CS 106B, Lecture 19
Binary Search Trees

reading:
*Programming Abstractions in C++, Chapter 16*
Plan for Today

• Continue our discussion of Binary Search Trees
  – Implement add
  – Discuss remove

• MiniBrowser! New Assignment!
Adding to a BST

• Suppose we want to add new values to the BST below.
  – Where should the value 14 be added?
  – Where should 3 be added? 7?
  – If the tree is empty, where should a new value be added?

• What is the general algorithm?
• Draw what a binary search tree would look like if the following values were added to an initially empty tree in this order:
Exercise: add

• Write a function `add` that adds a given integer value to the BST.
  – Add the new value in the proper place to maintain BST ordering.
• `tree.add(root, 49);`
void add(TreeNode* & node, int value) {
    if (node == nullptr) {
        node = new TreeNode(value);
    } else if (node->data > value) {
        add(node->left, value);
    } else if (node->data < value) {
        add(node->right, value);
    }
}

• Must pass the current node by reference for changes to be seen.
Free Tree

• To avoid leaking memory when discarding a tree, we must free the memory for every node.
  – Like most tree problems, often written *recursively*
  – must free the node itself, and its left/right subtrees
  – this is another *traversal* of the tree
    • should it be pre-, in-, or post-order?
• Suppose we want to **remove** values from the BST below.
  – Removing a leaf like 4 or 22 is easy.
  – What about removing 2? 19?
  – How can you remove a node with two large subtrees under it, such as 15 or 9?

• What is the general algorithm?
Cases for removal

1. a leaf:
2. a node with a left child only:
3. a node with a right child only:

Replace with nullptr
Replace with left child
Replace with right child

remove(root, 17);
remove(root, 55);
remove(root, 29);
Cases for removal

4. a node with **both** children: replace with **min from right**
   (replacing with **max from left** would also work)

```plaintext
remove(root, 55);
```

```
root

55

29  87
17  42  60  91

72
```

```
root

60

29  87
17  42  72  91
```
Exercise: remove

• Add a function `remove` that accepts a root pointer and removes a given integer value from the tree, if present. Remove the value in such a way as to maintain BST ordering.

  • `remove(root, 73);`
  • `remove(root, 29);`
  • `remove(root, 87);`
  • `remove(root, 55);`
remove solution

// Removes the given value from this BST, if it exists.
// Assumes that the given tree is in valid BST order.
void remove(TreeNode*& node, int value) {
    if (node == nullptr) {
        return;
    } else if (value < node->data) {
        remove(node->left, value); // too small; go left
    } else if (value > node->data) {
        remove(node->right, value); // too big; go right
    } else {
        // value == node->data; remove this node!
        // (continued on next slide)
        ...
    }
}
// value == node->data; remove this node!
if (node->right == nullptr) {
    // case 1 or 2: no R child; replace w/ left
    TreeNode* trash = node;
    node = node->left;
    delete trash;
} else if (node->left == nullptr) {
    // case 3: no L child; replace w/ right
    TreeNode* trash = node;
    node = node->right;
    delete trash;
} else {
    // case 4: L+R both; replace w/ min from right
    int min = getMin(node->right);
    remove(node->right, min);
    node->data = min;
}
Announcements

• Assignment 4 is due today

• Assignment 5 will be released later today
  – More time to complete it, but this assignment will be significantly longer than the others you've seen this quarter
  – As a rough guide, part c took SLs about four times as long to solve as part a, so don't wait until the last minute

• You will get assignment 3 feedback on today

• Please give feedback (if you have the next 30 minutes free): cs198.stanford.edu

• Exam logistics
  – Midterm today, July 25, from 7:00-9:00PM in Hewlett 200