

Trees 6A

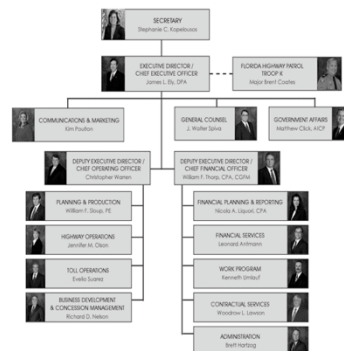
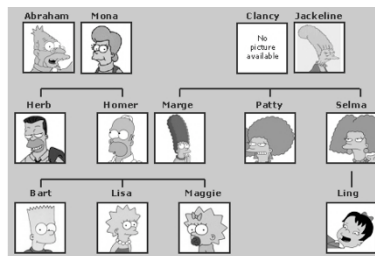
Binary Trees



15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

1

Trees



15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

Binary Trees

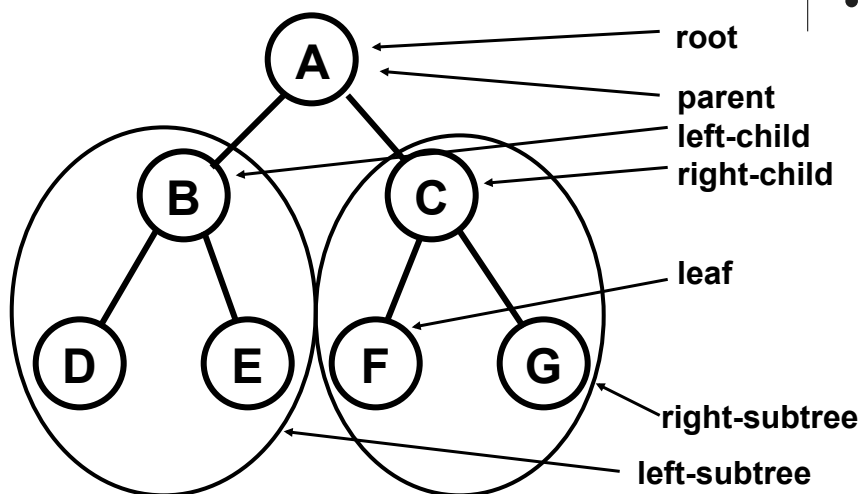


- A binary tree is either empty or it contains a root node and left- and right-subtrees that are also binary trees.
 - A binary tree is a nonlinear data structure.
- The top node of a tree is called the root.
- Any node in a binary tree has at most **2** children.
- Any node in a binary tree has exactly one parent node (except the root).

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

3

Tree Terminology



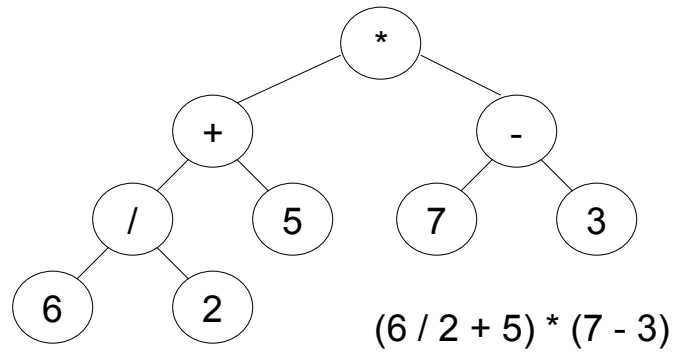
15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

4

Types of Binary Trees



Expression Trees



15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

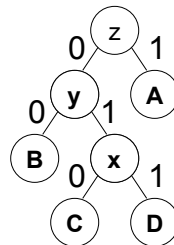
5

Types of Binary Trees



Huffman Trees

A	45%
B	30%
C	20%
D	5%



A	1
B	00
C	010
D	011

1001010100 = ABACAB

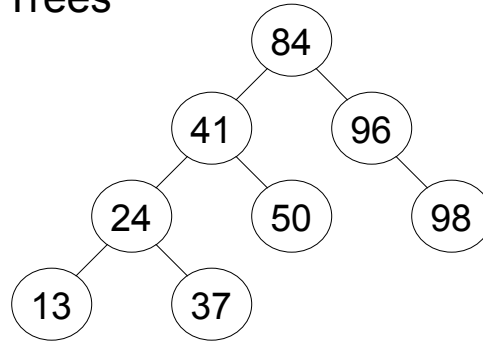
15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

6

Types of Binary Trees



Binary Search Trees



15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

7

More Terminology

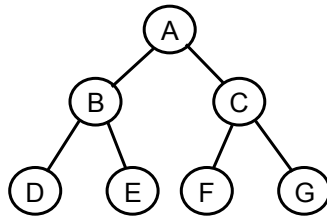


- A full binary tree is a binary tree such that
 - all leaves have the same level, and
 - every non-leaf node has 2 children.
- A complete binary tree is a binary tree such that
 - every level of the tree has the maximum number of nodes possible except possibly the deepest level, and
 - at the deepest level, the nodes are as far left as possible.

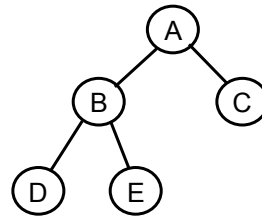
15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

8

Examples



FULL
(and COMPLETE)



COMPLETE

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

9

Binary Trees & Recursion



- Consider two nodes in a tree, X and Y.
- X is an ancestor of Y if
 - X is the parent of Y, or
 - X is the ancestor of the parent of Y.

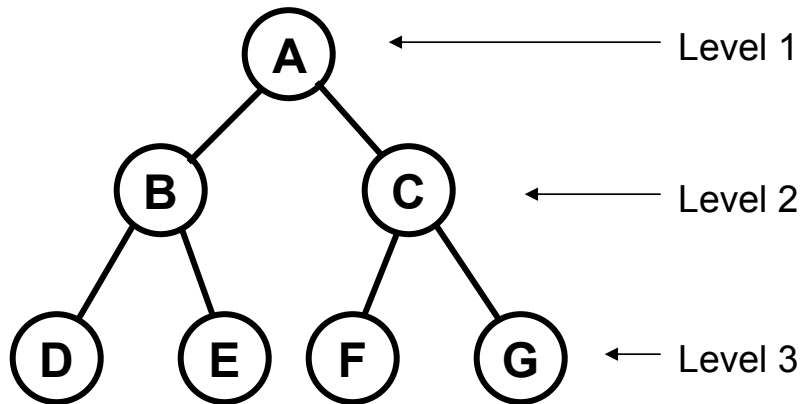
It's RECURSIVE!

- X is a descendant of Y if

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

10

Binary Trees Levels



15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

11

Binary Trees - Levels



- The level of a node Y is
 - BASE CASE
 - RECURSIVE CASE

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

12

Binary Tree - Height

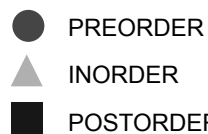


- The height of a binary tree T is the number of nodes in the longest path from the root node to a leaf node.
 - BASE CASE
 - RECURSIVE CASE

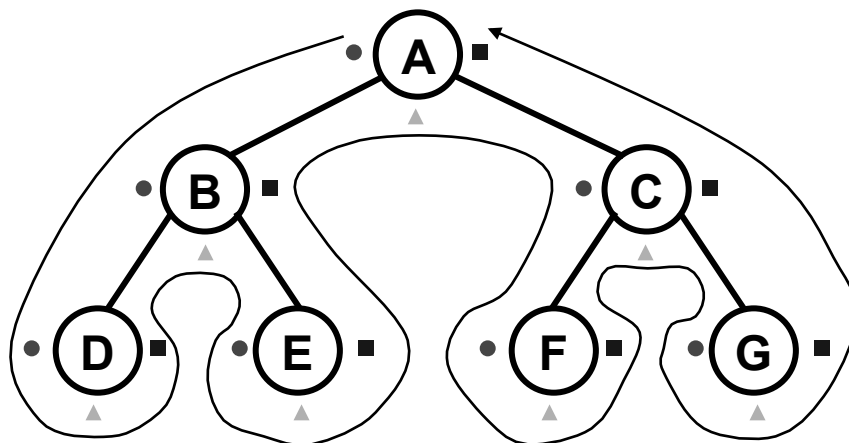
15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

13

Binary Tree Traversals



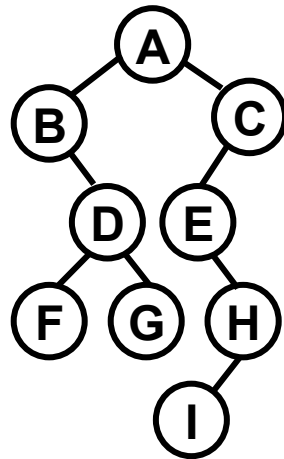
PREORDER	ABDECFG
INORDER	DBEAFCG
POSTORDER	DEBFGCA



15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

14

Traversal Example



preorder

inorder

postorder

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

15

Traversals are Recursive

- Preorder traversal
 1. Visit the root.
 2. Perform a preorder traversal of the left subtree.
 3. Perform a preorder traversal of the right subtree.
- Inorder traversal
 1. Perform an inorder traversal of the left subtree.
 2. Visit the root.
 3. Perform an inorder traversal of the right subtree.
- Postorder traversal
 1. Perform a postorder traversal of the left subtree.
 2. Perform a postorder traversal of the right subtree.
 3. Visit the root.

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

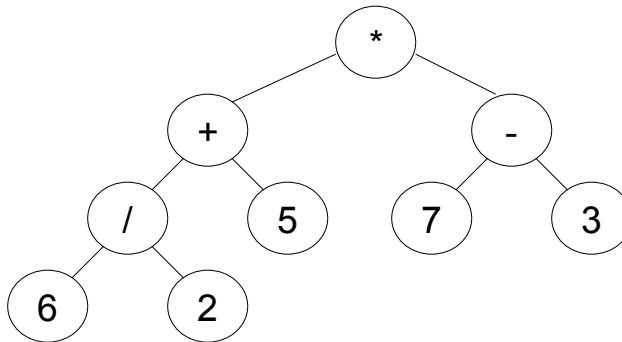
16

Traversals

on Expression Trees



- What do you get when you perform a postorder traversal on an expression tree?



15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

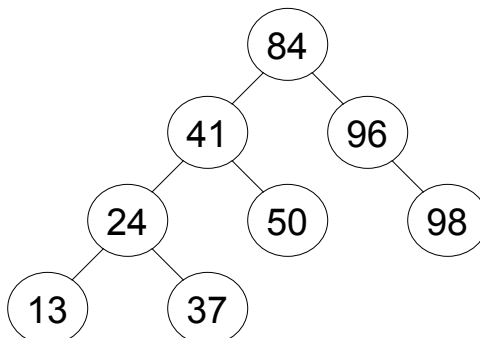
17

Traversals

on Binary Search Trees



- What do you get when you perform an **INORDER** traversal on a binary search tree?



15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

18

Implementing a binary tree



- Use an array to store the nodes.
 - mainly useful for complete binary trees (more on this soon)
- Use a variant of a linked list where each data element is stored in a node with links to the left and right children of that node
- Instead of a head reference, we will use a root reference to the root node of the tree.

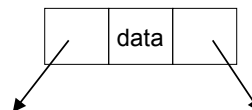
15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

19

Binary Tree Node (BTNode inner class)



```
private static class BTNode<E> {  
    private E data;  
    private BTNode<E> left;  
    private BTNode<E> right;  
  
    public BTNode(E element) {  
        data = element;  
        left = null;  
        right = null;  
    }  
}
```



15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

20

BinaryTree class



```
public class BinaryTree<E> {  
  
    private BTreeNode root;  
  
    public BinaryTree() {  
        root = null;  
    }  
}
```

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

21

BinaryTree class



```
public BinaryTree(E element,  
    BinaryTree<E> leftTree,  
    BinaryTree<E> rightTree) {  
    root = new BTreeNode<E>(element);  
    if (leftTree != null)  
        root.left = leftTree.root;  
    if (rightTree != null)  
        root.right = rightTree.root;  
}
```

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

22

BinaryTree class

```
public BinaryTree<E> getLeftSubtree() {  
    if (root != null && root.left != null)  
        return new BinaryTree<E>(root.left);  
    else  
        return null;  
}  
  
protected BinaryTree(BTNode<E> rootRef) {  
    root = rootRef;  
}
```

short-circuit evaluation

special constructor for this method only

same idea for right subtree

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

23

Preorder Traversal

```
public String preorder() {  
    return preorder(root);  
}  
  
private String preorder(Node<E> node) {  
    String result = "";  
    if (node != null) {  
        result += node.data + " ";  
        result += preorder(node.left) + " ";  
        result += preorder(node.right) + " ";  
    }  
    return result;  
}
```

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

24

Binary Search Trees

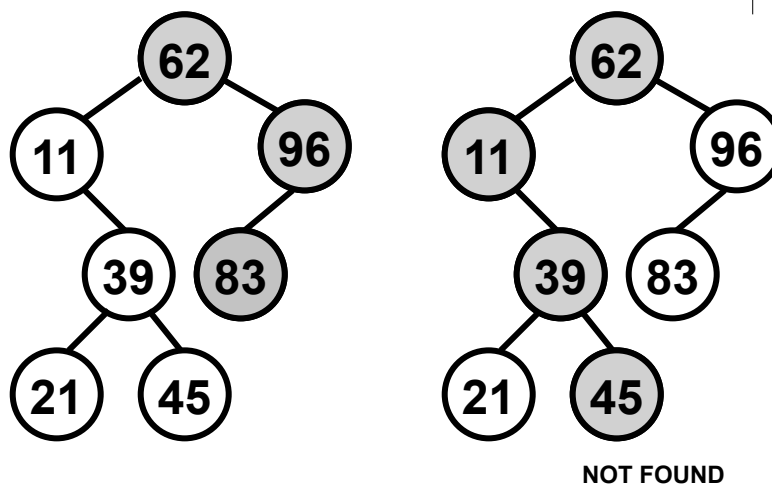


- A binary tree T is a binary search tree if
 - T is empty, or
 - T has two subtrees, T_L and T_R , such that
 - All the values in T_L are less than the value in the root of T ,
 - All the values in T_R are greater than the value in the root of T , and
 - T_L and T_R are binary search trees

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

25

Searching for 83 and 42



15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

26



Finding data in a BST

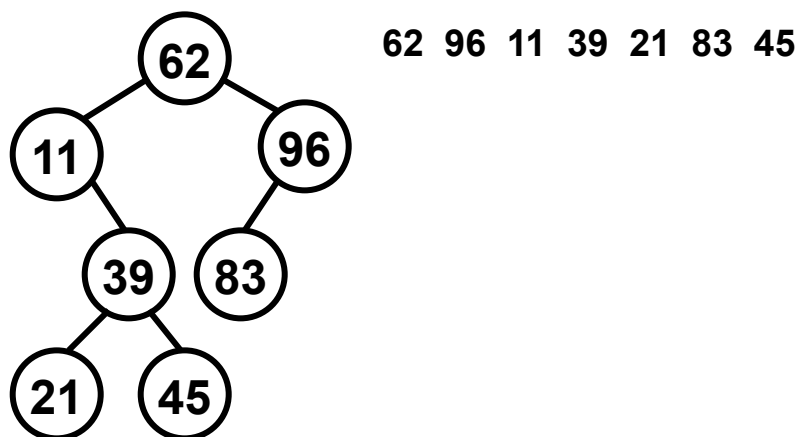
- If the root is null, return null.
- Compare the root value to the target.
- If they are equal, return a reference to the data in the root.
- Otherwise if the target is less than the root value, return the result of the search on the left subtree.
- Otherwise (since the target is greater than the root value), return the result of the search on the right subtree.

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

27



Inserting into a BST



15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

28

Inserting data into a BST



- If the root is null, replace empty tree with new tree containing the new element and return true.
- If the item is equal to the root data, return false.
- If the item is less than the root data, insert the new element into the left subtree and return the result of that insert operation.
- If the item is greater than the root data, insert the new element into the right subtree and return the result of that insert operation.

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

29

Efficiency of insert on a full BST



- A full binary search tree with height H has how many nodes?
 - $N = 1 + 2 + 4 + \dots + 2^{H-1} = 2^H - 1$
 - Thus, $H = \log_2(N+1)$.
- An insert will take $O(\log N)$ time on a full BST since we have to examine one node at each level in the worst case before we find the insert point, and there are H levels.

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

30

Efficiency of insert on a BST



- Are all BSTs full? **NO!**
Insert the following numbers in order into a BST: 11 21 39 45 62 83 96
What do you get?
- An insert will take $O(N)$ time in the worst case on an arbitrary BST since there may be up to N levels in a BST with N nodes.
 - We will take a brief look at balanced search trees later this semester.

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

31

Removing from a BST



General idea:

- Start at the root and search for the item to remove by progressing one level at a time until either:
 - the item is found
 - we reach a leaf and the item is not found
- If the item is found, remove the node and repair the tree so it is still a BST.

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

32

Removing from a BST



- If the root is null, return null.
- Compare the item to the root data.
- If the item is less than the root data, return the result of removing the data from the left subtree.
- If the item is greater than the root data, return the result of removing the data from the right subtree.

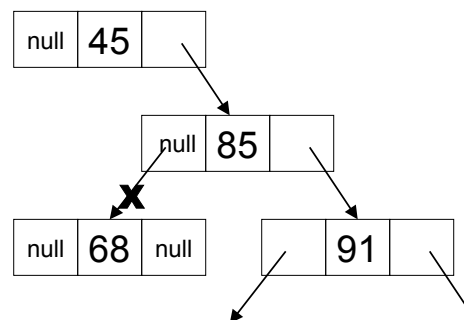
(cont'd)

Removing from a BST

Data value is found at local root



- Store the local root data in a temp variable.
- If the local root has no children, set the parent reference to this root to null.
- Example:
Remove 68

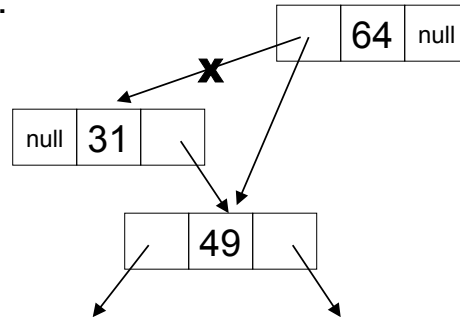


Removing from a BST

Data value is found at root (cont'd)



- If the local root has one child, set the parent reference to this root to the reference to this root's only child.
- Example:
Remove 31.



15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

35

Removing from a BST

Data value is found at root (cont'd)



- If the local root has two children:
 - If this root's left child has no right child
 - Copy the data from this root's left child into this root.
 - Set the left reference of this root to the reference to the left child of this root's left child.
 - Otherwise:
 - Find the rightmost node in the right subtree of this root's left child (a.k.a. the inorder predecessor).
 - Copy the data from the rightmost node from the previous step into this root's data field and then remove the original data by setting its parent to reference its left child.

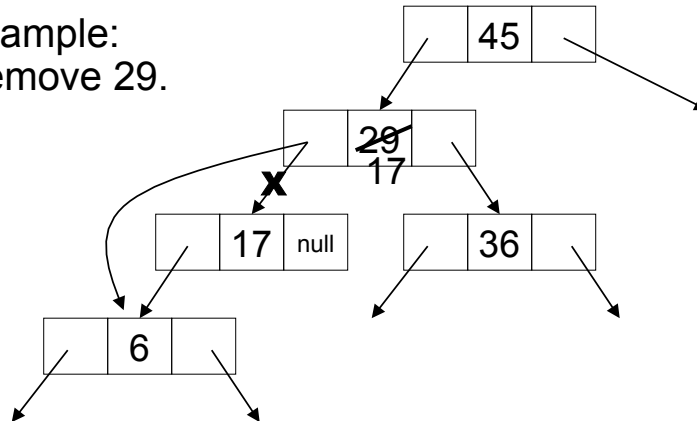
15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

36

Removing from a BST

Data value is found at root (cont'd)

- Example:
Remove 29.



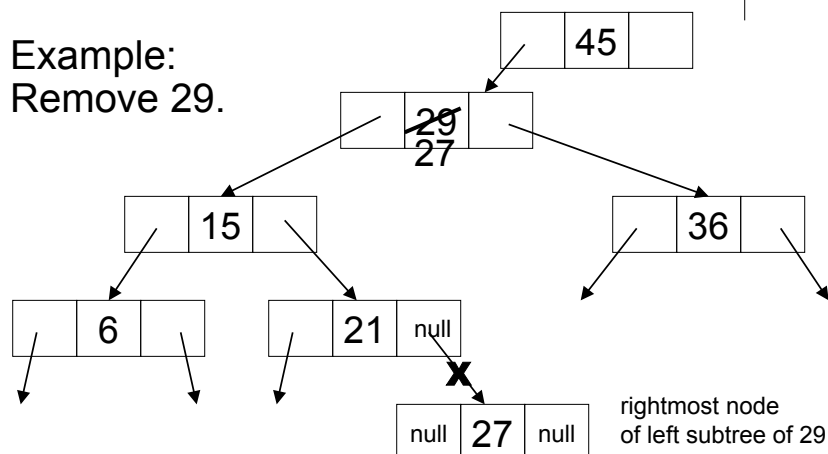
15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

37

Removing from a BST

Data value is found at root (cont'd)

- Example:
Remove 29.



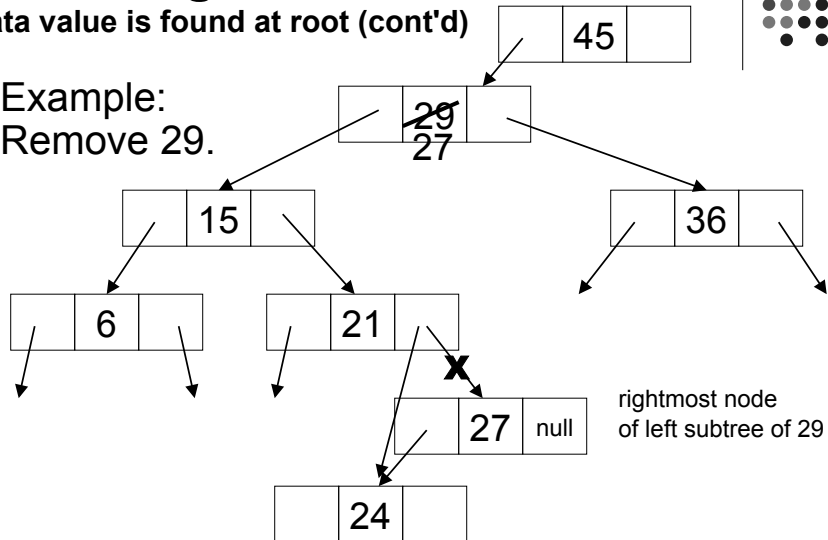
15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

38

Removing from a BST

Data value is found at root (cont'd)

- Example:
Remove 29.



15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

39

Removing from a BST

Data value is found at root (cont'd)

- If we remove the node referenced by our local root variable, how do we change the left or right field of its parent (if this node does not have a pointer back to the parent)?
 - Solution: Return a reference to the replacement reference (or null) as the result and let the previous invocation store the result in its local root node (which is the parent of the root node we want to remove).

15-121 Introduction to Data Structures, Carnegie Mellon University - CORTINA

40