Finding Minimum and Maximum

• Now let’s think about how to find the minimum element in a BST?

• The key idea is to simply follow the left child of each node, because that’s where the smaller elements are.

• Let’s write code to implement this.
Question

• How can you design an algorithm to check if a tree is a BST?
CS 187: Programming with Data Structures (Spring 2010)

Lecture 20: Binary Trees

Rui Wang
Deletion

• This is much more complicated than search and insertion
  – It may make your head spin
• While insertion always inserts as leaf, deletion can happen on an interior node.
• Let’s look at a few examples.
Deletion

• Similar to before, need to search for the node to be deleted first. Once that’s done, there are several cases to consider:
  1. The node to be deleted is a leaf (no children).
  2. The node to be deleted has one child.
  3. The node to be deleted has to children.

• Workshop Applet
• Case 1 and 2 are simple.
Case 3: Double-Children Node

• Now the fun begins
• You can’t do simple replacements. Why?

• Some node has to replace the deleted node. But which one?
• Note that you must make sure this is still a BST.
Case 3: Double-Children Node

- It turns out that we can use the **in-order successor** to replace the deleted node.

- **What’s in-order successor?**
  - The successor of the deleted node in in-order traversal.
  - In other words, the “leftmost” (smallest) element of the right subtree.
Case 3: Double-Children Node

• How to find the in-order successor?

• Why does this work?
  – Is the tree still a BST?
  – What about the in-order successor’s children?

• Is there another node that can be used to replace the to-be-deleted node?
Case 3: Double-Children Node

• How to find the in-order successor?
  – The smallest element in the right subtree

• Why does this work?
  – Is the tree still a BST? → Yes
  – What about the in-order successor’s children?
    The in-order successor cannot have a left child, so it’s either a leaf node or it has a right child.

• Is there another node that can be used to replace the to-be-deleted node?
  Yes: the in-order predecessor!
Case 3: Double-Children Node

• Workshop applet demo.

• Since this is relatively involved, there is no need to write down code.

• However, given a BST and a node to be deleted, you **MUST** know how to delete it (which node will replace it), and draw the result of deletion.
The Efficiency of Binary Trees

• In general, the search, insertion, and deletion costs are all $O(\log N)$.
  – This assumes the tree is somehow balanced.
  – Be careful with the degenerate case.

• How does this compare to ordered array and linked list?
Additional Topics

• Trees represented as arrays
  – Indexing scheme.

• Duplicate keys
  – More complicated.

• Huffman Coding
  – Efficient data compression
  – PKZIP, JPEG, MP3
How can we balance a tree?

• Before moving on, let’s take a break by watching an interesting YouTube Video.
How can we balance a tree?

• So far we’ve seen that the tree balance can greatly influence the tree search cost.
• Let’s look at a worst case example.

• Here we will define balance as the relative height of the left subtree vs. the right subtree.

• We will learn about self-balancing trees.
• **Rotation** is an operation that can help achieve balancing while still preserving BST property.
• An example of right rotation.
Rotation

• **Rotation** is an operation that can help achieve balancing while still preserving BST property.
• A general example.
Rotation

• **Rotation** is an operation that can help achieve balancing while still preserving BST property.
• Let’s verify that this is still a BST.
• Let’s write code to implement the rotation.

• What about left rotation?
AVL Tree

- **AVL** is the simplest self-balancing tree.
- The idea is to keep track of the difference in height of each subtree. Example?
- This difference must be within a range of [-1,1].
- When inserting/removing nodes, if the balance is violated, it uses rotation to bring the tree back to balance. How?
AVL Tree

• We define the balance measure as $h_R - h_L$
• The tree is ok if this measure is within $[-1,1]$, otherwise:
  • Balancing a Left-Left tree
  • Balancing a Left-Right tree
  • Balancing a Right-Right tree
  • Balancing a Right-Left tree
  • Demo
AVL Tree

- It can be shown that if an AVL tree contains $N$ nodes, the height of the tree is bounded by
  \[ 1.44 \log_2(N) \]

- This means even in the worst case, the search/insert/delete costs are $O(\log N)$, so it never degenerates to the $O(N)$ case.
Announcement

• Quiz 8 is available in SPARK.