Hierarchies

Examples of Hierarchical Structures
“In the Wild”

- Many Tournaments
  - [Look @ NFL bracket]

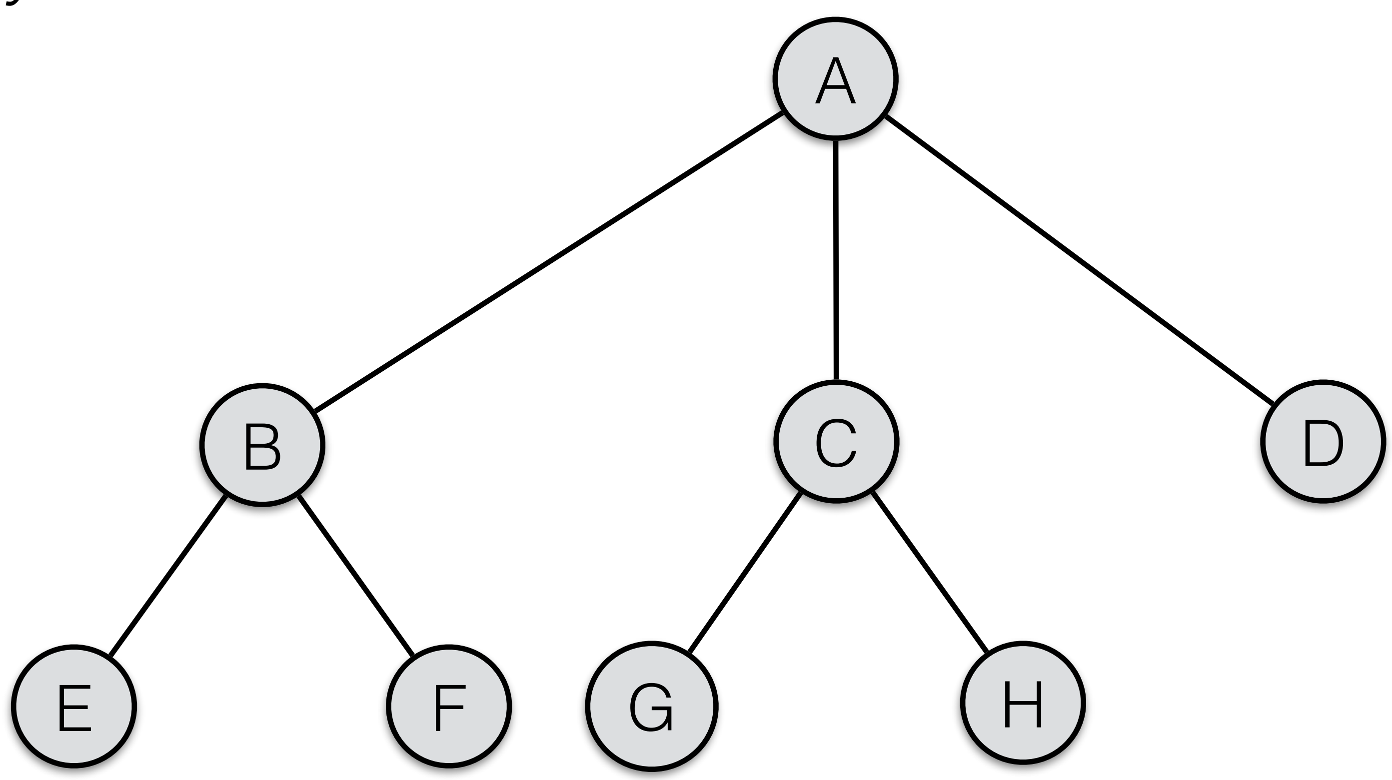
- XML and HTML documents
  - [Demo document structure in browser with Firebug]

- Folders nested inside folders nested inside … on computers
  - [Show (1) Finder, and (2) “tree” command]

Trees

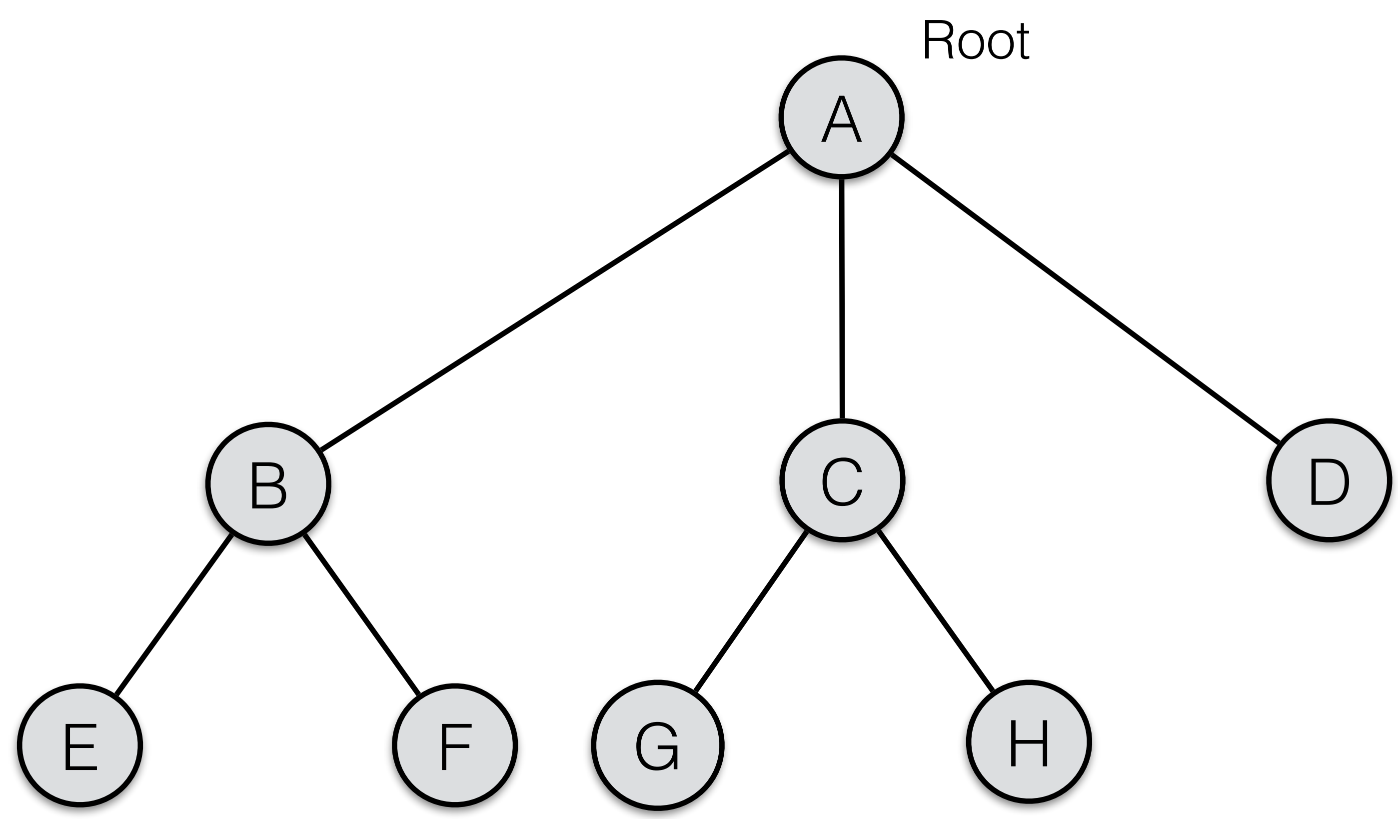
Trees

- In computer science, we can represent hierarchical relationships using a data structure called a ***tree***.
- A tree is built up from ***nodes***.
- Terminology is taken from *family trees*



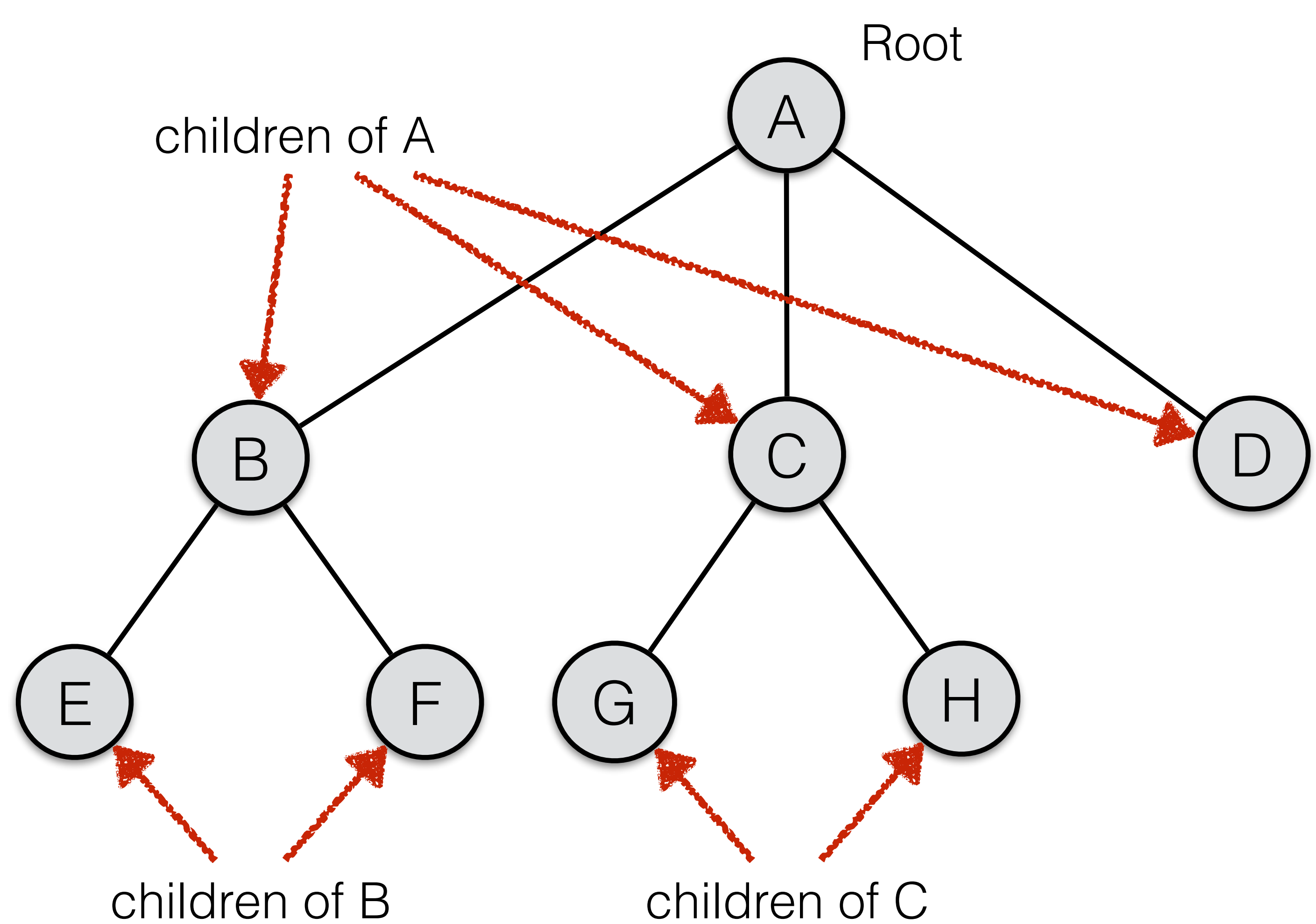
Trees

- Top node == **root**



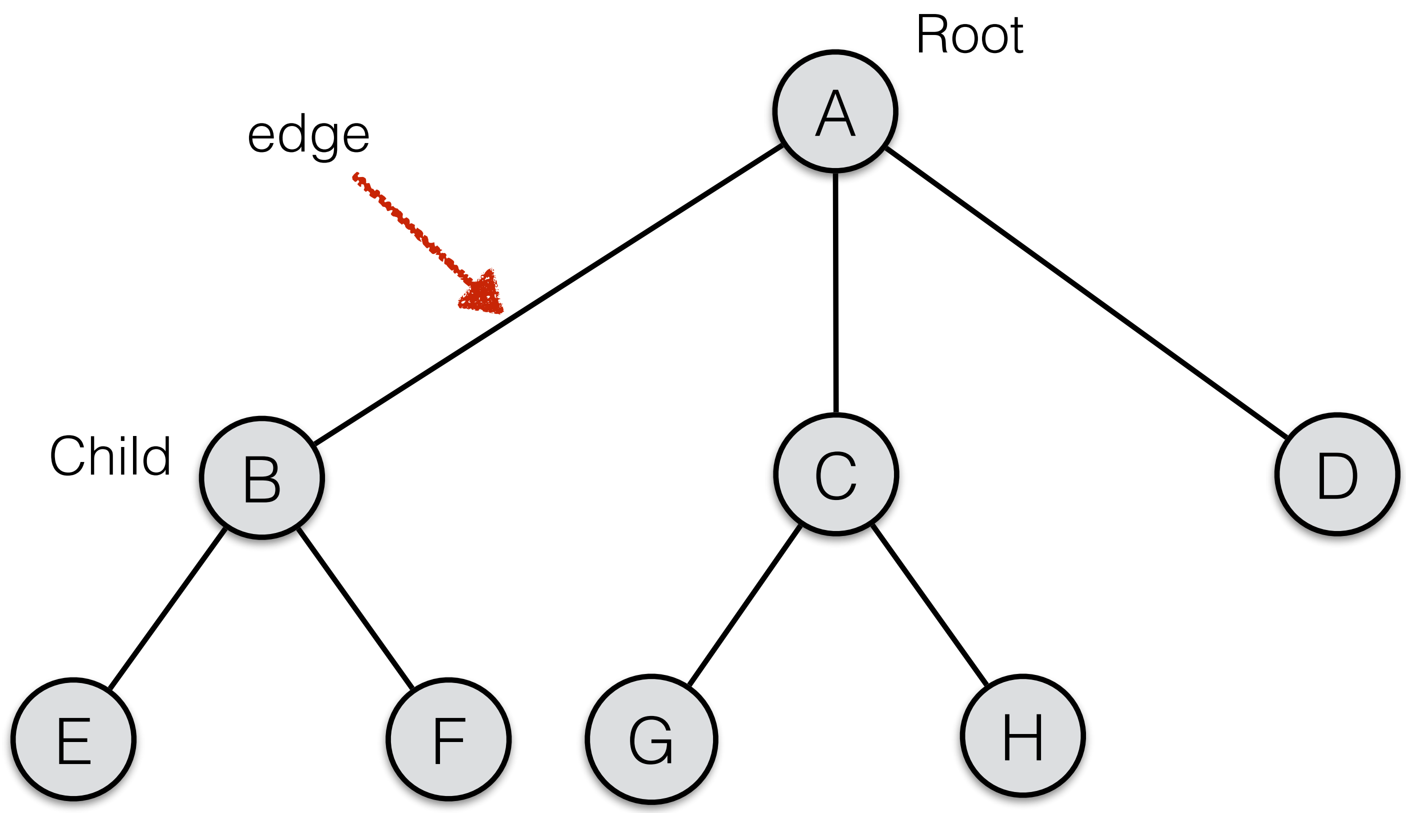
Trees

- Each node has zero or more **children**



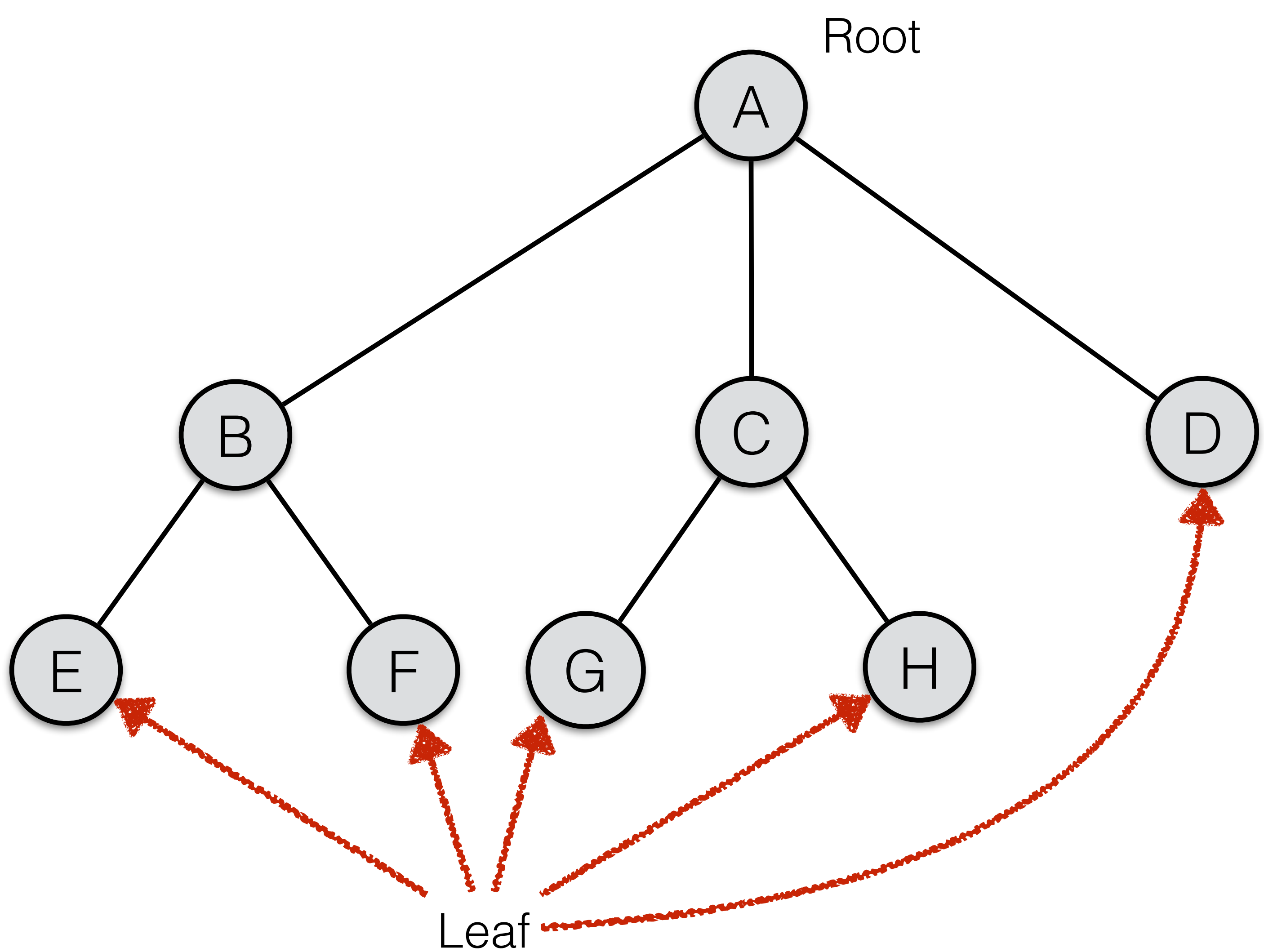
Trees

- An **edge** connects a node and a child



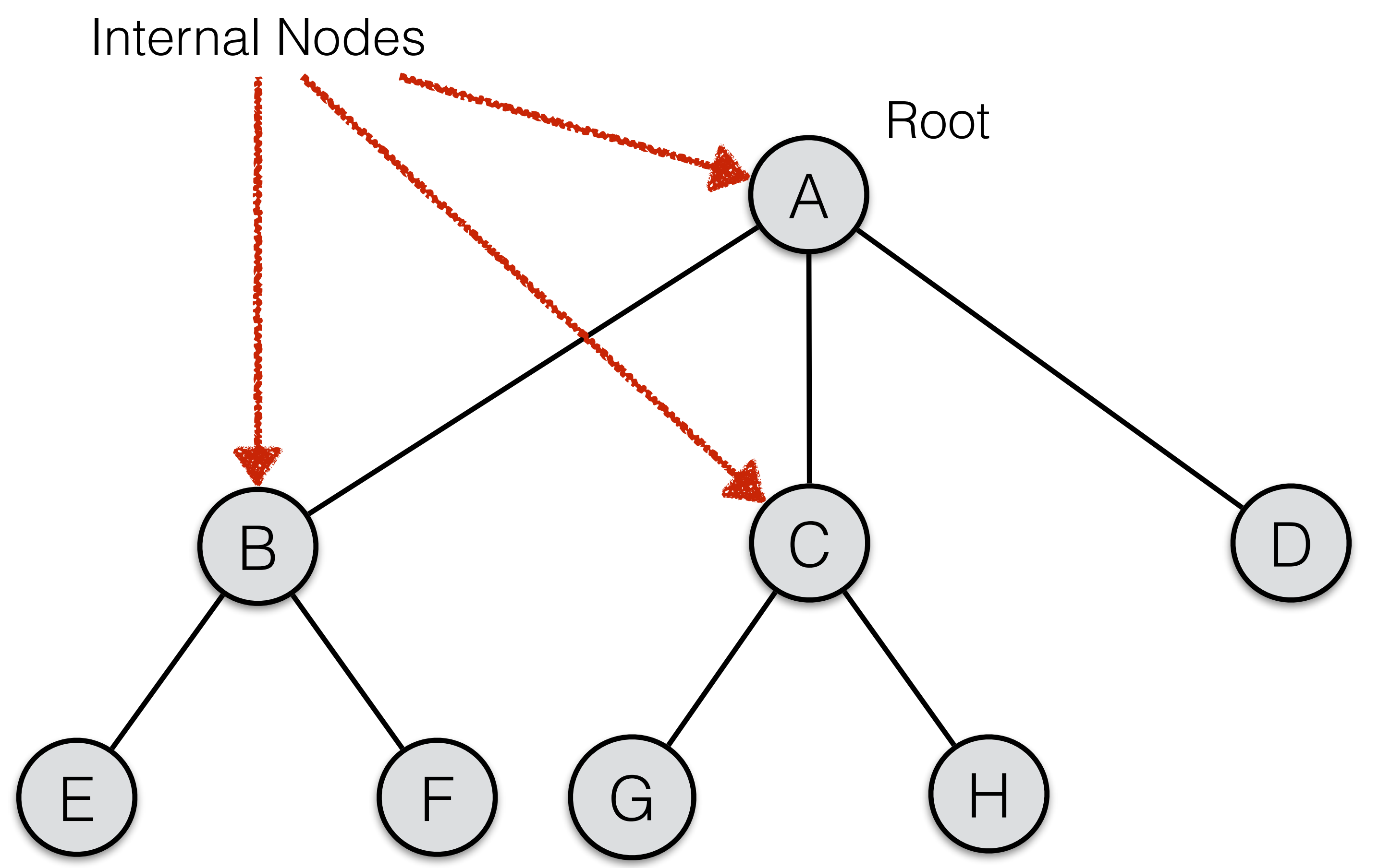
Trees

- Nodes with no children are called **external nodes** (or **leaves**)



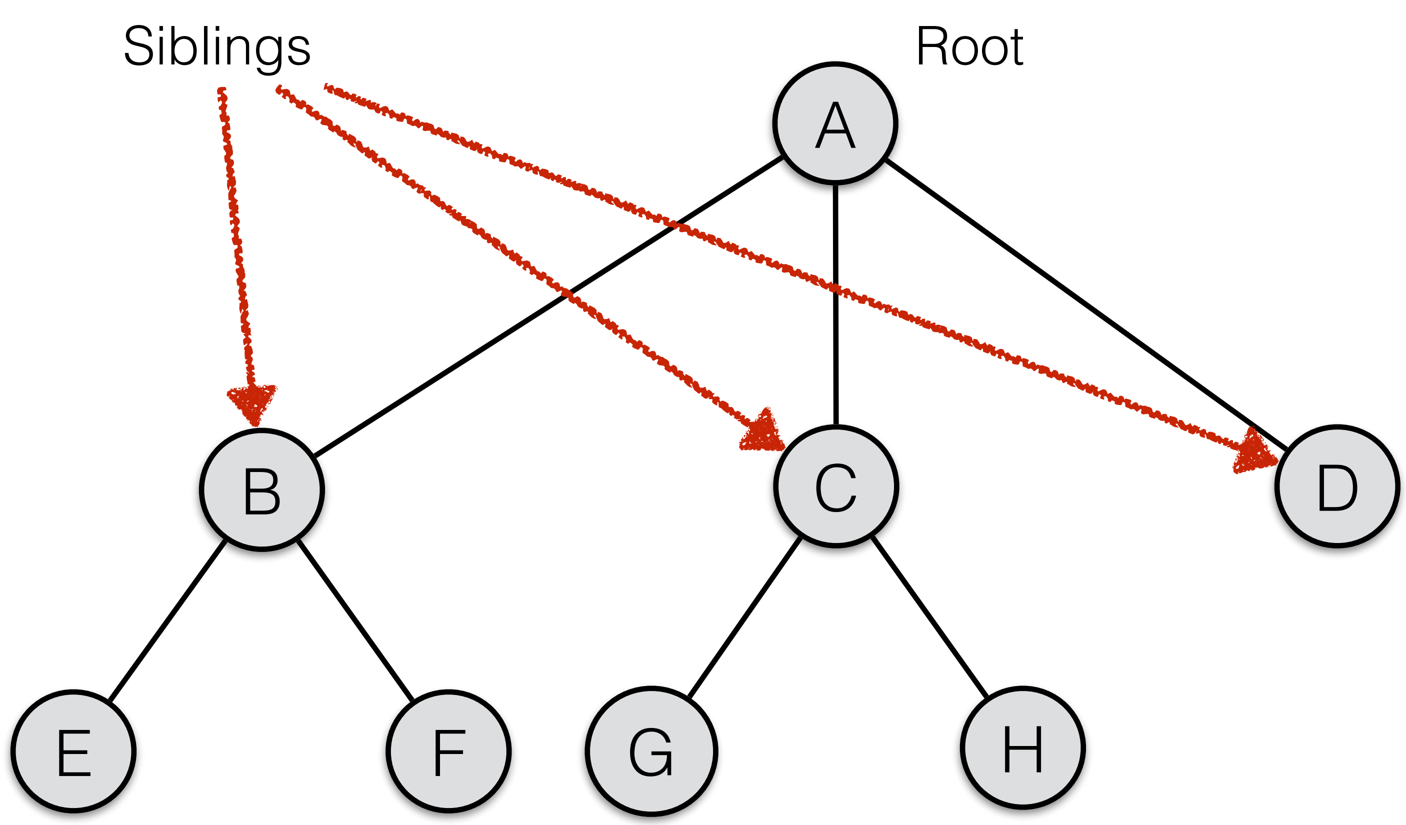
Trees

- Nodes with children are called **internal nodes**



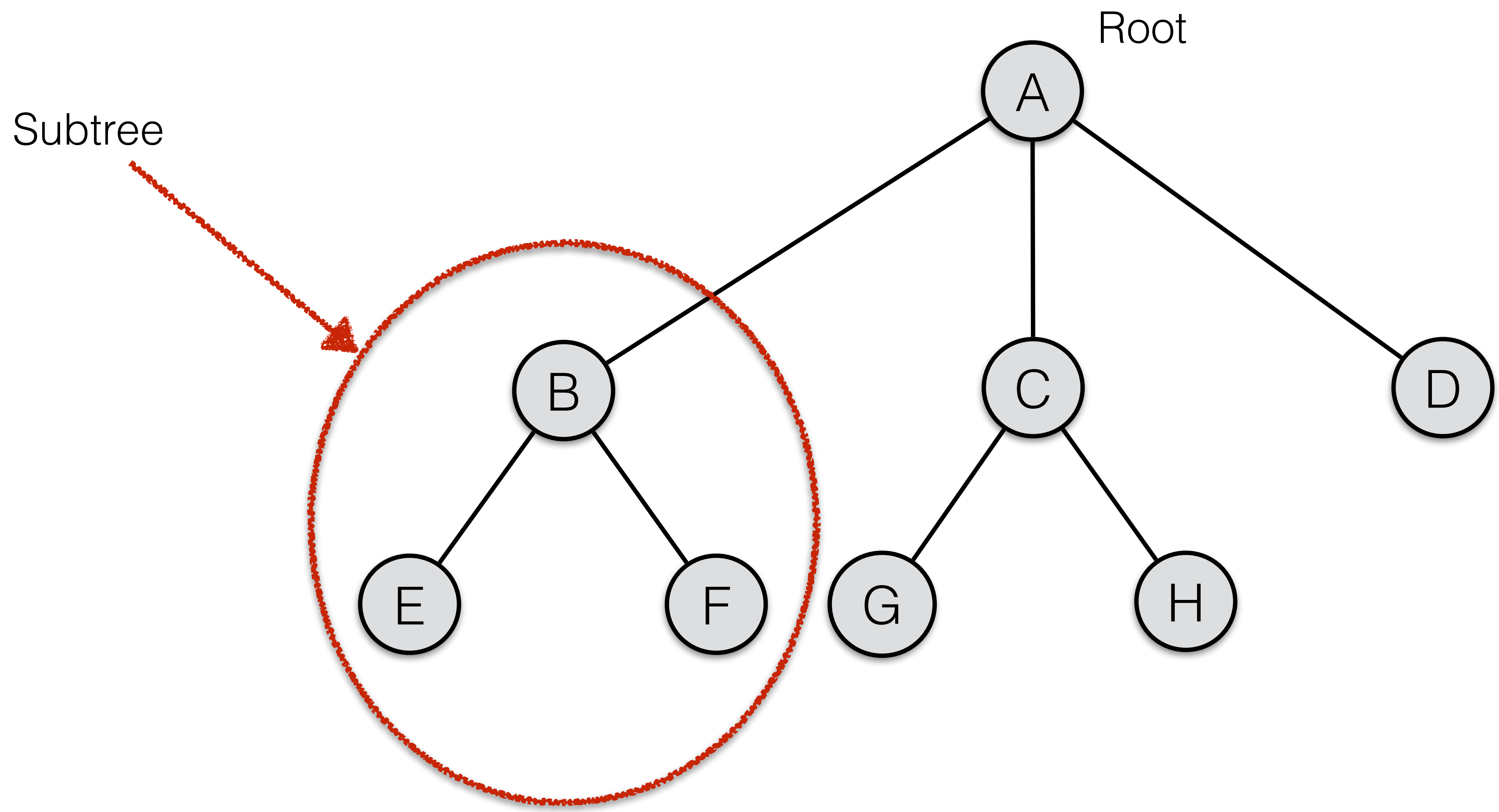
Trees

- A child has exactly one parent (no more/no less).
    - <u>Exception</u>: the *root* has no parent
- Nodes with the same parent are **siblings**



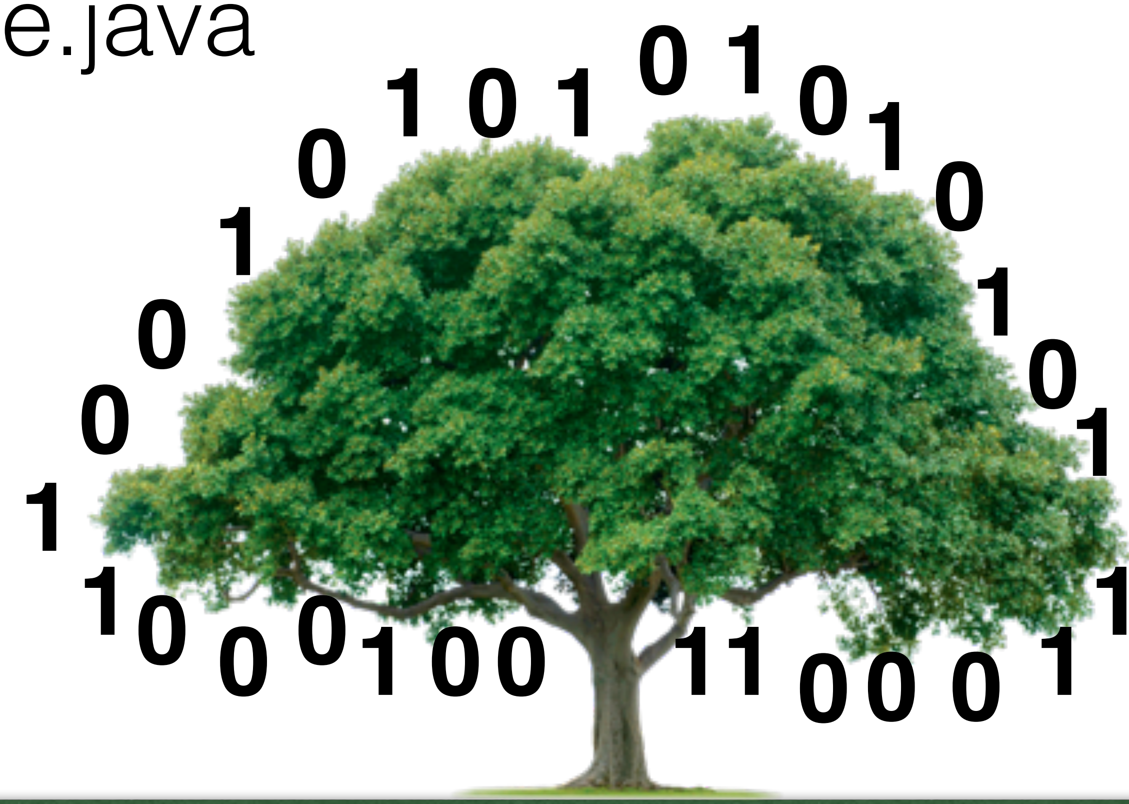
Trees

- As to be expected, a (family) tree consists of **Ancestors** and **descendants**
- A **subtree** of a node consists of all decedents of that node (including the node itself)
- NOTICE: It is just a tree!



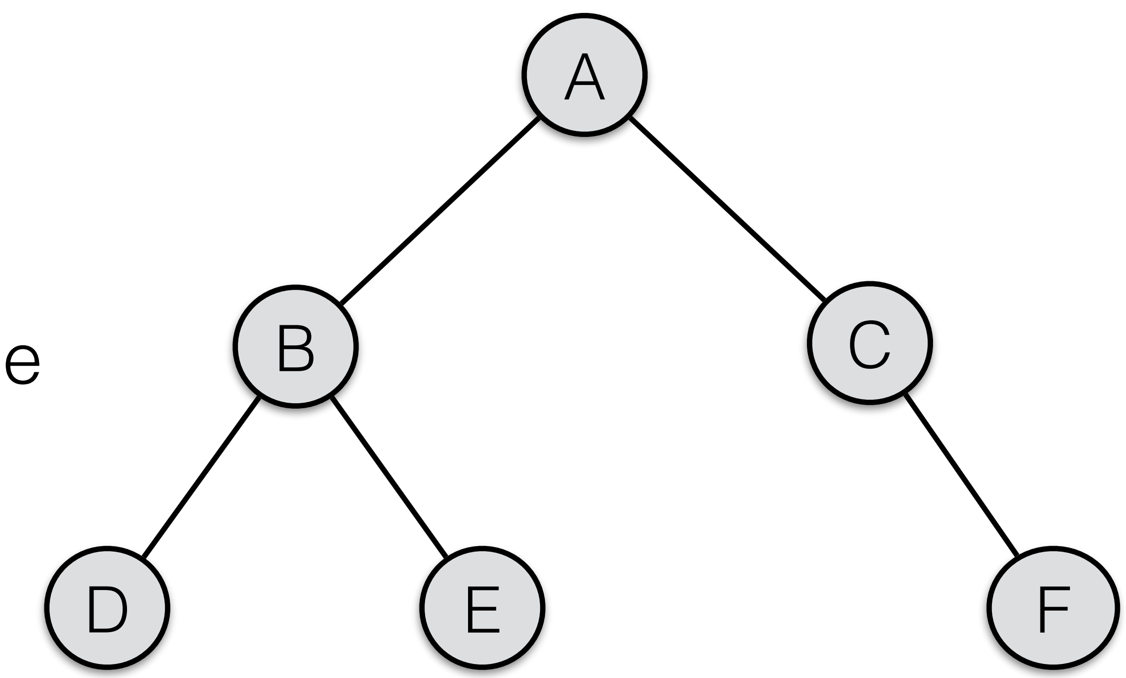
Binary Trees

Code: BinaryTree.java



Binary Trees

- A special kind of tree: each node has 0, 1, or 2 children.
- A tree node has a **left** and **right** “branch”
  - could be a child *OR* null!
- A tree node also has **data**



- Simply methods could be used to determine whether the tree has leaves, whether a node is an inner node or a leaf, etc.
- NOTE: Most tree code is recursive…

An Aside: Recursion Refresher

"an algorithm/approach to solve a problem by solving a smaller instance of the same problem, unless the problem is so small that we can just solve it directly"

Recursion: factorials

- **Problem**: How to compute factorial function?
  - Recall: factorial(n) = n! = 1*2*3*…*n
  - Special case: 0! = 1
- Example:
  - 5! = 1 * 2 * 3 * 4 * 5 = 120
- Ideas for solving this problem in general?

Recursion: factorials

re-write:

$$
\begin{aligned}
n! &= 1 * 2 * 3 * \ldots * (n-2) * (n-1) * n \\
&= n * (n-1) * (n-2) * \ldots * 3 * 2 * 1 \\
&= n * (n-1)!
\end{aligned}
$$

so

$$
\begin{aligned}
5! &= 5 * 4! && = 120 \\
4! &= 4 * 3! && = 24 \\
3! &= 3 * 2! && = 6 \\
2! &= 2 * 1! && = 2 \\
1! &= 1 * 0! && = 1 \\
0! &= 1 && = 1
\end{aligned}
$$

Recursion: factorials

- **Solution**: Multiply n * the factorial of (n-1); keep “recursing” until we hit the base case (i.e., 0! = 1).

```
public static int factorial(int n) {
    if(n == 0) {
        return 1;
    }
    return n * factorial(n-1);
}

public static void main(String[] args) {
    int result = factorial(0);
    System.out.println("factorial(0) = " + result);

    result = factorial(1);
    System.out.println("factorial(1) = " + result);

    result = factorial(5);
    System.out.println("factorial(5) = " + result);

    result = factorial(10);
    System.out.println("factorial(10) = " + result);
}
```

Recursion: factorials

- **Bonus Problem**: Just for fun, how could you solve the factorial problem *iteratively*?!

```
public static int iterativeFactorial(int n) {
    int result = 1;
    for(int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

public static void main(String[] args) {
   System.out.println("iterativeFactorial(0) = " + iterativeFactorial(0));

   System.out.println("iterativeFactorial(1) = " + iterativeFactorial(1));

   System.out.println("iterativeFactorial(5) = " + iterativeFactorial(5));

   System.out.println("iterativeFactorial(10) = " + iterativeFactorial(10));
}
```

Recursion: "Nuggets of Truth"

1. Each recursive call should be on a smaller instance of the same problem, that is, a smaller subproblem.
2. The recursive calls must eventually reach a base case, which is solved without further recursion.

Back to Binary Trees

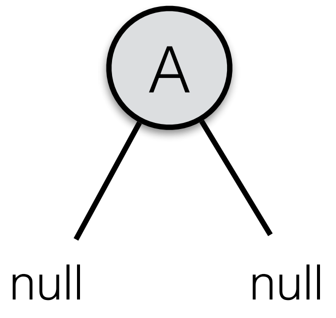
Binary Trees

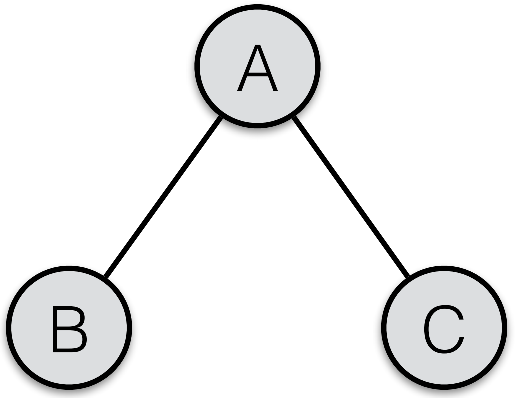
```java
/**
 * Generic binary tree, storing data of a parametric data in each node
 */
public class BinaryTree<E> {
    private BinaryTree<E> left, right;  // children; can be null
    E data;

    /**
     * Constructs leaf node -- left and right are null
     */
    public BinaryTree(E data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }

    /**
     * Constructs inner node
     */
    public BinaryTree(E data, BinaryTree<E> left, BinaryTree<E> right) {
        this.data = data;
        this.left = left;
        this.right = right;
    }

    // . . .
}
```

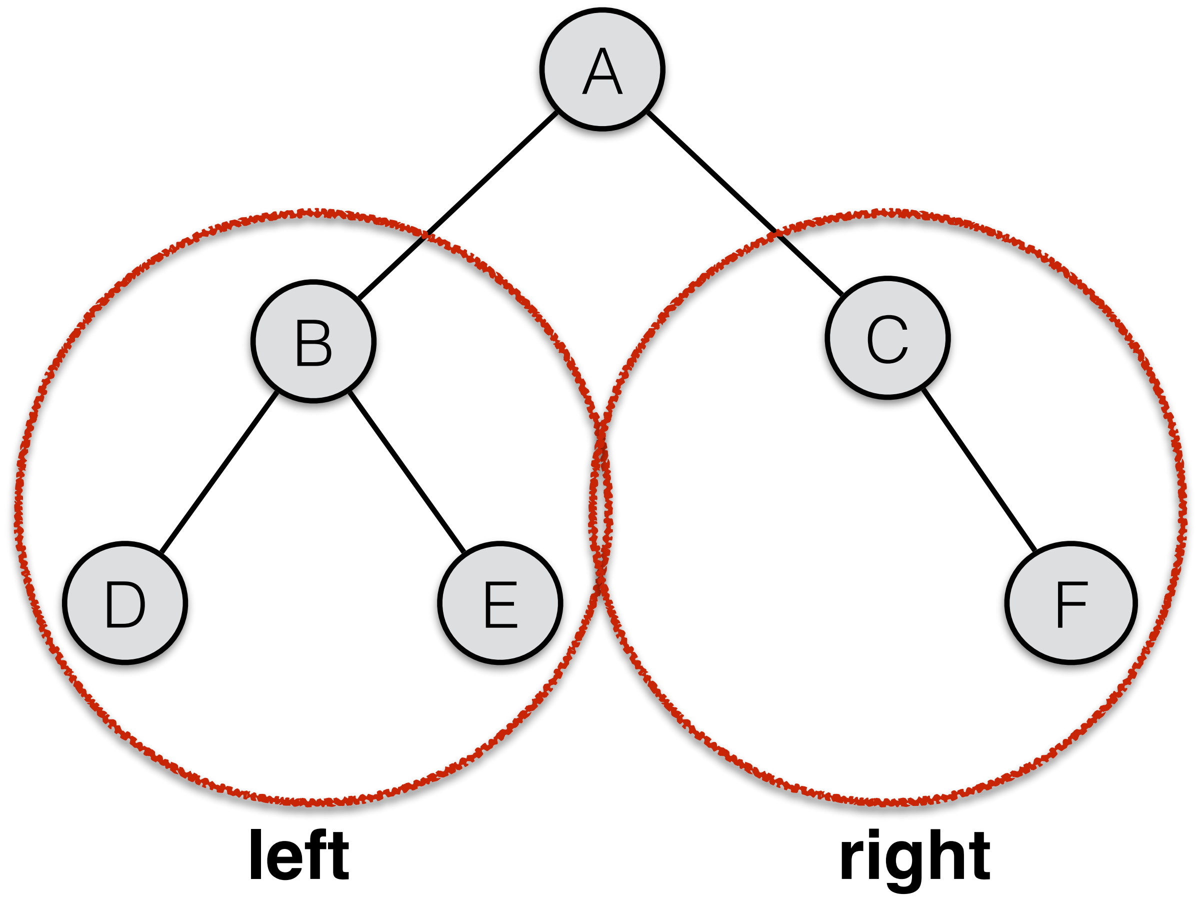




Binary Trees: size()

- **Problem**: How to compute the size of a binary tree (i.e., # of nodes in it)?
- **Hint**: Most tree code is recursive!
- **Solution**: The size of a tree (# nodes in it) is the size of its **left** tree plus that of its **right** tree plus 1 (the parent)

```java
/**
 * Number of nodes (inner and leaf) in tree
 */
public int size() {
    int num = 1;
    if (hasLeft()) num += left.size();
    if (hasRight()) num += right.size();
    return num;
}
```



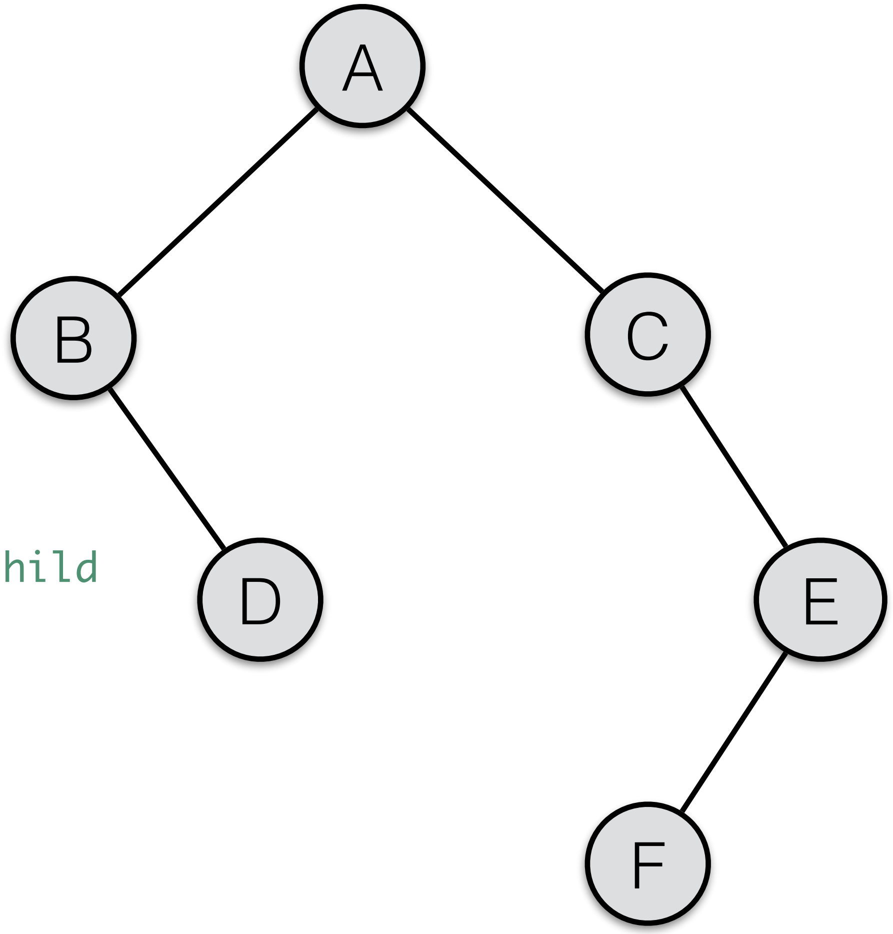
Binary Trees: height()

- **Problem**: How to compute the height of a binary tree (i.e., longest path from root to a leaf)?

- **Hint**: Again, think recursion!

- **Solution**: The height of a tree (longest path from root to a leaf) is the max of the height of the **left** subtree and the **right** subtree (+ 1).

```java
/**
 * Longest length to a leaf node from here
 */
public int height() {
    if (isLeaf()) return 0;
    int h = 0;
    if (hasLeft()) h = Math.max(h, left.height());
    if (hasRight()) h = Math.max(h, right.height());
    return h+1;                     // inner: one higher than highest child
}
```

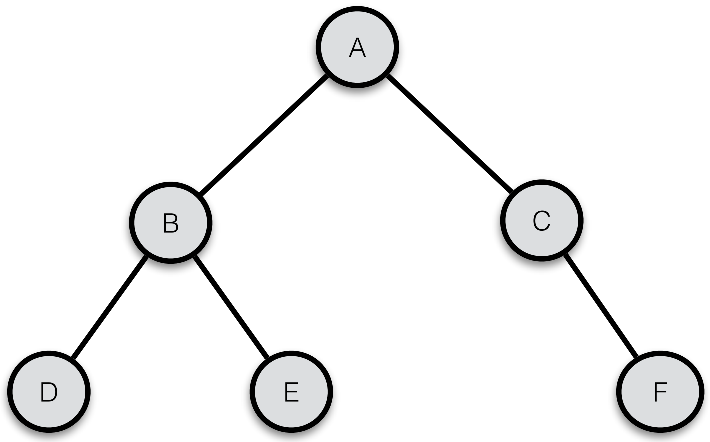


Binary Trees: fringe()

- **Problem**: How to compute the fringe of a binary tree (i.e., a left-to-right list of the leaves of the tree)?

- **Hint**: Yet again, think recursion!

- **Solution**: Recurse left *then* right. Keep recursing until you find a leaf. Since we go left *then* right, add nodes to our fringe list as we encounter leaves — this collects the leaves in order (left-to-right)!



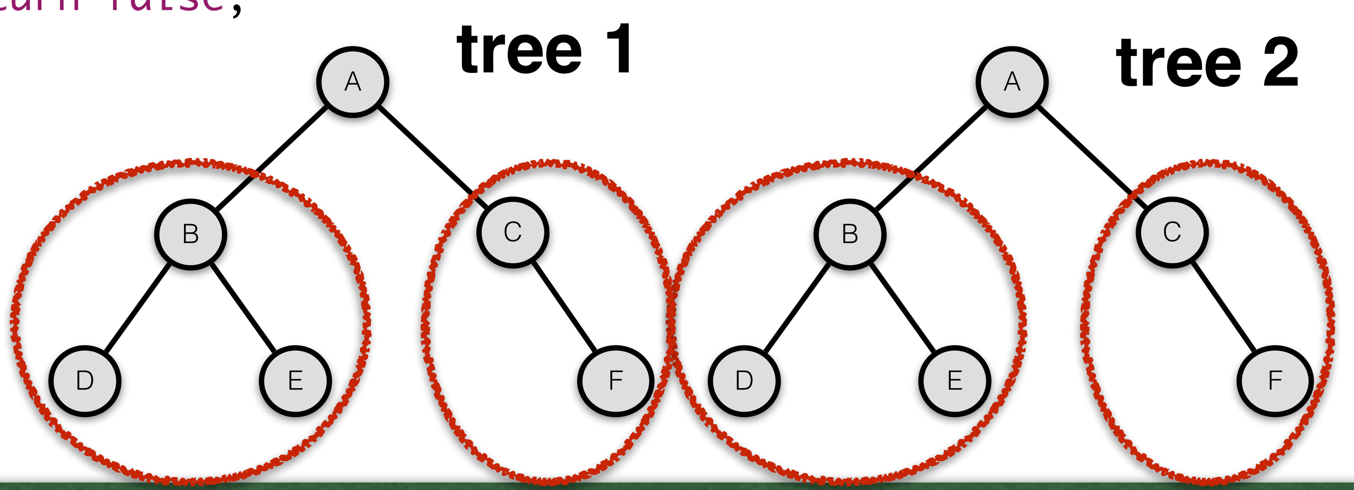

```java
/**
 * Leaves, in order from left to right
 */
public ArrayList<E> fringe() {
    ArrayList<E> f = new ArrayList<E>();
    addToFringe(f);
    return f;
}

/**
 * Helper for fringe, adding fringe data to the list
 */
public void addToFringe(ArrayList<E> fringe) {
    if (isLeaf()) {
        fringe.add(data);
    }
    else {
        if (hasLeft()) left.addToFringe(fringe);
        if (hasRight()) right.addToFringe(fringe);
    }
}
```

Binary Trees: equalsTree()

- **Problem**: How to determine if two trees are equal?
- **Hint**: psst…. were you thinking recursion?! ;)
- **Solution**: First, check some attributes about the tree
  - If the current tree "has a" left while the other tree doesn't, they can't be equal…
  - Same for the right…
  - If the data at a particular node isn't equal, then the trees aren't equal.
  - recursively check the left and right like this!

```java
/**
 * Same structure and data?
 */
public boolean equalsTree(BinaryTree<E> t2) {
    if (hasLeft() != t2.hasLeft() || hasRight() != t2.hasRight()) return false;
    if (!data.equals(t2.data)) return false;
    if (hasLeft() && !left.equalsTree(t2.left)) return false;
    if (hasRight() && !right.equalsTree(t2.right)) return false;
    return true;
}
```



Binary Trees: toString()

- **Problem**: How to “print out” a tree structure?

- **Solution**: Starting at the root, traverse the tree (parent, then left, then right), appending the “data” to a String (starts as “”) — notice that the traversal happens recursively!

```java
/**
 * Returns a string representation of the tree
 */
public String toString() {
    return toStringHelper("");
}

/**
 * Recursively constructs a String representation of the tree from this node,
 * starting with the given indentation and indenting further going down the tree
 */
public String toStringHelper(String indent) {
    String res = indent + data + "\n";
    if (hasLeft()) res += left.toStringHelper(indent+"  ");
    if (hasRight()) res += right.toStringHelper(indent+"  ");
    return res;
}
```

Binary Trees: parseNewick()

- **Problem**: How to build a tree structure from a String?

- **Solution**: We provide a simple (not too robust) version of a Newick format parser.
  - The format consists of
    - commas between children
    - parenthesis enclosing a node's left/right
  - For example, "((a,b)c,(d,e)f)g;"
    - root "g"
    - "g" that has two children "c" and "f",
    - "c" has leaf children "a" and "b", and likewise
    - "f" has leaf children"d" and "e".

Binary Trees: parseNewick()

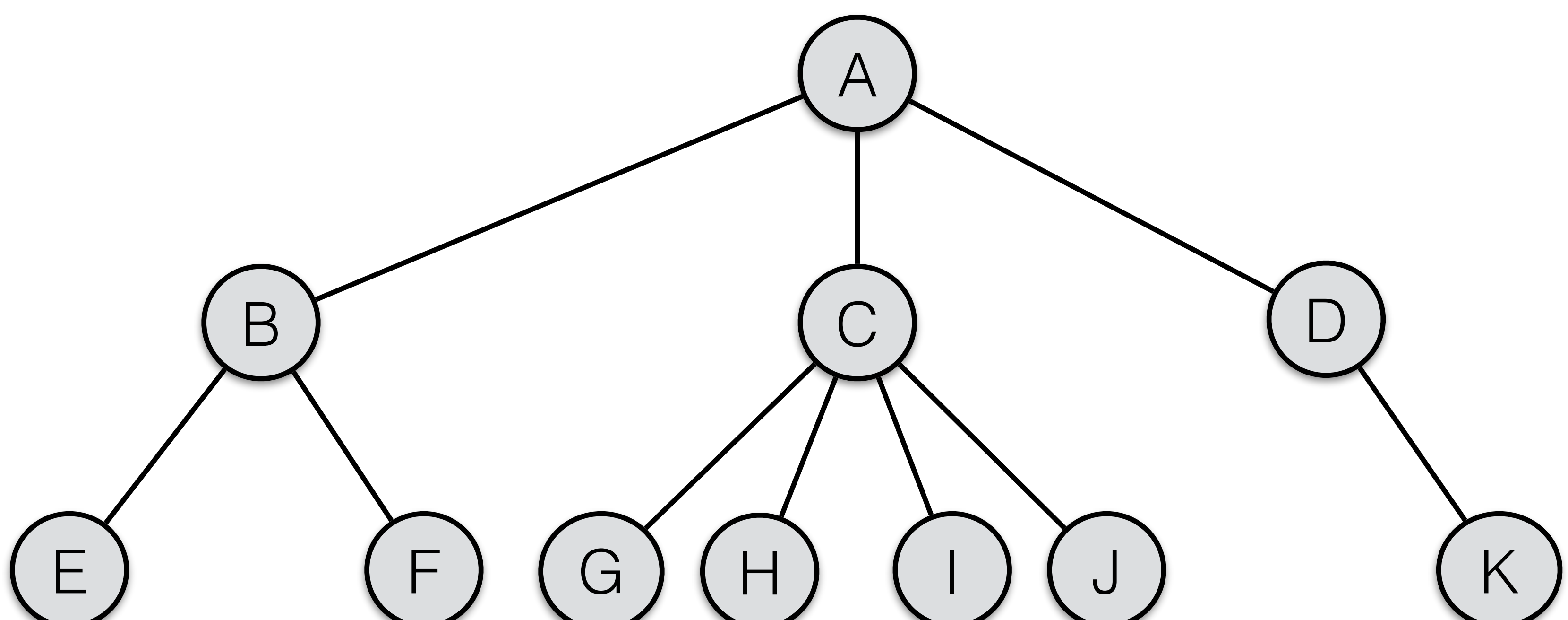
```java
/**
 * Very simplistic binary tree parser based on Newick representation
 * Assumes that each node is given a label; that becomes the data
 * Any distance information (following the colon) is stripped
 * <tree> = "(" <tree> "," <tree> ")" <label> [":"<dist>]
 *        | <label> [":"<dist>]
 * No effort at all to handle malformed trees or those not following these strict requirements
 */
public static BinaryTree<String> parseNewick(String s) {
    BinaryTree<String> t = parseNewick(new StringTokenizer(s, "(,)", true));
    // Get rid of the semicolon
    t.data = t.data.substring(0,t.data.length()-1);
    return t;
}

/**
 * Does the real work of parsing, now given a tokenizer for the string
 */
public static BinaryTree<String> parseNewick(StringTokenizer st) {
    String token = st.nextToken();
    if (token.equals("(")) {
        // Inner node
        BinaryTree<String> left = parseNewick(st);
        String comma = st.nextToken();
        BinaryTree<String> right = parseNewick(st);
        String close = st.nextToken();
        String label = st.nextToken();
        String[] pieces = label.split(":");
        return new BinaryTree<String>(pieces[0], left, right);
    }
    else {
        // Leaf
        String[] pieces = token.split(":");
        return new BinaryTree<String>(pieces[0]);
    }
}
```

General Trees

General Trees

- Can have **any #** of children.
- Sometimes order **doesn't** matter
  - Ex. files in a folder
- Sometimes order **does** matter!
  - Ex. order of elements in an HTML document
- When we represent trees, we end up imposing an order on children (whether it is important or not…)



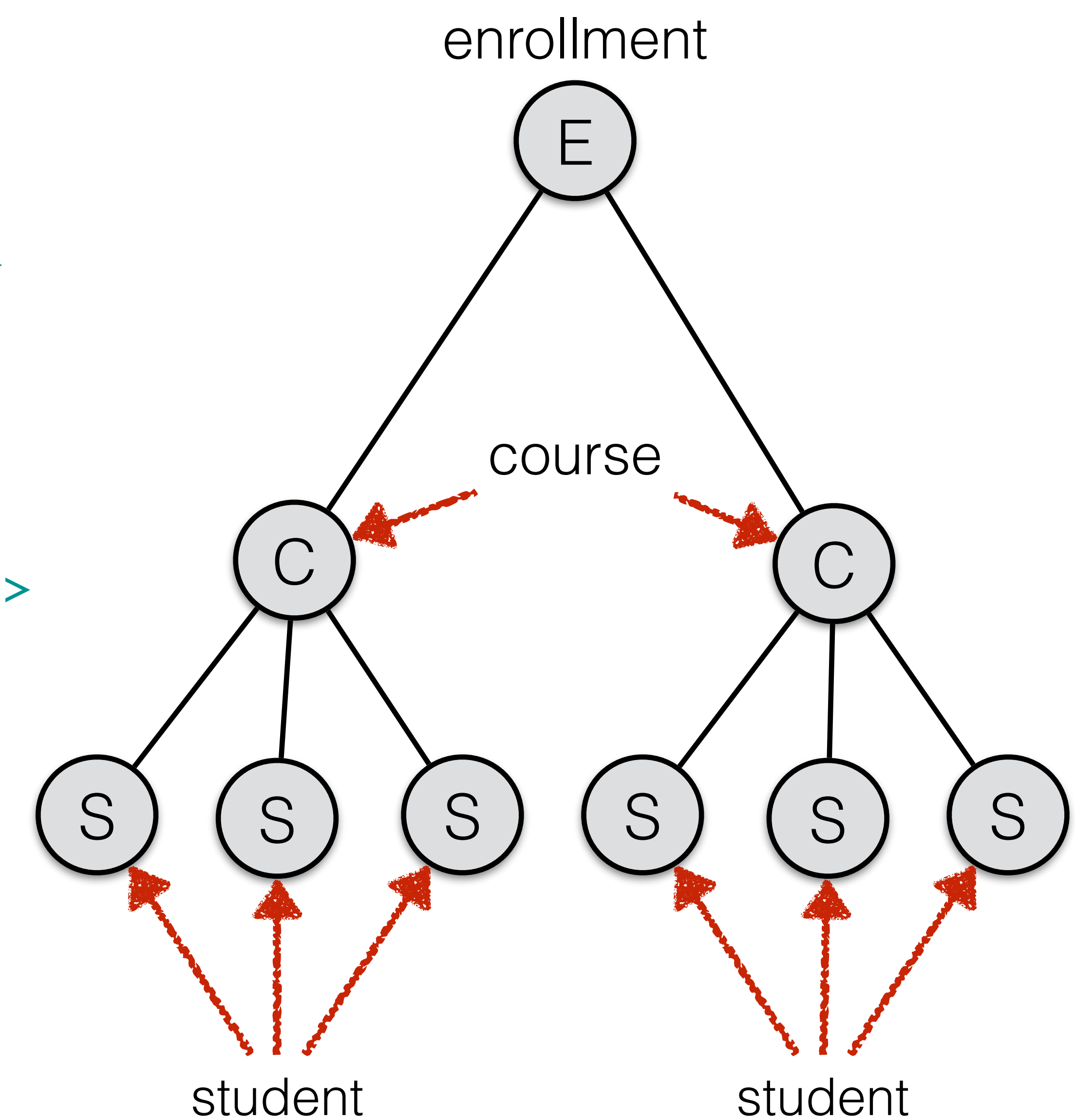
General Trees: DOM

- An example of a general tree is the **Document Object Model** (DOM).
- DOM is a convention for interacting w/ objects in **HTML** / **XML** documents.
- **Nodes** in the DOM are organized in a **tree structure**.

General Trees: DOM

- An **XML** example: test.xml
  - Each node is associated with an open/close tag pair or w/ text
    - Ex. <course> … </course>

```
<enrollment>
  <course department="CS" number="1" term="12F">
    <student name="Alice" year="13" />
    <student name="Bob" year="15" />
    <student name="Charlie" year="13" />
  </course>
  <course department="CS" number="10" term="12F">
    <student name="Delilah" year="12" />
    <student name="Elvis" year="14" />
    <student name="Flora" year="14" />
  </course>
</enrollment>
```



General Trees: DOM

- An **HTML** example: test.html

```
<html>
  <body>
    <h1>First and only chapter</h1>
    <p>He was a very <b>bold</b> man.</p>
    <p>He followed a <a href="http://www.dartmouth.edu/">hyperlink</a>
       off into the wild.</p>
    <img src="smiley.png" alt="smile!" />
    <p>The end.</p>
  </body>
</html>
```

I'll leave it as an exercise for you to sketch the tree specified by this HTML document — it is similar to the XML test file.

General Trees: DOM

- **ExampleXML.java** implements a DOM parser which can do things like count the # of students in a course (XML) or extract links/images (HTML).
  - Similar idea as how you parsed a simple message to update a sketch!
  - [In class demo]
- Uses Java's built-in "**Node**" class
  - .getNodeType()
    - DOCUMENT_NODE
    - ELEMENT_NODE
    - TEXT_NODE
  - .getName()
    - an element node has a name

General Trees: DOM

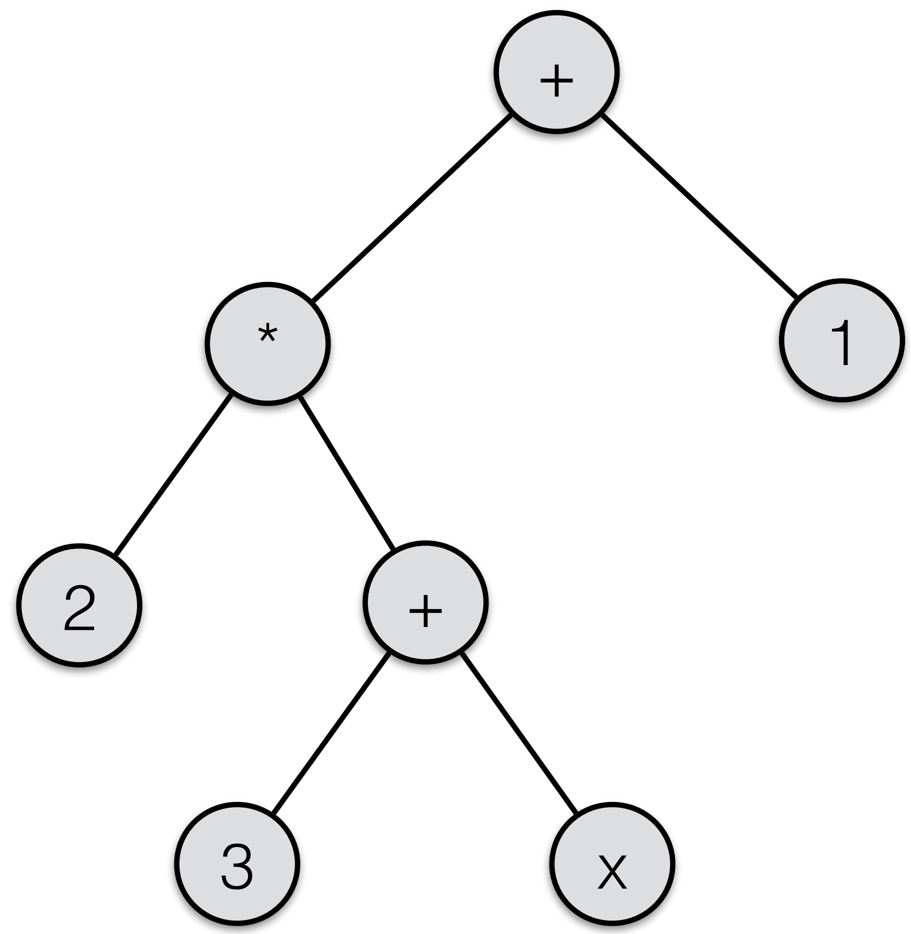
- .getAttributes() returns a special type of list of attributes for an element node
  - can get a particular item — .item(i)
  - can get length of attr. list — .getLength()
  - .getNodeName() and .getNodeValue()
- Attributes have name/value pairs (.getNodeName(), .getNodeValue())

- TEXT NODES have a value — .getNodeValue()
- Nodes can have children
  - (*start*) .getFirstChild() —> (*traverse*) .getNextSibling() …
- .printTree() is similar to the binary tree printTree() method, but has to handle different node types and recursing for *each* child in its child list.

Expression Trees

Expression Trees

1+2*(3+x)



Expression Trees

- Expression.java (interface)
  - eval()
  - deriv()
- ExpressionDriver.java
  - demonstrates building and using Expression objects
- Constant.java
- Variable.java
- Sum.java, Difference.java, Product.java, Quotient.java
  - perform *eval()* by evaluating their operands and performing an operation on them
  - Only difference between these classes is the *operator*
  - Thus, we create BinaryOp.java
- UnassignedVariableException.java — customized exception