Trees

CSE 373
Data Structures
Winter 2007

Why Do We Need Trees?

• Lists, Stacks, and Queues are linear relationships
• Information often contains hierarchical relationships
  › File directories or folders
  › Moves in a game
  › Hierarchies in organizations
• Can build a tree to support fast searching

Tree Jargon

• root
• nodes and edges (aka vertices and arcs)
• leaves
• parent, children, siblings
• ancestors, descendants
• subtrees
• path, path length
• height, depth

Definition and Tree Trivia

• A tree is a set of nodes, i.e., either
  › it’s an empty set of nodes, or
  › it has one node called the root from which zero or more trees (subtrees) descend
• A tree with N nodes always has N-1 edges (prove it by induction)
• A node has a single parent
• Two nodes in a tree have at most one path between them

More definitions

• Leaf (aka external) node: node without children
• Internal node: a node that is not a leaf
• Siblings: two nodes with the same parent
More Tree Jargon

- **Length of a path** = number of edges
- **Depth of a node N** = length of path from root to N
- **Height of node N** = length of longest path from N to a leaf
- **Depth of tree** = depth of deepest node
- **Height of tree** = height of root

Paths

- Can a non-zero path from node N reach node N again?
  - No. Trees can never have cycles (loops)
- Does depth (height) of nodes in a non-zero path increase or decrease?
  - Depth always increases in a non-zero path
  - Height always decreases in a non-zero path

More Jargon.....

- If there is a path from node u to node v, u is an ancestor of v
- Yes but... path in which direction?
  - Better to say:
    - Recursive definition: u is an ancestor of v if u = v or u is an ancestor of the parent of v
- Similar definition for descendant

Tree Operations

- The usual (size(), isEmpty(),...)
- Accessor methods
  - root(); error if the tree is empty
  - parent(v); error if v is the root
  - children(v); returns an iterable collection (i.e., ordered list) of children
- Queries (isRoot() etc...)
- How about iterators (or positions?)

Implementation of Trees (1)

- One possible pointer-based implementation
  - tree nodes with value and a pointer to each child
  - but how many pointers should we allocate space for?
  - OK if we use a pointer to a "collection" of children
  - But how should the "collection" be implemented? (doubly linked list?)
  - Should there be a parent link or not?

Implementation of Trees (2)

- A more flexible pointer-based implementation
  - 1st Child / Next Sibling List Representation
  - Each node has 2 pointers: one to its first child and one to next sibling
  - Can handle arbitrary number of children
  - Having a parent link is an orthogonal decision
### Arbitrary Branching

A tree with arbitrary branching.

- **A**
- **B**
- **C**
- **D**
- **E**
- **F**

### Binary Trees

- Every node has at most two children
  - Most popular tree in computer science
  - (But n-way branching common in databases, file structures; e.g., B-trees)
- Given N nodes, what is the minimum depth of a binary tree?
  - At depth $d$, you can have $N = 2^d$ to $N = 2^{d+1}-1$ nodes
  - $2^d \leq N \leq 2^{d+1}-1$ implies $d_{\text{min}} = \lceil \log_2 N \rceil$

### Minimum depth vs node count

- At depth $d$, you can have $N = 2^d$ to $2^{d+1}-1$ nodes
- Minimum depth $d$ is $O(\log N)$

### Maximum depth vs node count

- What is the maximum depth of a binary tree?
  - Degenerate case: Tree is a linked list!
  - Maximum depth = $N-1$
- Goal: Would like to keep depth at around $\log N$ to get better performance than linked list for operations like Find

### A degenerate tree

A linked list with high overhead and few redeeming characteristics.

### Traversing Binary Trees

- The definitions of the traversals are recursive definitions. For example:
  - Visit the root
  - Visit the left subtree (i.e., visit the tree whose root is the left child) and do this recursively
  - Visit the right subtree (i.e., visit the tree whose root is the right child) and do this recursively
- Traversal definitions can be extended to general (non-binary) trees
Traversing Binary Trees

- Preorder: Node, then Children (starting with the left) recursively + * + A B C D
- Inorder: Left child recursively, Node, Right child recursively A + B * C + D
- Postorder: Children recursively, then Node A B + C * D +