Rooted Trees

Collection of nodes, one of which is the root.
• Nodes ≠ root have unique parent node.
• Each nonroot can reach the root by following parent links one or more times.

Important Definitions
• If node p is the parent of node c, then c is a child of p.
• Leaf: no children; interior node has children.
• Path = list of nodes \((m_1, m_2, \ldots, m_k)\) such that each is the parent of the following node.
  □ It is “from \(m_1\) to \(m_k\).”
  □ Length of the path = \(k - 1\), the number of links, not nodes.
• If there is a path from \(m\) to \(n\), then \(m\) is an ancestor of \(n\) and \(n\) is a descendant of \(m\).
  □ Note \(m = n\) is possible.
  □ Proper ancestors, descendants exclude the possibility \(m = n\).
• Height of a node \(n\) is the length of the longest path from \(n\) to a leaf.
  □ Height of a tree is the height of its root.
• Depth of a node \(n\) is the length of the path from the root to \(n\).
• The subtree rooted at node \(n\) is all the descendants of \(n\) (including \(n\), of course!).
• The children of a given node are often ordered “from the left.”
  □ If so, and child \(c_1\) is to the left of child \(c_2\), then all nodes in the subtree rooted at \(c_1\) are said to be “to the left” of those in the subtree of \(c_2\).
• Nodes may have labels, which are values associated with the nodes.

Example: Expression trees: Labels are operands or operators.

• Leaf = operand; interior node = operator.

• Children are roots of the subexpressions to which the operator is applied.

Leftmost-Child, Right-Sibling Tree Representation

Each node has a pointer to

1. Its leftmost child.
2. Its right-sibling = node immediately to the right having the same parent.

• Advantage: represents trees without limit on number of children.

• Disadvantage: to find ith child of node n you must traverse list of right-sibling pointers starting at the leftmost child of n.

□ Nevertheless, this representation is the preferred approach in most cases.

Recursions on Trees

Many algorithms to process trees are designed with a basis = leaves and induction = interior nodes.

Example: Expressions in:

• infix (common form — operator between operands),

• prefix (operator before operands, like function calls without parentheses),

• postfix (operator after operands — important for compilers, because it gives the order in which computer must do things).

A recursive algorithm to convert from infix expression trees to postfix:

Basis: For a leaf, just print the operand.
**Induction:** At an interior node, having an operator:

- For each child, in order from the left, apply the algorithm at the child.
- Finally, list the operator.

**Example:** Infix expression \((a + b) \times c + (d \times e)\).

- Expression tree (note parentheses are not needed):

```
        +
      /   \
    *     *
  /     /  \
c  d   e  \\
 +   \\
a  b  
```

Result of Recursive algorithm: \(ab + c \times de \times +\).

**Preorder, Postorder Traversals**

Two common ways to explore a tree.

- Assume some "action" is to be taken at each node, e.g. printing its label.

- **Postorder**:

  **Basis:** Visit a leaf by performing the action there.

  **Induction:** Visit an interior node by visiting all its children, from the left, then performing the action at the node.

  **Example:** If the action is to list the label, postorder traversal converts the example expression tree to its equivalent postfix expression.

- **Preorder**:

  **Basis:** Visit a leaf by performing the action there.

  **Induction:** Perform the action at the node, then visit its children, from the left.

  **Example:** If the action is to print labels, the result of a preorder traversal of the previous example tree is \(+ \times + abc \times de\).
Structural Induction

- **Basis**: leaves (one-node trees).
- **Induction**: interior nodes (trees with ≥ 2 nodes). Assume the statement holds for the subtrees at the children of the root and prove the statement for the whole tree.
- A shorthand for induction on height of a tree or number of nodes in a tree.

**Example**: Consider the LMC-RS (leftmost-child, right-sibling) data structure for trees.

- \( S(T) \): \( T \) has one more NULL pointer than nodes.

**Basis**: \( T \) consists of a single node. It has neither a LMC nor a RS, so 2 NULL pointers and 1 node. Hence the basis holds.

**Induction**: \( T \) has a root \( r \) and one or more subtrees \( T_1, T_2, \ldots, T_k \). By the inductive hypothesis, each of these \( k \) trees *by itself* has one more NULL than node.

  - Think: “excess” is \( k \).
  - When we include the root, we add one node and one NULL pointer (the root’s LMC).
  - Excess is still \( k \).
- However, when they are children of a common node, the LMC pointers of the first \( k-1 \) become non-NULL.
  - Excess is reduced to 1, proving \( S(T) \).