CompSci 201
Search Trees and Recursion

Susan Rodger
March 6, 2020
P is for …

• **Patterns**
  • Object-oriented design: from decorator to …

• **Password**
  • From changeme to oh-oh

• **Phishing**
  • From changeme to bitcoin!
Announcements

• Assignment P4 DNA-Link
  • Part 1 due yesterday! – Analysis, Partner form
  • Part 2 due March 19 – Code and more Analysis

• There is Discussion for Monday, March 16
  • There is no Pre-Discussion before

• APT-5 out after break
Plan for DBSB

• Binary Trees
  • Search and more: best of array and linked lists
    • $O(1)$ insert and $O(\log n)$ search

• Understanding structure and recursion
  • List has one node and another list
  • Tree has one node/root and two subtrees

• Comparing two objects
Binary Search Trees

• Nodes have left/right references: similar prev/next
  • At each node: <= goes left, > goes right

• How do we search?

• How do we insert?

• Insert: “koala”
Binary Search Trees

- Nodes have left/right references: similar prev/next
  - At each node: <= goes left, > goes right

- How do we search?
- How do we insert?

- Insert: “koala”
Tree Terminology

- **Root**: "top node", has no parent
  - "macaque". Subtrees also have a root
- **Leaf**: bottom nodes, have no children
  - "baboon", "lemur", "organutan"
- **Path**: sequence of parent-child nodes
  - "macaque", "chimp", "lemur"
A TreeNode by any other name…

- What does this look like? Doubly linked list?

```java
public class TreeNode {
    TreeNode left;
    TreeNode right;
    String info;
    TreeNode(String s, //
               TreeNode llink, TreeNode rlink){
        info = s;
        left = llink;
        right = rlink;
    }
}
```
Trees: Concepts and Code

• In a search tree: property holds at every node
  • Nodes in left subtree are < (or <=)
  • Nodes in right subtree are >

• To search or add: if not found?
  • Look left if <=
  • Look right if >
  • Iterative or recursive
Tree Performance

• Search for any value. Compare to root and …
• Similar to binary search. $O(\log N)$ if tree "good"
  • Trees are generally well-behaved, but !!!
  • Guarantee? Balanced tree: AVL or Red-Black

• We get $O(\log N)$ search and …
  • No shifting to add, find leaf
Good Search Trees and Bad Trees

http://www.9wy.net/onlinebook/CPrimerPlus5/ch17lev1sec7.html
Printing a Search Tree

- Think of search trees as recursive/hierarchical
  - Empty OR Root/Node with two subtrees
  - What do we know about subtrees? Also tree!

```java
public void print(TreeNode root) {
    if (root != null) {
        print(root.left);
        System.out.println(root.info);
        print(root.right);
    }
}
```
public void print(TreeNode root) {
    if (root != null) {
        print(root.left);
        System.out.println(root.info);
        print(root.right);
    }
}
Print the tree - Recursion

public void print(TreeNode root) {
    if (root != null) {
        print(root.left);
        System.out.println(root.info);
        print(root.right);
    }
}

OUTPUT:
baboon chimp lemur

Believe in recursion
Print the tree - Recursion

```java
public void print(TreeNode root) {
    if (root != null) {
        print(root.left);  // 2
        System.out.println(root.info);  // 2
        print(root.right);
    }
}
```

OUTPUT:
baboon chimp lemur macaque
Print the tree - Recursion

OUTPUT:
baboon chimp lemur macaque monkey orangutan tamarin

Believe in recursion
Print the tree - Recursion

```java
public void print(TreeNode root) {
    if (root != null) {
        print(root.left);
        System.out.println(root.info);
        print(root.right);
    }
}
```
If you don’t believe in recursion yet, let’s see all the steps.

A green check mark will mean we have finished all the recursion for a node.
Print the tree - Recursion

```java
30  public void print(TreeNode root) {
31      if (root != null) {
32          print(root.left);
33          System.out.println(root.info);
34          print(root.right);
35  }
36}
```

OUTPUT:
Print the tree - Recursion

```
public void print(TreeNode root) {
    if (root != null) {
        print(root.left);
        System.out.println(root.info);
        print(root.right);
    }
}
```

OUTPUT:
baboon chimp lemur macaque monkey orangutan tamarin
Print the tree - Recursion

```java
public void print(TreeNode root) {
    if (root != null) {
        print(root.left); // 1
        System.out.println(root.info); // 2
        print(root.right); // 3
    }
}
```
Constructing and Printing Tree

- Code and visualize: constructor has 3 parameters
  - Info, Left Subtree, Right Subtree

```java
public static void main(String[] args) {
    TreeNode root =
        new TreeNode("mango",
            new TreeNode("durian",
                new TreeNode("apple",null,null),
                new TreeNode("grapefruit",null,null)),
            new TreeNode("pear",
                new TreeNode("orange",null,null),
                new TreeNode("tangerine",null,null)));
    print(root);
}
```
public static void main(String[] args) {
    TreeNode root =
        new TreeNode("mango",
                new TreeNode("durian",
                        new TreeNode("apple", null, null),
                        new TreeNode("grapefruit", null, null)),
                new TreeNode("pear",
                        new TreeNode("orange", null, null),
                        new TreeNode("tangerine", null, null)));
    print(root);
}
Visualize

- A different tree:
  - Left subtree?
  - Right subtree

```java
public static void main(String[] args) {
    TreeNode root =
    new TreeNode("mango",
    new TreeNode("durian",
    new TreeNode("apple",null,null),
    new TreeNode("grapefruit",null,null)),
    new TreeNode("pear",
    new TreeNode("orange",null,null),
    new TreeNode("tangerine",null,null)));
    print(root);
}
```
Three recursive calls

- "apple" node
- null, null
- Up to "durian"
Standard Tree Traversals

• Pre-, In-, and Post-order
  • When is root visited? Before, in-between, after
• Analogy: traveling the border: down, under, up
  • [https://coursework.cs.duke.edu/201spring20/classcode](https://coursework.cs.duke.edu/201spring20/classcode)
  • See TreeDemo.java, alphabetical for search tree
Inorder traversal – the print we just did

- Analogy: traveling the border: under a node
- Useful to print the elements in order

```java
public void inOrder(TreeNode root) {
    if (root != null) {
        inOrder(root.left);
        System.out.println(root.info);
        inOrder(root.right);
    }
}
```

baboon, chimp, lemur, macaque, monkey, orangutan, tamarin
Preorder traversal

- Analogy: traveling the border: on the way down
- Useful to read and write trees: Huffman

```java
public void preOrder(TreeNode root) {
    if (root != null) {
        System.out.println(root.info);
        preOrder(root.left);
        preOrder(root.right);
    }
}
```

macaque, chimp, baboon, lemur, monkey, tamarin, orangutan
Postorder traversal

• Analogy: traveling the border: on the way up
• Useful to destroy/delete trees

```java
public void postOrder(TreeNode root) {
    if (root != null) {
        postOrder(root.left);
        postOrder(root.right);
        System.out.println(root.info);
    }
}
```

baboon, lemur, chimp, orangutan, tamarin, monkey, macaque
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
WOTO

Richard Stallman

• Created "free" software foundation
  • Speech not beer
  • Wrote Gnu C compiler
  • No Linux without gcc
  • MacArthur award, Hopper award

• Maybe world's best programmer?

You and I we exist for ourselves, fundamentally. We should care about others but each human being is a source of value, each human being deserves things. And so if you lose control over your computing, that's bad for you, directly bad for you. So my first reaction is to say: Oh, what a shame; I hope you recover the control over your computing and the way you do that is to stop using the non-free software.
Not Everything is Comparable
Java-isms for comparing

• We can compare int, double, char
  • Using ==, and !=, and <, <=, >, >=
  • Primitives use conventional symbols

• *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 ==
    - We had to use `s.equals(t)` for strings/objects

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

- The int convention also used in C++, C, others
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>

    Note: parameter is Point and not Object

    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
  
  [Code]
  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;
  }

  [URL]
  http://stackoverflow.com/questions/2654839/rounding-a-double-to-turn-it-into-an-int-java
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
CompPoint.java in Action

- [https://coursework.cs.duke.edu/201spring20/classcode](https://coursework.cs.duke.edu/201spring20/classcode)

- We can sort collection of CompPoint objects, what's printed?
  - What if we change the `.compareTo` method?

```java
public static void main(String[] args) {
    ArrayList<CompPoint> list = new ArrayList<>();
    list.add(new CompPoint(2, 7));
    list.add(new CompPoint(2, 5));
    Collections.sort(list);
    for(CompPoint p : list) {
        System.out.println(p);
    }
}
```
WOTO

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.”