CSE373: Data Structures & Algorithms
Lecture 6: Binary Search Trees continued

Aaron Bauer
Winter 2014
Announcements

• HW2 out, due beginning of class Wednesday
• Two TA sessions next week
  – Asymptotic analysis on Tuesday
  – AVL Trees on Thursday
• MLK day Monday
Previously on CSE 373

• Dictionary ADT
  – stores (key, value) pairs
  – *find, insert, delete*

• Trees
  – terminology
  – traversals
More on traversals

```java
void inOrderTraversal(Node t) {
    if (t != null) {
        inOrderTraversal(t.left);
        process(t.element);
        inOrderTraversal(t.right);
    }
}
```

Sometimes order doesn’t matter
- Example: sum all elements

Sometimes order matters
- Example: print tree with parent above indented children (pre-order)
- Example: evaluate an expression tree (post-order)
Binary Search Tree

- **Structure property ("binary")**
  - Each node has \( \leq 2 \) children
  - Result: keeps operations simple

- **Order property**
  - All keys in left subtree smaller than node’s key
  - All keys in right subtree larger than node’s key
  - Result: easy to find any given key
Are these BSTs?
Are these BSTs?
Data find(Key key, Node root)
    if(root == null)
        return null;
    if(key < root.key)
        return find(key,root.left);
    if(key > root.key)
        return find(key,root.right);
    return root.data;
}
Find in BST, Iterative

Data find(Key key, Node root) {
    while (root != null && root.key != key) {
        if (key < root.key)
            root = root.left;
        else (key > root.key)
            root = root.right;
    }
    if (root == null)
        return null;
    return root.data;
}
Other “Finding” Operations

- Find *minimum* node
  - “the liberal algorithm”
- Find *maximum* node
  - “the conservative algorithm”
- Find *predecessor*
- Find *successor*
Insert in BST

insert(13)
insert(8)
insert(31)

(New) insertions happen only at leaves – easy!
Deletion in BST

Why might deletion be harder than insertion?
Deletion

• Removing an item disrupts the tree structure

• Basic idea: **find** the node to be removed, then “fix” the tree so that it is still a binary search tree

• Three cases:
  – Node has no children (leaf)
  – Node has one child
  – Node has two children
Deletion – The Leaf Case

delete(17)
Deletion – The One Child Case

delte(15)
Deletion – The Two Child Case

delete(5)

What can we replace 5 with?
Deletion – The Two Child Case

Idea: Replace the deleted node with a value guaranteed to be between the two child subtrees

Options:
• **successor** from right subtree: \texttt{findMin(node.right)}
• **predecessor** from left subtree: \texttt{findMax(node.left)}
  – These are the easy cases of predecessor/successor

Now delete the original node containing **successor** or **predecessor**
• Leaf or one child case – easy cases of delete!
Lazy Deletion

- Lazy deletion can work well for a BST
  - Simpler
  - Can do “real deletions” later as a batch
  - Some inserts can just “undelete” a tree node

- But
  - Can waste space and slow down find operations
  - Make some operations more complicated:
    - How would you change \texttt{findMin} and \texttt{findMax}?
BuildTree for BST

• Let’s consider `buildTree`
  – Insert all, starting from an empty tree

• Insert keys 1, 2, 3, 4, 5, 6, 7, 8, 9 into an empty BST
  – If inserted in given order, what is the tree?
  – What big-O runtime for this kind of sorted input?
  – Is inserting in the reverse order any better?

\[ O(n^2) \]

Not a happy place
**BuildTree for BST**

- Insert keys 1, 2, 3, 4, 5, 6, 7, 8, 9 into an empty BST
- What we if could somehow re-arrange them
  - median first, then left median, right median, etc.
  - 5, 3, 7, 2, 1, 4, 8, 6, 9
- What tree does that give us?
- What big-O runtime?

\[ O(n \log n), \text{ definitely better} \]
Unbalanced BST

- Balancing a tree at build time is insufficient, as sequences of operations can eventually transform that carefully balanced tree into the dreaded list

- At that point, everything is $O(n)$ and nobody is happy
  - find
  - insert
  - delete
Balanced BST

Observation

• BST: the shallower the better!
• For a BST with $n$ nodes inserted in arbitrary order
  – Average height is $O(\log n)$ – see text for proof
  – Worst case height is $O(n)$
• Simple cases, such as inserting in key order, lead to the worst-case scenario

Solution: Require a **Balance Condition** that

1. Ensures depth is always $O(\log n)$ – strong enough!
2. Is efficient to maintain – not too strong!
Potential Balance Conditions

1. Left and right subtrees of the root have equal number of nodes

   Too weak!
   Height mismatch example:

2. Left and right subtrees of the root have equal height

   Too weak!
   Double chain example:
Potential Balance Conditions

3. Left and right subtrees of every node have equal number of nodes

   Too strong!
   Only perfect trees ($2^n - 1$ nodes)

4. Left and right subtrees of every node have equal height

   Too strong!
   Only perfect trees ($2^n - 1$ nodes)
The AVL Balance Condition

Left and right subtrees of every node have heights differing by at most 1

Definition: \( \text{balance}(\text{node}) = \text{height}(\text{node}.\text{left}) - \text{height}(\text{node}.\text{right}) \)

AVL property: for every node \( x \), \(-1 \leq \text{balance}(x) \leq 1\)

- Ensures small depth
  - Will prove this by showing that an AVL tree of height \( h \) must have a number of nodes exponential in \( h \)

- Efficient to maintain
  - Using single and double rotations
The AVL Tree Data Structure

Structural properties
1. Binary tree property
2. Balance property:
   balance of every node is between -1 and 1

Result:
   **Worst-case** depth is $O(\log n)$

Ordering property
   - Same as for BST
An AVL tree?
An AVL tree?
The shallowness bound

Let $S(h) =$ the minimum number of nodes in an AVL tree of height $h$
– If we can prove that $S(h)$ grows exponentially in $h$, then a tree with $n$ nodes has a logarithmic height

• Step 1: Define $S(h)$ inductively using AVL property
  – $S(-1)=0$, $S(0)=1$, $S(1)=2$
  – For $h \geq 1$, $S(h) = 1+S(h-1)+S(h-2)$

• Step 2: Show this recurrence grows really fast
  – Can prove for all $h$, $S(h) > \phi^h - 1$ where
    $\phi$ is the golden ratio, $(1+\sqrt{5})/2$, about 1.62
  – Growing faster than $1.6^h$ is “plenty exponential”
    • It does not grow faster than $2^h$
Before we prove it

• Good intuition from plots comparing:
  – \( S(h) \) computed directly from the definition
  – \( ((1+\sqrt{5})/2)^h \)
• \( S(h) \) is always bigger, up to trees with huge numbers of nodes
  – Graphs aren’t proofs, so let’s prove it
The Golden Ratio

\[ \phi = \frac{1 + \sqrt{5}}{2} \approx 1.62 \]

This is a special number

- Aside: Since the Renaissance, many artists and architects have proportioned their work (e.g., length:height) to approximate the golden ratio: If \((a+b)/a = a/b\), then \(a = \phi b\)

- We will need one special arithmetic fact about \(\phi\):

\[
\phi^2 = \left( \frac{1 + \sqrt{5}}{2} \right)^2 \\
= \left( 1 + 2 \times \frac{\sqrt{5}}{2} + 5 \right) / 4 \\
= \left( 6 + 2 \times \frac{\sqrt{5}}{2} \right) / 4 \\
= \left( 3 + \sqrt{5} / 2 \right) / 2 \\
= 1 + (1 + 5^{1/2}) / 2 \\
= 1 + \phi
\]
The proof

Theorem: For all $h \geq 0$, $S(h) > \phi^h - 1$

Proof: By induction on $h$

Base cases:

$S(0) = 1 > \phi^0 - 1 = 0$
$S(1) = 2 > \phi^1 - 1 \approx 0.62$

Inductive case ($k > 1$):

Show $S(k+1) > \phi^{k+1} - 1$ assuming $S(k) > \phi^k - 1$ and $S(k-1) > \phi^{k-1} - 1$

$S(k+1) = 1 + S(k) + S(k-1)$ by definition of $S$

$> 1 + \phi^k - 1 + \phi^{k-1} - 1$ by induction

$= \phi^k + \phi^{k-1} - 1$ by arithmetic ($1-1=0$)

$= \phi^{k-1} (\phi + 1) - 1$ by arithmetic (factor $\phi^{k-1}$)

$= \phi^{k-1} \phi^2 - 1$ by special property of $\phi$

$= \phi^{k+1} - 1$ by arithmetic (add exponents)
Good news

Proof means that if we have an AVL tree, then \( \text{find} \) is \( O(\log n) \)

- Recall logarithms of different bases \( > 1 \) differ by only a constant factor

But as we insert and delete elements, we need to:
1. Track balance
2. Detect imbalance
3. Restore balance

Is this AVL tree balanced?
How about after \( \text{insert}(30) \)?
An AVL Tree

Track height at all times!
AVL tree operations

• AVL find:
  – Same as BST find

• AVL insert:
  – First BST insert, then check balance and potentially “fix” the AVL tree
  – Four different imbalance cases

• AVL delete:
  – The “easy way” is lazy deletion
  – Otherwise, do the deletion and then have several imbalance cases (we will likely skip this but post slides for those interested)