Trees II

Binary Search Trees

Announcement

- Final Exam
  - Wednesday, February 25, 2004
  - 8:00am – 10:00 am
  - 70-3435
- Please report all exam conflicts now!

Project 2 Notes

- Writeup now on the Web
- Due Dates:
  - Minimum submission due Friday, Feb 6th
    - Entry.java & Document.java
    - Partial implementation of Directory.java provided
  - Final submission due Sunday, Feb 15th
    - A little more than a week after the minimum!
    - Complete Directory.java & VFSystem.java
    - Integration tests WILL be performed.

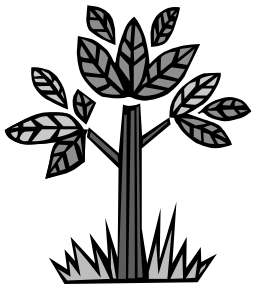
Announcement

- Exam 2
- Wednesday, Feb 4th
- Covering:
  - Java IO
  - Recursion
  - Asymptotic Analysis
  - Searching
  - Sorting

Announcement

- Office hour tomorrow cancelled.
  - 2-3

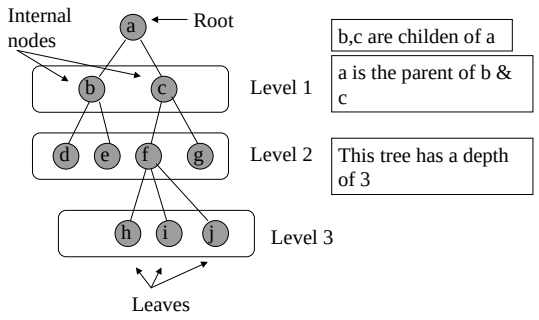
Trees



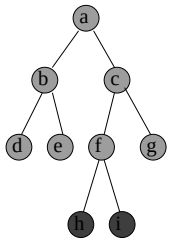
I think that I shall never see
A poem lovely as a tree

-- J. Kilmer

Anatomy of a Tree



Anatomy of a Binary Tree



- Binary Tree
  - Each node has at most 2 children
  - Children of nodes in a binary tree are referred to as left child or right child
    - Node h is the left child of Node f
    - Node i is the right child of Node f

Implementing a Binary Tree

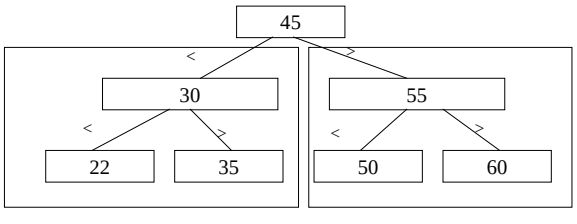
- Define a binary tree node object
- Each node can be seen as the root of a Binary Tree.

Binary Search Trees (BST)

- Another use for binary trees
- Efficient storage for search / retrieval of "sortable" data.
- Basic idea
  - Left subtree contains nodes with data less than data at node
  - Right subtree contains nodes with data greater than data at node

Binary Search Trees

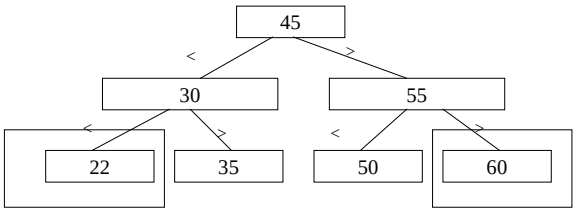
- Branching can also imply ordering of node data



All values less than 45 All values greater than 45

Binary Search Trees

- Branching can also imply ordering of node data



All values less than 22 All values greater than 55

Comparable

- Data in BST need to be compared
  - Comparable interface:
    - public int **compareTo**(Object o)
      - Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.
      - Assumes that o is of same class of object being compared to.

BST as Sets

- What if a data item is equal to the data at a node?
  - We'll assume that the BST represents a set
    - Each element can be present in the tree only once.
    - No duplicates

Binary Search Trees (BST)

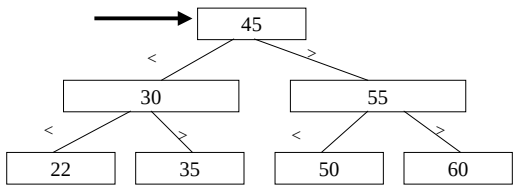
- Class BSTNode
  - extends BTNode
  - Modify BTNode to accept Comparable as data.
  - Additional methods
    - Insert – Add a node to the BST
    - Find – Find an object in the BST
    - Remove – Remove a Node from a BST
    - Print – all objects in a BST

Binary Search Trees (BST)

- Insert
  - Must make sure that we insert the node into the proper place in the tree.
  - At each node
    - If the data of the node being inserted is < the data of the node into which we are inserting, new node must be placed into the left subtree
    - If the data of the node being inserted is > the data of the node into which we are inserting, new node must be placed into the left subtree

Binary Search Trees

- Let's say we are inserting a node with data = 42



Binary Search Trees (BST)

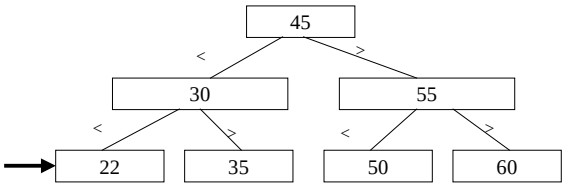
```
public void insert (BSTNode N)
{
    Comparable D = N.getData();
    if (D.compareTo(data) < 0)
       leftChild.insert (N);
    else
       rightChild.insert (N);
}
```

Binary Search Trees

- What if we have no left or right child?

Binary Search Trees

- Let's say we are inserting a node with data = 23

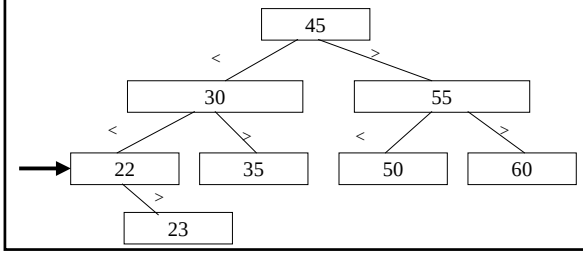


Binary Search Trees (BST)

- Insert
  - When you need to insert into the left (right) child of a node that has no left (right) child
    - Simply define the node to be inserted to be the new left (right) child.

Binary Search Trees

- Let's say we are inserting a node with data = 23



Binary Search Trees (BST)

- Insert (corrected code)

```
public void insert (BSTNode N)
{
    Comparable D = N.getData();
    if (D.compareTo(data) < 0) {
       if (leftChild != null) leftChild.insert (N);
       else setLeft (N);
    } else {
       if (rightChild != null) rightChild.insert (N);
       else setRight (N);
    }
}
```

Binary Search Trees (BST)

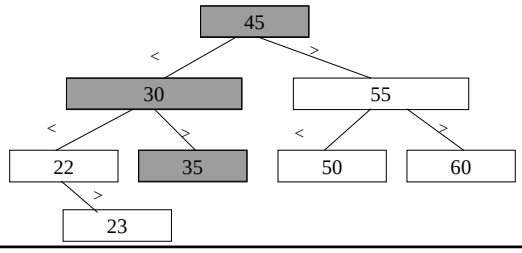
- Insert
- Questions?

Binary Search Trees (BST)

- Find
  - Find and return a node in the BST that has a particular data value
  - Return null if data is not in the BST
  - Basic idea
    - At each node:
      - If data is equal to node data, return node
      - If data is < than node data, call find on left subtree
      - If data is > than node data, call find on right subtree

Binary Search Trees

- Let's say we are looking for = 35

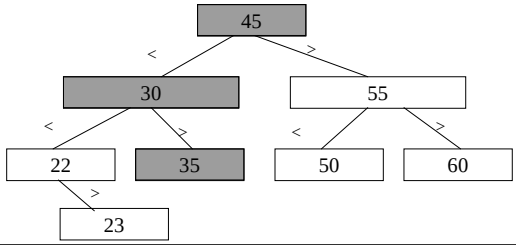


Beware of this guy!

- What if we have no left or right child?

Binary Search Trees

- Let's say we are looking for = 37



Binary Search Trees

- Find
  - If you get to a node with no left(right) subtree to search, the search data must not be in the BST!

Binary Search Trees (BST)

- Find
  - Final algorithm:
    - At each node:
      - If data is equal to node data, return node
      - If data is < than node data,
        - » If node has a left subtree call find on left subtree
        - » Otherwise return null
      - If data is > than node data,
        - » If node has a right subtree call find on right subtree
        - » Otherwise return null.

Binary Search Trees (BST)

- Find

- Questions

Binary Search Trees (BST)

- So far
  - Binary Search Trees
  - Insert
  - Find

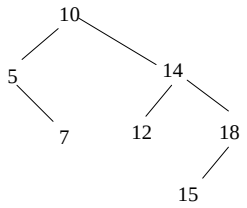
  - Next: Remove & Print

Binary Search Tree

- Removal
  - Remove the node with a given data value
  - Basic idea:
    - First find the node with the given data value using find.
    - Remove the node..but
      - Take care to reattach any children of the deleted node to the deleted node's parent.

Binary Search Tree

- Example:

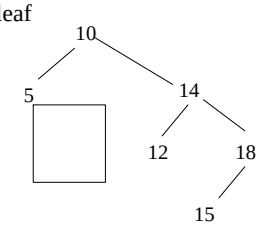


Binary Search Tree

- Removal
  - First find node to be removed
  - 5 cases:
    1. Data not in tree – No node to delete
    2. Node to be deleted is a leaf
    3. Node to be deleted has only a right child
    4. Node to be deleted has only a left child
    5. Node to be deleted has both children

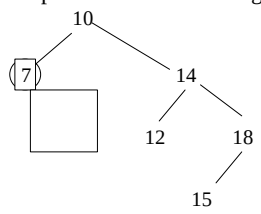
Binary Search Tree

- Removal: Case 2
  - Node to be deleted is a leaf
  - Simply remove it!



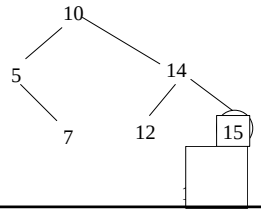
Binary Search Tree

- Removal: Case 3
  - Node only has a right child
  - Replace node with it's right child



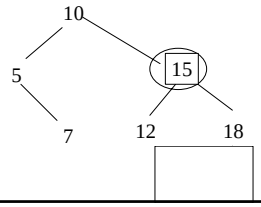
Binary Search Tree

- Removal: Case 4
  - Node only has a left child
  - Replace node with it's left child



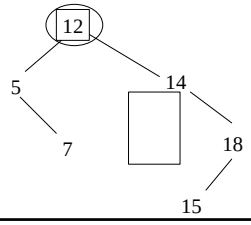
Binary Search Tree

- Removal: Case 5
  - Node has both children
  - Replace node with leftmost node of right subtree.



Binary Search Tree

- Removal: Case 5
  - Node has both children
  - Replace node with leftmost node of right subtree.



Binary Search Tree

- Let's write this in code:

```
public BSTNode getLeftmostNode()
{
    if (left == null) return this;
    else return
  (left.getLeftmostNode());
}
```

Binary Search Tree

```
public void remove (Comparable C)
{
   // Find the node to be removed
   BSTNode N = find (C);

   // start the removal
   if (N != null) {
       // get the parent
       BSTNode P = N.getParent();
```

Binary Search Tree

```
// case 2: leaf
if ((N.getLeft() == null) &&
    (N.getRight() == null)) {
   if (N = P.getLeft())
      P.setLeft (null);
   else
      P.setRight (null);
}
```

Binary Search Tree

```
// case 3: only left child
if (((N.getLeft() != null) && (N.getRight() == null)) {
   BSTNode L = N.getLeft()
   if (P != null) {
     if (N == P.getLeft())
        parent.setLeft (L);
     else
        parent.setRight (L);
   }
   else {
      // N is the root
      N.setData (L.getData());
      N.setRight (L.getRight());
      N.setLeft (L.getLeft());
   }
}
```

Binary Search Tree

```
// case 4: only right child
if (((N.getLeft() == null) && (N.getRight() != null)) {
   BSTNode R = N.getRight()
   if (P != null) {
     if (N == P.getLeft())
        parent.setLeft (R);
     else
        parent.setRight (R);
   }
   else {
      // N is the root
      N.setData (R.getData());
      N.setRight (R.getRight());
      N.setLeft (R.getLeft());
   }
}
```

Binary Search Tree

```
// case 5: has both children
if (((N.getLeft() != null) && (N.getRight() != null)) {

   // Get leftmost node of right tree
   BSTNode LM = N.getRight().getLeftmostNode();

   // replace the node's data
   N.setData (LM.getData());

   // remove the leftmost node of right tree
   BSTNode LMP = LM.getParent();
   if (LM == LMP.getLeft()) LMP.setLeft (null);
   else LMP.setRight(LM.getRight());
}
}}
```

Binary Search Trees (BST)

- Removal
  - Questions

Binary Search Trees (BST)

- Print
  - Use a traversal with the result of visiting a node be a print
  - What kind of traversal?

Binary Search Trees (BST)

- Print

```
public void print()
{
     inorder();
}
```

Binary Search Trees (BST)

- Summary
  - Binary Search Trees
    - Insert
    - Find
    - Remove
    - Print
  - Questions?