CSCI 136
Data Structures & Advanced Programming

Jeannie Albrecht
Lecture 21
April 11, 2014

Administrative Details

- Lab 7
  - Due Monday at noon
- Questions?

2

Last Time

- Learned about ordered structures (Ch 11)
  - Talked about OrderedVector and OrderedList
  - Main advantage is that the data is always sorted
    - Restrict add method so that objects are always added "in the right spot"
    - Easy to find min, max, median
  - Be careful not to use mutable keys!

3

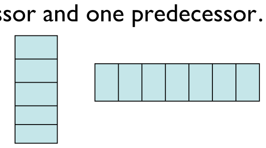
Today's Outline

- Begin learning about trees
  - Very important data structure!

4

Data Structures so far…

- So far all data has been stored in a *linear* fashion
  - Stacks, queues
  - Even arrays, vectors, SLLs are visualized using linear structures
- By linear we mean that each element has only one successor and one predecessor…



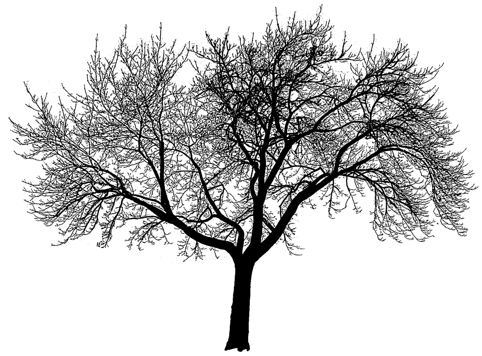
5

6

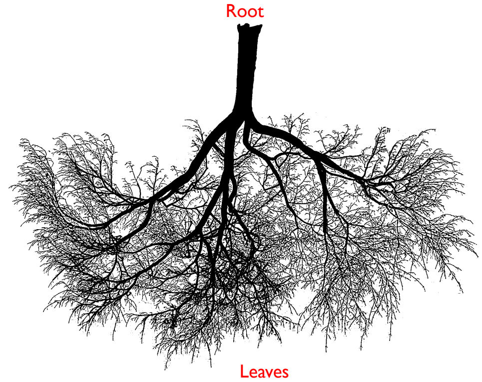
Introducing Trees

- A tree is a data structure where elements can have multiple successors (called children)
- But still only one predecessor (called parent)

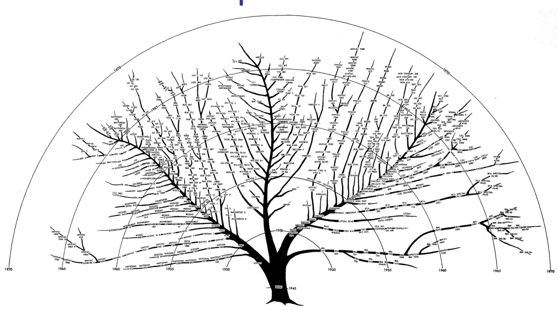
7



9



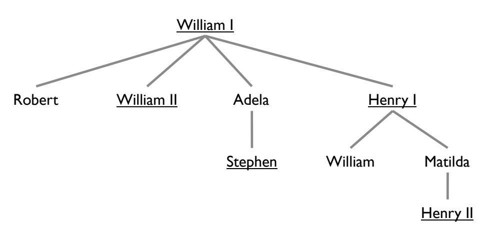
"Computer Tree"



11

Family trees

House of Normandy, Battle of Hastings, 1066

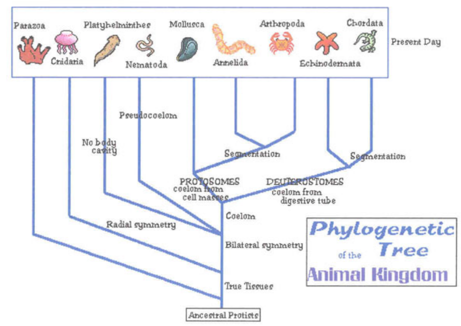


12

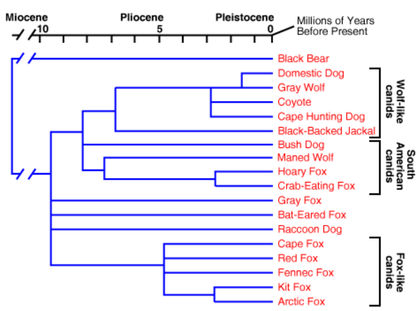
Other Trees

- Phylogenetic tree
- Directories of files
- Game tree
  - Build tree
  - Search for moves with high likelihood of winning
- Expression trees (we'll come back to these in a bit)

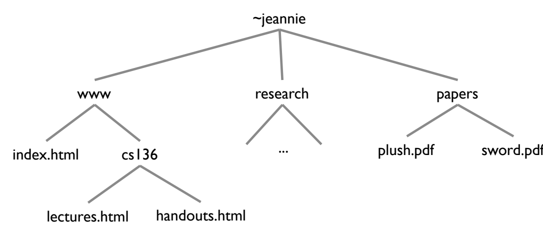
13

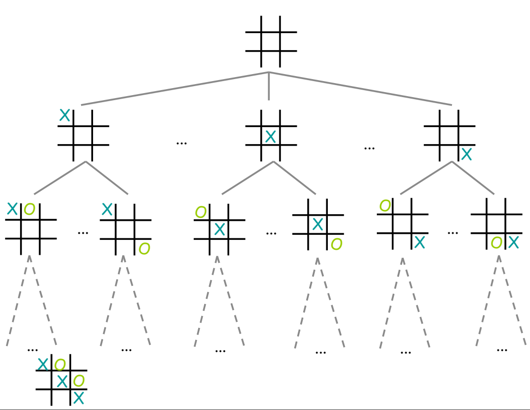


14



15

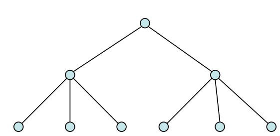




17

Trees in CS 136

- A tree is a data structure where elements can have multiple successors (but generally only one predecessor)



18

Tree Features

- Hierarchical relationship
- Root at the top
- Leaf at the bottom
- Interior nodes in middle
- Parents, children, ancestors, descendants, siblings
- Degree: max number of children per node
- Depth of node n: number of edges from root to n
- Height: max depth (across all nodes)

19

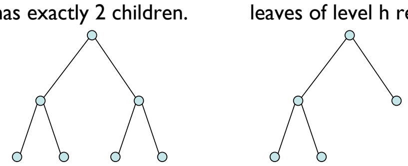
Binary Trees

- Degree of all nodes <= 2
- Recursive nature of tree
  - Base case: Empty
  - Rec. case: Root with left and right subtrees (also BTs)
- SLL: Recursive nature was captured by elements (SLLEs) that pointed to other elements (SLLEs)
- Binary Tree: No second element class; single BinaryTree class does it all!

20

Full vs. Complete

- Full tree – A full binary tree of height h has *leaves only* on level h, and each internal node has exactly 2 children.
- Complete tree – A complete binary tree of height h is a full tree with 0 or more **rightmost** leaves of level h removed.

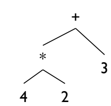


All full trees are complete, but not all complete trees are full!

21

Expression Trees

4 * 2 + 3




```
BinaryTree<String> fourTimesTwo =
                    new BinaryTree<String>("*",
                    new BinaryTree<String>("4"),
                    new BinaryTree<String>("2"));
BinaryTree<String> plusTree =
                    new BinaryTree<String>("+",
                    fourTimesTwo,
                    new BinaryTree<String>("3"));
```

22

Expression Trees

- General strategy
  - Make a binary tree (BT) for each leaf node
  - Move from bottom to top, creating BTs
  - Eventually reach the root
  - Call "evaluate" on final BT
- Example
  - How do we make a binary expression tree for (((4+3)*(10-5))/2)
    - Postfix notation: 4 3 + 10 5 - * 2 /

23

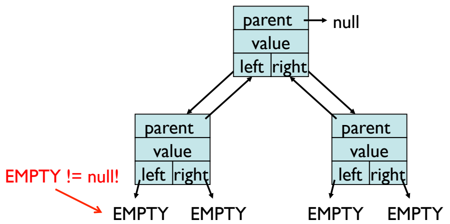
```
public int evaluate(BinaryTree<String> expr) {
    if (expr.height() == 0) {
      return Integer.parseInt(expr.value());
    } else {
      int left = evaluate(expr.left());
      int right = evaluate(expr.right());
      String op = expr.value();
      if (op.equals("+")) return left + right;
      if (op.equals("-")) return left - right;
      if (op.equals("*")) return left * right;
      if (op.equals("/")) return left / right;
      Assert.fail("Bad op");
      return -1;  //Why do we need this?
   }
}
```

BinaryExpressionTree.java

24

Implementing BinaryTree

- BinaryTree class
  - Instance variables
    - BT parent, BT left, BT right, Object value



Implementing BinaryTree

- Methods (on board – see Ch 12 for more info):
- (All “left” methods have equivalent “right” methods)
  - public BinaryTree()
    - // generates an empty node (EMPTY)
    - // parent and value are null, left=right=this
  - public BinaryTree(E value)
    - // generates a tree with a non-null value and two empty (EMPTY) subtrees
  - public BinaryTree(E value, BinaryTree<E> left, BinaryTree<E> right)
    - // returns a tree with a non-null value and two subtrees
  - public void setLeft(BinaryTree<E> newLeft)
    - // sets left subtree to newLeft
    - // re-parents newLeft (if not null) by calling newLeft.setParent(this)
  - protected void setParent(BinaryTree<E> newParent)
    - // sets parent subtree to newParent
    - // called from setLeft and setRight to keep all “links” consistent