CS 146: Trees

Acknowledgements: Some of this presentation is from R. Mok CS 146 Trees
boolean checkBST(TreeNode n){
    return checkBST(n, null, null);
}

boolean checkBST(TreeNode n, Integer min, Integer max){
    if(n == null){
        return true;
    }
    if((min != null && n.data <= min) ||
        (max != null && n.data > max)){
        return false;
    }
    if(!checkBST(n.left, min, n.data) ||
        !checkBST(n.right, n.data, max)){
        return false;
    }
    return true;
}
Tree Day 2

- Homework is due!
- Office hours next week moved to Friday 1-3pm
- Saturday Project by 11:59pm
The Binary Search Tree: Contains

- Does a binary search tree contain a target value?

- **Search recursively** starting at the root node:
  - If the target value is less than the node’s value, then search the node’s left subtree.
  - If the target value is greater than the node’s value, then search the node’s right subtree.
  - If the values are equal, then yes, the target value is contained in the tree.
  - If you “run off the bottom” of the tree, then no, the target value is not contained in the tree.
private boolean contains(AnyType x, BinaryNode<AnyType> t) {
    if (t == null) return false;

    int compareResult = x.compareTo(t.element);

    if (compareResult < 0) {
        return contains(x, t.left);
    }

    else if (compareResult > 0) {
        return contains(x, t.right);
    }

    else { // Match
        return true;
    }
}
To **insert** a target value into the tree:

- Proceed as if you are checking whether the tree contains the target value.

As you’re recursively examining left and right subtrees, if you **encounter a null link** (either a left link or a right link), then **that’s where the new value should be inserted**.

- Create a new node containing the target value and replace the null link with a link to the new node.
- So the new node is attached to the **last-visited node**.
The Binary Search Tree: Insert, *cont’d*

- If the target value is already in the tree, either:
  - Insert a duplicate value into the tree.
  - Don’t insert but “update” the existing node.
The Binary Search Tree: Insert

**Figure 4.21** Binary search trees before and after inserting 5
private BinaryNode<AnyType> insert(AnyType x, BinaryNode<AnyType> t) {
    // Create a new node to be attached
    // to the last-visited node.
    if (t == null) {
        return new BinaryNode<>(x, null, null);
    }

    int compareResult = x.compareTo(t.element);

    // Find the insertion point.
    if (compareResult < 0) {
        t.left = insert(x, t.left);
    }
    else if (compareResult > 0) {
        t.right = insert(x, t.right);
    }
    else {
        // Duplicate: do nothing.
    }

    return t;
}
The Binary Search Tree: Remove

- After removing a node from a binary search tree, **the remaining nodes must still be in order**.

- **No child case**: The target node to be removed is a leaf node.
  - Just remove the target node.
The Binary Search Tree: Remove, cont’d

- **One child case**: The target node to be removed has one child node.
  - Change the parent’s link to the target node to point instead to the target node’s child.

![Diagram](image.png)  
*Figure 4.23*  Deletion of a node (4) with one child, before and after
The Binary Search Tree: Remove, *cont’d*

- **Two children case:** The target node to be removed has two child nodes.
  - This is the complicated case.

- How do we restructure the tree so that the order of the node values is preserved?
Recall what happens you remove a list node.

- Assume that the list is sorted.

```
0  1  2  3  4  5  6  7  8  9
```

- If we delete target node 5, which node takes its place?

```
0  1  2  3  4  6  7  8  9
```

- The replacement node is the node that is immediately after the target node in the sorted order.
A somewhat convoluted way to do this:

- Replace the target node’s value with the successor node’s value.

- Then remove the successor node, which is now “empty”.

```
   0  1  2  3  4  5  6  7  8  9
0  1  2  3  4  6 6  7  8  9
```
The same convoluted process happens when you remove a node from a binary search tree.

- The successor node is the node that is immediately after the deleted node in the sorted order.
- Replace the target node’s value with the successor node’s value.
- Remove the successor node, which is now “empty”.

```
0  1  2  3  4  5  6  7  8  9
0  1  2  3  4  6  7  8  9
0  1  2  3  4  6  7  8  9
```
The Binary Search Tree: Remove, cont’d

☐ If you have a target node in a binary search tree, where is the node that is its immediate successor in the sort order?
   - The successor’s value is ≥ than the target value.
   - It must be the minimum value in the right subtree.

☐ General idea:
   - Replace the value in the target node with the value of the successor node.
     - The successor node is now “empty”.
   - Recursively delete the successor node.
The Binary Search Tree: Remove, *cont’d*

- Replace the value of the target node 2 with the value of the successor node 3.
- Now recursively remove node 3.

*Figure 4.24* Deletion of a node (2) with two children, before and after

- The second deletion is easy because the node has no left child.
private BinaryNode<AnyType> remove(AnyType x, BinaryNode<AnyType> t) {
    if (t == null) return t;
    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) {
        t.element = findMin(t.right).element;
        t.right = remove(t.element, t.right);
    }
    else {  
        t = (t.left != null) ? t.left : t.right;
    }
    return t;
}
The Binary Search Tree Animations

- Download Java applets from http://www.informit.com/content/images/0672324539/downloads/ExamplePrograms.ZIP
- The binary search tree applet is in Chap08/Tree
- Run with the appletviewer application that is in your java/bin directory:
  appletviewer Tree.html
Questions on Binary Trees

1) Recall: Height of tree = longest path from root to leaf (count edges)

2) For binary tree of height $h$:
   1) Max # of leaves?
   2) Max # of nodes?
   3) Min # of leaves?
   4) Min # of nodes?

3) For $n$ nodes, we cannot do better than $O(?)$ height, and we want to avoid $O(?)$ height.
Questions on Binary Trees

4) Algorithm to find height of a tree with root root?
5) Are these BSTs?
Questions on Binary Trees

6) Delete Node 15?

delete(15)
Questions on Binary Trees

7) Delete Node 5?

What can we replace 5 with?
Implement a function to check if a binary tree is balanced. For the purposes of this question, a balanced tree is defined to be a tree such that the heights of the two subtrees of any node never differ by more than one.

Given a sorted (increasing order) array with unique integer elements, write an algorithm to create a binary search tree with minimal height.