Announcements

- Homework 2 released on website
  - Due Oct. 6th at 5:40 PM (7 days)
- Homework 1 solutions posted
- Post homework to Shared Files, Homework #2
Review

- Review of scope
- Stack applications examples
- Stack implementation (easy)
- Queue ADT definition and implementation
Today’s Plan

• Lists, Stacks, Queues in Linux
• Introduction to Trees
  • Definitions
  • Tree Traversal Algorithms
• Binary Trees
Lists, Stacks, Queues in Linux

- Linux:
  - processes stored in Linked List
  - FIFO scheduler schedules jobs using queue
  - function calls push memory onto stack
Drawbacks of Lists

• So far, the ADT’s we’ve examined have been linear
• \(O(N)\) for simple operations
• Can we do better?
  • Recall binary search: \(\log N\) for find  :-)
  • But list must be sorted. \(N \log N\) to sort  :-(
Trees

- Extension of Linked List structure:
  - Each node connects to multiple nodes
- Example usages include file systems, Java class hierarchies
- Fast searchable collections
Tree Terminology

- Just like Linked Lists, **Trees** are collections of **nodes**
- Conceptualize trees upside down (like family trees)
  - the top node is the **root**
  - nodes are connected by **edges**
  - edges define **parent** and **child** nodes
  - nodes with no children are called **leaves**
• Nodes that share the same parent are **siblings**

• A **path** is a sequence of nodes such that the next node in the sequence is a child of the previous
More Tree Terminology

• a node’s **depth** is the length of the path from root

• the **height** of a tree is the maximum depth

• if a path exists between two nodes, one is an **ancestor** and the other is a **descendant**
Tree Implementation

- Many possible implementations
- One approach: each node stores a list of children

```
public class TreeNode<T> {
    T Data;
    Collection<TreeNode<T>> myChildren;
}
```
Tree Traversals

- Suppose we want to print all nodes in a tree
- What order should we visit the nodes?
  - **Preorder** - read the parent before its children
  - **Postorder** - read the parent after its children
Preorder vs. Postorder

- // parent before children
  preorder(node x)
  print(x)
  for child : myChildren
    preorder(child)

- // parent after children
  postorder(node x)
  for child : myChildren
    postorder(child)
  print(x)
Binary Trees

- Nodes can only have two children:
  - left child and right child
- Simplifies implementation and logic
- `public class BinaryNode<T> {`
  `T element;`
  `BinaryNode<T> left;`
  `BinaryNode<T> right;`
- Provides new **inorder** traversal
Inorder Traversal

- Read left child, then parent, then right child
- Essentially scans *whole* tree from left to right
- `inorder(node x)`
  - `inorder(x.left)`
  - `print(x)`
  - `inorder(x.right)`
Binary Tree Properties

- A binary tree is **full** if each node has 2 or 0 children.
- A binary tree is **perfect** if it is full and each leaf is at the same depth.
- That depth is $O(\log N)$. 
Expression Trees

- Expression Trees are yet another way to store mathematical expressions
  - \(((x + y) \times z)/300\)
- Note that the main mathematical operators have 2 operands each
- Inorder traversal reads back infix notation
- Postorder traversal reads postfix notation
Decision Trees

- It is often useful to design decision trees.
- Left/right child represents yes/no answers to questions.
Reading

- This class: Weiss 4.1-4.2
- Next class: Weiss 4.3