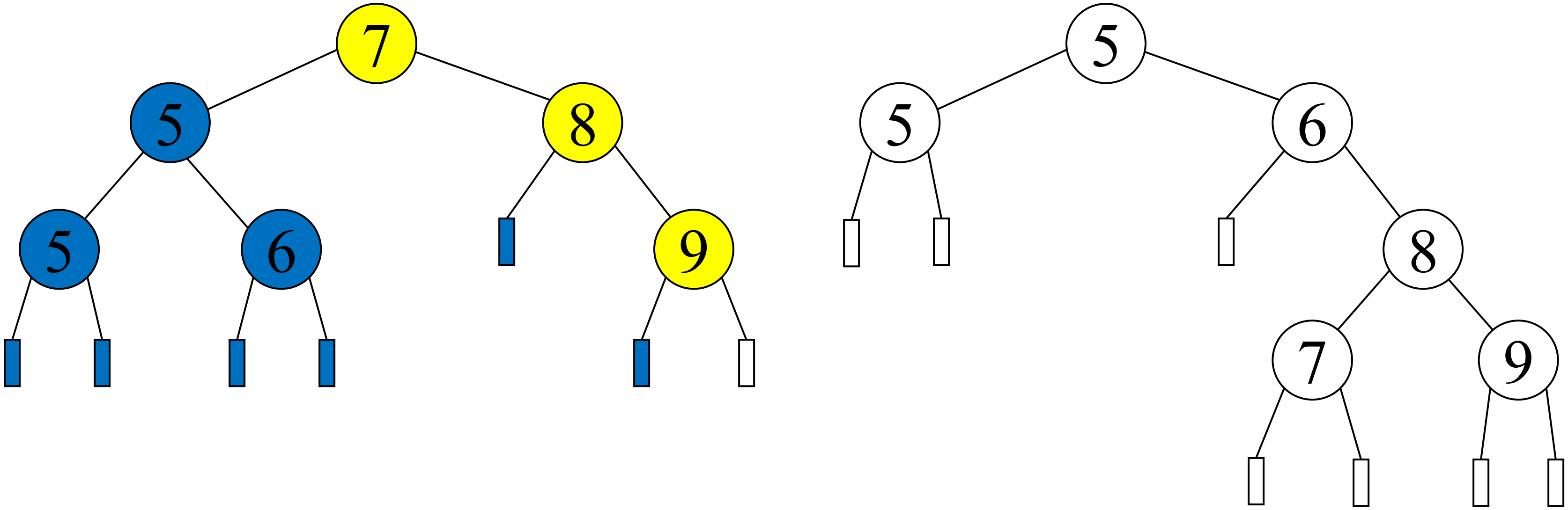
Preorder Tree Walk (example)

7, 5, 5, 6, 8, 9



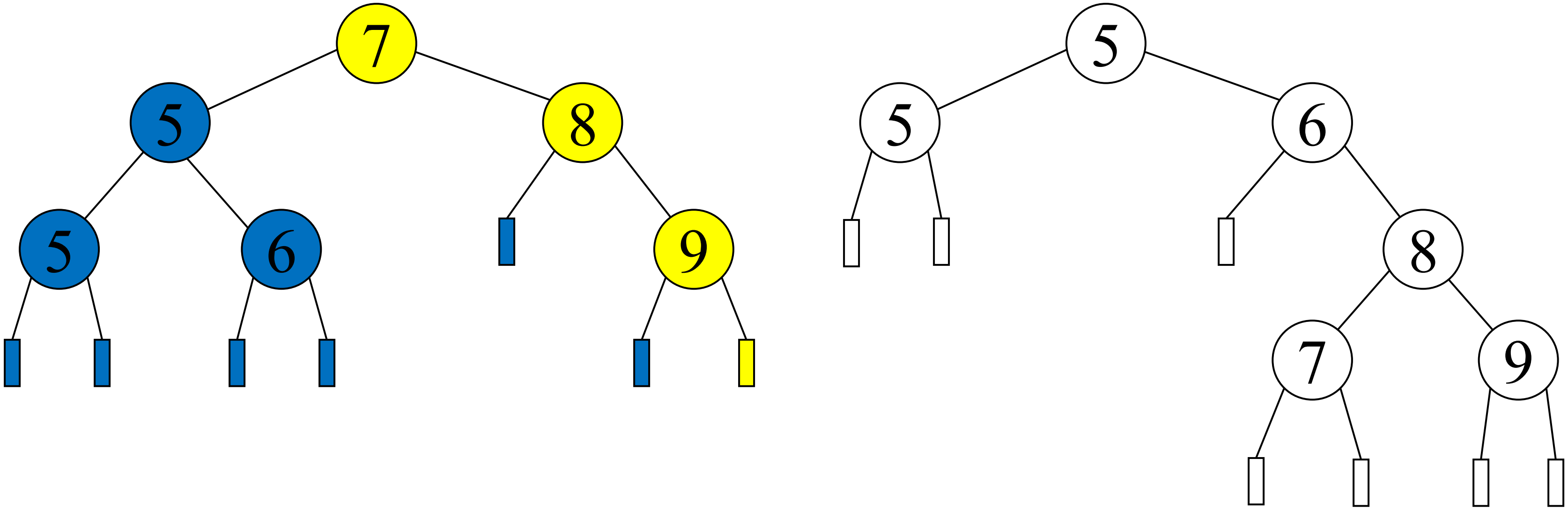
Preorder Tree Walk (example)

7, 5, 5, 6, 8, 9



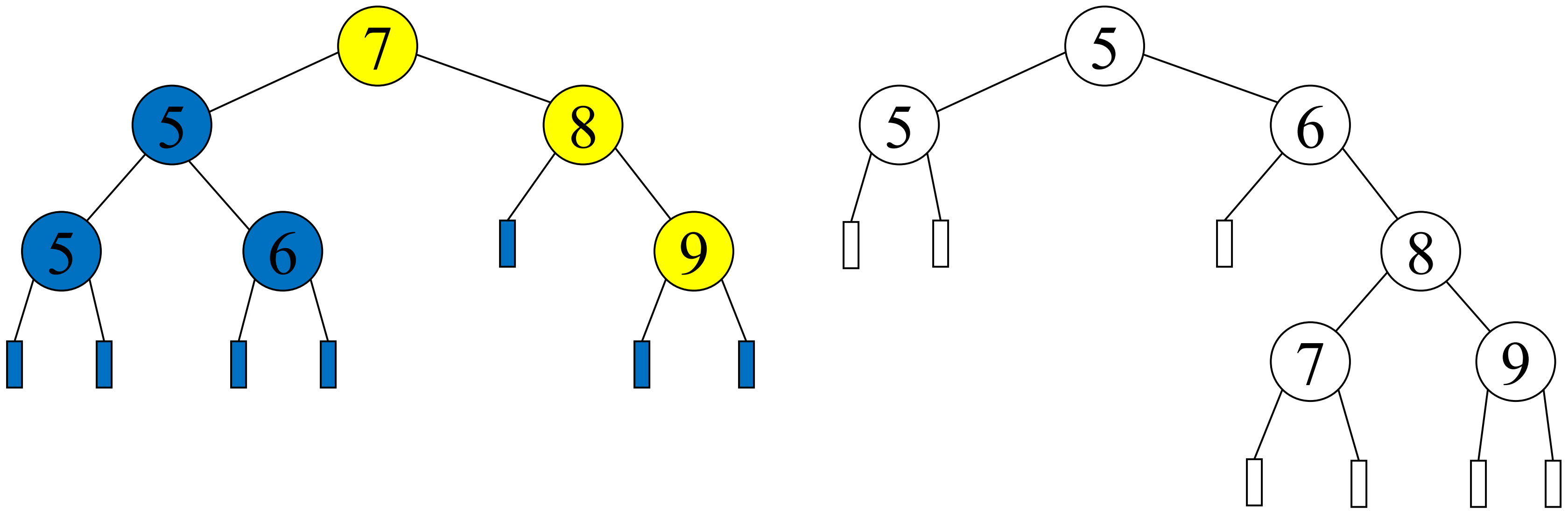
Preorder Tree Walk (example)

7, 5, 5, 6, 8, 9



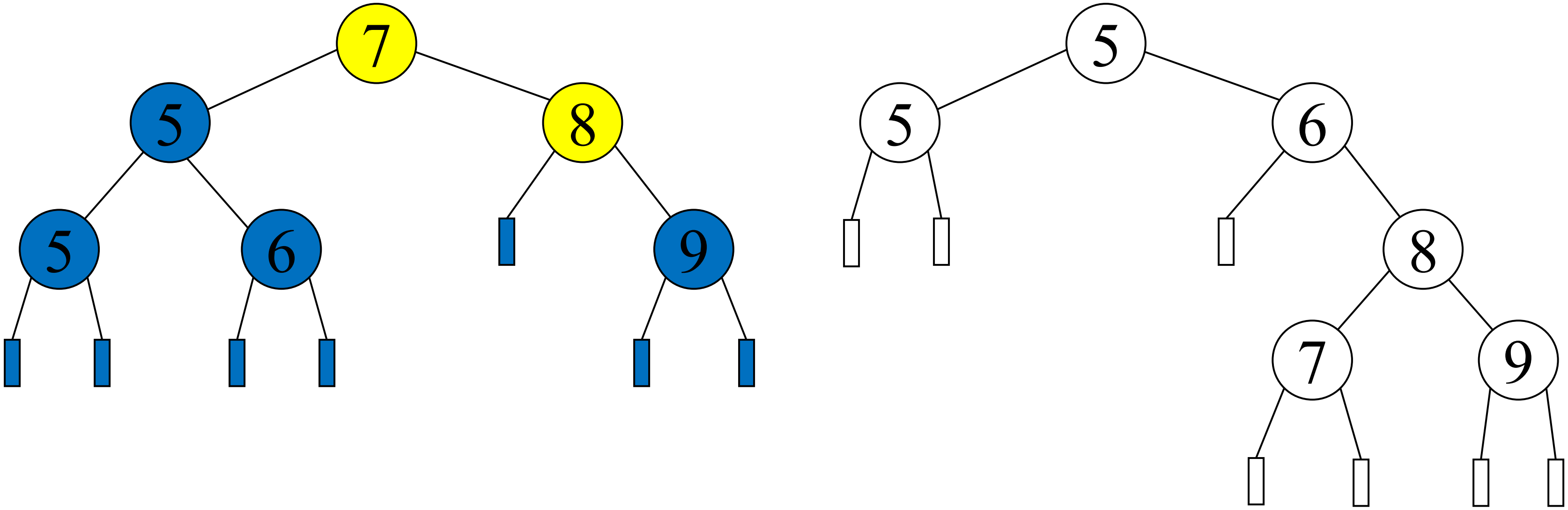
Preorder Tree Walk (example)

7, 5, 5, 6, 8, 9



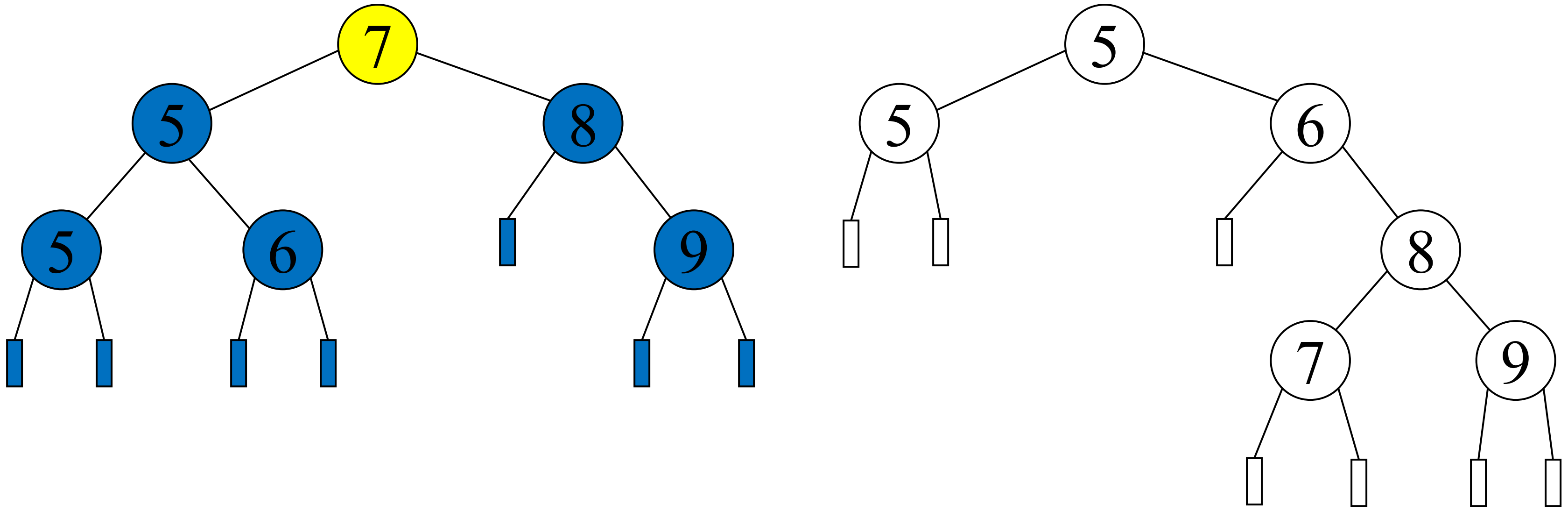
Preorder Tree Walk (example)

7, 5, 5, 6, 8, 9



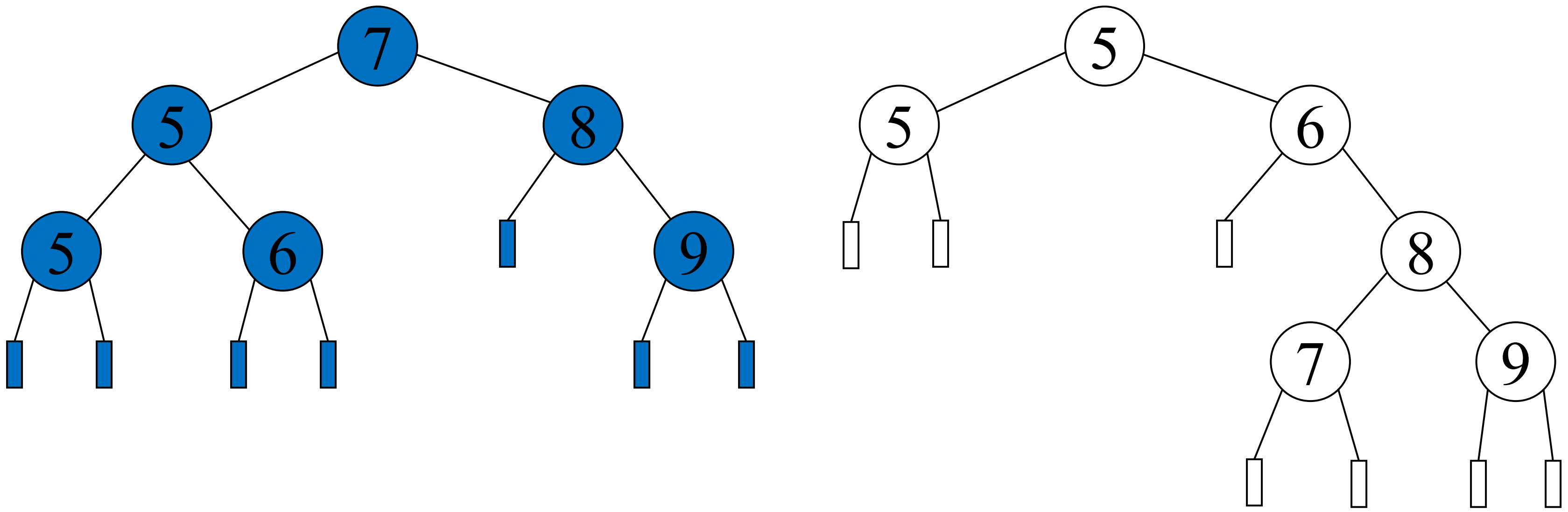
Preorder Tree Walk (example)

7, 5, 5, 6, 8, 9



Preorder Tree Walk (example)

7, 5, 5, 6, 8, 9



Tree Walks (Postorder Walk)

```
Postorder-Tree-Walk(x)
if x ≠ null then
    Postorder-Tree-Walk(x.left)         //left[x]
    Postorder-Tree-Walk(x.right)        //right[x]
    print(x.key)                        //key[x]
```

Time Complexity

```
Inorder-Tree-Walk(x)
    if x $\neq$ null then
        Inorder-Tree-Walk(x.left)        //left[x]
        print(x.key)                     //key[x]
        Inorder-Tree-Walk(x.right)       //right[x]
```

Inorder-Tree-Walk takes $\Theta(n)$ time.
Preorder-Tree-Walk takes $\Theta(n)$ time.
Postorder-Tree-Walk takes $\Theta(n)$ time.
Here n is the number of tree nodes.

ASU®

Ira A. Fulton Schools of Engineering

Arizona State University

Binary Search Trees, Part 3

- Binary Search Trees, Representation
- Tree Walks
- Search, Min, Max, Successor
- Insertion
- Deletion

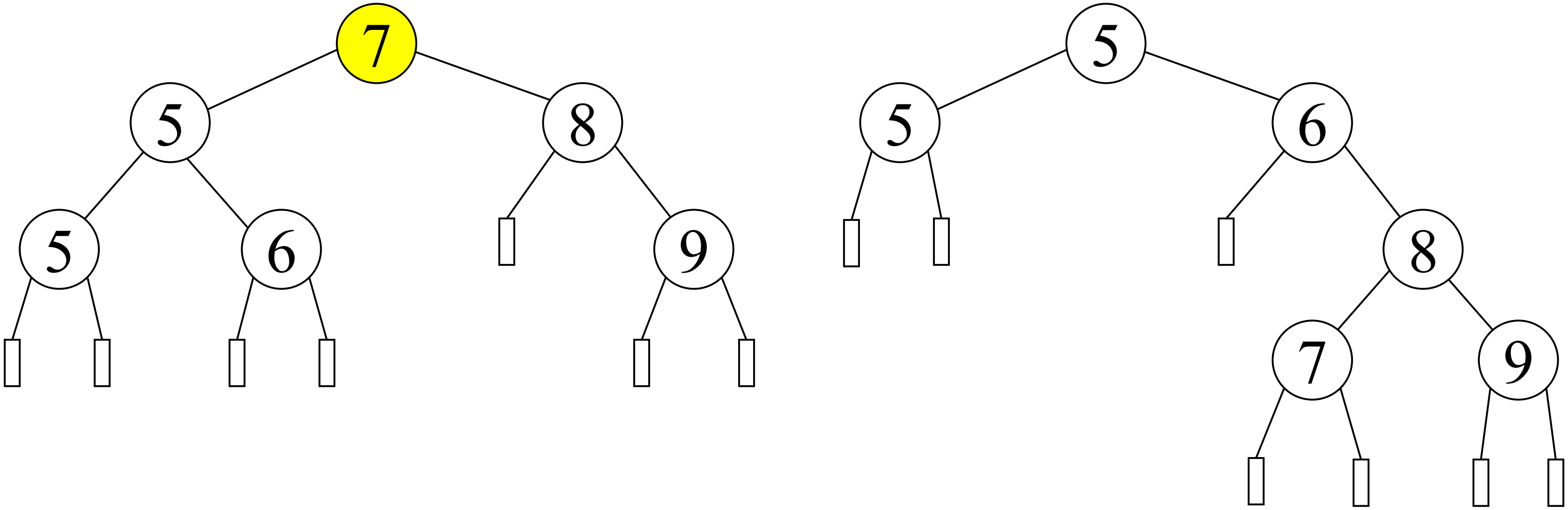
Tree Search

```
Tree-Search(x, data)
    if x == null or x.key == data then
        return x
    if data < x.key then
        return Tree-Search(x.left, data)
    else return Tree-Search(x.right, data)
```

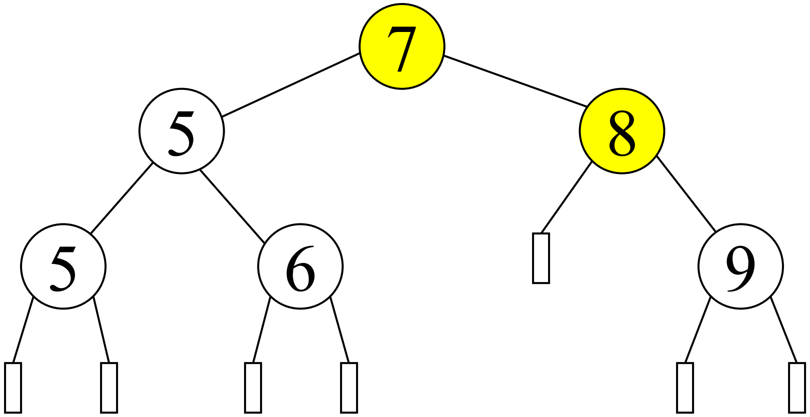
```
Iterative-Tree-Search(x, data) if x == null or x.key == data then
    while x ≠ null and x.key ≠ data do
        if data < x.key then
            x = x.left
        else
            x = x.right
    return x
```

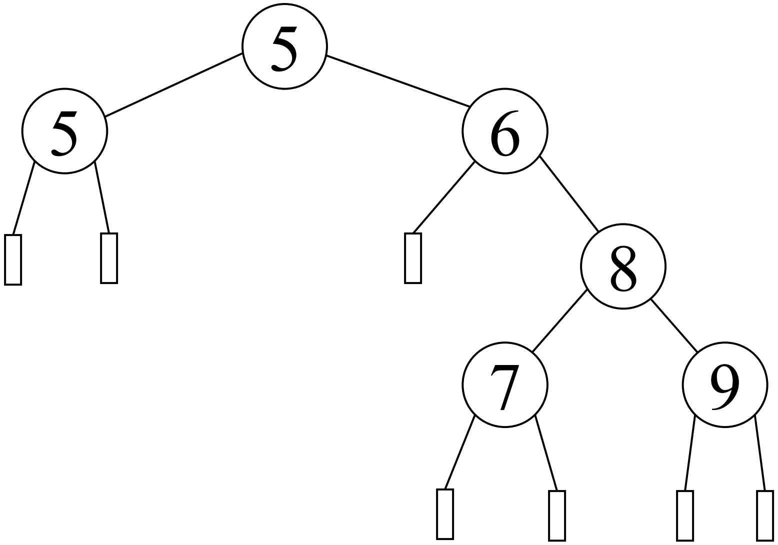
Running time: O(tree height)

Tree-Search(root, 9)

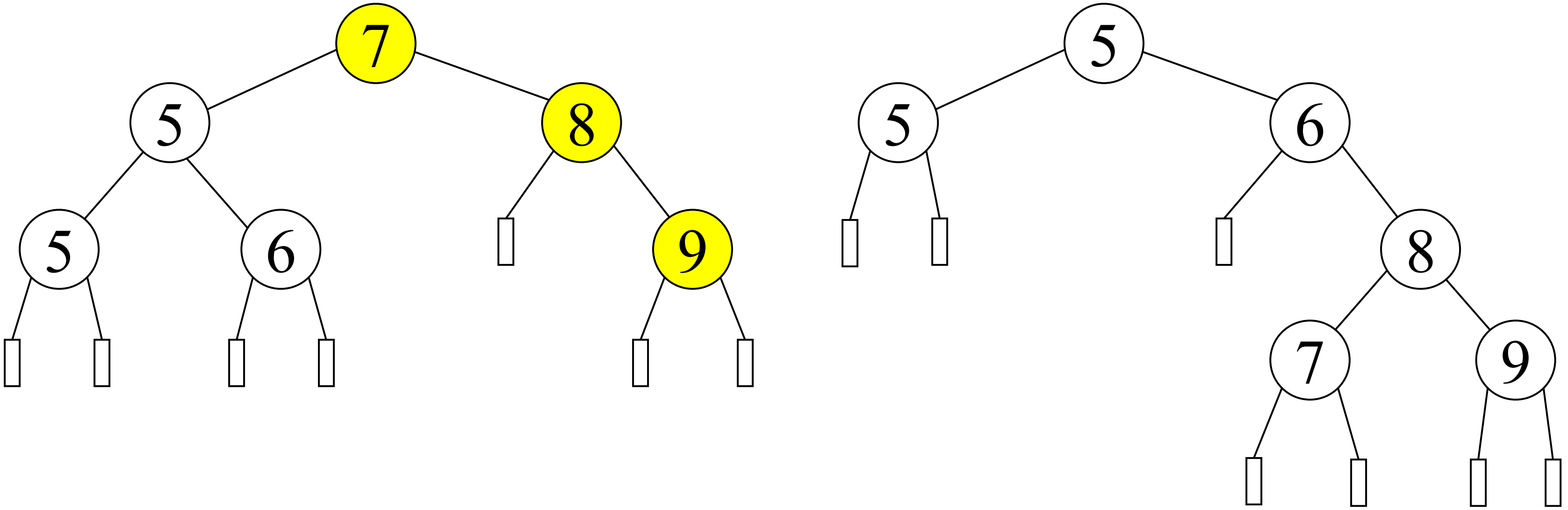


Tree-Search(root, 9)



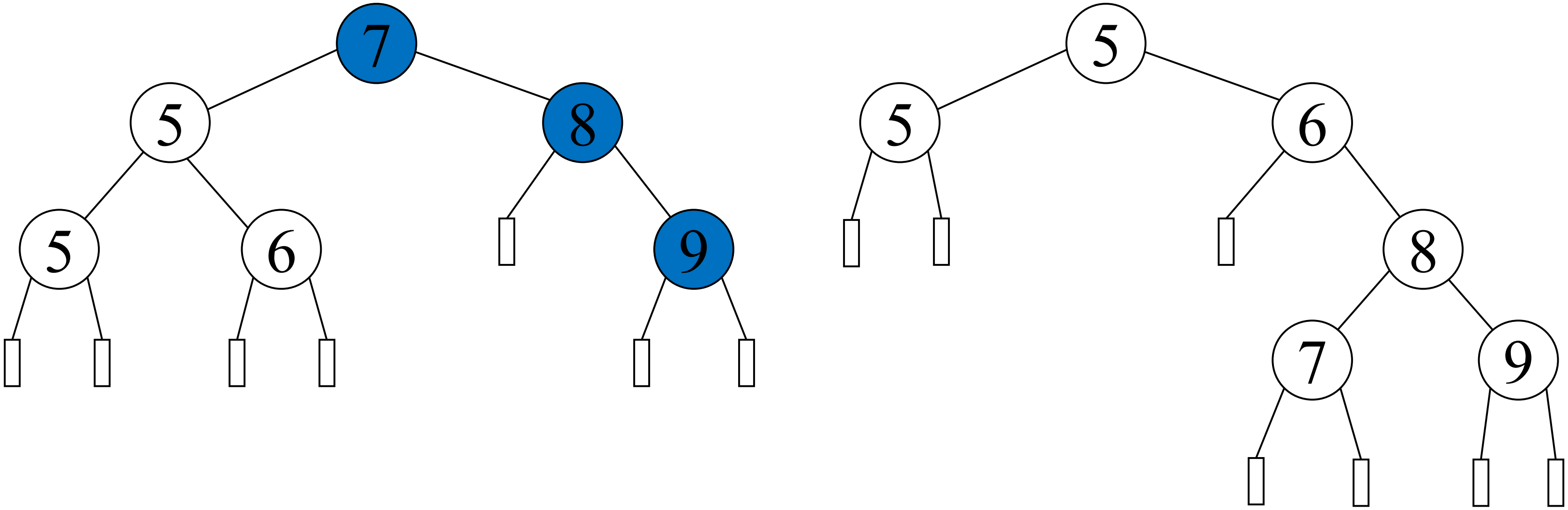


Tree-Search(root, 9)

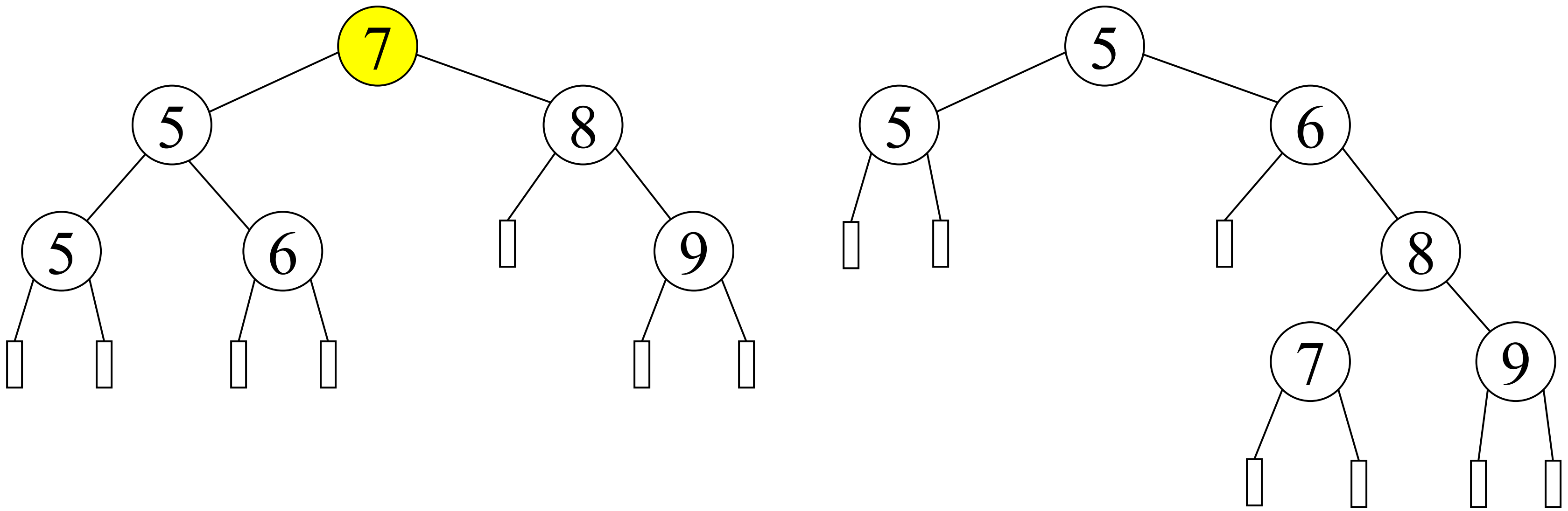


Tree-Search(root, 9)

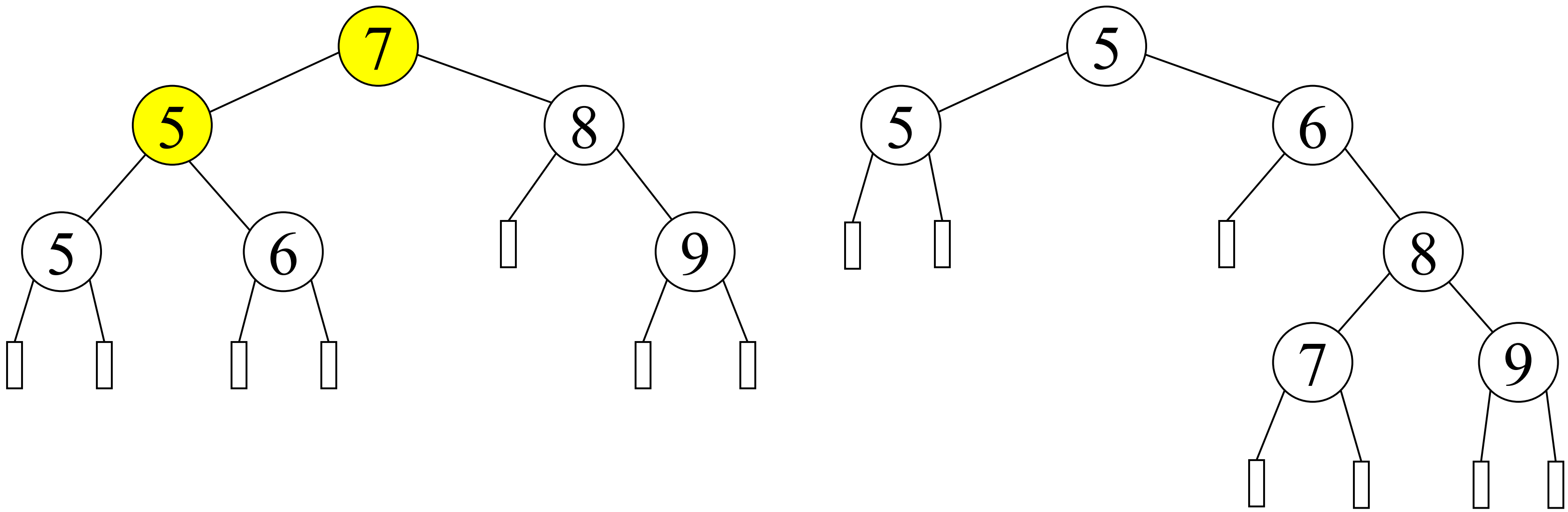
Return the address of the node with key=9



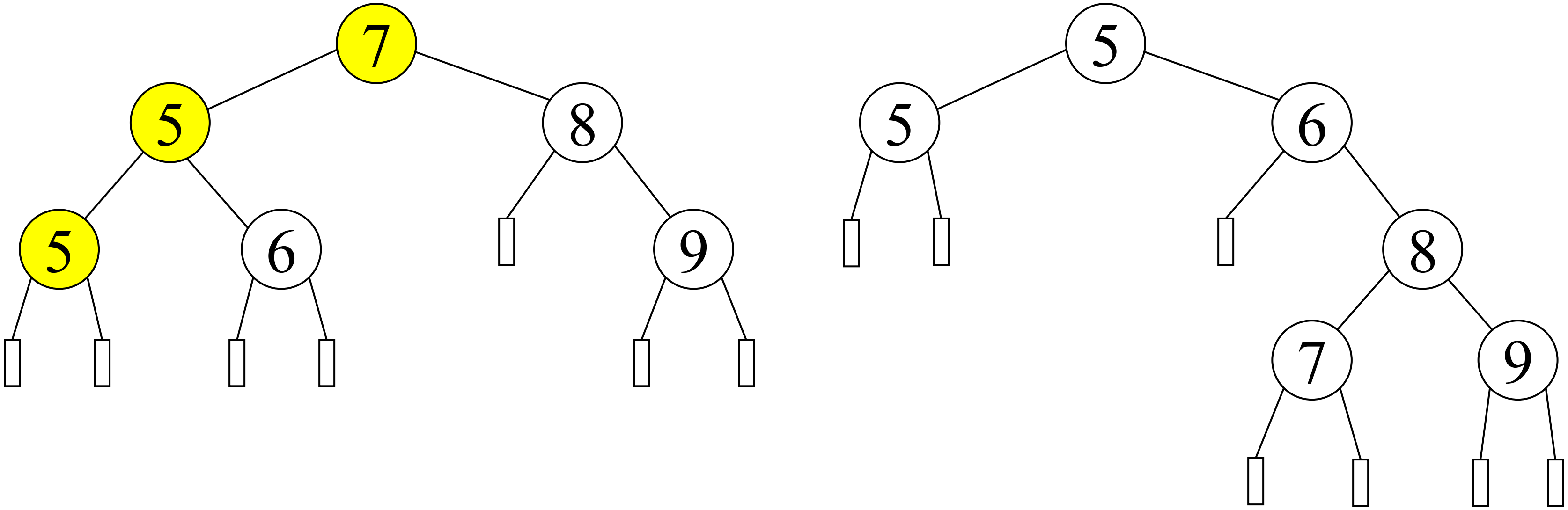
Tree-Search(root, 4)



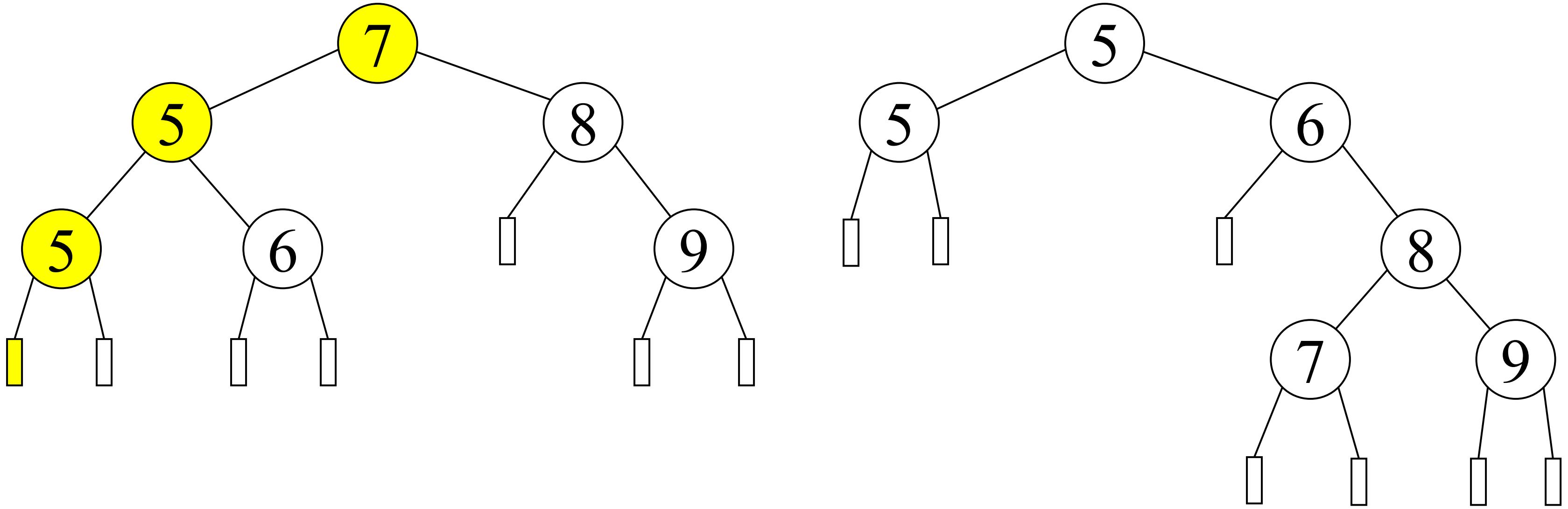
Tree-Search(root, 4)



Tree-Search(root, 4)

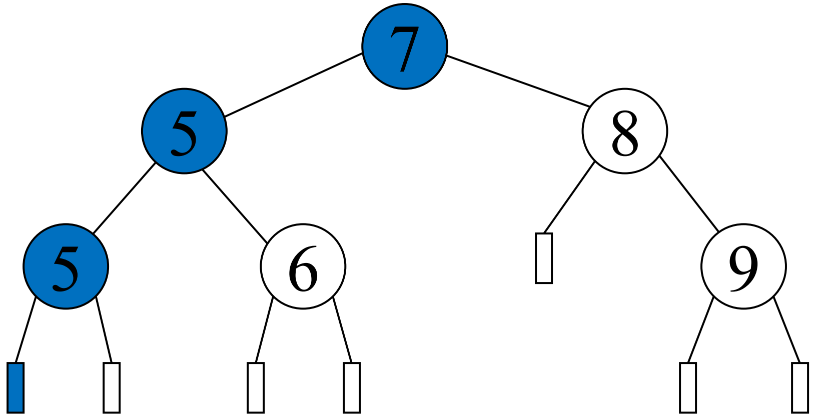


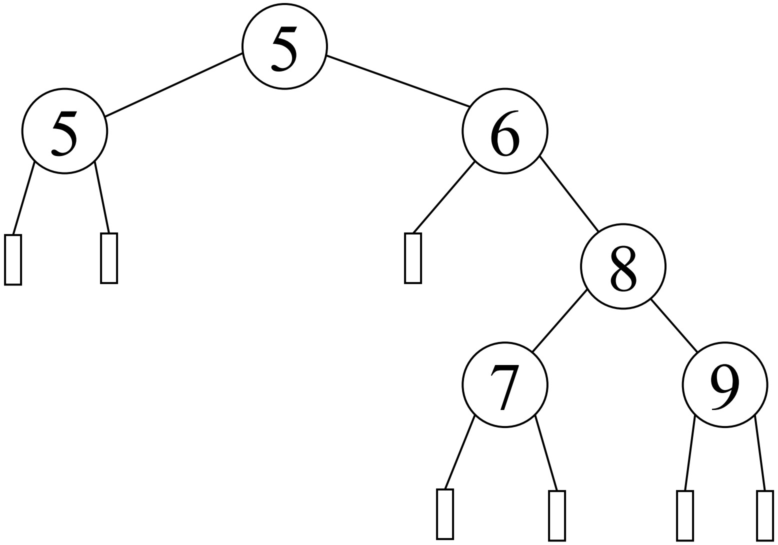
Tree-Search(root, 4)



Tree-Search(root, 4)

Return NULL





Tree-Minimum and Tree-Maximum

Tree-Minimum(x)

```
while $x \neq$ null and $x$.left $\neq$ null do
    $x$ = $x$.left
return $x$
```

Tree-Maximum(x)

```
while $x \neq$ null and $x$.right $\neq$ null do
    $x$ = $x$.right
return $x$
```

Running time: O(tree height)

Tree-Successor(x)

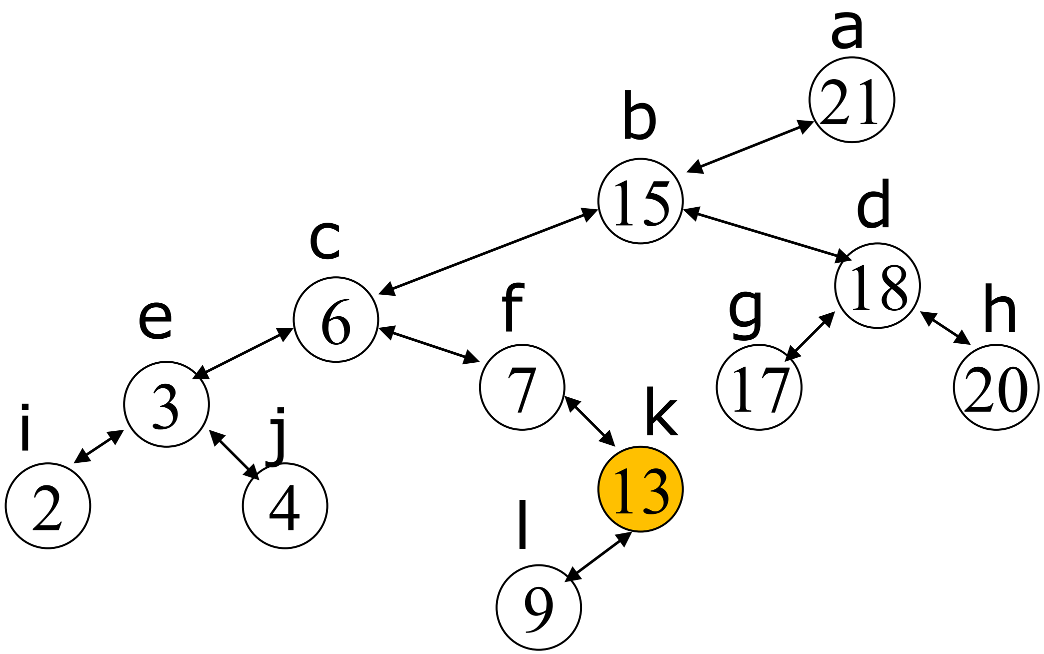
Successor(x)

```
if $x.right \neq null$ then
    return Tree-Minimum(x.right)
$y = x.parent$
while $y \neq null$ and $x == y.right$ do
    $x = y$
    $y = y.parent$
return y
```

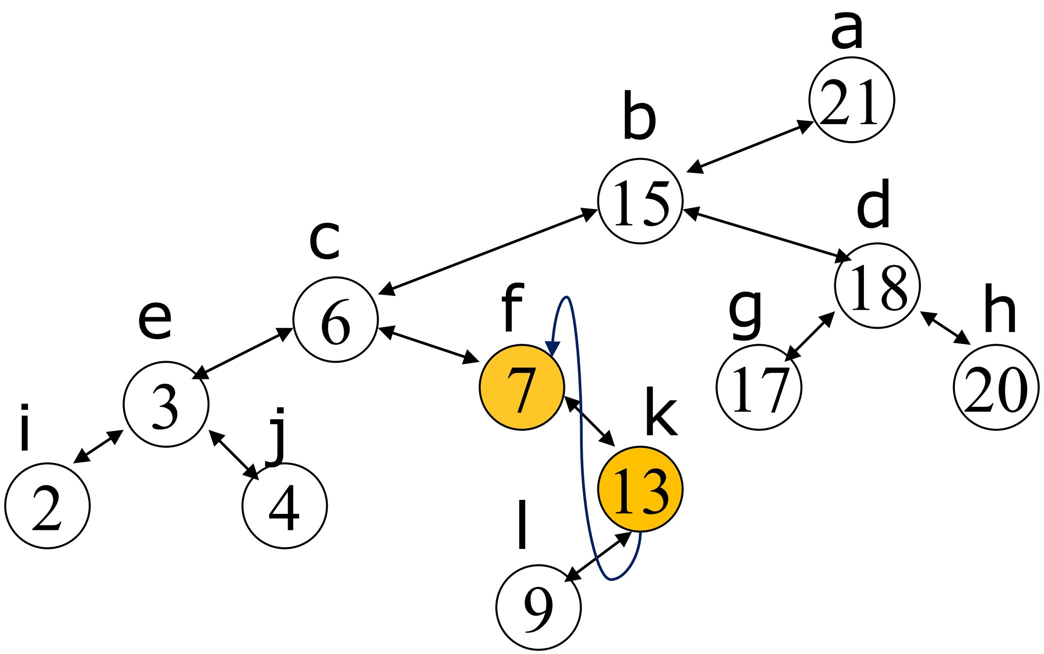
Condition: $x \neq null$

Running time: O(tree height)

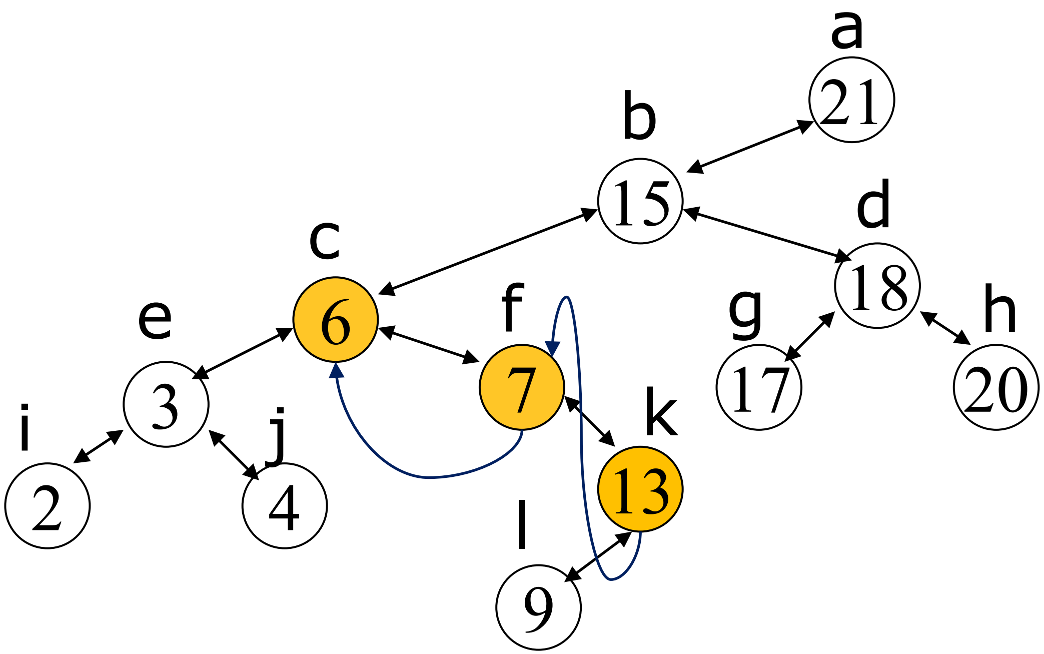
Example: Tree-Successor(k)



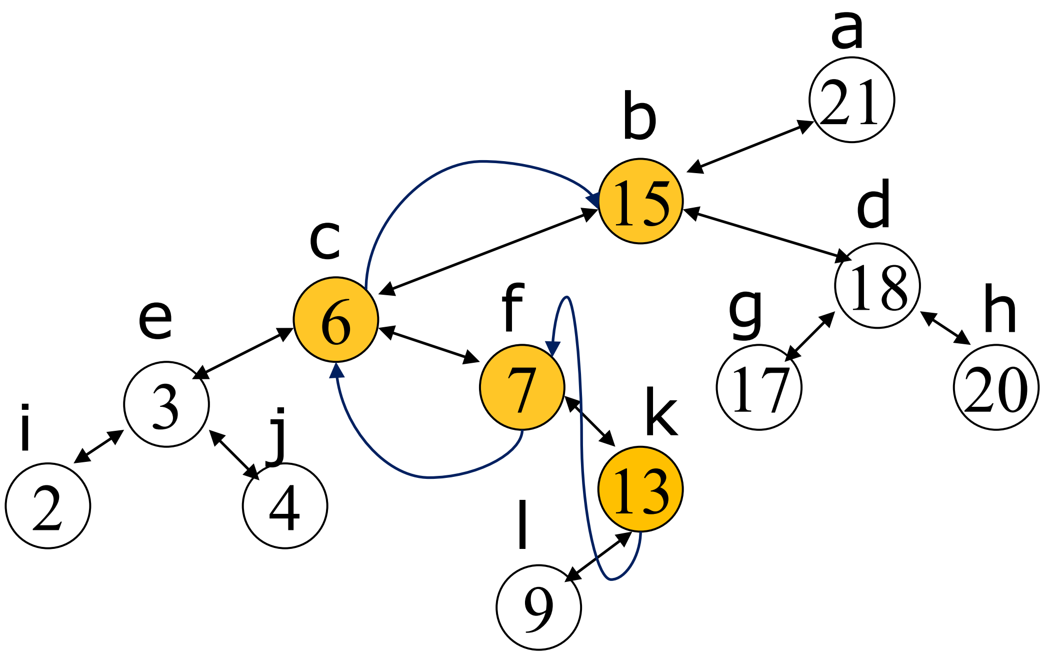
Example: Tree-Successor(k)



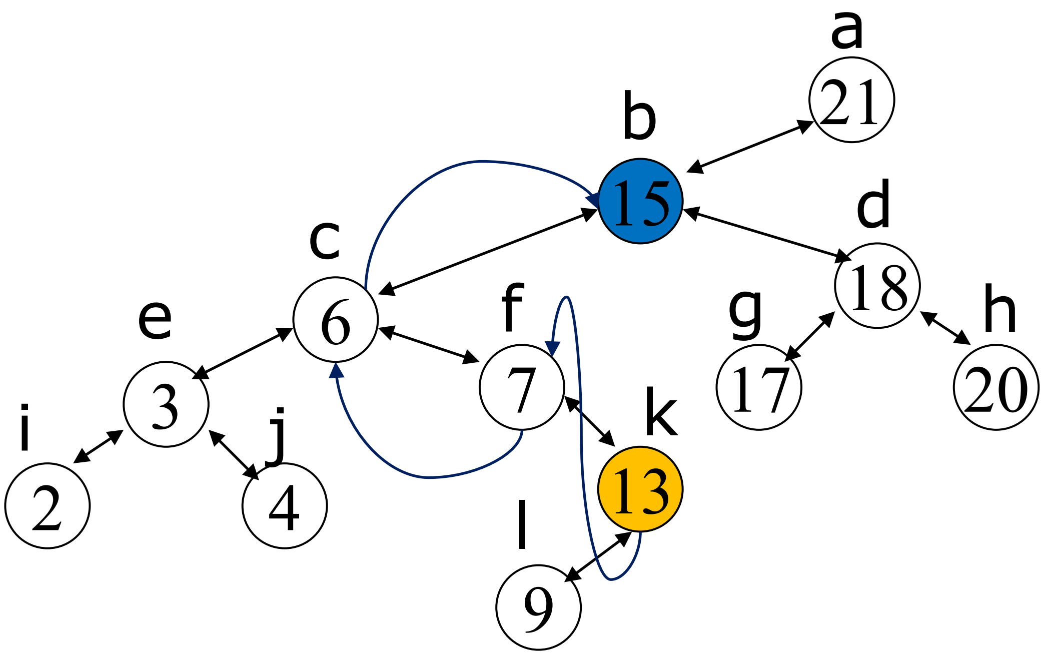
Example: Tree-Successor(k)



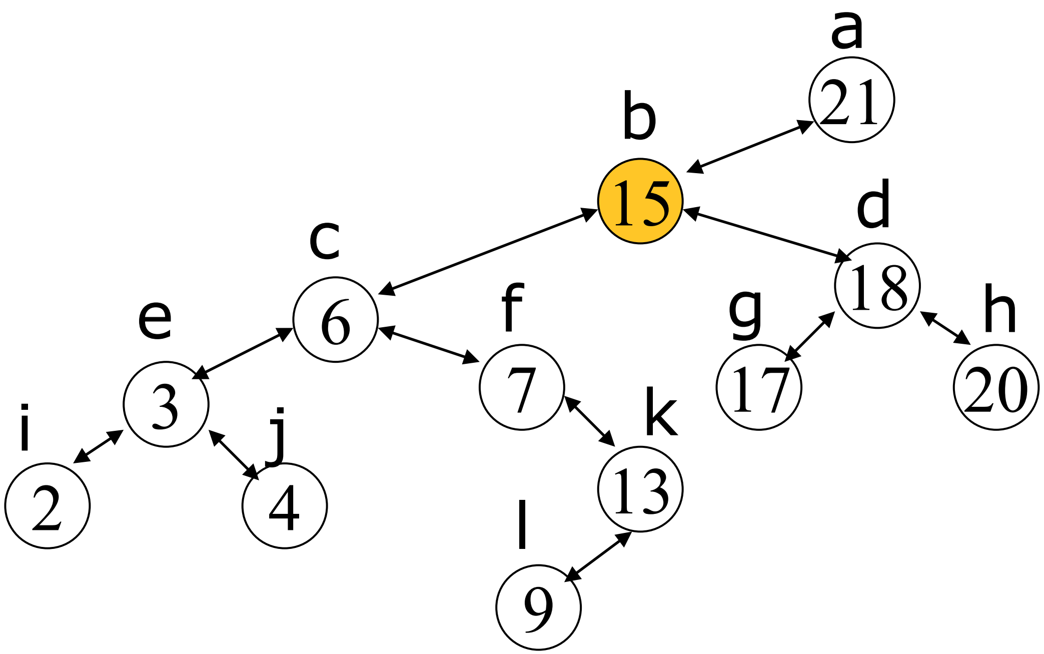
Example: Tree-Successor(k)



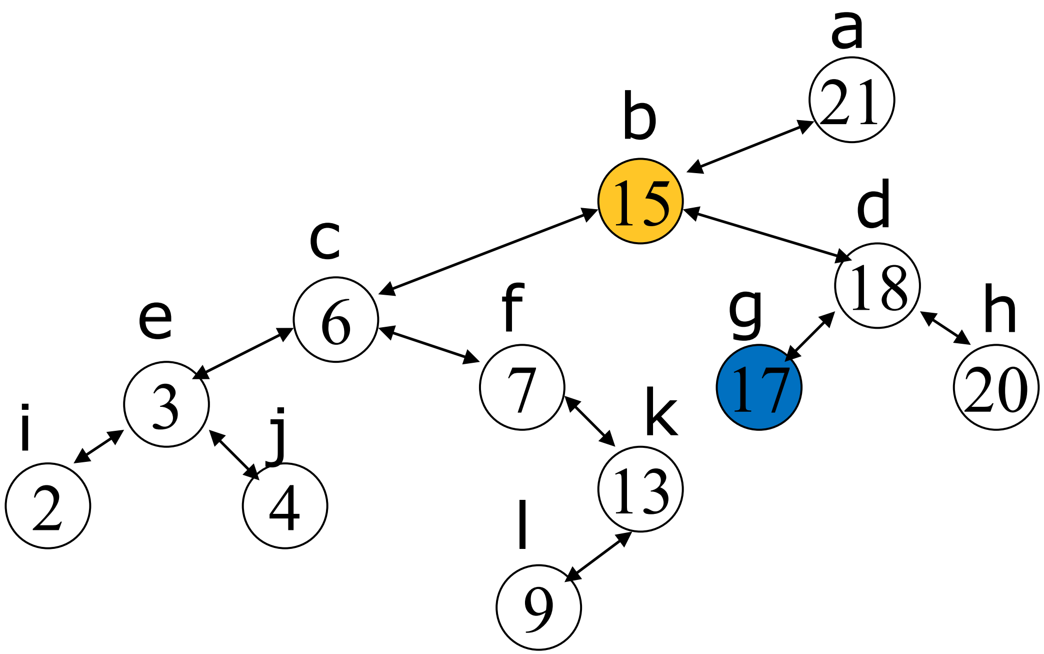
Example: Tree-Successor(k)



Example: Tree-Successor(b)



Example: Tree-Successor(b)



ASU®

Ira A. Fulton Schools of Engineering

Arizona State University

Binary Search Trees, Part 4

- Binary Search Trees, Representation
- Tree Walks
- Search, Min, Max, Successor
- Insertion
- Deletion

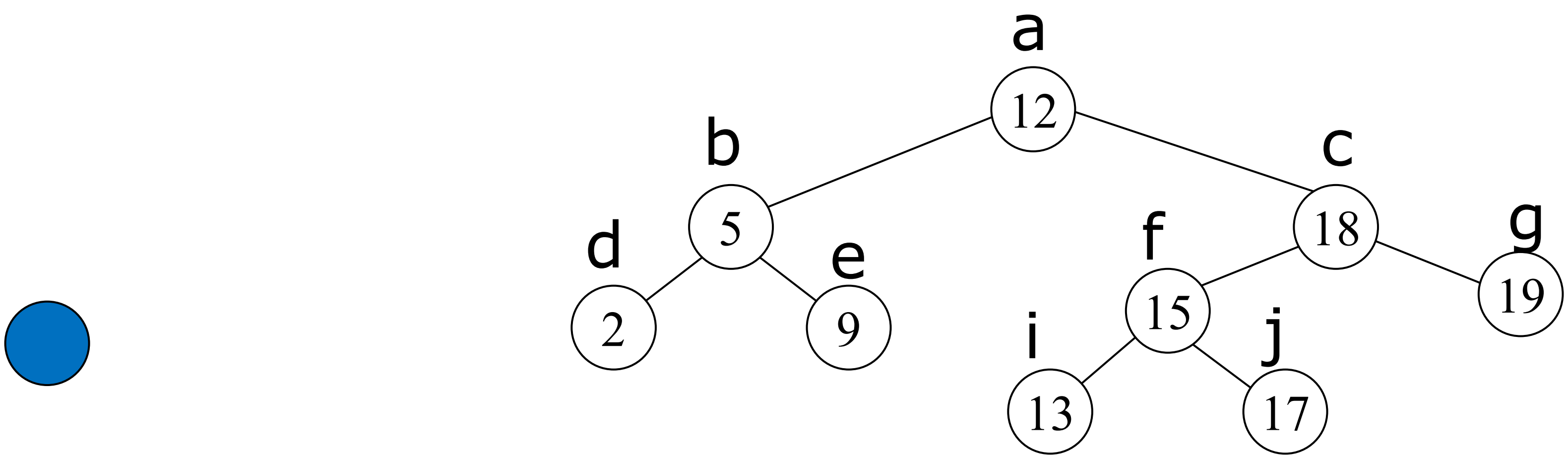
Tree Insert

```
Tree-Insert(T, z)
y = NULL
x = T.root
while $x \neq$ NULL
    y = x
    if z.key < x.key then
        x = x.left
    else x = x.right
z.parent = y
if y == NULL then
    T.root = z
else if z.key < y.key then
    y.left = z
else y.right = z
```

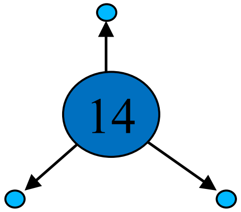
```
z = new-treenode(value, null, null, null)
    z.key = value
    z.parent = null
    z.left := null
    z.right := null
```

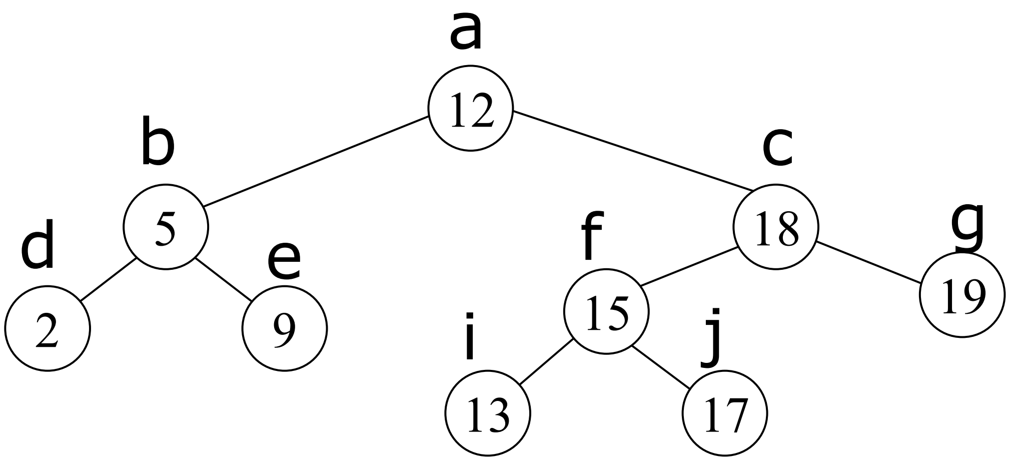
Running time: O(tree height)

Tree-Insert Example: Insert a node with key 14

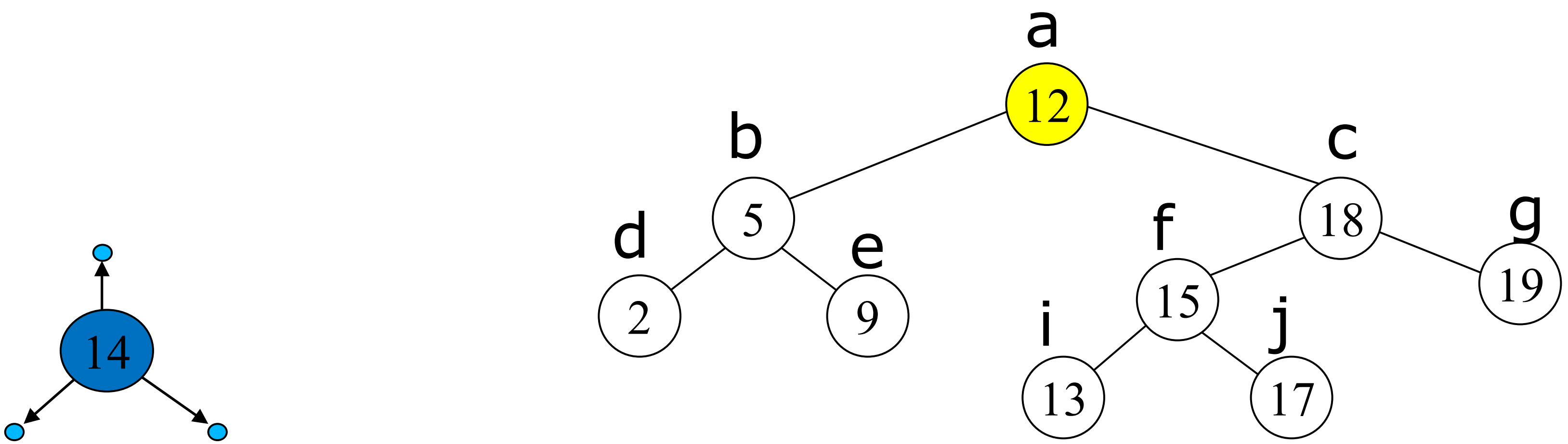


Tree-Insert Example: Insert a node with key 14

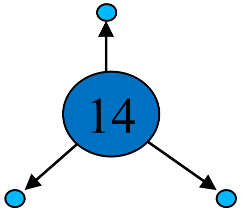


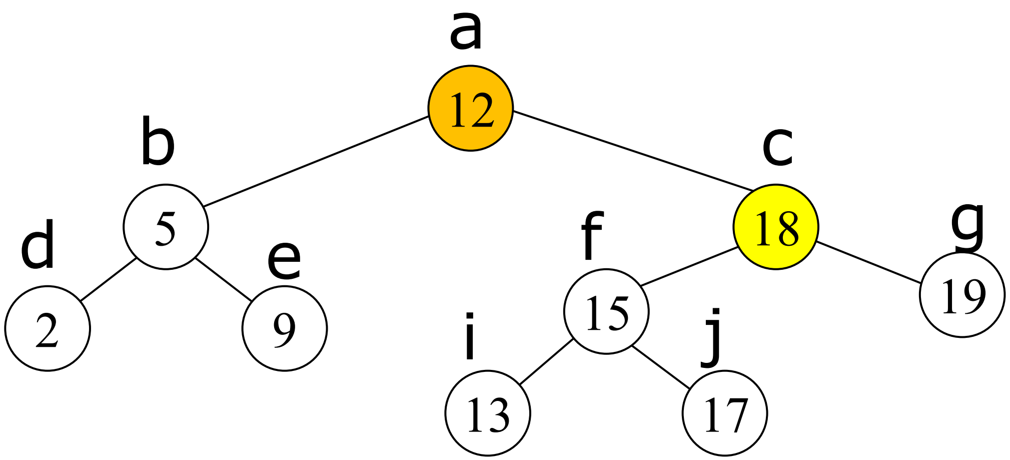


Tree-Insert Example: Insert a node with key 14

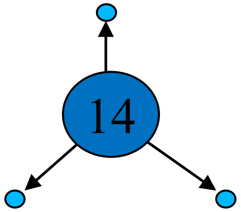


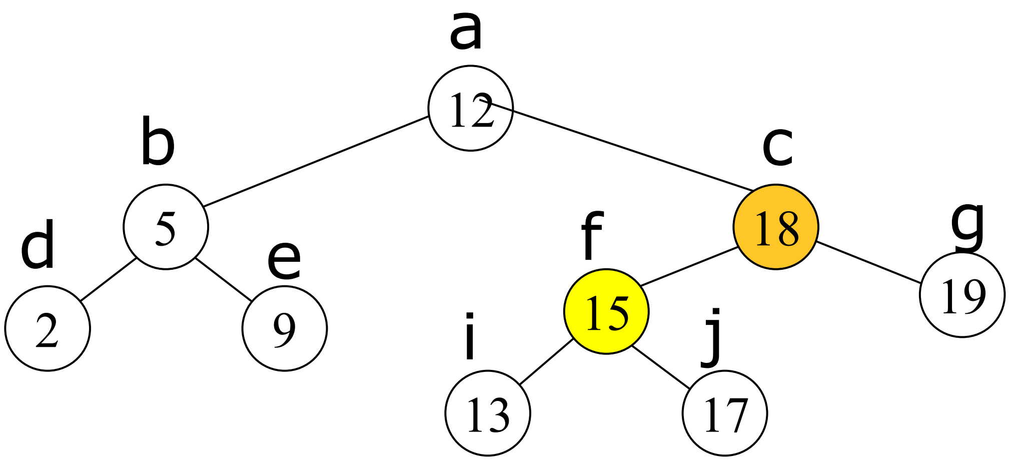
Tree-Insert Example: Insert a node with key 14



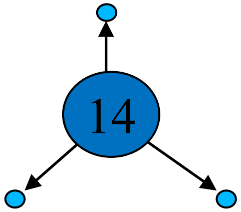


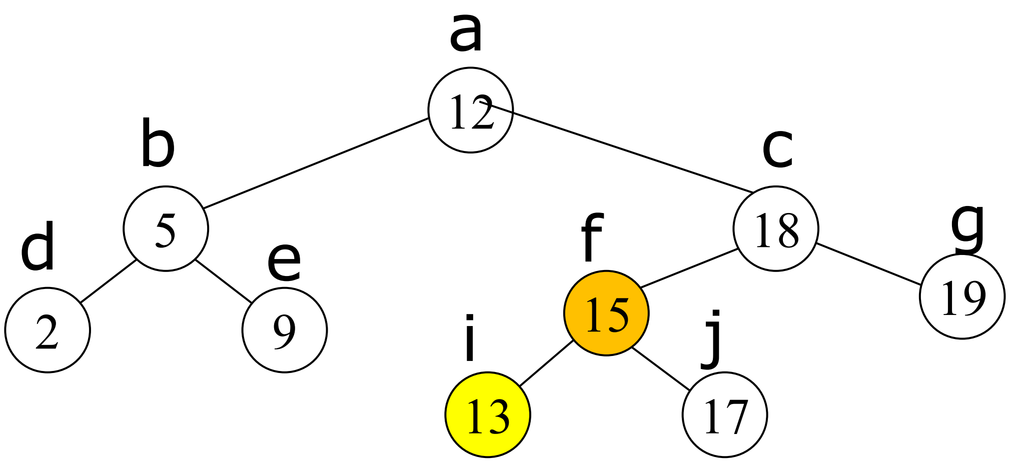
Tree-Insert Example: Insert a node with key 14



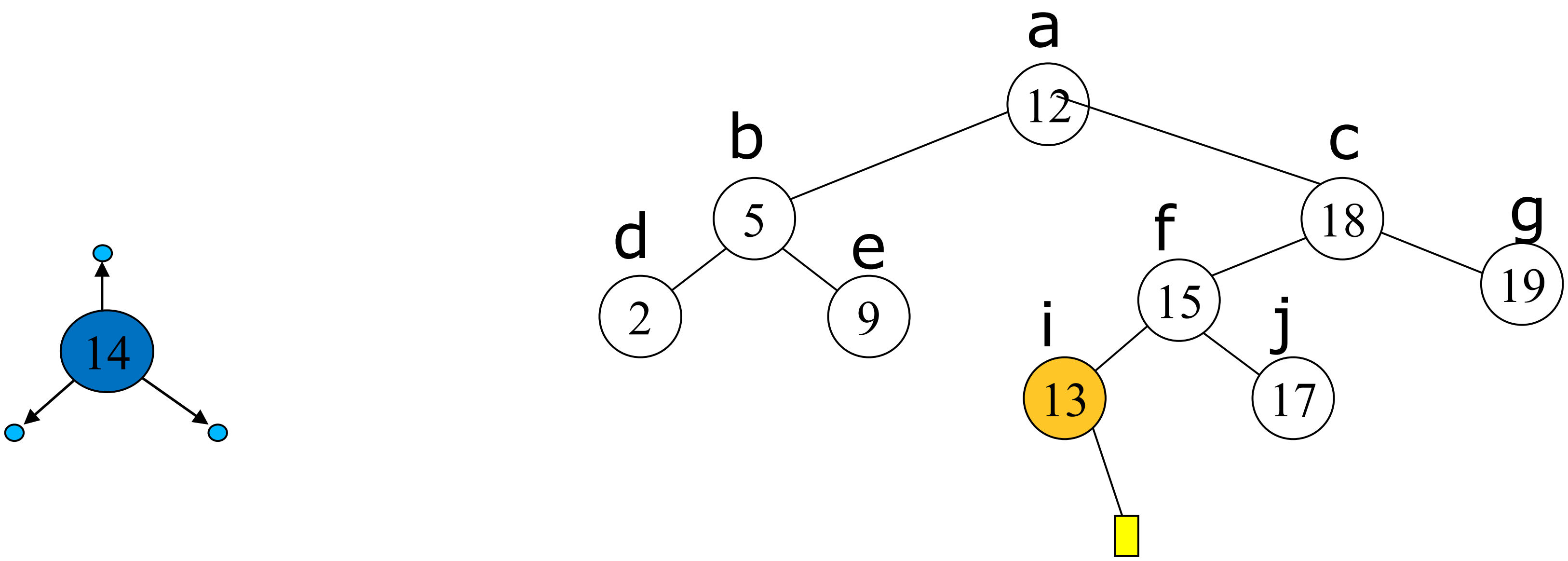


Tree-Insert Example: Insert a node with key 14

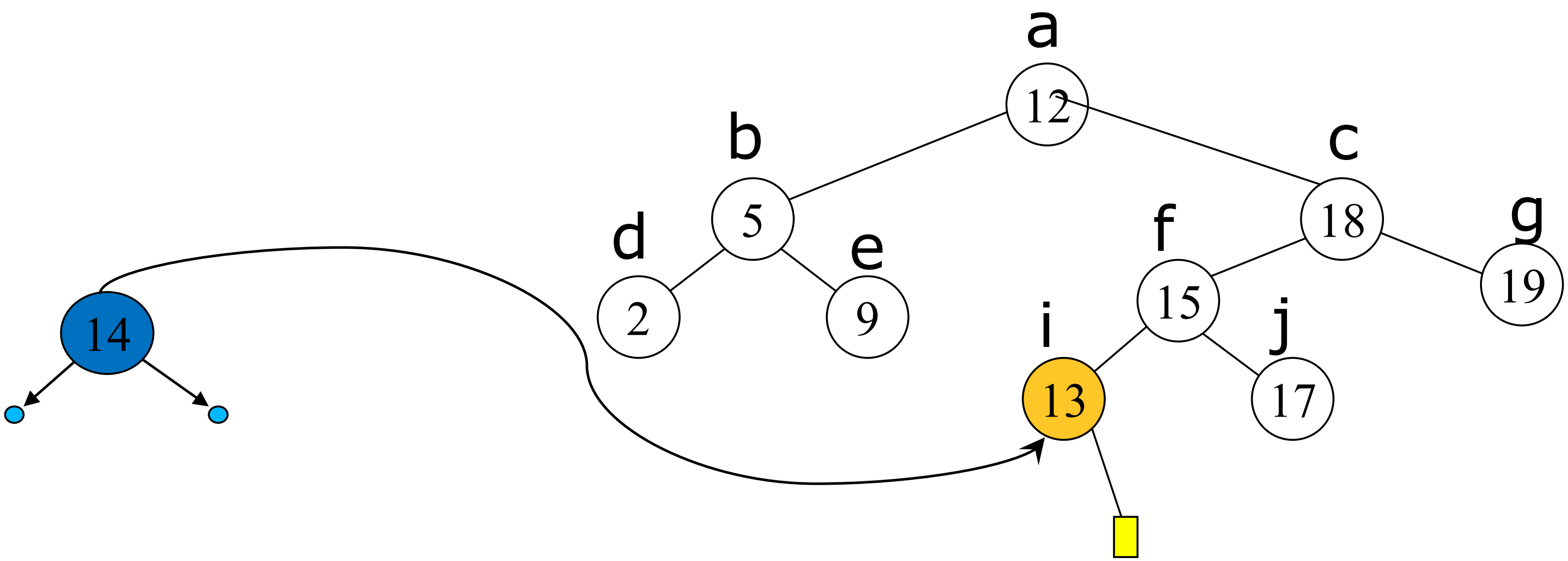




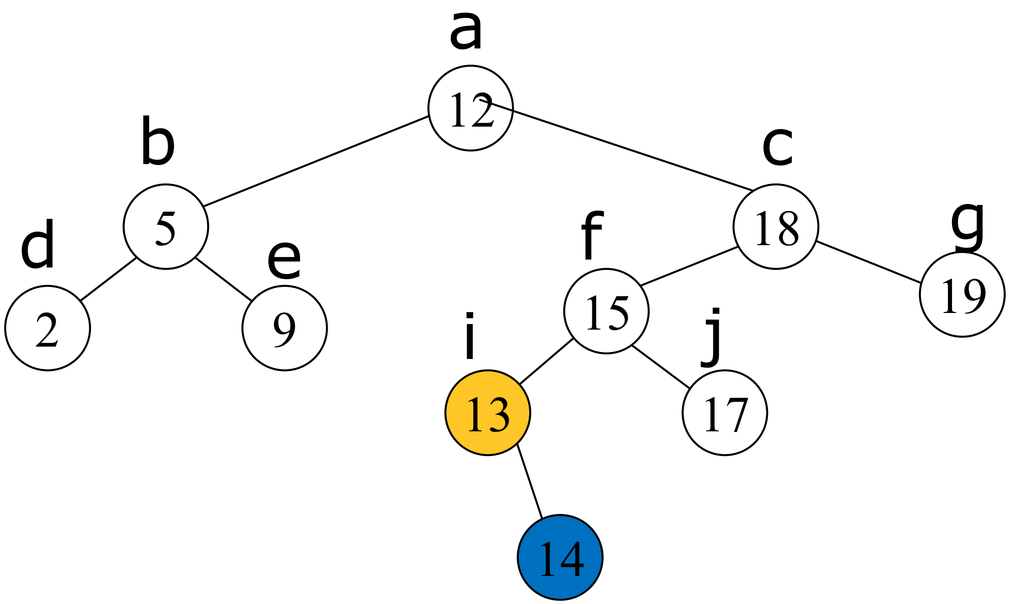
Tree-Insert Example: Insert a node with key 14



Tree-Insert Example: Insert a node with key 14



Tree-Insert Example: Insert a node with key 14



ASU®

Ira A. Fulton Schools of

Engineering

Arizona State University

Binary Search Trees, Part 4

- Binary Search Trees, Representation
- Tree Walks
- Search, Min, Max, Successor
- Insertion
- Deletion

BST Deletion: Case 1

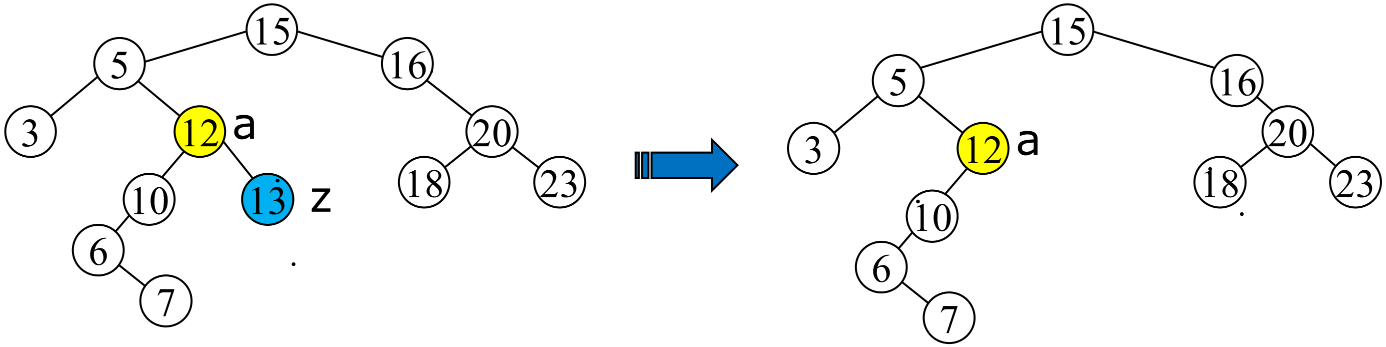
Deleting the node pointed to by z

(1) Both children of z are null:

Use null to replace z. Need to change pointer(s) at z' parent node. What happens if the parent is null? z is root. The tree becomes empty.

1a: When z->parent != null && z==z->parent.right

z->parent.right := null



BST Deletion: Case 1

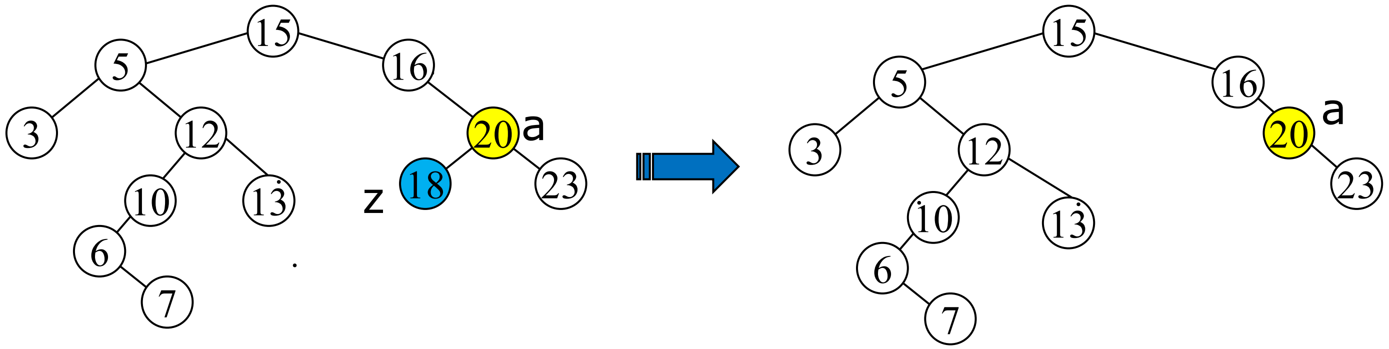
Deleting the node pointed to by z

(1) Both children of z are null:

Use null to replace z. Need to change pointer(s) at z' parent node. What happens if the parent is null? z is root. The tree becomes empty.

1b: When z->parent != null && z=z->parent.left

z->parent.left := null



BST Deletion: Case 1

Deleting the node pointed to by z

(1) Both children of z are null:

Use null to replace z. Need to change pointer(s) at z' parent node. What happens if the parent is null? z is root. The tree becomes empty.

1c: When z->parent == null

T.root:= null

BST Deletion: Case 2 (LL)

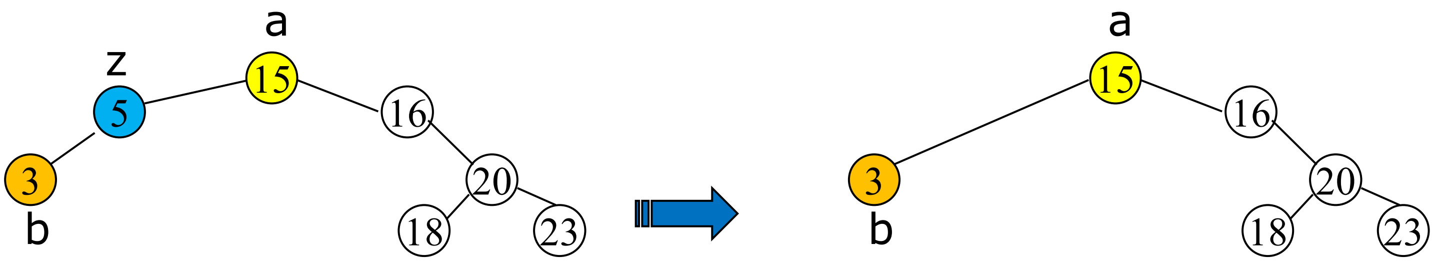
Deleting the node pointed to by z

(2) Exactly one child of z is null:

Change pointers at z's parent and non-null child. What happens if z's parent is null? z's non-null child becomes the root of the tree.

2a: z->parent != null && z==z->parent.left && z->left != null

z->parent.left := z->left; z->left.parent := z->parent;



BST Deletion: Case 2 (LR)

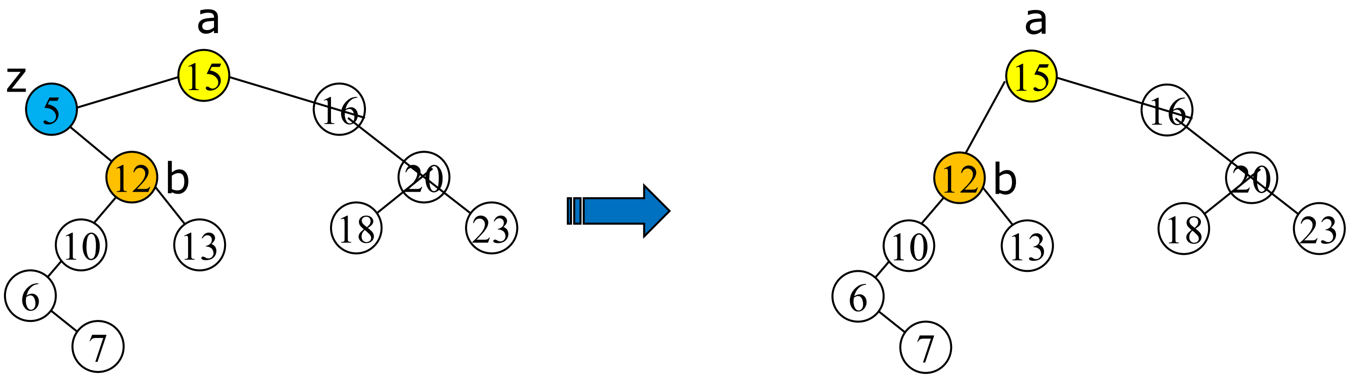
Deleting the node pointed to by z

(2) Exactly one child of z is null:

Change pointers at z’s parent and non-null child. What happens if z’s parent is null? z’s non-null child becomes the root of the tree.

2b: z->parent != null && z==z->parent.left && z->right != null

z->parent.left := z->right; z->right.parent := z->parent;



BST Deletion: Case 2 (RL)

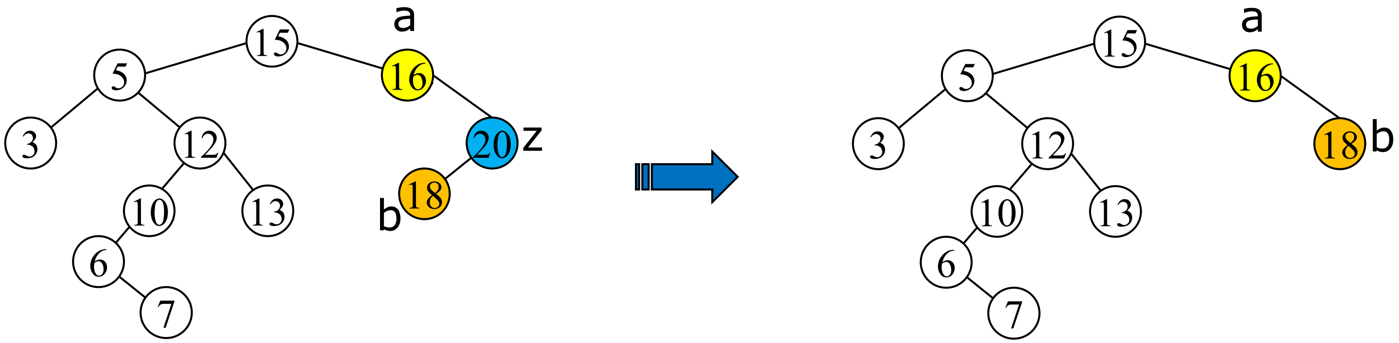
Deleting the node pointed to by z

(2) Exactly one child of z is null:

Change pointers at z’s parent and non-null child. What happens if z’s parent is null? z’s non-null child becomes the root of the tree.

2c: z->parent != null && z==z->parent.right && z->left != null

z->parent.right := z->left; z->left.parent := z->parent;



BST Deletion: Case 2 (RR)

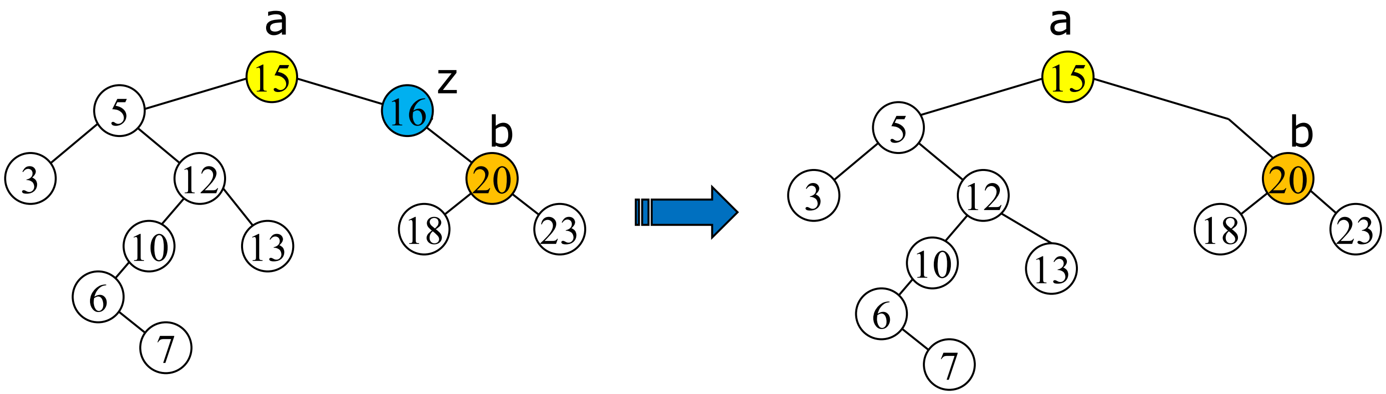
Deleting the node pointed to by z

(2) Exactly one child of z is null:

Change pointers at z’s parent and non-null child. What happens if z’s parent is null? z’s non-null child becomes the root of the tree.

2d: z->parent != null && z==z->parent.right && z->right != null

z->parent.right := z->right; z->right.parent := z->parent;



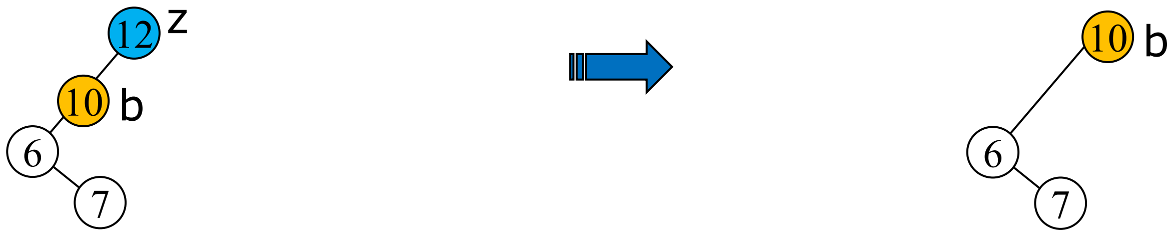
BST Deletion: Case 2 (NL)

Deleting the node pointed to by z

(2) Exactly one child of z is null:

Change pointers at z’s parent and non-null child. What happens if z’s parent is null? z’s non-null child becomes the root of the tree.

```
2e: z->parent == null && z->left != null
     T.root := z->left;
```



BST Deletion: Case 2 (NR)

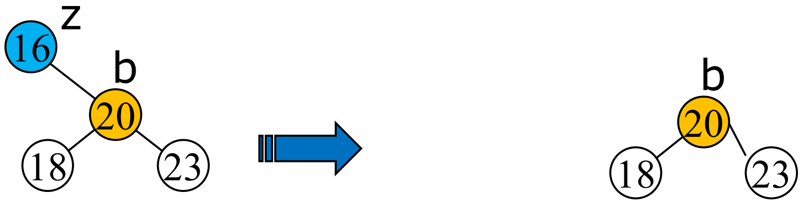
Deleting the node pointed to by z

(2) Exactly one child of z is null:

Change pointers at z's parent and non-null child. What happens if z's parent is null? z's non-null child becomes the root of the tree.

2f: z->parent == null && z->right != null

T.root:= z->right;



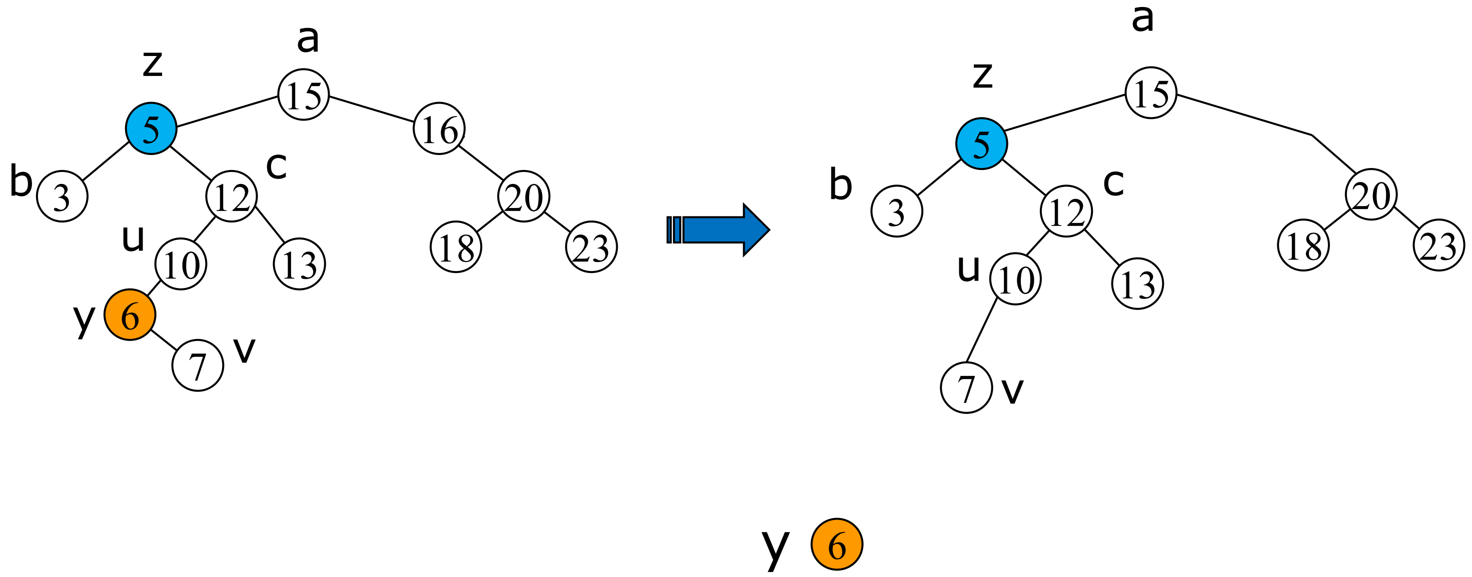
BST Deletion: Case 3

(3) No child of z is null:

Let y be the successor of z. Then the left child of y is null. Cut y out first.

But we deleted the wrong node.

We wanted to delete node z…



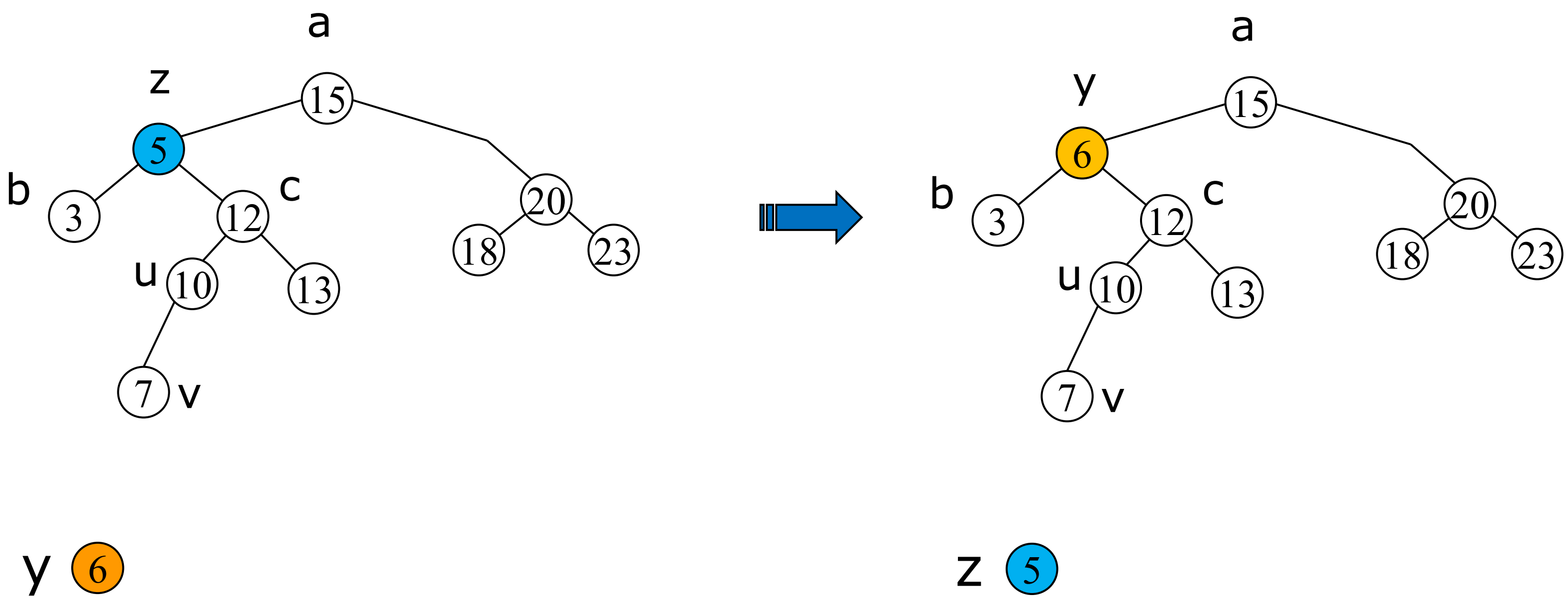
BST Deletion: Case 3

(3) No child of z is null:

Let y be the successor of z. Then the left child of y is null. Cut y out first.

But we deleted the wrong node.

How to fix it? Replace z by y.



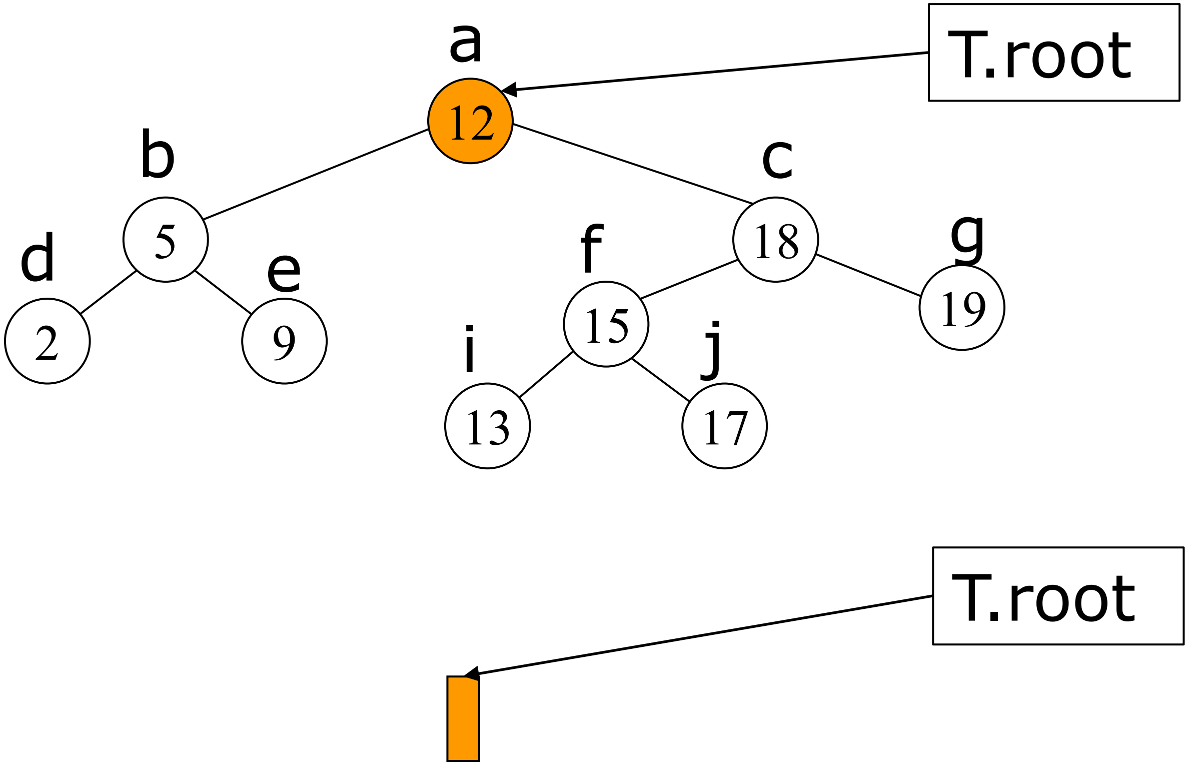
Tree Transplant

```
Transplant(T, u, v)
    if u.parent == null then
        T.root = v
    elseif u == u.parent.left
        u.parent.left = v
    else u.parent.right = v
    if $v \neq$ null then
        v.parent = u.parent
```

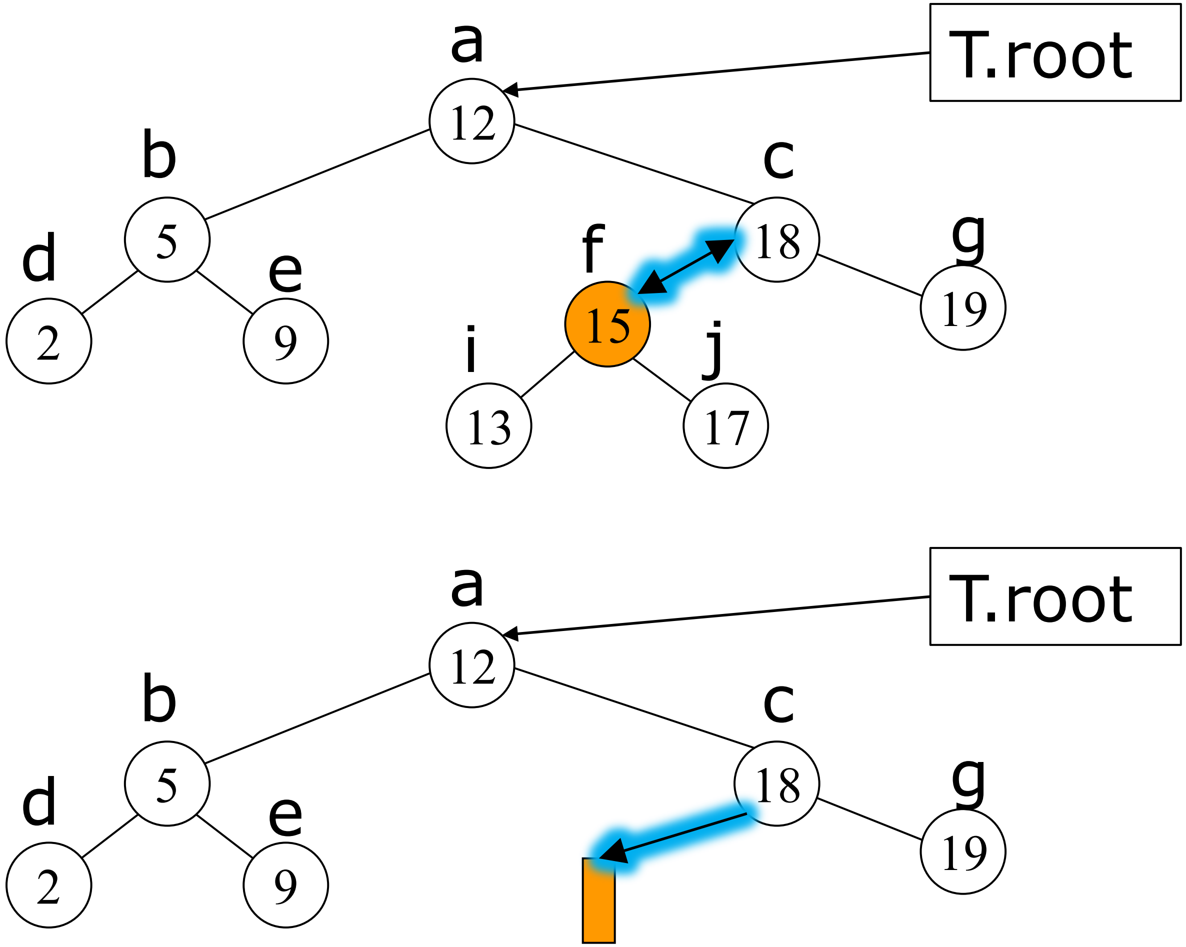
Condition: $T \neq$ null, $u \neq$ null, v is not an ancestor of u. Note that v can be null

Effect: Replaces the subtree rooted at u with the subtree rooted at v.

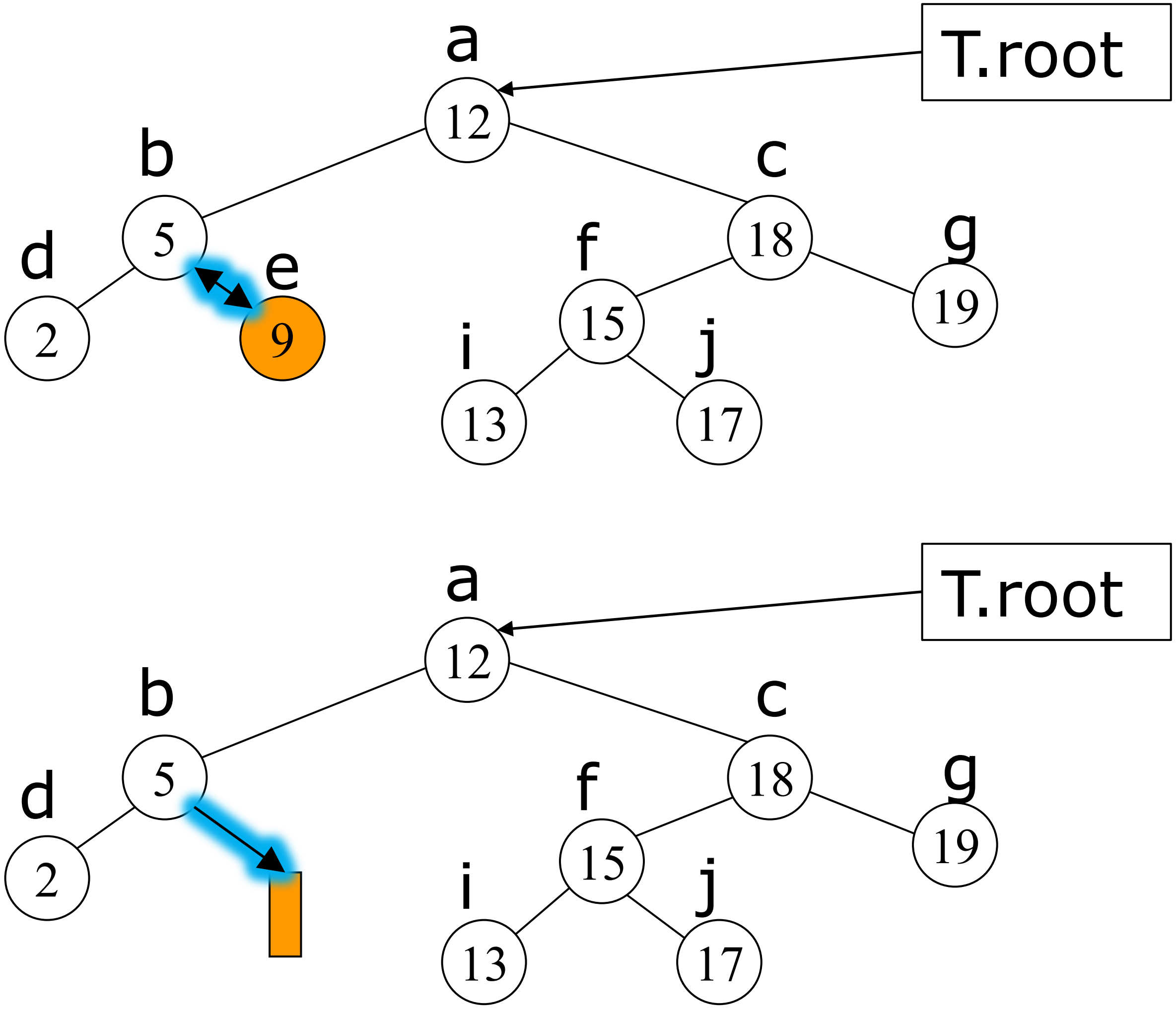
Example: Transplant(T, a, null)



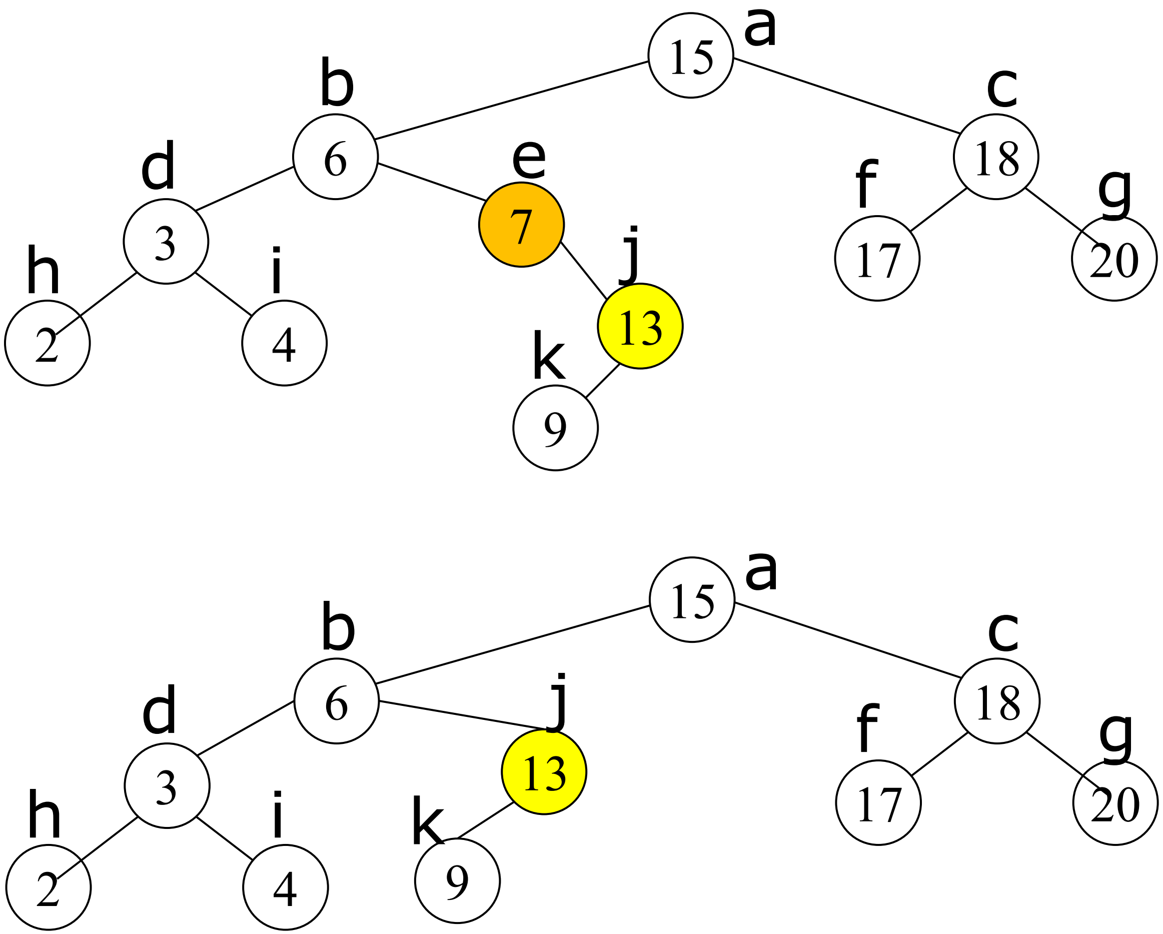
Example: Transplant(T, f, null)



Example: Transplant(T, e, null)



Example: Transplant(T, e, e.right)



BST Deletion

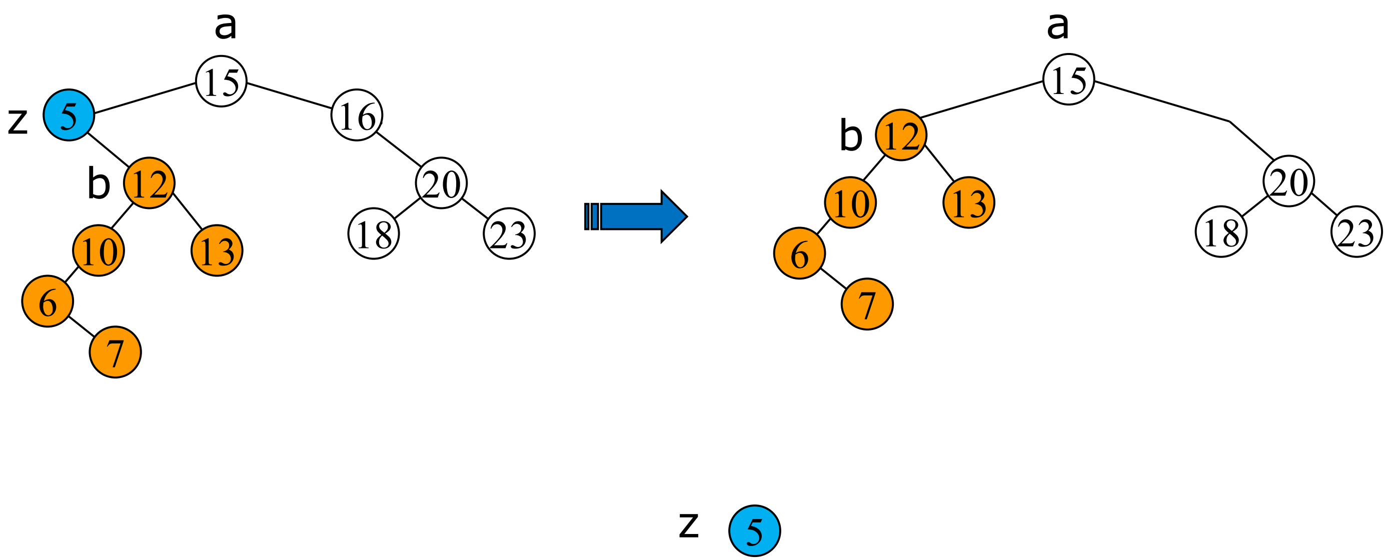
```
Tree-Delete(T, z)
if z.left == null then
    Transplant(T, z, z.right)
elseif z.right == null then
    Transplant(T, z, z.left)
else
    y = Tree-Minimum(z.right)
    if y.parent ≠ z then
        Transplant(T, y, y.right)
        y.right = z.right
        z.right.parent = y
    Transplant(T, z, y)
    y.left = z.left
    z.left.parent = y
```

Running time: O(tree height)

BST Deletion: Case of Line 2

Left child of z is null:

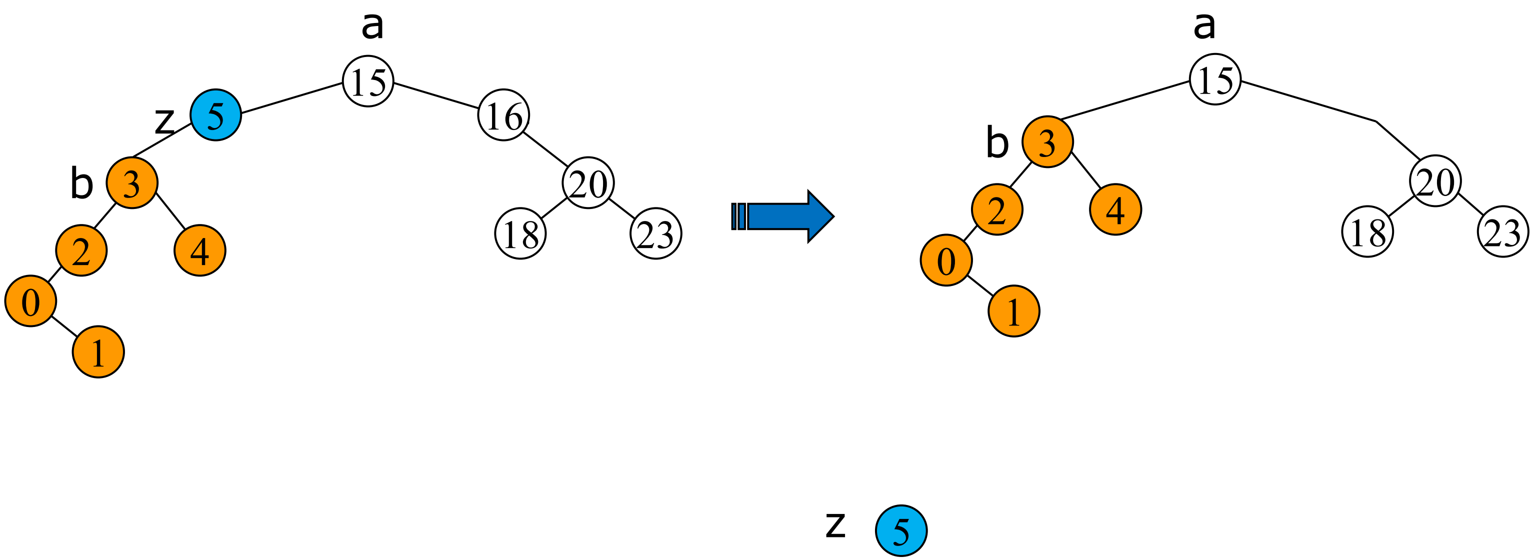
Note that the parent field of (deleted) z is still a, and the right child of z is still b.
These values will never be used again.



BST Deletion: Case of Line 4

Right child of z is null:

Note that the parent field of (deleted) z is still a, and the left child of z is still b.
These values will never be used again.



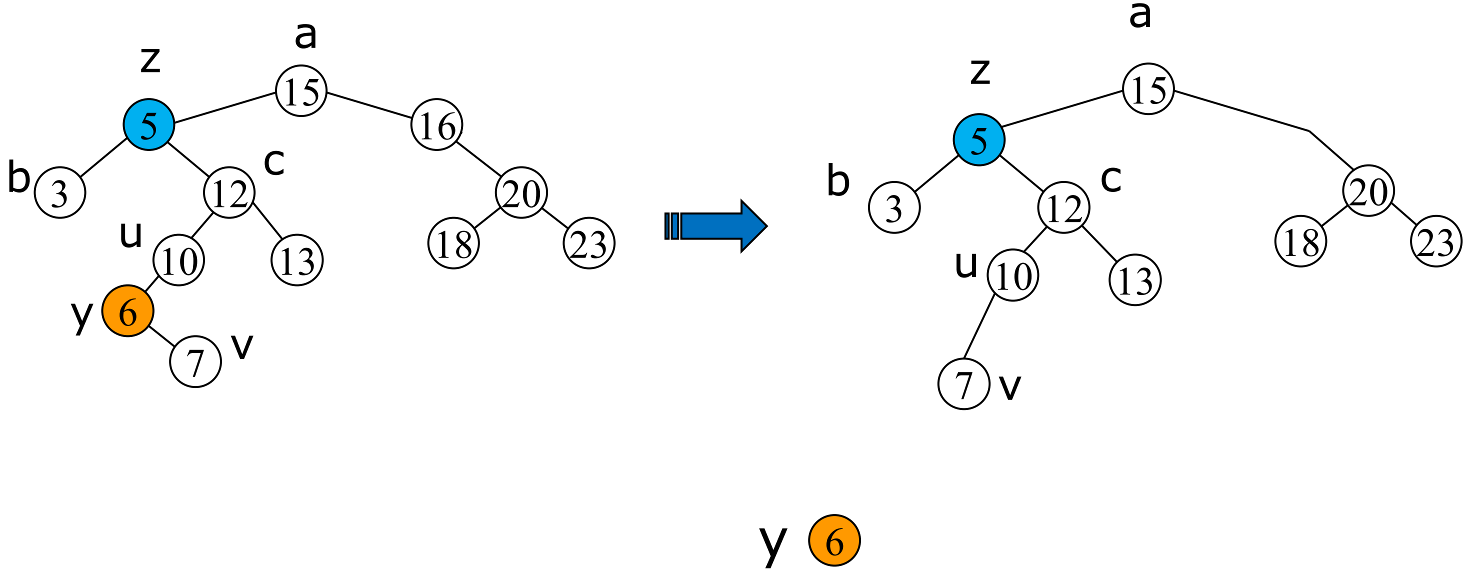
BST Deletion: Line 5

(3) No child of z is null:

Let y be the successor of z. Then the left child of y is null. Cut y out first.

But we deleted the wrong node.

We wanted to delete node z…



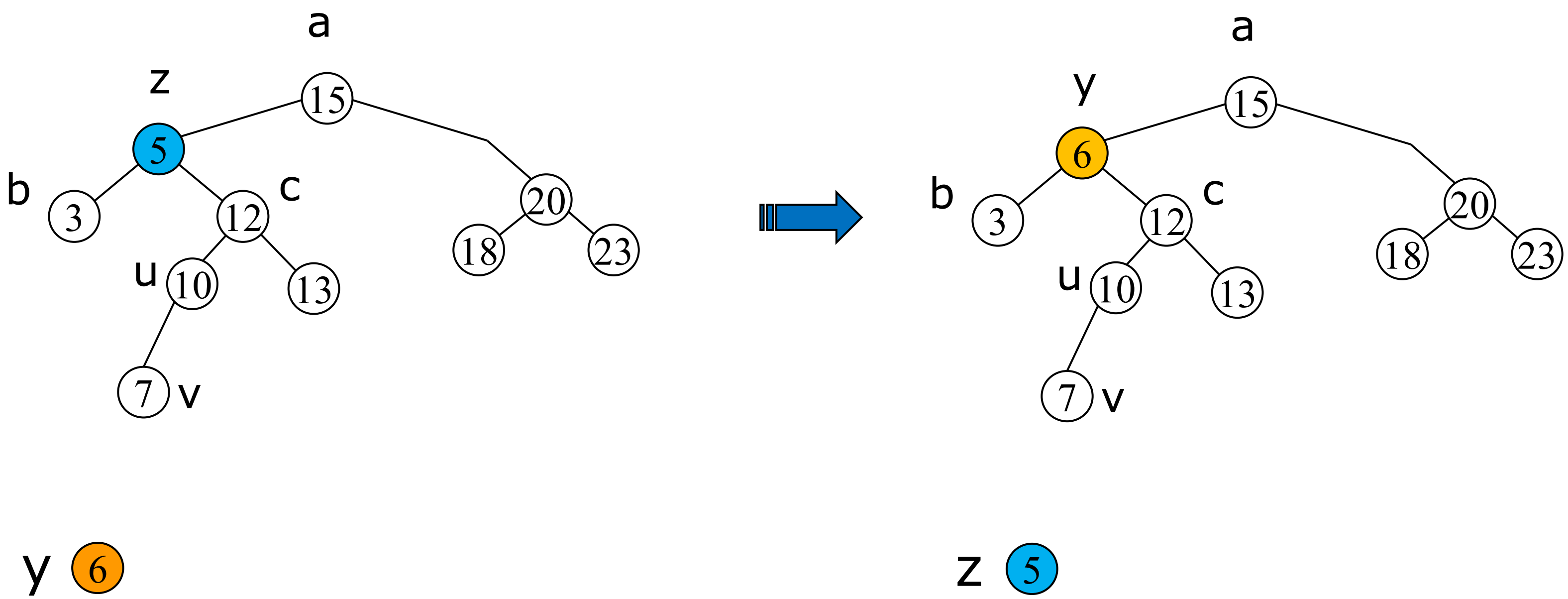
BST Deletion: Line 5

(3) No child of z is null:

Let y be the successor of z. Then the left child of y is null. Cut y out first.

But we deleted the wrong node.

How to fix it? Replace z by y.



ASU®

Ira A. Fulton Schools of Engineering

Arizona State University