PFTFBH

- Binary Search trees
  - Fundamental Data Structure
  - Recursive and leads to recurrence relations to analyze performance
- Review some concepts from Recitation/Discussion
  - Revisit doubly-linked lists and see how Java uses both trees and lists in java.util
- Understanding the DNA Assignment
  - Motivation for linked lists
  - What you do and how you can go about doing it

How does cutAndSplice Work?

- Find enzyme 'gat'
  - Replace with DNA splicee 'gggtttaaa'
- Strings and StringBuilders, complexity of A + B
  - Either A + B or B, does this make a difference?
  - What if done N times?
Plan of Action: re trees

- Trees from top to bottom
- Why trees are useful, tradeoffs in types of trees
  - How trees are implemented
  - Where trees are used, data structures and algorithms
- We’ll concentrate on binary trees
  - A tree can be empty
  - A tree consists of a (root (subtree) (subtree))
  - Analyzing tree functions with recurrences

From doubly-linked lists to binary trees

- Re-imagine prev/next, no longer linear
  - Similar to binary search, everything less goes left, everything greater goes right
- How do we search?
- How do we insert?

Binary Trees for performance reasons

- Search and insert: toward the best of both worlds
  - Linked list: efficient insert/delete, inefficient search
  - ArrayList: efficient (binary) search if sorted, but shift to insert
- Binary trees: efficient insert, delete, and search
  - Not just for searching, used in many contexts,
    - Game trees, collisions, cladistics, genomics, quad trees,
    - search in $O(\log n)$ like sorted array
      - Average case. Note: worst case can be avoided!
    - insertion/deletion $O(1)$, once location found
Good Search Trees and Bad Trees

http://www.9wy.net/onlinebook/CPrimerPlus5/ch17lev1sec7.html

https://git.cs.duke.edu/201fall16/set-examples/tree/master

- What about ISimpleSet interface
  - How does this compare to java.util?
  - Occam’s razor v. KISS v. ...

- What does a simple implementation look like?
  - What are complexity repercussions: add, contains
  - What about iterating?

- How do TreeSet and TrieSet and BSTSet compare?
  - We can look at these to study code
  - Some recursive, some iterative

Why Study Binary Trees? In 201?

- Basis for understanding search
  - Fundamental algorithm and data structure
  - Basis for other trees, red-black, 2-3, quad, ...

- Simple structure for demonstrating understanding of recursion and analysis
  - Also powerful, now basis for hashmap bucket in Java 8, and much more

- Expected as basic knowledge of computer science

A TreeNode by any other name...

public class TreeNode
{
    TreeNode left;
    TreeNode right;
    String info;
    TreeNode(String s, TreeNode llink, TreeNode rlink)
    {
        info = s;
        left = lllink;
        right = rlink;
    }
}
Printing a search tree in order

- **When is root printed?**
  - After left subtree, before right subtree.

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

- **Inorder traversal**

Visit tree using preorder traversal

- **When is root printed?**
  - Before left subtree, before right subtree.

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

- **Preorder traversal**

Visit tree using postorder traversal

- **When is root printed?**
  - After left subtree, after right subtree.

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

- **Postorder traversal**
Tree Questions


- What aspects of trees aren't straightforward in terms of understanding terminology?
- Let's look at terminology and code as part of next steps

Review: tree terminology

- Binary tree is a structure:
  - empty
  - root node with left and right subtrees

- Tree Terminology
  - parent and child: A is parent of B, E is child of B
  - leaf node has no children, internal node has 1 or 2 children
  - path is sequence of nodes (edges), N₀ N₁ N₂ ... Nₖ
    - Nᵢ is parent of Nᵢ₊₁
  - depth (level) of node: length of root-to-node path
    - level of root is 1 (measured in nodes)
  - height of node: length of longest root-to-leaf path
    - height of tree is height of root

Tree functions

- Compute height of a tree, what is complexity?
  - Length of 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));
      }
    }
    ```
- Modify function to compute number of nodes in a tree, does complexity change?
  - What about computing number of leaf nodes?

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
    boolean isBalanced(Tree root) {
      if (root == null) return true;
      return isBalanced(root.left) && isBalanced(root.right) &&
        Math.abs(height(root.left) - height(root.right)) <= 1;
    }
    ```
What is complexity?

- Assume trees “balanced” in analyzing complexity
  - Roughly half the nodes in each subtree
  - Leads to easier analysis

- How to develop recurrence relation?
  - What is T(n)? Time func executes on n-node tree
  - What other work? Express recurrence, solve it

- How to solve recurrence relation
  - Plug, expand, plug, expand, find pattern
  - Proof requires induction to verify correctness

Recurrence relation

- Let T(n) be time for height to execute (n-node tree)
  - T(n) = T(n/2) + T(n/2) + O(1)
  - T(n) = 2 T(n/2) + 1
  - T(n) = 2 [T(n/4) + 1] + 1
  - T(n) = 4T(n/4) + 2 + 1
  - T(n) = 8T(n/8) + 4 + 2 + 1, eureka!
  - T(n) = 2^k T(n/2^k) + 2^k - 1 why is this true?
  - T(n) = nT(1) + O(n) is O(n), if we let n=2^k

- Let T(n) be time for isBalanced on n-node tree
  - T(n) = 2 T(n/2) + O(n), why? Solution?

What does insertion look like?

- Simple recursive insertion into tree (accessed by root)
  - root = insert("foo", root);

```java
TreeNode insert(TreeNode t, String s) {
  if (t == null) t = new Tree(s, null, null);
  else if (s.compareTo(t.info) <= 0)
    t.left = insert(t.left, s);
  else
    t.right = insert(t.right, s);
  return t;
}
```

https://git.cs.duke.edu/201fall16/set-examples/tree/master

- What about ISimpleSet interface
  - How does this compare to java.util?
  - What about Java source? Can we look at it?

- What does a simple implementation look like?
  - What are complexity repercussions: add, contains
  - What about iterating?

- Scenarios where linked lists better?
  - Consider N adds and M contains operations
  - Move to front heuristic?
Notes on tree insert and search

- Note: in each recursive insert call, the parameter t in the called clone is either the left or right pointer of some node in the original tree
  - Why is this important?
  - The idiom \( t = \text{treeMethod}(t, \ldots) \) used
- When good trees go bad, what happens and why?
  - Insert alpha, beta, gamma, delta, epsilon, ...
  - Where does gamma go?
    - Can we avoid this case? Yes!
  - What to prefer? Long/stringy or short/bushy

Removal from tree?

- For insertion we can use iteration (see BSTSet)
  - Look below, either left or right
    - If null, stop and add
    - Otherwise go left when <=, else go right when >
- Removal is tricky, depends on number of children
  - Straightforward when zero or one child
  - Complicated when two children, find successor
    - See set code for complete cases
    - If right child, straightforward
    - Otherwise find node that's left child of its parent (why?)

Exploring Discussion/Recitation

- How do we verify? How do we analyze?
  - \([1, 2, 2, 3, 3, 4, 4, 4, 4, 4, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7]\)
  - Correctness and Performance

```java
public Node createGaussList(int n){
    if (n == 1) return new Node(1,null);
    Node first = createGaussList(n-1);
    Node last = first;
    while (last.next != null){
        last = last.next;
    }
    last.next = createNlist(n);
    return first;
}
```
Doubly Linked Lists

- Why do we have some lists with nodes to previous and next nodes?
  - Easier to write code, don't need before, current, after ... [link to Wikipedia]
  - See examples in Recitation
- Used in Java through Java 7 for HashMap
- Still used in LinkedList, easy traversal from front or back, also LinkedHashSet
- See DNA LinkStrand with singly linked list

LinkedHashMap in code

- [Link to code example]
  - Note that each hash "bucket" uses a search tree to store (key,value) pairs where keys have same hashcode
  - Search tree nodes linked using doubly-linked list with before and after pointers