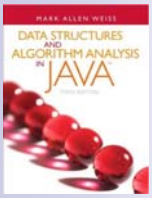
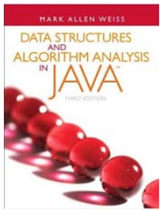


FIU FLORIDA INTERNATIONAL UNIVERSITY

Trees (III)

Dr. Antonio L. Bajuelos

FIU School of Computing & Information Sciences



COP-3530 - Data Structures

Module #3: Trees (part III)

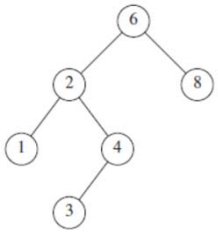
Outline:

- **Binary Search Trees (BST).**
- **Main operations on BST:**
  - **Insertion / Add.**
  - **Remove / Del.**
  - **Find.**
  - **Complexity analysis.**
- **Java implementation of BST.**
- **Build a BST.**
- **An application of BST.**

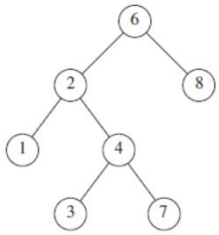
2

Binary Search Trees (BST)

- For simplicity, let assume that each node in the tree stores an integer and also assume that all the items are distinct.
- **Definition:** A **binary tree** is a **binary search tree** is that for every node, ***X***, in the tree:
  I. the values of all the items in its **left subtree are smaller than the item in *X***, and,
  II. the values of all the items in its **right subtree are larger than the item in *X***.
- **Examples:**



Binary Search Tree



Binary Tree

3

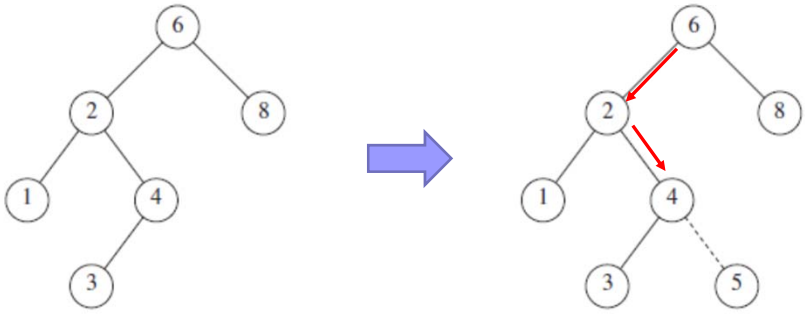
Binary Search Trees. Insertion

- To **insert/add** *x* into **BST** T proceed down the tree as you would with a contains.

  (i) If *x* is found, do nothing (or "update" something).

  (ii) Otherwise, insert *x* at the last spot on the path traversed.

- **Example:** insert(5)



4

Binary Search Trees. Insertion Java Code

```
/**
 * Internal method to insert into a subtree.
 * @param x the item to insert.
 * @param t the node that roots the subtree.
 * @return the new root of the subtree.
 */
private BinaryNode<AnyType> insert( AnyType x, BinaryNode<AnyType> t )
 {
    if( t == null )
      return new BinaryNode<>( x, null, null );
    int compareResult = x.compareTo( t.element );
    if( compareResult < 0 )            //goto left subtree
        t.left = insert( x, t.left );
    else if( compareResult > 0 )     //goto right subtree
        t.right = insert( x, t.right );
    else
        ; // Duplicate; do nothing
    return t;
}
```

5

Binary Search Trees. Remove

- As is common with many data structures, the hardest operation is deletion.

(i) If the node is a leaf, it can be deleted immediately.

(ii) If the node has one child, the node can be deleted after its parent adjusts a link to bypass the node

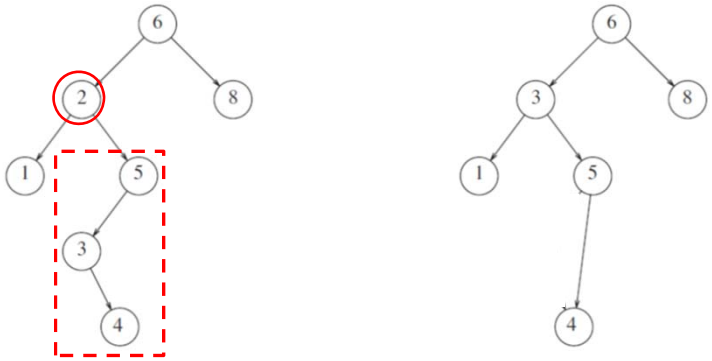
- **Example:** Case (ii): remove(4)

6

Binary Search Trees. Remove

- (iii) If the node has two children.
  - Replace the data of this node with the smallest data of the right subtree (which is easily found)
  - Recursively delete that node (which is now empty).

- **Example:** Case (iii): remove(2)

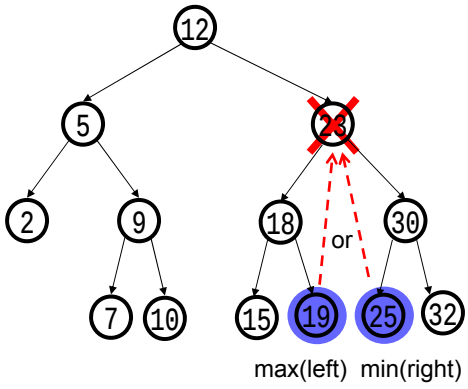


7

Binary Search Trees. Remove

- (iii) If the node has two children.
  - Replace the data of this node with the smallest data of the right subtree (which is easily found)
  - Recursively delete that node (which is now empty).

- **Example (alternative):** Case (iii): remove(23)



8

Binary Search Trees. Remove Java Code

```
/**
* Internal method to remove from a subtree.
 * @param x the item to remove.
* @param t the node that roots the subtree.
* @return the new root of the subtree.
*/
private BinaryNode<AnyType> remove( AnyType x, BinaryNode<AnyType> t )
{
    if( t == null )
        return t; // Item not found; do nothing
    int compareResult = x.compareTo( t.element );
    if( compareResult < 0 )
        t.left = remove( x, t.left );
    else if( compareResult > 0 )
        t.right = remove( x, t.right );
    else if( t.left != null && t.right != null ) // Two children
    {
        t.element = findMin( t.right ).element;
        t.right = remove( t.element, t.right );
    }
    else
        t = ( t.left != null ) ? t.left : t.right;
    return t;
}
```

9

Binary Search Trees. Java Code

- See **Java** Code implementation of **Binary Search Tree** in:

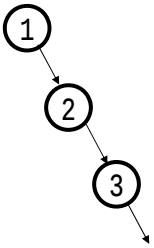
http://users.cis.fiu.edu/~weiss/cop3530_sum09/Day13.java

(Prof. Mark Weiss Implementation)

10

Insert & Remove. Worst case running time.

- For the **BST – Find, Insert** and **Remove** are **O(N) (worst case)**
- **Example:**



11

Building a BST (buildBST)

- Suppose we need to build a **BST** from de following ordered sequence of integers:

  1, 2, 3, 4, 5, 6, 7, 8, 9

- How to insert this sequence into an **empty BST**?
  - **First option:** insert the elements in a given order.

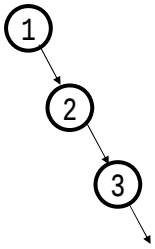
    What's the tree?

    Not a good tree!

    we want to avoid height = *N* (max height)

    What the running time?

    $O(N^2)$



  - **Second option:** insert the elements in the reverser order?

12

Building a BST (buildBST)

- Suppose we need to build a **BST** from de following ordered sequence of integers:

  1, 2, 3, 4, 5, 6, 7, 8, 9

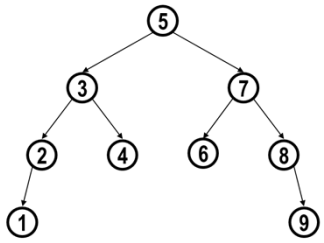
- How to insert this sequence into an **empty BST**?
  - **Third option:** re-arrange elements: median first, then left median, right median, etc. (**5, 3, 7, 2, 1, 4, 8, 6, 9**)

What's the tree?

Good tree! (balanced tree)

What the running time?

O(NlogN) for building the **BST**!!!



- **Important:** The expected depth of any node is **O(logN)**

13

Applications of Binary Trees (BT) and BST

- A **BST** is a prominent data structure used in many systems programming applications for representing and managing dynamic sets.
- The **Average case** complexity of **Find**, **Insert**, and **Remove** operations is **O(log*N*)**, where *N* is the number of nodes in the tree.
- **Examples:**
  - A **Binary Search Partition** is used in almost every 3D video game to determine what objects need to be rendered.
  - **BST** is used in Unix kernels for managing a set of virtual memory areas.
  - **BST** may also be used to solve some of database problems, for example, indexing.
  - . . .

14


14