TREES

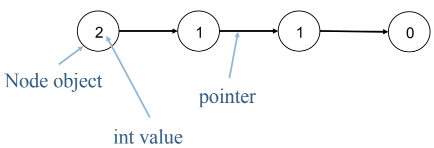
Lecture 12
CS2110 – Fall 2017

Important Announcements

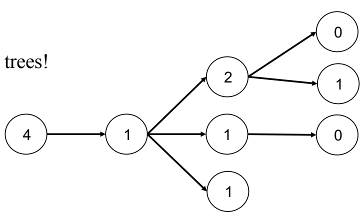
2

- A4 is out now and due two weeks from today. Have fun, and start early!

A picture of a singly linked list:



Today: trees!



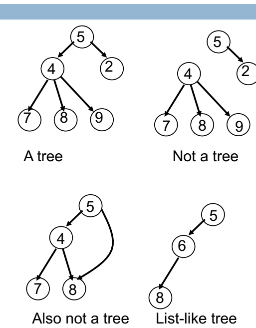
3

Tree Overview

4

Tree: data structure with nodes, similar to linked list

- Each node may have zero or more *successors* (children)
- Each node has exactly one *predecessor* (parent) except the *root*, which has none
- All nodes are reachable from *root*

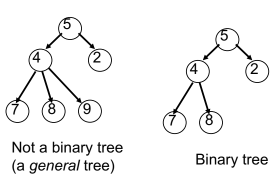


Binary Trees

5

A *binary tree* is a particularly important kind of tree where every node as at most two children.

In a binary tree, the two children are called the *left* and *right* children.

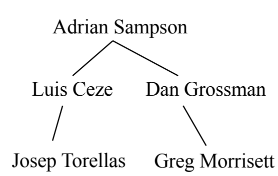


Binary trees were in A1!

6

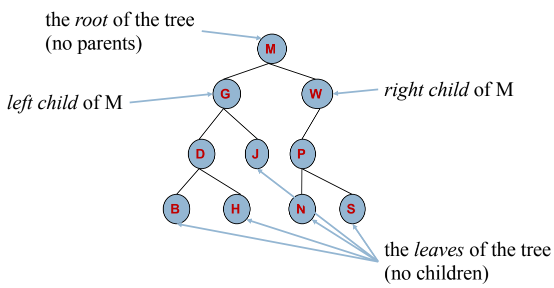
You have seen a binary tree in A1.

A PhD object has one or two advisors. (Confusingly, my advisors are my “children.”)



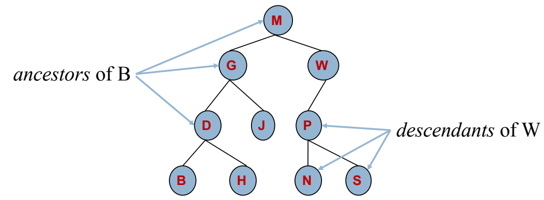
Tree Terminology

7



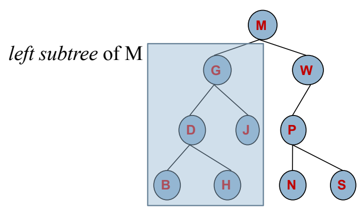
Tree Terminology

8



Tree Terminology

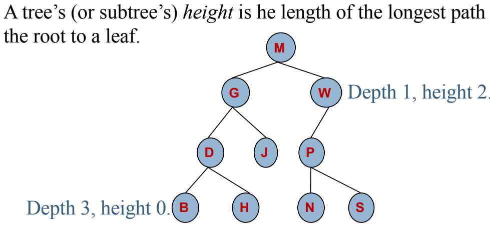
9



Tree Terminology

10

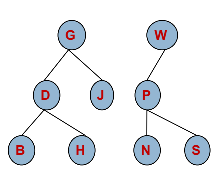
A node's *depth* is the length of the path to the root.
A tree's (or subtree's) *height* is he length of the longest path from the root to a leaf.



Tree Terminology

11

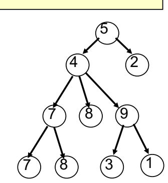
Multiple trees: a *forest*.



Class for general tree nodes

```
class GTreeNode<T> {
    private T value;
    private List<GTreeNode<T>> children;
    //appropriate constructors, getters,
    //setters, etc.
}
```

Parent contains a list of its children

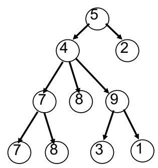


General tree

Class for general tree nodes

```
class GTreeNode<T> {
    private T value;
    private List<GTreeNode<T>> children;
    //appropriate constructors, getters,
    //setters, etc.
}
```

Java.util.List is an interface!
It defines the methods that all implementation must implement.
Whoever writes this class gets to decide what implementation to use — ArrayList? LinkedList? Etc.?



General tree

Class for binary tree node

```
class TreeNode<T> {
   private T value;
   private TreeNode<T> left, right;

   /** Constructor: one-node tree with datum x */
   public TreeNode (T d) { datum= d; left= null; right= null;}

   /** Constr: Tree with root value x, left tree l, right tree r */
   public TreeNode (T d, TreeNode<T> l, TreeNode<T> r) {
      datum= d; left= l; right= r;
   }
}
```

Either might be null if the subtree is empty.

more methods: getValue, setValue, getLeft, setLeft, etc.

Binary versus general tree

In a binary tree, each node has up to two pointers: to the left subtree and to the right subtree:

- One or both could be **null**, meaning the subtree is empty (remember, a tree is a set of nodes)

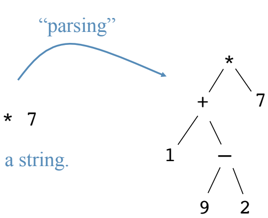
In a general tree, a node can have any number of child nodes (and they need not be ordered)

- Very useful in some situations ...
- ... one of which may be in an assignment!

An Application: Syntax Trees

`(1 + (9 – 2)) * 7`

A Java expression as a string.



An expression as a tree.

Applications of Tree: Syntax Trees

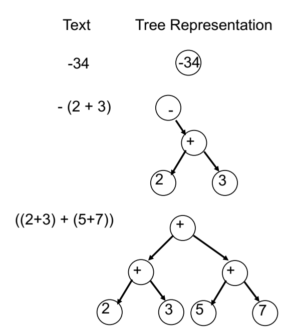
- Most languages (natural and computer) have a recursive, hierarchical structure
- This structure is *implicit* in ordinary textual representation
- Recursive structure can be made *explicit* by representing sentences in the language as trees: Abstract Syntax Trees (ASTs)
- ASTs are easier to optimize, generate code from, etc. than textual representation
- A parser converts textual representations to AST

Applications of Tree: Syntax Trees

In textual representation:
Parentheses show hierarchical structure

In tree representation:
Hierarchy is explicit in the structure of the tree

We'll talk more about expressions and trees in next lecture

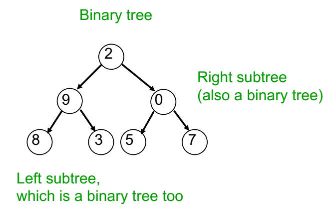


A Tree is a Recursive Thing

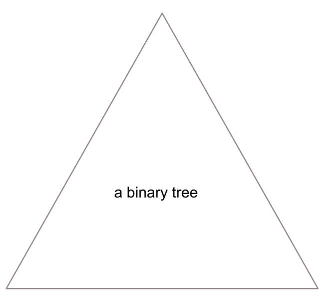
19

A binary tree is either `null` or an object consisting of a value, a left binary tree, and a right binary tree.

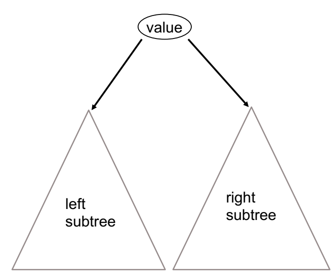
Looking at trees recursively



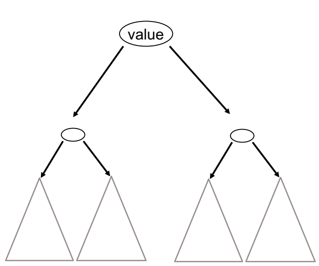
Looking at trees recursively



Looking at trees recursively



Looking at trees recursively



A Recipe for Recursive Functions

24

Base case:

If the input is "easy," just solve the problem directly.

Recursive case:

Get a smaller part of the input (or several parts).
Call the function on the smaller value(s).
Use the recursive result to build a solution for the full input.

Recursive Functions on Binary Trees

25

Base case:
- empty tree (null)
- or, possibly, a leaf

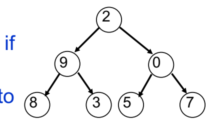
Recursive case:
- Call the function on each subtree.
- Use the recursive result to build a solution for the full input.

Searching in a Binary Tree

26

```
/** Return true iff x is the datum in a node of tree  t*/
public static boolean treeSearch(T x, TreeNode<T> t) {
   if (t == null) return false;
   if (x.equals(t.datum)) return true;
   return treeSearch(x, t.left) || treeSearch(x, t.right);
}
```

- Analog of linear search in lists: given tree and an object, find out if object is stored in tree
- Easy to write recursively, harder to write iteratively



Searching in a Binary Tree

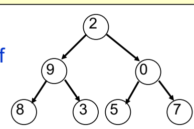
27

```
/** Return true iff x is the datum in a node of tree  t*/
public static boolean treeSearch(T x, TreeNode<T> t) {
   if (t == null) return false;
   if (x.equals(t.datum)) return true;
   return treeSearch(x, t.left) || treeSearch(x, t.right);
}
```

VERY IMPORTANT!

We sometimes talk of t as the root of the tree.

But we also use t to denote the whole tree.



Some useful methods – what do they do?

28

```
/** Method A ??? */
public static boolean A(Node n) {
   return n != null  &&  n.left == null  &&  n.right == null;
}
/** Method B ???  */
public static int B(Node n) {
   if (n== null) return -1;
   return 1 + Math.max(B(n.left), B(n.right));
}
/** Method C ???  */
public static int C(Node n) {
   if (n== null) return 0;
   return 1 + C(n.left) + C(n.right);
}
```

Some useful methods

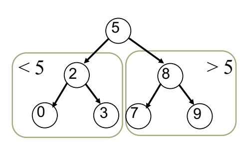
29

```
/** Return true iff node n is a leaf */
public static boolean isLeaf(Node n) {
   return n != null  &&  n.left == null  &&  n.right == null;
}
/** Return height of node n (postorder traversal) */
public static int height(Node n) {
   if (n== null) return -1; //empty tree
   return 1 + Math.max(height(n.left), height(n.right));
}
/** Return number of nodes in n (preorder traversal) */
public static int numNodes(Node n) {
   if (n== null) return 0;
   return 1 + numNodes(n.left) + numNodes(n.right);
}
```

Binary Search Tree (BST)

30

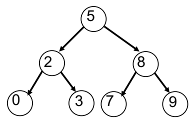
A *binary search tree* is a binary tree that is **ordered** and **has no duplicate values.** In other words, for *every* node:
- All nodes in the left subtree have values that are less than the value in that node, and
- All values in the right subtree are greater.



A BST is the key to making search way faster.

Binary Search Tree (BST)

31



Compare binary tree to binary search tree:

```
boolean searchBT(n, v):
  if n==null, return false
  if n.v == v, return true
  return searchBST(n.left, v)
      || searchBST(n.right, v)
```

2 recursive calls

```
boolean searchBST(n, v):
  if n==null, return false
  if n.v == v, return true
  if v < n.v
    return searchBST(n.left, v)
  else
    return searchBST(n.right, v)
```

1 recursive call

Building a BST

32

- To insert a new item:
  - Pretend to look for the item
  - Put the new node in the place where you fall off the tree

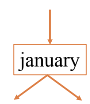
Building a BST

33

january

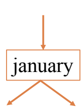
Building a BST

34



Building a BST

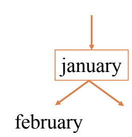
35



february

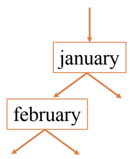
Building a BST

36



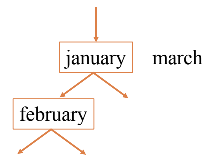
Building a BST

37



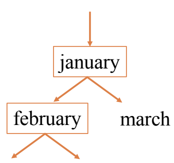
Building a BST

38



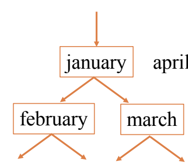
Building a BST

39



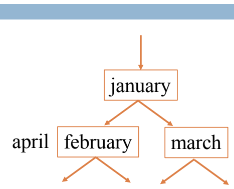
Building a BST

40



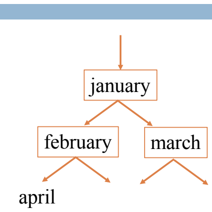
Building a BST

41

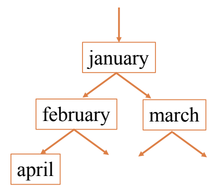


Building a BST

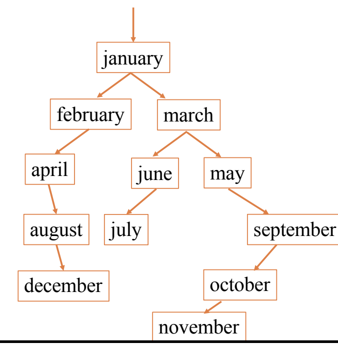
42



Building a BST



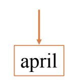
Building a BST



Inserting in Alphabetical Order

april

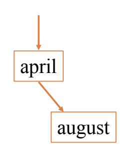
Inserting in Alphabetical Order



Inserting in Alphabetical Order

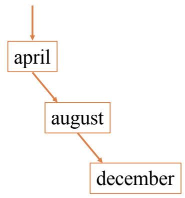
august

Inserting in Alphabetical Order



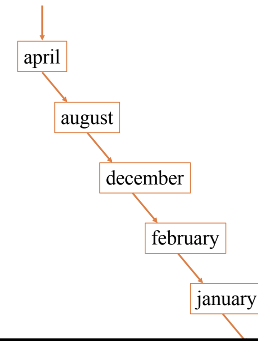
Inserting in Alphabetical Order

49



Inserting in Alphabetical Order

50



Insertion Order Matters

51

- A *balanced* binary tree is one where the two subtrees of any node are about the same size.
- Searching a binary search tree takes O(h) time, where h is the height of the tree.
- In a balanced binary search tree, this is O(log n).
- But if you insert data in sorted order, the tree becomes imbalanced, so searching is O(n).

Printing contents of BST

52

Because of ordering rules for a BST, it's easy to print the items in alphabetical order

- Recursively print left subtree
- Print the node
- Recursively print right subtree

```
/** Print BST t in alpha order */
private static void print(TreeNode<T> t) {
   if (t== null) return;
   print(t.left);
   System.out.print(t.value);
   print(t.right);
}
```

Tree traversals

53

"Walking" over the whole tree is a tree traversal

- Done often enough that there are standard names

Previous example: in-order traversal

- Process left subtree
- Process root
- Process right subtree

Note: Can do other processing besides printing

Other standard kinds of traversals

- preorder traversal
  - Process root
  - Process left subtree
  - Process right subtree
- postorder traversal
  - Process left subtree
  - Process right subtree
  - Process root
- level-order traversal
  - Not recursive: uses a queue (we'll cover this later)

Useful facts about binary trees

54

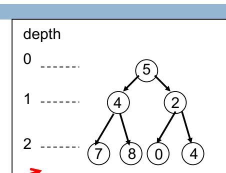
Max # of nodes at depth d: 2^d

If height of tree is h

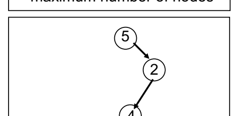
- min # of nodes: $h + 1$
- max #of nodes in tree:
- $2^0 + \ldots + 2^h = 2^{h+1} - 1$

Complete binary tree

- All levels of tree down to a certain depth are completely filled



Height 2, maximum number of nodes



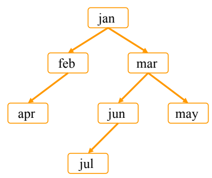
Height 2, minimum number of nodes

Things to think about

What if we want to *delete* data from a BST?

A BST works great as long as it's *balanced*.

There are kinds of trees that can *automatically* keep themselves balanced as you insert things!



Tree Summary

- A *tree* is a recursive data structure
  - Each node has 0 or more successors (*children*)
  - Each node except the *root* has exactly one predecessor (*parent*)
  - All node are reachable from the *root*
  - A node with no children (or empty children) is called a *leaf*
- Special case: *binary tree*
  - Binary tree nodes have a left and a right child
  - Either or both children can be empty (null)
- Trees are useful in many situations, including exposing the recursive structure of natural language and computer programs