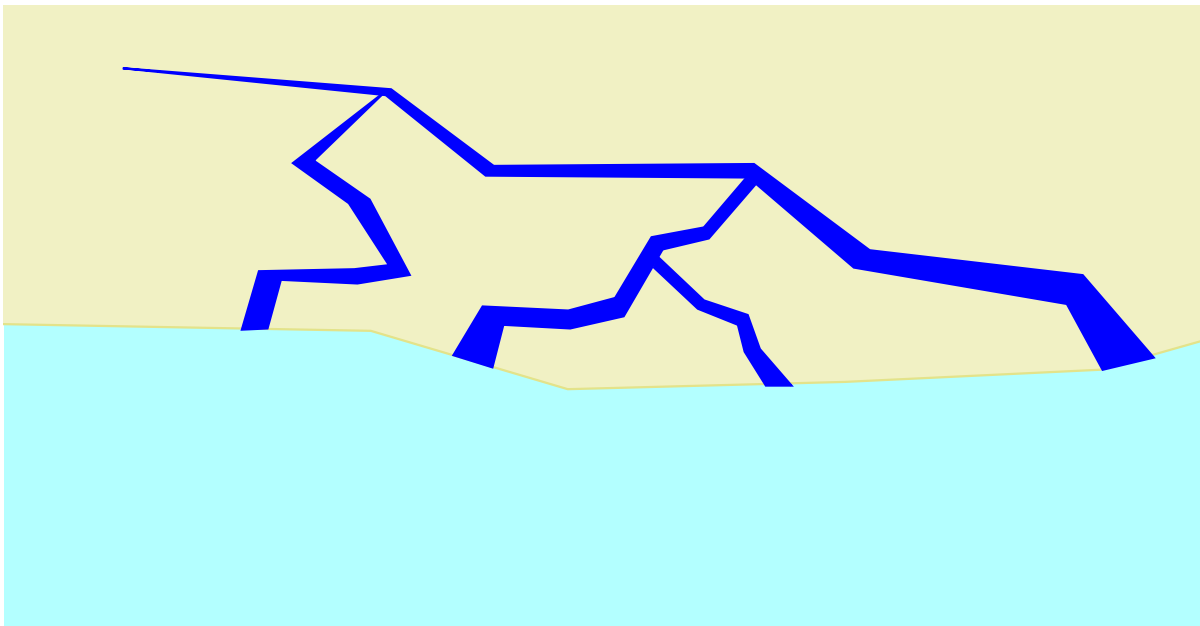


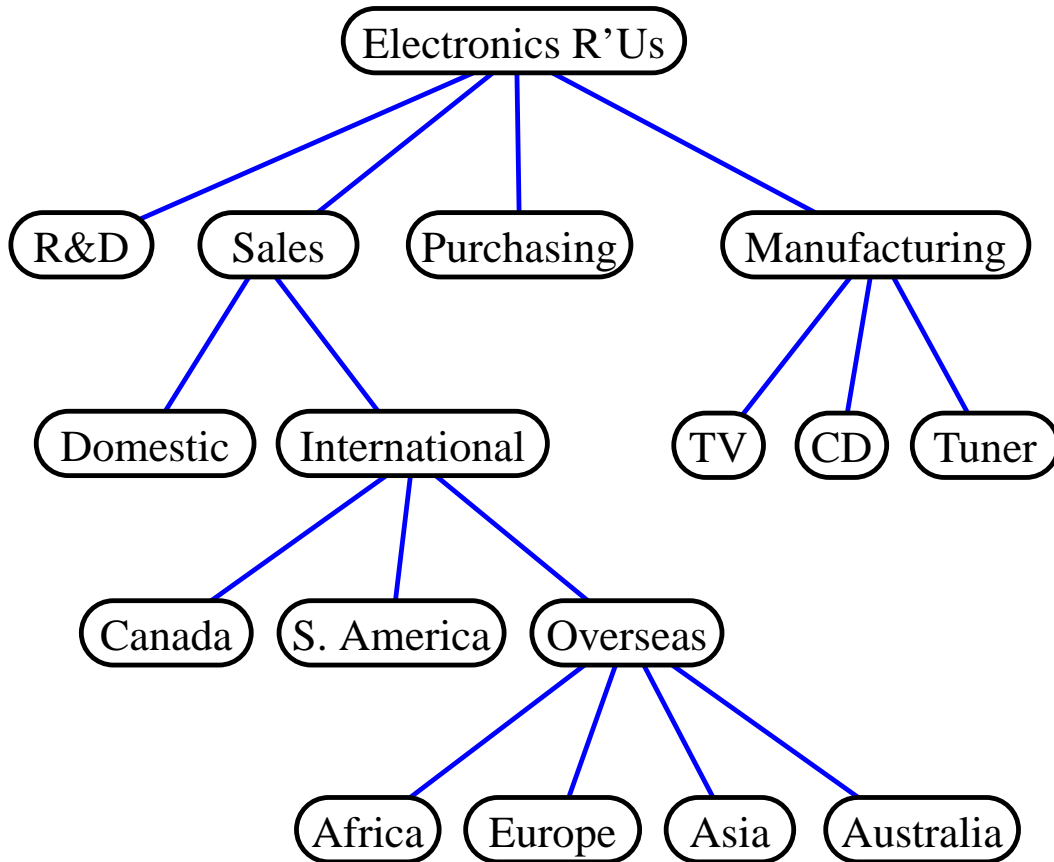
TREES

- trees
- binary trees
- traversals of trees
- template method pattern
- data structures for trees

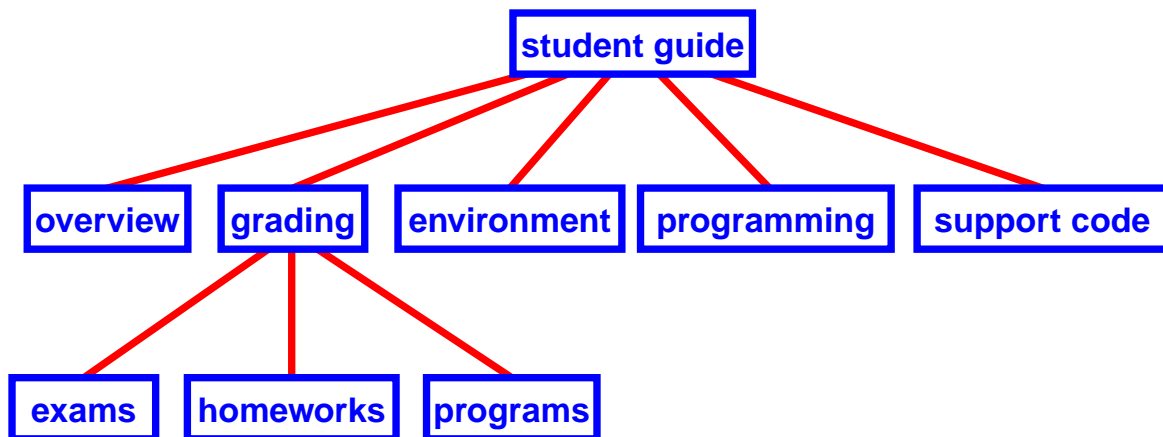


Trees

- a **tree** represents a hierarchy
 - organization structure of a corporation

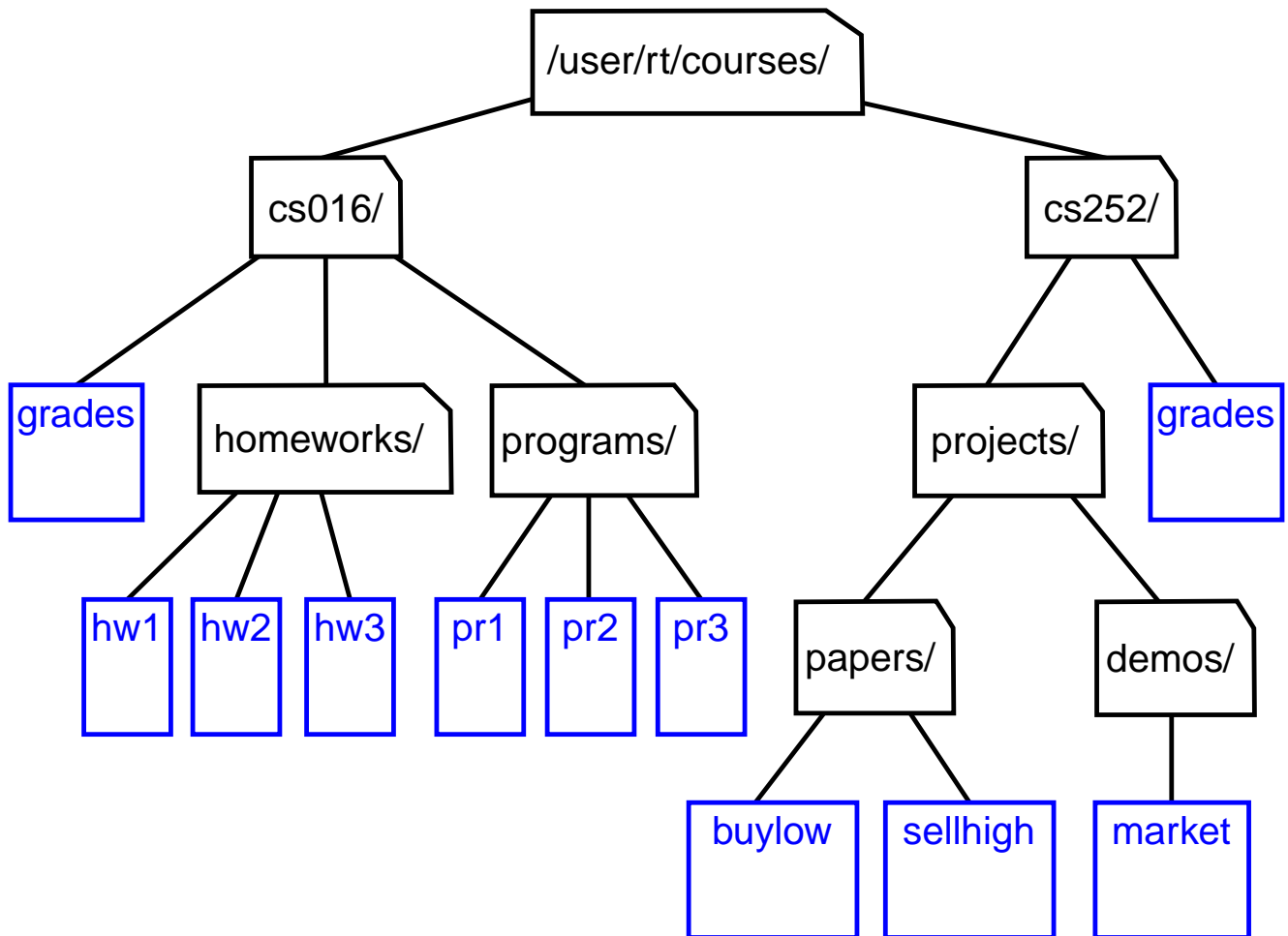


- table of contents of a book



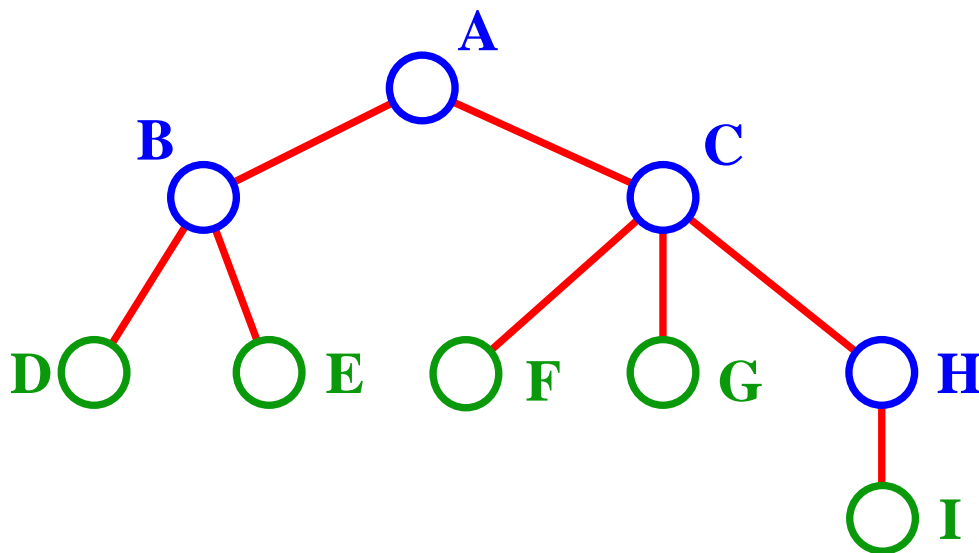
Another Example

- Unix or DOS/Windows file system



Terminology

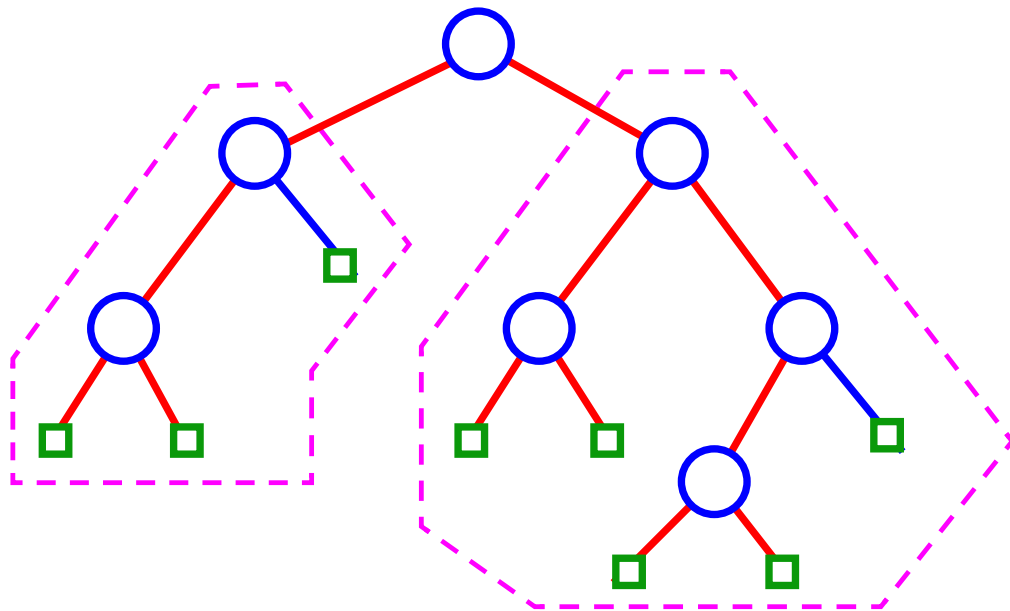
- **A** is the *root* node.
- **B** is the *parent* of D and E.
- **C** is the *sibling* of B
- **D** and **E** are the *children* of B.
- **D, E, F, G, I** are *external nodes*, or *leaves*.
- **A, B, C, H** are *internal nodes*.
- The *depth (level)* of **E** is **2**
- The *height* of the tree is **3**.
- The *degree* of node **B** is **2**.



Property: (# edges) = (#nodes) - 1

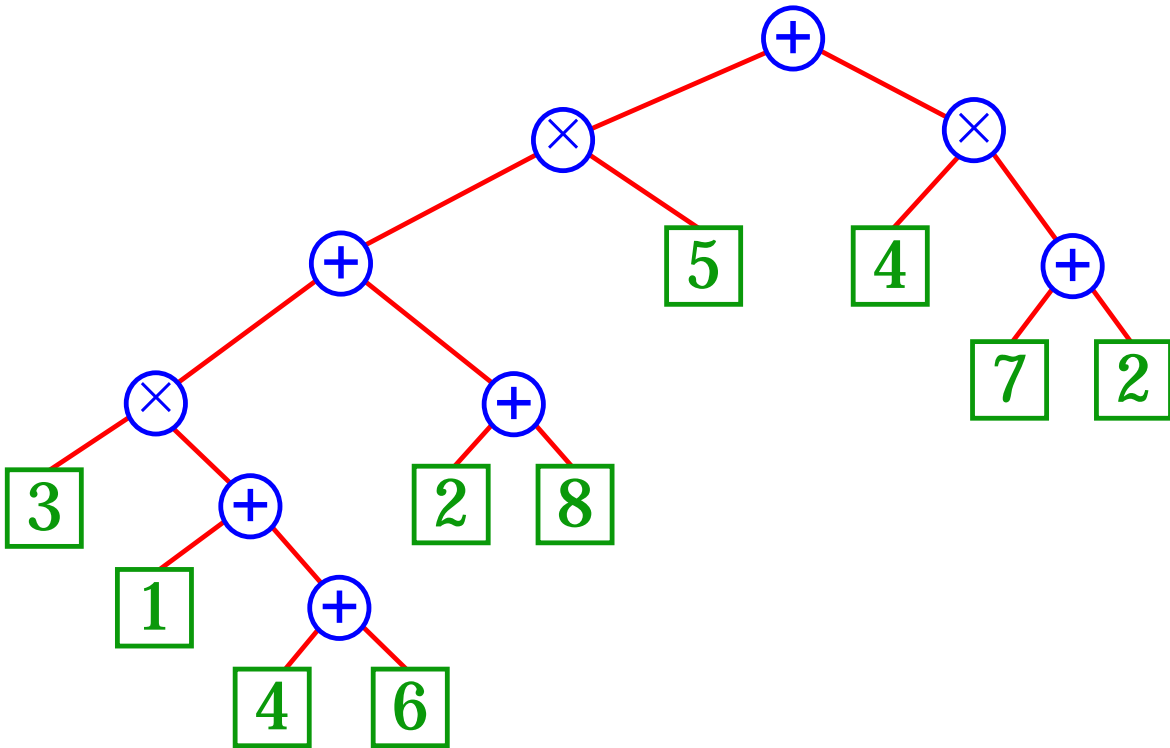
Binary Trees

- **Ordered tree:** the children of each node are ordered.
- **Binary tree:** ordered tree with all internal nodes of **degree 2**.
- Recursive definition of binary tree:
- A **binary tree** is either
 - an **external node (leaf)**, or
 - an **internal node (the root)** and two binary trees (**left subtree** and **right subtree**)



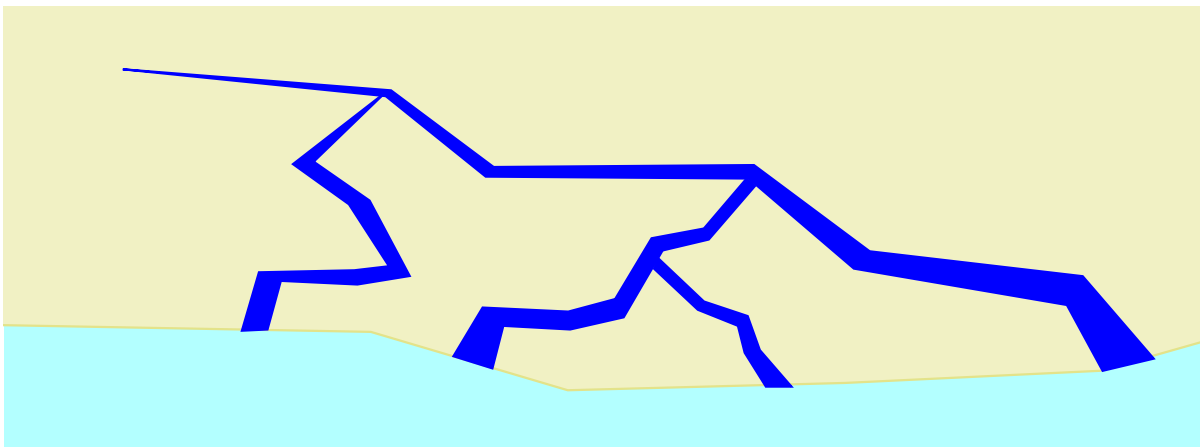
Examples of Binary Trees

- arithmetic expression



$$(((3 \times (1 + (4 + 6))) + (2 + 8)) \times 5) + (4 \times (7 + 2)))$$

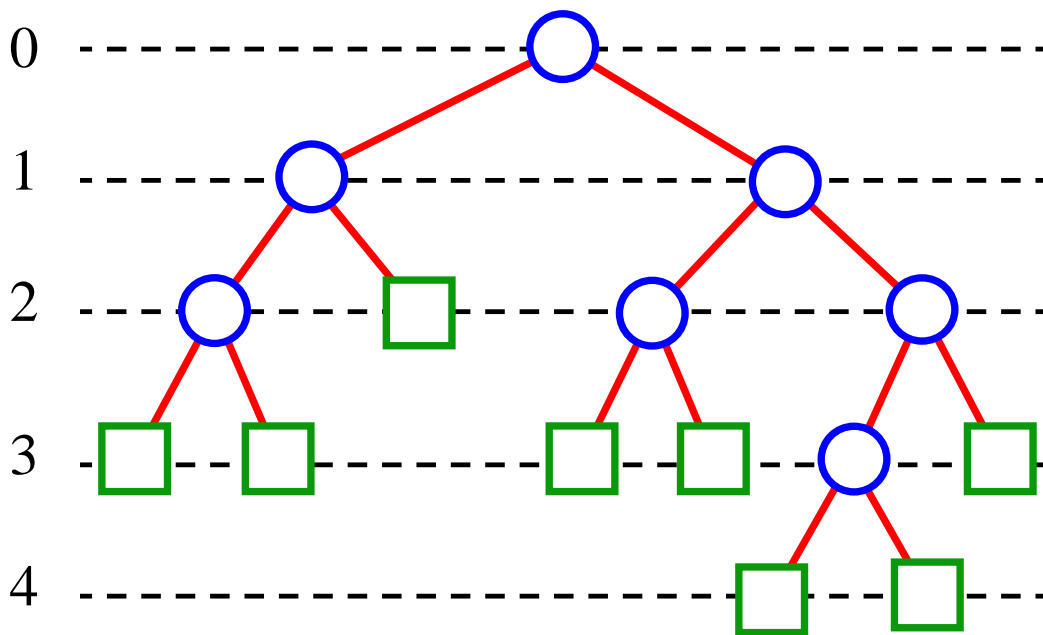
- river



Properties of Binary Trees

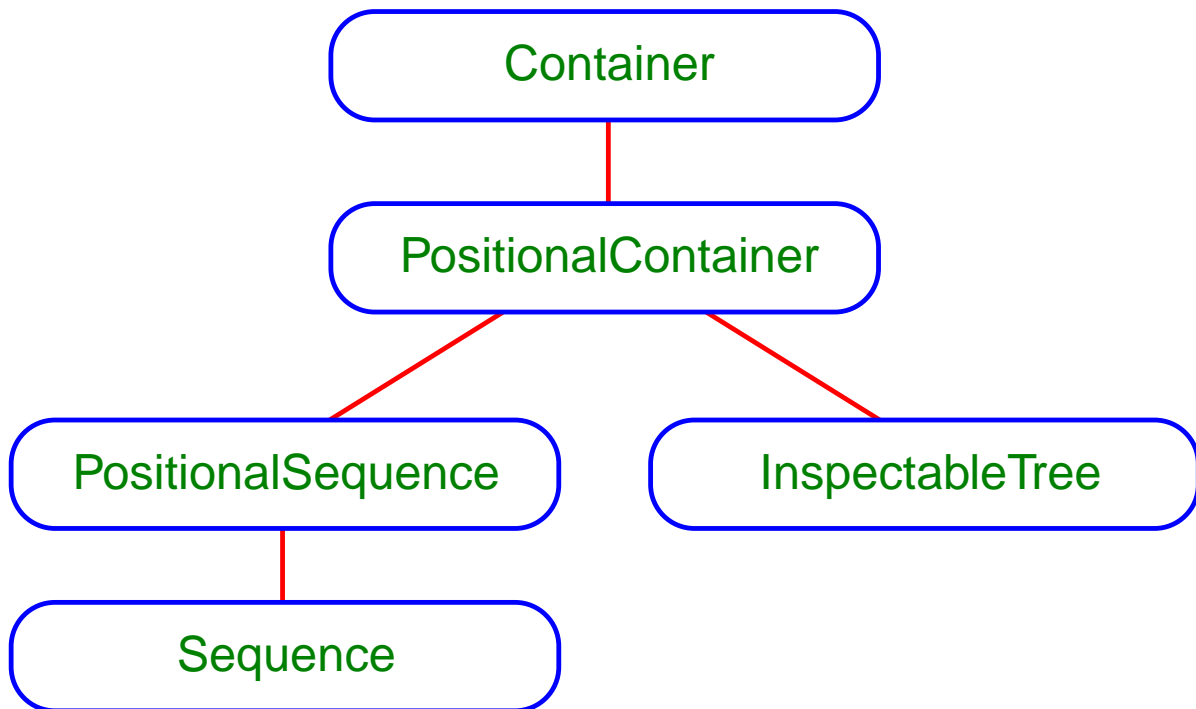
- (# external nodes) = (# internal nodes) + 1
- (# nodes at level i) $\leq 2^i$
- (# external nodes) $\leq 2^{(\text{height})}$
- (height) $\geq \log_2$ (# external nodes)
- (height) $\geq \log_2$ (# nodes) - 1
- (height) \leq (# internal nodes) = ((# nodes) - 1)/2

Level



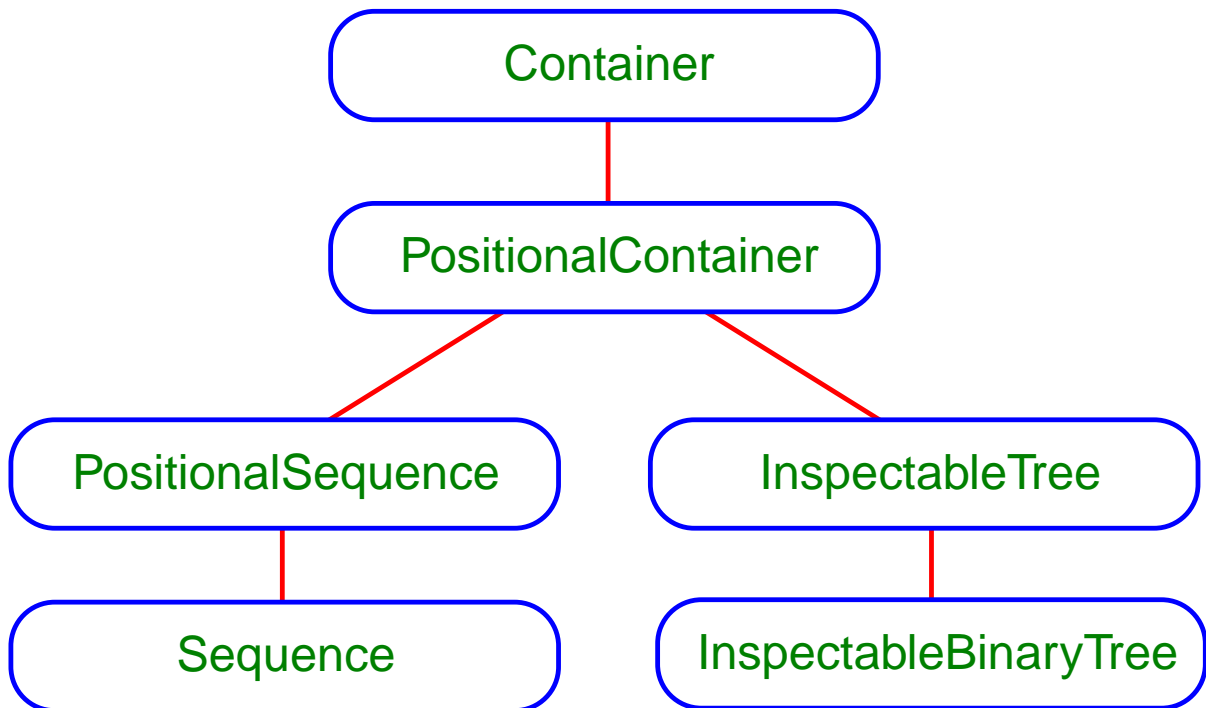
The Tree ADT

- the nodes of a tree are viewed as **positions**
- generic container methods
 - `size()`, `isEmpty()`, `elements()`, `newContainer()`
- positional container methods
 - `positions()`, `replace(p,e)`, `swap(p,q)`
- query methods
 - `isRoot(p)`, `isInternal(p)`, `isExternal(p)`
- accessor methods
 - `root()`, `parent(p)`, `children(p)`, `siblings(p)`
- update methods (application specific)



The Binary Tree ADT

- extends the tree ADT
- accessor methods
 - leftChild(p), rightChild(p), sibling(p)
- update methods
 - expandExternal(p), removeAboveExternal(p)
 - other application specific methods
- interface hierarchy of positional containers



Traversing Trees

- preorder traversal

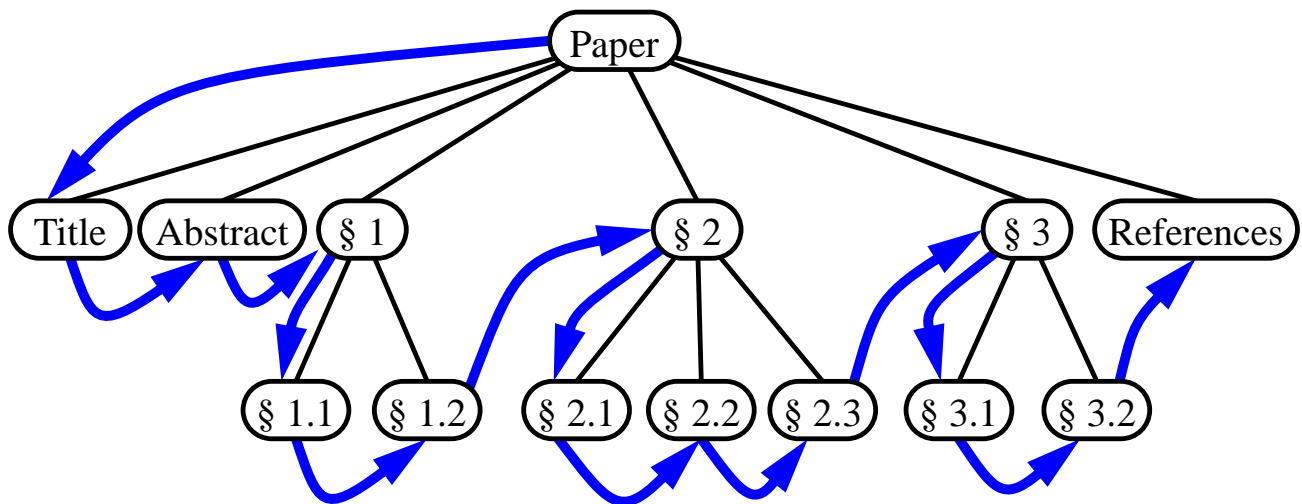
Algorithm preOrder(v)

“visit” node v

for each child w of v do

recursively perform preOrder(w)

- reading a document from beginning to end

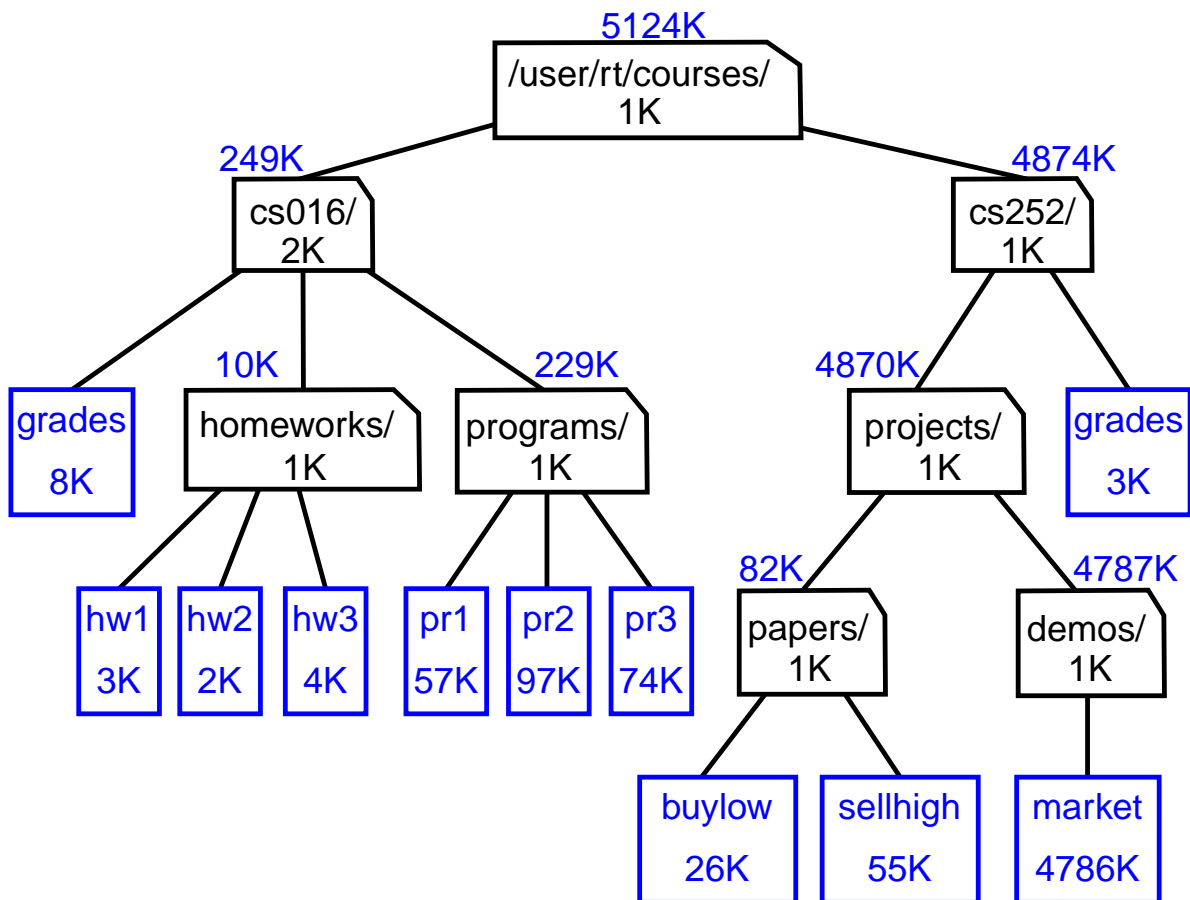


Traversing Trees

- `postorder` traversal

Algorithm `postOrder(v)`
for each child `w` of `v` do
 recursively perform `postOrder(w)`
“visit” node `v`

- `du` (disk usage) command in Unix



Evaluating Arithmetic Expressions

- specialization of a postorder traversal

Algorithm evaluateExpression(v)

if v is an external node

return the variable stored at v

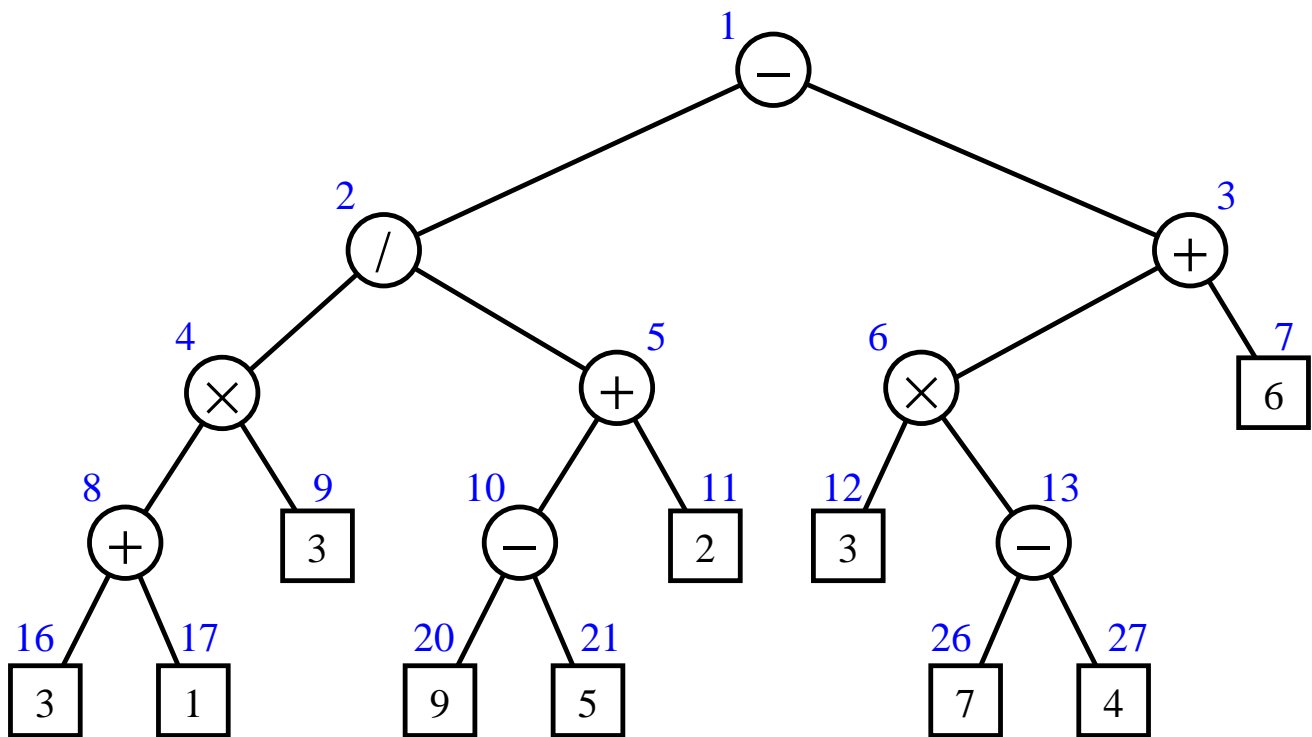
else

let o be the operator stored at v

x ← evaluateExpression(leftChild(v))

y ← evaluateExpression(rightChild(v))

return x o y



Traversing Trees

- **inorder** traversal of a **binary tree**

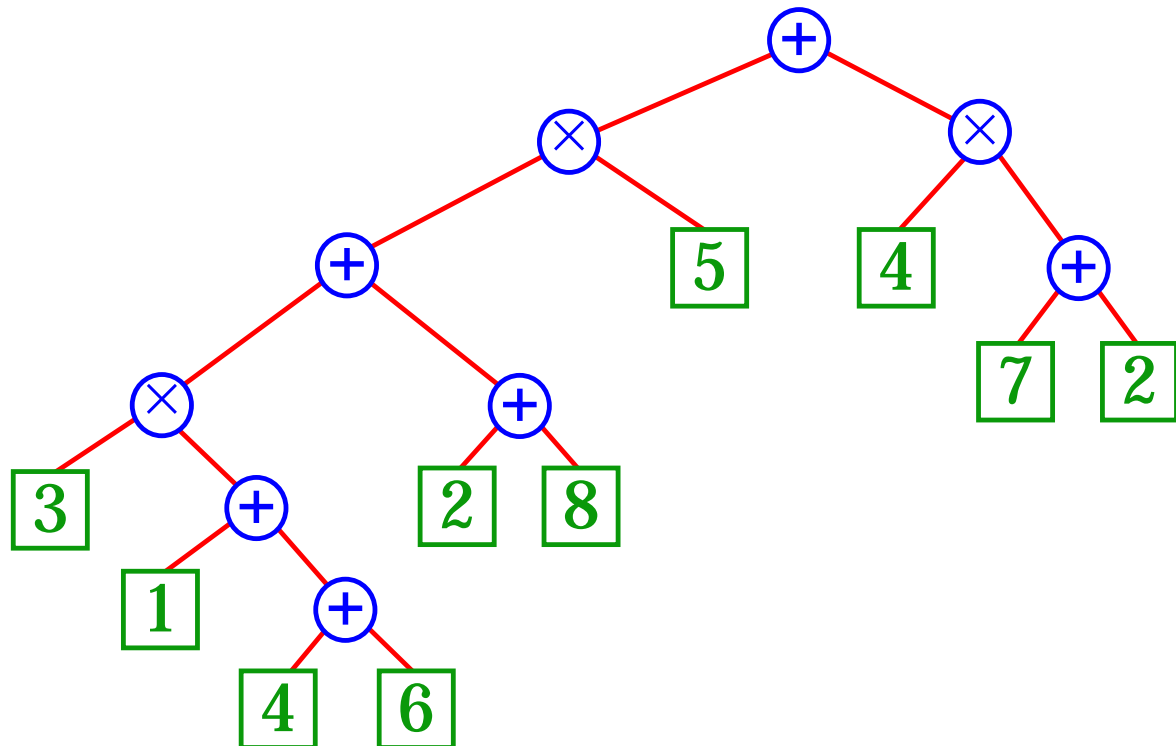
Algorithm `inOrder(v)`

recursively perform `inOrder(leftChild(v))`

“visit” node `v`

recursively perform `inOrder(rightChild(v))`

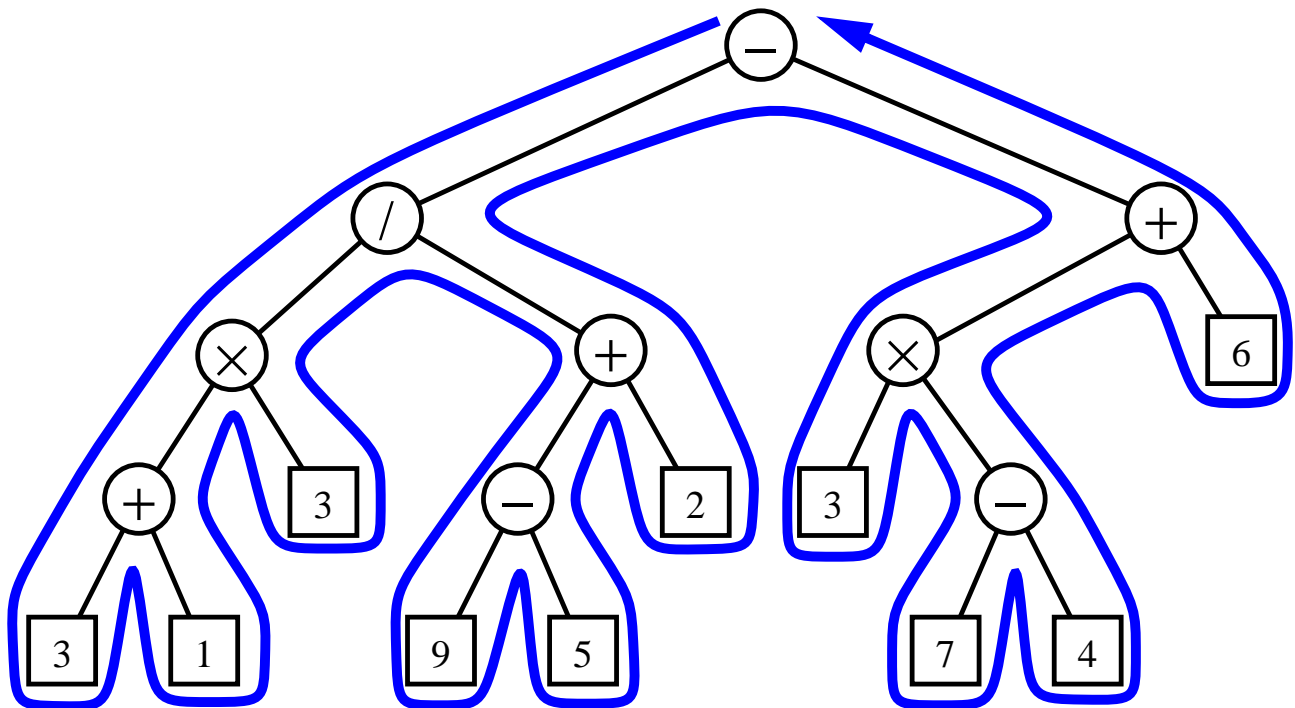
- printing an arithmetic expression
 - specialization of an inorder traversal
 - print “(“ before traversing the left subtree
 - print “)” after traversing the right subtree



$((((3 \times (1 + (4 + 6))) + (2 + 8)) \times 5) + (4 \times (7 + 2)))$

Euler Tour Traversal

- generic traversal of a binary tree
- the preorder, inorder, and postorder traversals are special cases of the Euler tour traversal
- “walk around” the tree and visit each node three times:
 - on the left
 - from below
 - on the right



Template Method Pattern

- generic computation mechanism that can be specialized by redefining certain steps
- implemented by means of an abstract Java class with **methods that can be redefined by its subclasses**

```
public abstract class BinaryTreeTraversal {  
  
    protected BinaryTree tree;  
  
    ...  
  
    protected Object traverseNode(Position p) {  
        TraversalResult r = initResult();  
        if (tree.isExternal(p)) {  
            external(p, r);  
        } else {  
            left(p, r);  
            r.leftResult = traverseNode(tree.leftChild(p));  
            below(p, r);  
            r.rightResult = traverseNode(tree.rightChild(p));  
            right(p, r);  
        }  
        return result(r);  
    }  
}
```

Specializing the Generic Binary Tree Traversal

- printing an arithmetic expression

```
public class PrintExpressionTraversal
    extends BinaryTreeTraversal {

    ...

    protected void external(Position p, TraversalResult r) {
        System.out.print(p.element());
    }

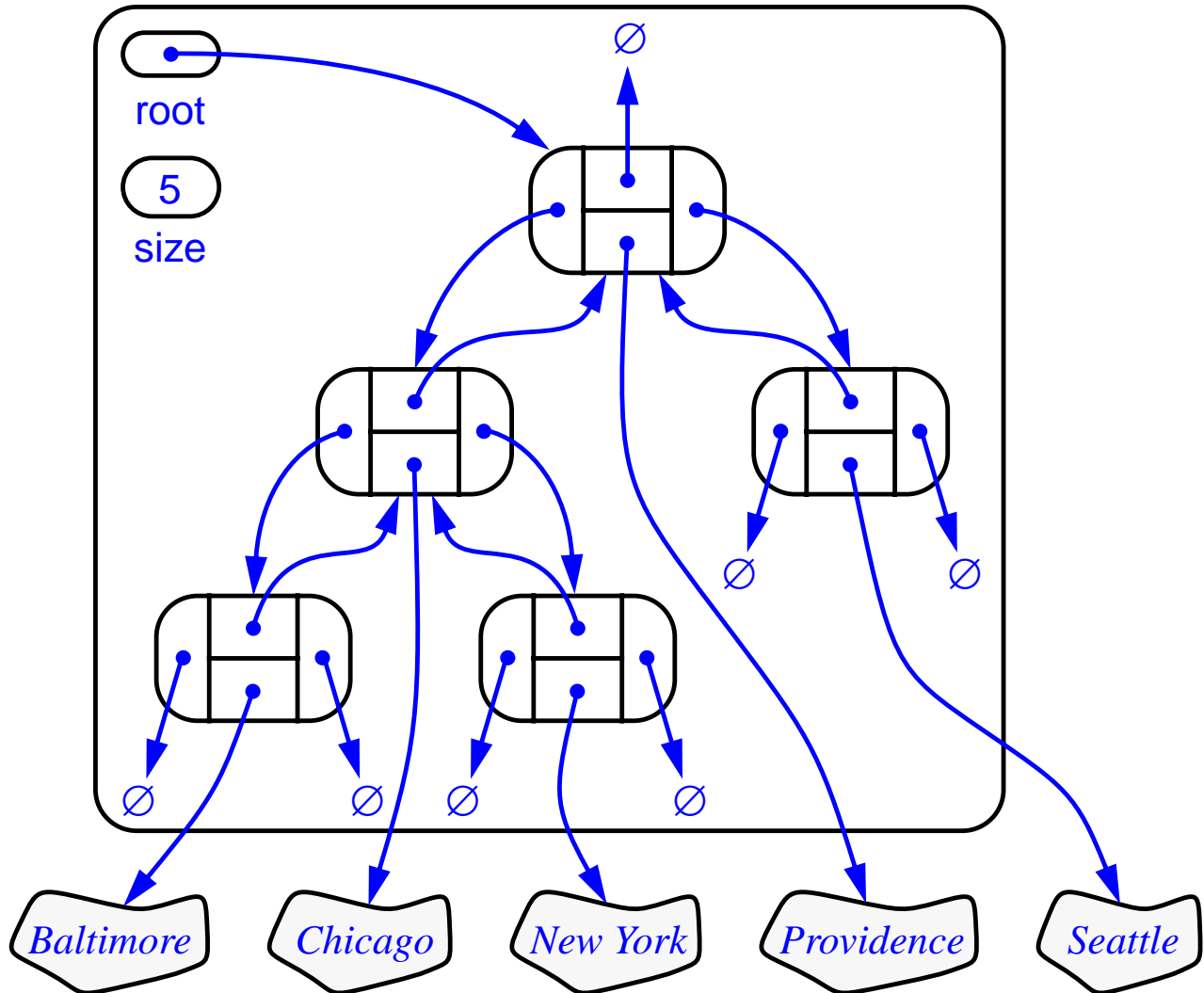
    protected void left(Position p, TraversalResult r) {
        System.out.print("(");
    }

    protected void below(Position p, TraversalResult r) {
        System.out.print(p.element());
    }

    protected void right(Position p, TraversalResult r) {
        System.out.print(")");
    }

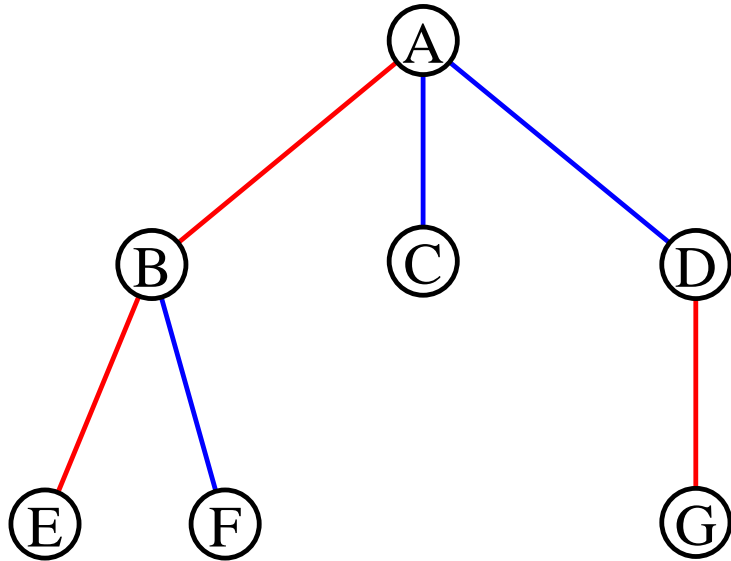
}
```


Linked Data Structure for Binary Trees



Representing General Trees

- tree T



- binary tree T' representing T

