Compsci 201
Binary Trees
Recurrence Relations

Owen Astrachan
ola@cs.duke.edu
October 24, 2018
O is for …

• Object Oriented
  • Programming with inheritance and classes
• Open Source
  • Copyright meets the Creative Commons
• O-Notation
  • Measuring in the limit
Plan for Today

• Binary Trees, Binary Search Trees
  • APTs, Recursion
  • Trees in Context: TreeSet

• Comparable: from trees to sorting
  • How to compare one thing to another

• Recurrence Relations
  • Measuring recursive algorithms/functions
Motivation for Trees

- HashSet and HashMap are $O(1)$ average
  - Astonishing! Search, insert, delete
  - No order for keys, sometimes order matters
  - **Worst-case?** Everything in same locker/bucket
    - Just in case? Use a tree in that locker/bucket

- **Search Trees:** TreeSet and TreeMap
  - $O(\log N)$ no matter what, average and worst
  - "Alphabetical" order and range queries
    - Find all keys in range [low,high] efficiently
Why Trees are $O(\log N)$

• With each query: eliminate half of tree
  • 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1
• Can ensure trees are balanced: TreeSet/TreeMap
  • Re-balance on add or delete
Java-isms for comparing

• We can compare int, double, char
  • Using ==, and !=, and <, <=, >, >=
  • Can write true != false, but *not* true > false

• Cannot write "apple" < "zebra"
  • Must compare objects using specific method
  • Objects must be *comparable*, that is they must implement the *Comparable* interface
Strings are Comparable

• Compare strings lexicographically, natural ordering, dictionary order
  • “zebra” > “aardvark” but “Zebra” < “aardvark”
  • Conceptual, cannot use < or > or ==

• "yak".compareTo(s) returns < 0, == 0, > 0
  • s is “zebra”, “yak”, and “toad”, respectively

• The int convention also used in C++, C, others
Not Everything is Comparable
Comparable in Java?

• String implements Comparable<String>
  "hello".compareTo("goodbye")

• Integer implements Comparable<Integer>
  new Integer(5).compareTo(new Integer(6))

• Cannot compare ArrayLists or arrays
  • Note: .equals works for ArrayList, not arrays
Don't do this at home: \((x,y) < (z,w)\)

- Can we compare Point objects?

- Let's look at the Java code that makes a Point comparable to another Point
  - `Point` implements `Comparable<Point>`
    - `public int compareTo(Point other)`
Build on What You Know

• How does .equals work?
  • Make sure you have the correct type
  • Cast, compare

```java
public boolean equals(Object o) {
    if (o == null || ! (o instanceof Point)) {
        return false;
    }
    Point p = (Point) o;
    return p.x == x && p.y == y;
}
```
Extend what you know

• This is method in Point class

Point implements Comparable<Point>

```java
public int compareTo(Point p) {
    if (this.x < p.x) return -1;
    if (this.x > p.x) return 1;
    // what must be true here?
    if (this.y < p.y) return -1;
    if (this.y > p.y) return 1
    return 0;
}
```
Useful math trick: Faster? Care?

● Use subtraction to help with return values

http://stackoverflow.com/questions/2654839/rounding-a-double-to-turn-it-into-an-int-java

```java
public int compareTo(Point p) {
    int deltaX = (int) Math.round(x - p.x);
    int deltaY = (int) Math.round(y - p.y);
    if (deltaX == 0) return deltaY;
    return deltaX;
}
```
Comparable Elements

- TreeSet<String>, TreeMap<String, Anything>
  - Tree elements must be comparable
    - Must implement Comparable<..>
  - It's possible to supply a Comparator, later

- Arrays.sort, Collections.sort
  - What algorithm is used in sorting?
  - Can change order of sort: Comparator, later
Jan Cuny

Program officer at National Science Foundation (NSF)
Leading #CSforAll initiatives.

2009 ABI Woman of Vision Award for Social Impact,
2016 Distinguished Educator Award

“All of today’s kids will need – along with reading, writing, and arithmetic – a basic understanding of computation and the role it plays across a wide range of disciplines.”
From Recursion to Recurrence

• Base case for trees: empty tree and maybe leaf
  • if (t == null) or
  • if (t.left == null && t.right == null)

• Make recursive calls on subtrees
  • Use results to create return value
  • Solve for one node, recursion does the rest
Tree function: Tree height

• Compute tree height (longest root-to-leaf path)

```java
int height(Tree root) {
    if (root == null) return 0;
    else {
        return 1 + Math.max(height(root.left), height(root.right));
    }
}
```

• Find height of left subtree, height of right subtree
  • Use results to determine height of tree
Complexity/Efficiency

• Intuitively: visit every node once for height: $O(N)$
  • How can we analyze this mathematically?

• Write a recurrence relation describing runtime
  • Eventually we will solve, but for now? Write

• Let $T(n)$ be time for height to run on n-node tree
  • Then $T(0) = O(1)$
  • Then $T(n) = O(1) + 2*T(n/2)$ balanced tree
Balanced Trees and Complexity

- A tree is height-balanced if
  - Left and right subtrees are height-balanced
  - Left and right heights differ by at most one
- Which trees are balanced?
Balanced Trees and Complexity

- A tree is height-balanced if
  - Left and right subtrees are height-balanced
  - Left and right heights differ by at most one

```java
public boolean isBalanced(Tree root) {
    if (root == null) return true;
    int lh = height(root.left);
    int rh = height(root.right);

    return isBalanced(root.left) &&
           isBalanced(root.right) &&
           Math.abs(lh - rh) <= 1;
}
```
Complexity of isBalanced

• We know that height(root) is $O(N)$ for $N$-node tree
  • Recurrence is $T(N) = 2T(N/2) + O(1)$

• Recurrence for isBalanced (average case)
  • $T(N)$ is time for isBalanced on $N$-node tree
  • Call height twice: $O(N)$, each tree $N/2$ nodes
  • Make two recursive calls $2T(N/2)$
• Recurrence: $T(N) = 2T(N/2) + O(N)$
Solve recurrence

- T(n) is time for isBalanced to run on n-node tree
  - T(n) = 2T(n/2) + n
  - = 2 [2 (T(n/4) + n/2) + n]
  - = 4T(n/4) + n + n = 4T(n/4) + 2n
  - = 4 [2T(n/8) + n/4] + 2n
  - = 8T(n/8) + 3n
  - = 2^kT(n/2^k) + kn  Eureka!
- Holds n = 1, 2, ... Let n = 2^k, so k=log_2 n
  - = nT(1) + n log(n)
- We now have solution to recurrence!
  - O(n log n) -- base 2, but base doesn't matter
Recurrence Relations

No need to derive, remember or look up

<table>
<thead>
<tr>
<th>Recurrence</th>
<th>Algorithm</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(n) = T(n/2) + O(1)$</td>
<td>binary search</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>$T(n) = T(n-1) + O(1)$</td>
<td>sequential search</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$T(n) = 2T(n/2) + O(1)$</td>
<td>tree traversal</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$T(n) = T(n/2) + O(n)$</td>
<td>quicksort partition</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$T(n) = 2T(n/2) + O(n)$</td>
<td>mergesort, quicksort</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>$T(n) = T(n-1) + O(n)$</td>
<td>selection or bubble sort</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>
WOTO

Tree APTs

- Must include TreeNode class in project src
  - Similar to ListNode, not submitted as code

- Helper method often useful: add a parameter
  - Can be ArrayList<> or String or ...

- For running/testing outside of APT system?
  - Create tree using code in class folder
  - [https://github.com/astrachano/classcode201fall18/blob/master/src/TreeDemo.java](https://github.com/astrachano/classcode201fall18/blob/master/src/TreeDemo.java)
LeafSum:

- [https://www2.cs.duke.edu/csed/newapt/leafsum.html](https://www2.cs.duke.edu/csed/newapt/leafsum.html)
- Sum all the values in leaves of tree
  - Base cases?
  - Recursive calls?
- What should big-Oh be?
  - N-node tree?
  - Visit each node ….
  - Balanced or stringy …
  - Similar to tree height
LeafSum correct?

• What do we do with null tree? Why?
  • From base-case to combining recursive calls
  • Recurrence expression?

```java
public class LeafSum {
    public int sum(TreeNode t) {
        if (t == null) return 0;
        // something is missing here!
        return sum(t.left) + sum(t.right);
    }
}
```
LeafSum Now Correct!

• What do we do with a leaf, why?
  • Why else isn’t needed, but ok
  • Why this is still O(n) for ALL TREES!

```java
public class LeafSum {
    public int sum(TreeNode t) {
        if (t == null) return 0;
        if (t.left == null && t.right == null) return t.info;
        return sum(t.left) + sum(t.right);
    }
}
```
AllPaths

- [https://www2.cs.duke.edu/csed/newapt/allpaths.html](https://www2.cs.duke.edu/csed/newapt/allpaths.html)
- Use instance variable or parameter to helper
  - Path is parameter to helper method
  - Invariant: path from root to current node

```java
public String[] paths(TreeNode root) {
    if (root != null) {
        doPaths(root, "");
    }
    Collections.sort(myList);
    return myList.toArray(new String[0]);
}
```
Administrivia

• Looking at calendar: what's coming
  • Assignments, APTs, APT quizzes

• APT Quiz1 Redux (system or performance issues)
  • Saturday-Monday

• Final exam reminder
  • Changes not happening
Using TreeNode internally

- LinkedList class used doubly-linked list
- TreeSet and TreeMap use balanced tree
  - [https://github.com/astrachano/diyad-treestuff](https://github.com/astrachano/diyad-treestuff)

- Source code: Balanced Red/Black tree where nodes have left, right, parent pointer
  - [http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/687fd7c7986d/src/share/classes/java/util/TreeMap.java](http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/687fd7c7986d/src/share/classes/java/util/TreeMap.java)