Recursive Data Structures and Grammars

- **Themes**
  - Recursive Description of Data Structures
  - Grammars and Parsing
  - Recursive Definitions of Properties of Data Structures
  - Recursive Algorithms for Manipulating and Traversing Data Structures

- **Examples**
  - Lists
  - Trees
  - Expressions and Expression Trees
Lists

❖ A 2-tuple

➢ The data (for this node), and the rest of the list
  ▪ List = () – empty list
  ▪ List = ( d, List ), where d is some element

❖ E.g.,

L = ()
L = (24, L) – now (24, ( ))
L = (3, L ) – now (3, ( 24, ( )))
Lists – Observations

- Can be modified in constant time (vs. $\Theta(n)$ for an array)
- Do *not* support random access. To get the $k^{th}$ element is $\Theta(n)$ (vs. $\Theta(1)$ for an array)
Stacks – Linear structures

- Linear structure – Last In First Out (LIFO)
- Relative order of elements is maintained
- Can be built on a list or array. All operations are constant-time
- E.g.:
  - The “undo” stack in an editor
  - The operands and operators in a scientific calculator
  - The function call stack in a running program
Stack – operations

- `push(S, x)` – insert x onto the “top” of S
- `pop(S)` – remove the element at the top
- `is_empty(S)` – true if stack is empty
Queue – Linear structure

- Linear structure – First In First Out (FIFO)
- Relative order of elements is maintained
- Can be built on a list or array. All operations are (can be) constant-time

- E.g.:
  - The print spooler
  - Your email server
Queue – operations

- insert(Q, x) – insert x at the end (back) of Q
- pop(Q) – remove the element at the front
- is_empty(Q) – true if queue is empty
n-Trees

- Finite collection of nodes
  - Each node has exactly one *parent*, but for the *root node*
  - Each node may have up to $n$ children
  - A *leaf node* has no children
  - An *interior node* is a node that is not a leaf

- Each *subtree* is a tree rooted at a given node
Binary Trees

- A *binary tree* is a 2-tree
  - Each node has, at most, 2 sub-trees
- A binary tree is
  - Empty, or
  - Consists of a node with 3 attributes:
    - value
    - left, which is a tree
    - right, which is a tree
- A subtree of a binary tree is a binary tree
Number of Nodes of a Binary Trees

- $\text{Nnodes}(T) = 0$ if $T$ is empty
- $\text{Nnodes}(T\.\text{left}) + \text{Nnodes}(T\.\text{right}) + 1$
Internal Path Length (IPL)

- The sum of the length of the paths from the root to every node in the tree

- Recursively:
  - \( IPL(T) = 0 \) if \( T \) is empty
  - \( IPL(T) = IPL(\text{left}(T)) + IPL(\text{right}(T)) + \text{num_nodes}(T) - 1 \)
Paths, Ancestors, Descendants

Given a sequence of nodes:

\[ m_1, m_2, m_3, \ldots, m_k, \]

where \( m_i \) is parent of \( m_{i+1} \) \( \forall \ i < k \)

- \( m_i \) is an ancestor of \( m_j \) if \( i \leq j \)
- \( m_i \) is a proper ancestor of \( m_j \) if \( i < j \)
- \( m_i \) is a descendant of \( m_j \) if \( i \geq j \)
- \( m_i \) is a proper descendant of \( m_j \) if \( i > j \)
- \( m_1, m_2, m_3, \ldots, m_k \) is a path from \( m_1 \) to \( m_k \), of length \( k-1 \)
Height of Binary Trees

- Height of tree $T$, $H(T)$, is the max length of the paths from the root all of the leaves.
- $Height(T) = -1$ if $T$ is empty.
- $max( Height(T.left), Height(T.right) ) + 1$
Depth of a Node in a Tree

- The *depth*, or *level* of a node is the length of the path from the root to that node.
Internal Path Length

- Sum of the depths of all of the nodes in a tree
  - Alternatively, the sum of the level of every node in the tree
- Recursively
  - IPL(T) = 0 if T is empty
  - IPL(T) = IPL(T.left) + IPL(T.right) + Nnodes(T)-1
External Format for Binary Trees

- `<bintree> → []
  → [<value>,<bintree>,<bintree>]`
- `[],
- `[1,[],[]],
- `[2,[1,[],[]],[]], [2,[],[1,[],[]]]`
- `[3, [2,[1,[],[]],[]], []], [3, [2,[],[1,[],[]]],[]]`
- `[3, [1,[],[]], [1,[],[]]],
- `[3, [],[2,[1,[],[]],[]]], [3, [],[2,[],[1,[],[]]]]`
Recurrence for the Number of Binary Trees

Let $T_n$ be the number of binary trees with $n$ nodes.

$T_0 = 1$, $T_1 = 1$, $T_2 = 2$, $T_3 = 5$

$$T_n = \sum_{k=0}^{n-1} T_k T_{n-k-1}$$
Ordered Trees

- **Binary Search Tree (BST)**
  - Binary Tree
  - All elements in $T\rightarrow$left are $< T\rightarrow$value
  - All elements in $T\rightarrow$right are $\geq T\rightarrow$value

- Each subtree of a BST is a BST
Partially Ordered Trees

- **Heap (binary)**
  - *Complete* binary tree
    - So, height = $\Theta(n)$
  - Nodes are assigned some measure, a *priority*
  - No node has lower priority than its children
Inorder traversal

- Recursively visit nodes in T.left
- visit root
- Recursively visit nodes in T.right

- An in order traversal of a BST lists the elements in sorted order. Proof by induction.
Expression Trees

- Basic arithmetic expressions can be represented by a binary tree
  - Internal nodes are operators
  - Leaf nodes are operands
- Consider $2 \times (3 + 5)$:
- Note that parentheses don’t appear. Expression trees are not ambiguous, as infix expressions are (can be).
Expression Tree – In-order Traversal

- An in-order traversal yields $2 * 3 + 5$
- We put parentheses around every operation to make this correct: $(2 * (3 + 5))$
  (not all are needed)
Pre- and Post-Order Traversal

- Always go left-right
- Pre (me first):
  - Visit node
  - Traverse left subtree
  - Traverse right subtree
- Post (me last):
  - Traverse left subtree
  - Traverse right subtree
  - Visit node
Expression Trees – Pre- and Post-Order

- Pre: * 2 + 3 5
- Post: 2 3 5 + *

- Note: Parentheses never needed
Ambiguous Grammars (preview)
(blank, for notes)