Introduction to Trees

D. Thiebaut
CSC212 — Fall 2014
A bit of History & Data Visualization: The Book of Trees.
We Concentrate on Binary-Trees, Specifically, Binary-Search Trees (BST)
How Will Java Trees be related to Java Linked-Lists?
A doubly-linked list…

A binary tree…
Definitions
Recursive Definition

1. An empty structure is a tree…

2. If \( t_1, t_2, \ldots, t_k \) are disjunct trees, then the structure whose children are the roots of \( t_1, t_2, \ldots, t_k \) is a tree.

3. Only structures built following Rules 1 and 2 above are trees.
Recursive Definition: 1

This is a tree
Recursive Definition: 2

If these are trees... Then, that's a tree...
More Definitions
Definitions

- **Height** = number of nodes in the longest path from root to leaf.

- **Internal Node (Inner Node)**

- **Root**

- **Node**

- **Leaf**

- **Parent**

- **Child**

- **Grandparent**

- **Ancestor**
Edge

A Path = Collection of Edges Connecting Nodes

Definitions
Definitions

Node of degree 3

Node of degree 2

Node of degree 0
Definitions
Questions:

1. What is the **height** of the tree?
2. What is the **level** of Node E?
3. What is the **degree** of D?
4. What is the **degree** of H?
5. How many **leaves**?
6. How many **inner nodes**?
Binary Search Trees

BST

Diagram:

- Root
  - Left
  - Right

- 30
  - 20
    - 10
    - 25
  - 45
    - 40
    - 50
Examples
Do BSTs make you think of something we saw recently?

Why create a new data structure?
Java Implementation
class IntBSTNode {

    public int key;
    public IntBSTNode left, right;
    IntBSTNode( int el, IntBSTNode l, IntBSTNode r ) {
        key = el;
        left = l;
        right = r;
    }

    IntBSTNode( int el ) {
        this( el, null, null );
    }
}

NODE

root

left

right
public class IntBST {
    private IntBSTNode root;

    IntBST( int rootEl ) {
        root = null;
    }

    protected void visit( IntBSTNode p ) { ... }
    public IntBSTNode search( IntBSTNode p, int el ) { ... }
    public void breadthFirst( IntBSTNode p, int el ) { ... }
    public void inorderVisit( ) { ... }
    public void preorderVisit( ) { ... }
    public void postorderVisit( ) { ... }
    public void insert( int el ) { ... }
    public void delete( int el ) { ... }
    public void balance() { ... }
}
Note: The textbook cover much more important details!

Read Chapter 6!
Building a BST
IntBSTNode q, p = new IntBSTNode( 4 );

p = new IntBSTNode( 6, p, new IntBSTNode( 7 ) );
p = new IntBSTNode( 3, new IntBSTNode( 1 ), p );

q = new IntBSTNode( 13 );
q = new IntBSTNode( 14, q, null );
q = new IntBSTNode( 10, null, q );
tree.setRoot( new IntBSTNode( 8, p, q ) );
IntBSTNode q,

\[ p = \text{new IntBSTNode}(4) ; \]

\[ p = \text{new IntBSTNode}(6, p, \text{new IntBSTNode}(7)) ; \]

\[ p = \text{new IntBSTNode}(3, \text{new IntBSTNode}(1), p) ; \]

\[ q = \text{new IntBSTNode}(13) ; \]

\[ q = \text{new IntBSTNode}(14, q, \text{null}) ; \]

\[ q = \text{new IntBSTNode}(10, \text{null}, q) ; \]

\[ \text{tree.setRoot(new IntBSTNode}(8, p, q) ;) ; \]
IntBSTNode q, p = new IntBSTNode( 4 );

p = new IntBSTNode( 6, p, new IntBSTNode( 7 ) );
p = new IntBSTNode( 3, new IntBSTNode( 1 ), p );

q = new IntBSTNode( 13 );
q = new IntBSTNode( 14, q, null );
q = new IntBSTNode( 10, null, q );
tree.setRoot( new IntBSTNode( 8, p, q ) );
IntBSTNode q, p = new IntBSTNode( 4 );

\[
p = \text{new IntBSTNode( 6, p, new IntBSTNode( 7 ) );}
\]

\[
p = \text{new IntBSTNode( 3, new IntBSTNode( 1 ), p );}
\]

q = new IntBSTNode( 13 );
q = new IntBSTNode( 14, q, null );
q = new IntBSTNode( 10, null, q );
tree.setRoot( new IntBSTNode( 8, p, q ) );
IntBSTNode q, p = new IntBSTNode(4);

p = new IntBSTNode(6, p, new IntBSTNode(7));
p = new IntBSTNode(3, new IntBSTNode(1), p);

q = new IntBSTNode(13);
q = new IntBSTNode(14, q, null);
q = new IntBSTNode(10, null, q);
tree.setRoot(new IntBSTNode(8, p, q));
IntBSTNode q, p = new IntBSTNode( 4 );

p = new IntBSTNode( 6, p, new IntBSTNode( 7 ) );

\textcolor{orange}{p = new IntBSTNode( 3, new IntBSTNode( 1 ), p );}

q = new IntBSTNode( 13 );
q = new IntBSTNode( 14, q, null );
q = new IntBSTNode( 10, null, q );
tree.setRoot( new IntBSTNode( 8, p, q ) );
```java
IntBSTNode q, p = new IntBSTNode(4);

p = new IntBSTNode(6, p, new IntBSTNode(7));
p = new IntBSTNode(3, new IntBSTNode(1), p);

q = new IntBSTNode(13);
q = new IntBSTNode(14, q, null);
q = new IntBSTNode(10, null, q);
tree.setRoot( new IntBSTNode(8, p, q) );
```
IntBSTNode q, p = new IntBSTNode(4);

p = new IntBSTNode(6, p, new IntBSTNode(7));
p = new IntBSTNode(3, new IntBSTNode(1), p);

q = new IntBSTNode(13);
q = new IntBSTNode(14, q, null);
q = new IntBSTNode(10, null, q);
tree.setRoot(new IntBSTNode(8, p, q));
IntBSTNode q, p = new IntBSTNode( 4 );

p = new IntBSTNode( 6, p, new IntBSTNode( 7 ) );
p = new IntBSTNode( 3, new IntBSTNode( 1 ), p );

q = new IntBSTNode( 13 );
a = new IntBSTNode( 14, a, null );
q = new IntBSTNode( 10, null, q );
tree.setRoot( new IntBSTNode( 8, p, q ) );
IntBSTNode  q, p = new IntBSTNode( 4 );

p = new IntBSTNode( 6, p, new IntBSTNode( 7 ) );
p = new IntBSTNode( 3, new IntBSTNode( 1 ), p );

q = new IntBSTNode( 13 );
q = new IntBSTNode( 14, q, null );
a = new IntBSTNode( 10, null, q );

tree.setRoot( new IntBSTNode( 8, p, q ) );
Searching A BST
Exercise

1) Search for these keys:
   - 8
   - 3
   - 6
   - 13
   - 20

2) Devise an iterative algorithm that searches for key in a BST with root root

3) Devise a recursive solution
public IntBSTNode search( int el ) {
    IntBSTNode p = root;
    while ( p != null ) {
        if ( el == p.key )
            return p;
        if ( el < p.key )
            p = p.left;
        else
            p = p.right;
    }
    return null;
}
public IntBSTNode recSearch( int el ) {
    return recSearch( root, el );
}

private IntBSTNode recSearch( IntBSTNode p, int el ) {
    if ( p==null )
        return null;
    if ( el == p.key )
        return p;
    if ( el < p.key )
        return recSearch( p.left, el );
    else
        return recSearch( p.right, el );
}
[beowulf2]

DEMO TIME!

[beowulf2]
[11:53:56] ~$:  

Traversal:

The Action of Visiting Systematically all the Nodes of a Tree, in an Order that Reveals some Property of the Tree.
Breadth-First Traversal
Breadth-First Traversal
Breadth-First Traversal
Breadth-First Traversal
Breadth-First Traversal
Breadth-First Traversal
Queue

1 6 14
etc...
WE STOPPED HERE...
public void breadthFirst() {
    IntBSTNode p = root;
    Queue<IntBSTNode> queue = new LinkedList<IntBSTNode>();

    if (p == null)
        return;

    queue.add(p);
    while (!queue.isEmpty()){
        p = queue.poll();
        //--- do some work with the key.
        //---- Here we just print it...
        System.out.print(p.key + " ");
        if (p.left != null) queue.add(p.left);
        if (p.right != null) queue.add(p.right);
    }
}
public void breadthFirst() {
    IntBSTNode p = root;
    Queue<IntBSTNode> queue = new LinkedList<IntBSTNode>();

    if ( p == null )
        return;

    queue.add( p );
    while ( !queue.isEmpty() ){
        p = queue.poll();
        //--- do some work with the key.
        //---- Here we just print it...
        System.out.print( p.key + " " );
        if ( p.left != null ) queue.add( p.left );
        if ( p.right != null ) queue.add( p.right );
    }
}
public void breadthFirst() {
    IntBSTNode p = root;
    Queue<IntBSTNode> queue = new LinkedList<IntBSTNode>();
    if (p == null)
        return;
    queue.add(p);
    while (!queue.isEmpty()){
        p = queue.poll();
        //--- do some work with the key.
        //——- Here we just print it...
        System.out.print(p.key + " ");
        if (p.left != null) queue.add(p.left);
        if (p.right != null) queue.add(p.right);
    }
}
public void breadthFirst() {
    IntBSTNode p = root;
    Queue<IntBSTNode> queue = new LinkedList<IntBSTNode>();

    if (p == null)
        return;

    queue.add(p);
    while (!queue.isEmpty()){
        p = queue.poll();
        //--- do some work with the key.
        //---- Here we just print it...
        System.out.print(p.key + " ");
        if (p.left != null) queue.add(p.left);
        if (p.right != null) queue.add(p.right);
    }
}
public void breadthFirst() {
    IntBSTNode p = root;
    Queue<IntBSTNode> queue = new LinkedList<IntBSTNode>();

    if (p == null)
        return;

    queue.add(p);
    while (!queue.isEmpty()){
        p = queue.poll();
        //--- do some work with the key.
        //---- Here we just print it...
        System.out.print(p.key + " ");
        if (p.left != null) queue.add(p.left);
        if (p.right != null) queue.add(p.right);
    }
}
public void breadthFirst() {
    IntBSTNode p = root;
    Queue<IntBSTNode> queue = new LinkedList<IntBSTNode>();

    if (p == null)
        return;

    queue.add(p);
    while (!queue.isEmpty()){
        p = queue.poll();
        //--- do some work with the key.
        //---- Here we just print it...
        System.out.print(p.key + " ");
        if (p.left != null) queue.add(p.left);
        if (p.right != null) queue.add(p.right);
    }
}
public void breadthFirst() {
    IntBSTNode p = root;
    Queue<IntBSTNode> queue = new LinkedList<IntBSTNode>();
    queue.add(p);

    while (!queue.isEmpty()) {
        p = queue.poll();
        //--- do some work with the key.
        //---- Here we just print it...
        System.out.print(p.key + " ");
        if (p.left != null) queue.add(p.left);
        if (p.right != null) queue.add(p.right);
    }
}
public void breadthFirst() {
    IntBSTNode p = root;
    Queue<IntBSTNode> queue = new LinkedList<IntBSTNode>();

    if ( p == null )
        return;

    queue.add( p );
    while ( !queue.isEmpty() ){
        p = queue.poll();
        //--- do some work with the key.
        //---- Here we just print it...
        System.out.print( p.key + " " );
        if ( p.left != null ) queue.add( p.left );
        if ( p.right != null ) queue.add( p.right );
    }
}
public void breadthFirst() {
    IntBSTNode p = root;
    Queue<IntBSTNode> queue = new LinkedList<IntBSTNode>();
    if ( p == null )
        return;
    queue.add( p );
    while ( ! queue.isEmpty() ){
        p = queue.poll();
        //--- do some work with the key.
        //---- Here we just print it...
        System.out.print( p.key + " " );
        if ( p.left != null ) queue.add( p.left );
        if ( p.right != null ) queue.add( p.right );
    }
}
public void breadthFirst() {
    IntBSTNode p = root;
    Queue<IntBSTNode> queue = new LinkedList<IntBSTNode>();
    if ( p == null )
        return;

    queue.add( p );
    while ( ! queue.isEmpty() ){
        p = queue.poll();
        //--- do some work with the key.
        //---- Here we just print it...
        System.out.print( p.key + " " );
        if ( p.left != null ) queue.add( p.left );
        if ( p.right != null ) queue.add( p.right );
    }
}

public void breadthFirst() {
    IntBSTNode p = root;
    Queue<IntBSTNode> queue = new LinkedList<IntBSTNode>();
    if (p == null)
        return;
    queue.add(p);
    while (!queue.isEmpty()) {
        p = queue.poll();
        //--- do some work with the key.
        // Here we just print it...
        System.out.print(p.key + " ");
        if (p.left != null) queue.add(p.left);
        if (p.right != null) queue.add(p.right);
    }
}
public void breadthFirst() {
    IntBSTNode p = root;
    Queue<IntBSTNode> queue = new LinkedList<IntBSTNode>();
    if (p == null) return;
    queue.add(p);
    while (!queue.isEmpty()) {
        p = queue.poll();
        //--- do some work with the key.
        //--- Here we just print it...
        System.out.print(p.key + " ");
        if (p.left != null) queue.add(p.left);
        if (p.right != null) queue.add(p.right);
    }
}

1 6 8 3
Etc...
Depth-First Search Traversal
3 Ways to Visit

- **pre-order**
- **in-order**
- **post-order**
public void preOrderVisit() {
    preOrderVisit(root);
    System.out.println();
}

private void preOrderVisit(IntBSTNode p) {
    if (p==null)
        return;
    //--- do some work with the key.
    System.out.print(p.key + " ");
    preOrderVisit(p.left);
    preOrderVisit(p.right);
}
Predict Output of PreOrder...
public void inOrderVisit() {
    inOrderVisit(root);
    System.out.println();
}

private void inOrderVisit(IntBSTNode p) {
    if (p==null) {
        return;
    }
    inOrderVisit(p.left);

    //--- do some work with the key.
    System.out.print(p.key + " ");

    inOrderVisit(p.right);
}
Predict Output of InOrder...
public void postOrderVisit() {
    postOrderVisit(root);
    System.out.println();
}

private void postOrderVisit(IntBSTNode p) {
    if (p==null)
        return;
    postOrderVisit(p.left);
    postOrderVisit(p.right);

    //--- do some work with the key.
    System.out.print(p.key + " ");
}

Predict Output of PostOrder...
Insertion in a BST
Item to Insert: 12

Insertion

![Binary Search Tree Diagram]

The binary search tree after inserting the item 12.
Item to Insert: 12

Insertion

prev → p
Item to Insert: 12

Insertion
Item to Insert: 12

Insertion
Item to Insert: 12

Insertion
Item to Insert: 12

Insertion

```
   8
  / 
 3   10
 /   /  
1    6   14
 /     / 
4      7 13
```
Item to Insert: 12

Insertion
public void insert( int el ) {
    // is the tree empty? if so, create 1 node
    if ( root==null ) {
        root = new IntBSTNode( el );
        return;
    }

    // go down to the leaf that will be the parent
    // of the new node
    IntBSTNode p = root, prev = null;
    while ( p != null ) {
        prev = p;
        if ( p.key < el )
            p = p.right;
        else
            p = p.left;
    }

    // add element as new leaf of prev's node
    if ( el < prev.key )
        prev.left = new IntBSTNode( el );
    else
        prev.right = new IntBSTNode( el );
}
// create a new tree by using insert()
keys = new int[] { 8, 3, 6, 1, 4, 7, 10, 14, 13 };
tree.clear();
for ( int i=0; i< keys.length; i++ ) {
    tree.insert( keys[i] );
// create a new tree by using insert()
keys = new int[] { 1, 3, 4, 6, 8, 7, 10, 13, 14 };

tree.clear();
for (int i=0; i<keys.length; i++) {
    tree.insert(keys[i]);
Balancing BSTs is an Important Operation
Deletion in a BST
Case 1: Deleting a Leaf

```
     8
    / \  
   3   10
  / \   /  
 1   6 14
 / \  /   /
4   7 13  
```
Case 1: Deleting a Leaf
Case 2: Deleting an Internal Node
Locate the node containing the element to delete…
Find the rightmost child in the left subtree…
Find the rightmost child in the left subtree…
Find the rightmost child in the left subtree…

node
Found it!
Detach right sub-tree of node to delete…
Attach it to blue node...
Attach parent of yellow node to its left sub-tree
Done! The yellow node has been removed!
This is "Deletion by Merging"

It may change the balance of the tree…
Delete Examples

Original Tree

Delete 3
Delete Examples

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18</td>
</tr>
</tbody>
</table>
```
Delete Examples

Diagram:

- Node 8
- Node 3
- Node 1
- Node 4
- Node 6
- Node 0
- Node 2
- Node 13
- Node 10
- Node 14
- Node 16
- Node 20
- Node 18
Delete Examples
Delete 8

Delete Examples

Diagram:
- Node 8
  - Node 1
    - Node 0
      - Node 6
    - Node 2
  - Node 13
    - Node 10
      - Node 9
      - Node 12
    - Node 14
      - Node 16
      - Node 20
      - Node 18
Delete Examples
public void delete( int el ) {
    IntBSTNode tmp, node, p=root, prev = null;

    if ( root == null ) {
        System.out.println( "Trying to delete from an empty tree" );
        return;
    }

    // find the node containing el
    while ( p!=null && p.key != el ) {
        prev = p;
        if (p.key < el )
            p = p.right;
        else
            p = p.left;
    }

    if ( p==null ) {
        System.out.println( "Key " + el + " not in tree" );
        return;
    }
}
// we found the element. p is pointing to it. prev is pointing to
// parent (or is null if el is in root).
node = p;

// no right sub-tree?
if ( node.right == null ) {
    // yes, then pick the left subtree
    node = node.left;
}
else if ( node.left == null ) {
    // yes, then pick the right subtree
    node = node.right;
}
else {
    // two existing subtrees.
    // go to left subtree
    tmp = node.left;

    // and find the right-most node that doesn't have a right child.
    while ( tmp.right != null )
        tmp = tmp.right;

    // tmp points to right-most node (blue node)

    // attach right subtree of node we're deleting (yellow node)
    // to right most node (blue node)
    tmp.right = node.right;

    node = node.left;
}
if ( p==root )
    root = node;
el if ( prev.left == p )
    prev.left = node;
else
    prev.right = node;
}
Balancing a BST
If you start with an empty tree, in what order would you start inserting nodes to get a fully balanced tree?

Step 1:
Given the answer you found for Step 1
Devise a method for balancing an unbalanced tree
private void addNodeToArray(IntBSTNode p, ArrayList allNodes) {
    if (p == null) return;
    allNodes.add(p.key);
    addNodeToArray(p.left, allNodes);
    addNodeToArray(p.right, allNodes);
}

public void balance() {
    // visit the tree and copy all the nodes into an ArrayList
    ArrayList allNodes = new ArrayList();
    addNodeToArray(root, allNodes);

    // sort the ArrayList
    Collections.sort(allNodes);

    // recursively reconstruct the tree
    clear();
    recursiveBalance(allNodes, 0, allNodes.size() - 1);
}

private void recursiveBalance(ArrayList allNodes, int low, int high) {
    if (low <= high) {
        int mid = (low + high) / 2;
        insert((int) allNodes.get(mid));
        recursiveBalance(allNodes, low, mid - 1);
        recursiveBalance(allNodes, mid + 1, high);
    }
}