What is a Tree

- In computer science, a tree is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Applications:
  - Organization charts
  - File systems
  - Programming environments

Tree Terminology

- **Root**: node without parent (A)
- **Internal node**: node with at least one child (A, B, C, F)
- **External node** (a.k.a. leaf): node without children (E, I, J, K, G, H, D)
- **Ancestors of a node**: parent, grandparent, grand-grandparent, etc.
- **Depth of a node**: number of ancestors
- **Height of a tree**: maximum depth of any node (3)
- **Descendant of a node**: child, grandchild, grand-grandchild, etc.
- **Subtree**: tree consisting of a node and its descendants

---

Tree ADT

- **We use positions to abstract nodes**
- **Generic methods**:
  - integer size()
  - boolean isEmpty()
  - Iterator iterator()
  - Iterable positions()
- **Accessor methods**:
  - position root()
  - position parent(p)
  - Iterable children(p)
  - Integer numChildren(p)
- **Query methods**:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- **Additional update methods** may be defined by data structures implementing the Tree ADT
Java Interface

Methods for a Tree interface:

```java
/** An interface for a tree where nodes can have an arbitrary number of children. */
public interface Tree<E> extends Iterable<E> {
    Position<E> root();
    Position<E> parent(Position<E> p) throws IllegalArgumentException;
    Iterable<Position<E>> children(Position<E> p) throws IllegalArgumentException;
    int numChildren(Position<E> p) throws IllegalArgumentException;
    boolean isInternal(Position<E> p) throws IllegalArgumentException;
    boolean isExternal(Position<E> p) throws IllegalArgumentException;
    boolean isRoot(Position<E> p) throws IllegalArgumentException;
    int size();
    boolean isEmpty();
    Iterator<E> iterator();
    Iterable<Position<E>> positions();
}
```

Preorder Traversal

- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

**Algorithm** `preOrder(v)`

1. `visit(v)`
2. For each child `w` of `v`:
   - `preorder(w)`

```
Algorithm preOrder(v)
visit(v)
foreach child w of v
preorder(w)
```
Postorder Traversal

- In a postorder traversal, a node is visited after its descendants.
- Application: compute space used by files in a directory and its subdirectories.

Algorithm `postOrder(v)`

```
for each child w of v
    postOrder(w)
visit(v)
```

Binary Trees

- A binary tree is a tree with the following properties:
  - Each internal node has at most two children (exactly two for proper binary trees).
  - The children of a node are an ordered pair.
- We call the children of an internal node left child and right child.
- Alternative recursive definition: a binary tree is either
  - a tree consisting of a single node, or
  - a tree whose root has an ordered pair of children, each of which is a binary tree.

Applications:
- arithmetic expressions
- decision processes
- searching
Arithmetic Expression Tree

- Binary tree associated with an arithmetic expression
  - Internal nodes: operators
  - External nodes: operands
- Example: arithmetic expression tree for the expression \((2 \times (a - 1) + (3 \times b))\)

Decision Tree

- Binary tree associated with a decision process
  - Internal nodes: questions with yes/no answer
  - External nodes: decisions
- Example: dining decision
  - Want a fast meal?
    - Yes
    - How about coffee?
      - Yes: Starbucks
      - No: Chipotle
    - No
      - On expense account?
        - Yes: Gracie’s
        - No: Café Paragon
Properties of Proper Binary Trees

- **Notation**
  - \( n \): number of nodes
  - \( e \): number of external nodes
  - \( i \): number of internal nodes
  - \( h \): height

- **Properties:**
  - \( e = i + 1 \)
  - \( n = 2e - 1 \)
  - \( h \leq i \)
  - \( h \leq (n - 1)/2 \)
  - \( e \leq 2^h \)
  - \( h \geq \log_2 e \)
  - \( h \geq \log_2 (n + 1) - 1 \)

---

BinaryTree ADT

- The **BinaryTree ADT** extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT.
- **Additional methods:**
  - `position left(p)`
  - `position right(p)`
  - `position sibling(p)`

- **Update methods** may be defined by data structures implementing the BinaryTree ADT.

- The above methods return `null` when there is no left, right, or sibling of \( p \), respectively.
Inorder Traversal

- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: draw a binary tree
  - \( x(v) = \) inorder rank of \( v \)
  - \( y(v) = \) depth of \( v \)

Algorithm \( \text{inOrder}(v) \)

- if \( \text{left}(v) \neq \text{null} \)
  - \( \text{inOrder} \left( \text{left}(v) \right) \)
- \( \text{visit}(v) \)
- if \( \text{right}(v) \neq \text{null} \)
  - \( \text{inOrder} \left( \text{right}(v) \right) \)

Print Arithmetic Expressions

- Specialization of an inorder traversal
  - print operand or operator when visiting node
  - print “(“ before traversing left subtree
  - print “)” after traversing right subtree

Algorithm \( \text{printExpression}(v) \)

- if \( \text{left}(v) \neq \text{null} \)
  - \( \text{print} \left( \text{“} \right) \)
  - \( \text{inOrder} \left( \text{left}(v) \right) \)
  - \( \text{print} \left( \text{v.element}() \right) \)
  - \( \text{print} \left( \text{“} \right) \)
  - \( \text{inOrder} \left( \text{right}(v) \right) \)
  - \( \text{print} \left( \text{“} \right) \)

\( ((2 \times (a - 1)) + (3 \times b)) \)
Evaluate Arithmetic Expressions

- Specialization of a postorder traversal
  - recursive method returning the value of a subtree
  - when visiting an internal node, combine the values of the subtrees

Algorithm `evalExpr(v)`

```plaintext
if isExternal(v)
    return v.element()
else
    x ← evalExpr(left(v))
    y ← evalExpr(right(v))
    ◊ ← operator stored at v
    return x ◊ y
```

Euler Tour Traversal

- Generic traversal of a binary tree
- Includes a special cases the preorder, postorder and inorder traversals
- Walk around the tree and visit each node three times:
  - on the left (preorder)
  - from below (inorder)
  - on the right (postorder)
Linked Structure for Trees

- A node is represented by an object storing:
  - Element
  - Parent node
  - Sequence of children nodes
- Node objects implement the Position ADT

Linked Structure for Binary Trees

- A node is represented by an object storing:
  - Element
  - Parent node
  - Left child node
  - Right child node
- Node objects implement the Position ADT
Array-Based Representation of Binary Trees

- Nodes are stored in an array $A$

Node $v$ is stored at $A[\text{rank}(v)]$

- $\text{rank}(\text{root}) = 0$
- If node is the left child of parent(node),
  $\text{rank}(\text{node}) = 2 \cdot \text{rank}(\text{parent(node)}) + 1$
- If node is the right child of parent(node),
  $\text{rank}(\text{node}) = 2 \cdot \text{rank}(\text{parent(node)}) + 2$