

Introduction to Programming Methods



Last Time

Intro to data structures

- how to store your data in a convenient form
 - arrays, linked lists
 - started trees too
- Creation of a Small Tree

Creation of a Small Tree

```
TreeNode Integer root = new TreeNode(Integer(new Integer(20))
TreeNode Integer root.setLeft(
    new TreeNode(Integer(new Integer(5))
        new TreeNode(Integer(new Integer(9))) // left
        new TreeNode(Integer(new Integer(12))) // right
    );
```

Small tree, of "height" 3

- length of longest path from root

```
public int heightOfBinaryTree(TreeNode T) {
    if (node == null) return 0;
    else
        return 1 + Math.max(heightOfBinaryTree(node.getLeft()),
            heightOfBinaryTree(node.getRight()));
}
```

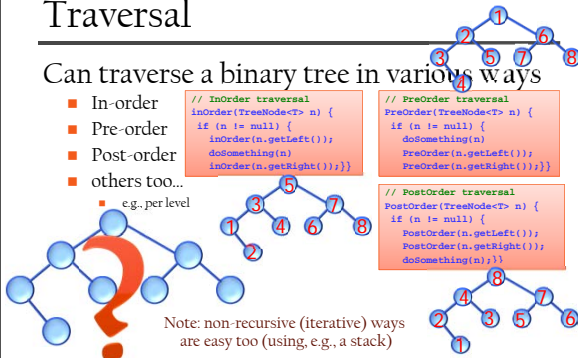


Traversal

Can traverse a binary tree in various ways

- In-order
- Pre-order
- Post-order
- others too...
 - e.g., per level

- e.g., per level



Note: non-recursive (iterative) ways
are easy too (using, e.g., a stack)

Balanced Binary Search Trees (AVL)

A particularly good type of tree if you want:

- to store n values (records) in $O(n)$
- to search, insert, delete in $O(\log n)$
 - obviously, a simple linked list won't do
 - ... but we discussed “binary search” earlier
 - data structure inspired by this algorithm
 - » keep data ordered
 - » keep data in log n levels
- note: will assume no two (or more) elements the same to avoid complication

Two Definitions

Binary search tree

- for any node n in the tree:
 - nodes in left subtree of n contain items less than the item of n
 - nodes in right subtree of n contain items greater than the item of n



Balanced tree

- for any node n:
 - $\text{height}(\text{left}) = \text{height}(\text{right}) (\pm 1)$



Balanced binary search tree

- you guessed it...

Search in Binary Search Tree

Start at the root, and visit subtrees

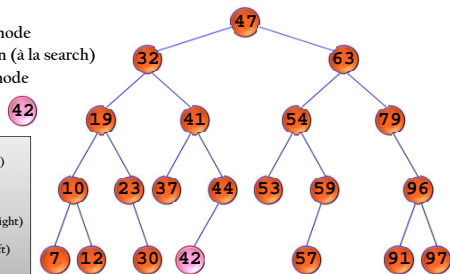
- cost?

```
TreeNode<T> findNode(TreeNode<T> n, T value)
// returns a node containing the item containing "value"
// or null if not found
{
    if (n == null) { return null; }
    else {
        // if found, return n
        if (value == n.getItem()) { return n; }
        // otherwise search the left or right subtree
        else if (value < n.getItem())
            { return findNode(n.getLeft(), value); }
        else { return findNode(n.getRight(), value); } // end if
    } // end if
} // end findNode
```

Insertion in Binary Search Tree?

Insert 42?
create new node
find position (à la search)
insert new node

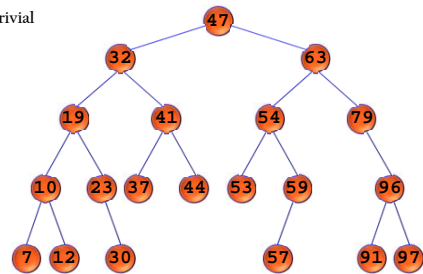
```
Pseudocode
Insert(Value x, Node n)
if (n == null)
    return new Node(x)
if (x < n.item)
    n.left = insert(x, n.left)
else if (x > n.item)
    n.right = insert(x, n.right)
return n
```



source of BST examples: J. Manuch

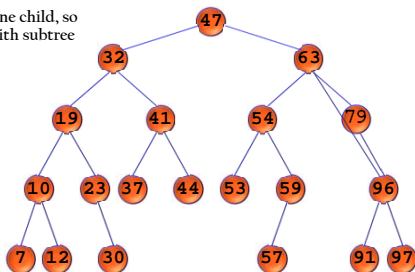
Deletion in BST – Case 1

Delete 30?
Leaf, so trivial



Deletion in BST – Case 2

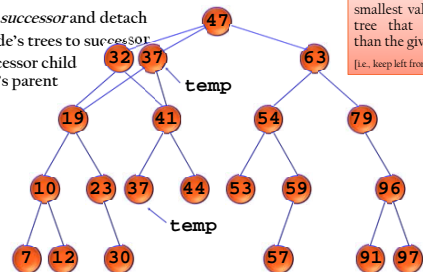
Delete 79?
Easy too: one child, so
replace with subtree



Deletion in BST – Case 3(a)

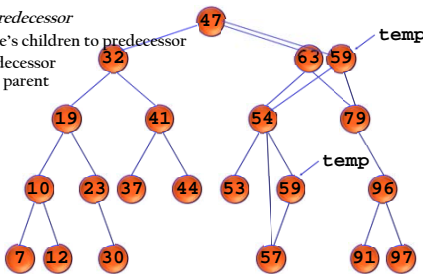
Delete 32?
first, find *successor* and detach
attach node's trees to successor
make successor child
of node's parent

The *successor* is the
smallest value in the
tree that is larger
than the given value.
[i.e., keep left from right child]



Deletion in BST – Case 3(b)

Delete 63?
shortcut *predecessor*
attach node's children to predecessor
attach predecessor
to node's parent



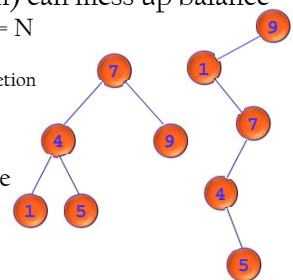
Now, What About Balancing?

Insertion (or deletion) can mess up balance

- worst case: height = N
- bad news for cost of search/insertion/deletion

Run balancing after
each operation

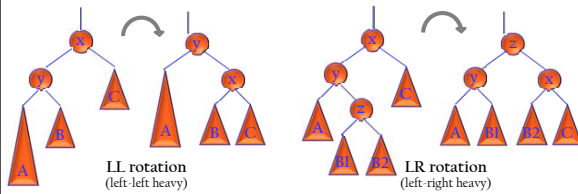
- to maintain balance
- in $O(\log N)$



Rebalancing

(Recursively) Check and correct balance

- only need to perform “rotations”
- important: does not alter BST property!



CS2 - INTRODUCTION TO PROGRAMMING METHODS

13

Rebalancing Pseudocode

```

IF tree is right heavy
  IF tree's right subtree is left heavy
    Perform RL rotation
  ELSE
    Perform RR rotation
ELSE IF tree is left heavy
  IF tree's left subtree is right heavy
    Perform LR rotation
  ELSE
    Perform LL rotation
    
```

CS2 - INTRODUCTION TO PROGRAMMING METHODS

14

So, Balanced BST is Best, Right?

Can do better, actually...

- if you perform lots of search
 - paying $O(\log n)$ each time is worse than... $O(1)$
 - but array is the only data structure with $O(1)$ access
 - but static, with size planned to be worst case scenario
 - » ex: phone numbers? 10 billions possible ones, not all used...



Hash tables is the answer

- roughly, array of llists with hash function
 - $O(1)$ for search, insert, and remove on average
 - but much slower to find min/max, range queries, ...

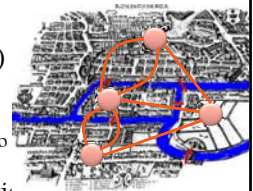
CS2 - INTRODUCTION TO PROGRAMMING METHODS

15

Extensions of Trees

Graphs are widely used too

- date back to Euler
- nodes and links (edges)
 - cycles allowed
 - edge can be directed or not
 - values assigned to edges too
 - “weighted” graphs
- useful in many applications
 - e.g., networks, automata, database dependencies, task scheduling (critical path analysis), mapquest/google map, even garbage collection in Java



CS2 - INTRODUCTION TO PROGRAMMING METHODS

16