Trees

CSE 2320 – Algorithms and Data Structures
Vassilis Athitsos
University of Texas at Arlington
Trees

• Trees are a natural data structure for representing several types of data.
  – Family trees.
  – Organizational chart of a corporation, showing who supervises who.
  – Folder (directory) structure on a hard drive.
Terminology

• Root: 0
• Path: 0-2-6, 1-3, 1-3-4
• Parent vs child
• Ascendants vs descendants:
  – ascendants of 3: 1,0 // descendants of 1: 5, 3,4
• Internal vs external nodes
  (non-terminal vs terminal nodes)
• Leaves: 5,4,6,7
• M-ary trees (Binary trees)
• General trees
• Subtree
Terminology

• If Y is the parent of X, then X is called a **child** of Y.
  – The root has no parents.
  – Every other node, except for the root, has exactly one parent.

• A node can have 0, 1, or more children.

• Nodes that have children are called **internal nodes** or **non-terminal nodes**.

• Nodes that have no children are called **terminal nodes**, **external nodes**, or **leaves**.
Terminology

• The **level** of the root is defined to be 0.
• The **level** of each node is defined to be 1+ the level of its parent.
• The **depth** of a node is the number of edges from the root to the node. (It is equal to the level of that node.)
• The **height** of a node is the number of edges from the node to the deepest leaf.
M-ary Trees

- An **M-ary tree** is a tree where every node is either a leaf or it has **exactly** M children.
- Example: **binary** trees, **ternary** trees, ...

Is this a binary tree?

Is this a binary tree?
M-ary Trees

- An **M-ary tree** is a tree where every node is either a leaf or it has **exactly** $M$ children.
- Example: **binary** trees, **ternary** trees, ...

This is **not** a binary tree, node 3 has 1 child.

This is a binary tree.
General Trees

• Review this topic at the end.

• In a general tree a node can have any number of children.

• How would you implement a general tree?
Types of binary trees

• **Perfect** – each internal node has exactly 2 children and all the leaves are on the same level.
  – E.g. ancestry tree (anyone will have exactly 2 parents).

• **Full** – every node has exactly 0 or 2 children.
  – E.g. tree generated by the Fibonacci recursive calls.
  – Binary tree.

• **Complete tree** – every level, except for possibly the last one is completely filled and on the last level, all the nodes are as far on the left as possible.
  – E.g. the heap tree.
  – Height: \( \lceil \log N \rceil \) and it can be stored as an array.

A **perfect binary tree** with $N$ nodes has:

- $\lceil \lg N \rceil + 1$ levels
- Height $\lceil \lg N \rceil$
- $\left\lfloor N/2 \right\rfloor$ leaves (half the nodes are on the last level)
- $\left\lceil N/2 \right\rceil$ internal nodes (half the nodes are internal)

$$\sum_{k=0}^{n} 2^k = 2^{n+1} - 1$$

<table>
<thead>
<tr>
<th>Level</th>
<th>Nodes per level</th>
<th>Sum of nodes from root up to this level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$2^0$ ($=1$)</td>
<td>$2^1 - 1$ ($=1$)</td>
</tr>
<tr>
<td>1</td>
<td>$2^1$ ($=2$)</td>
<td>$2^2 - 1$ ($=3$)</td>
</tr>
<tr>
<td>2</td>
<td>$2^2$ ($=4$)</td>
<td>$2^3 - 1$ ($=7$)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$i$</td>
<td>$2^i$</td>
<td>$2^{i+1} - 1$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$n$</td>
<td>$2^n$</td>
<td>$2^{n+1} - 1$</td>
</tr>
</tbody>
</table>
Properties of Full Trees

• A **full** binary tree (0/2 children) with **N internal nodes** has:
  
  – N+1 external nodes.
  
  – 2N edges (links).
  
  – height at least \( \lg N \) and at most \( N \):
    
    • \( \lg N \) if all external nodes are at the same level (perfect tree)
    
    • \( N \) if each internal node has one external child.
Number of nodes per level

```
0
/     /
1     2
/     /
8     5
/     /
6     7
```

- Level 0: 1 node
- Level 1: 2 nodes
- Level 2: 3 nodes
- Level 3: 3 nodes
- Level 4: 2 nodes
typedef struct node *link;
struct node {
    Item item;
    link left;
    link right;
};

Other possible fields:
- parent (type link)
- Size (of subtree rooted at this node) (type int)
Traversing a Binary Tree

- **Traversing** is the process of going through each node of a tree, and doing something with that node. Examples:
  - We can print the contents of the node.
  - We can change the contents of the node.
  - We can otherwise use the contents of the node in computing something.

- We have four standard choices for the order in which we visit nodes when we traverse a binary tree.
  - **Preorder (Root, L, R)**: we visit the node, then its left subtree, then its right subtree. (depth-first order)
  - **Inorder (L, Root, R)**: we visit the left subtree, then the node, then the right subtree. (depth-first order)
  - **Postorder (L, R, Root)**: we visit the left subtree, then the right subtree, then the node. (depth-first order)
  - **Level order**: all the nodes on the level going from 0 to the last level. (breadth-first)
Examples

List the nodes of the tree in:
Preorder (___, ___, ___): ________________________________
Inorder (___, ___, ___): ________________________________
Postorder (___, ___, ___): _______________________________
Level-order: ________________________________
Examples

List the nodes of the tree in:

Preorder (Root, Left, Right): 0, 1, 5, 3, 12, 6, 7, 10, 4, 8, 2.
Inorder (Left, Root, Right): 5, 3, 1, 0, 6, 12, 4, 10, 8, 7, 2.
Postorder (Left, Right, Root): 3, 5, 1, 6, 4, 8, 10, 2, 7, 12, 0.
Level-order: 0, 1, 12, 5, 6, 7, 3, 10, 2, 4, 8.
Recursive Tree Traversal

```c
void traverse_preorder(link h){
    if (h == NULL) return;
    do_something_with(h);
    traverse_preorder (h->left);
    traverse_preorder (h->right);
}

void traverse_inorder(link h){
    if (h == NULL) return;
    do_something_with(h);
    traverse_inorder (h->left);
    do_something_with(h);
    traverse_inorder (h->right);
}

void traverse_postorder(link h){
    if (h == NULL) return;
    traverse_postorder(h->left);
    traverse_postorder(h->right);
    do_something_with(h);
}
```
Class Practice

• Write the following (recursive or not) functions, in class:
  – Count the number of nodes in a tree
  – Compute the height of a tree
  – Level-order traversal – discuss/implement
  – Print the tree in a tree-like shape – discuss/implement

• Which functions are “similar” to the traversals discussed previously and to each other?
• These slides contain code from the Sedgewick book.
Recursive Examples

Counting the number of nodes in the tree:

```c
int count(link h) {
    if (h == NULL) return 0;
    int c1 = count(h->left);
    int c2 = count(h->right);
    return c1 + c2 + 1;
}
```

Computing the height of the tree:

```c
int height(link h) {
    if (h == NULL) return -1;
    int u = height(h->left);
    int v = height(h->right);
    if (u > v) {
        return u + 1;
    } else {
        return v + 1;
    }
}
```
Recursive Examples: print tree

Print the contents of each node (assuming that the items in the nodes are characters)

How will the output look like?

What type of tree traversal is this?

```c
void printnode(char c, int h) {
    int i;
    for (i = 0; i < h; i++)
        printf(" ");
    printf("%c\n", c);
}

void show(link x, int h) {
    if (x == NULL) {
        printnode("*", h);
        return;
    }
    printnode(x->item, h);
    show(x->left, h+1);
    show(x->right, h+1);
}
```
Recursive and Iterative Preorder Traversal (Sedgewick)

```c
void traverse(link h, void (*visit)(link)) {
    if (h == NULL) return;
    (*visit)(h);
    traverse(h->left, visit);
    traverse(h->right, visit);
}
```

```c
void traverse(link h, void (*visit)(link)) {
    STACKinit(max);
    STACKpush(h);
    while (!STACKempty())
    {
        (*visit)(h = STACKpop());
        if (h->right != NULL) STACKpush(h->right);
        if (h->left != NULL) STACKpush(h->left);
    }
}
```

Stack:
Print:
// Adapted from Sedgewick
void traverse(link h, void (*visit)(link)) {
    Queue Q = new_Queue(max); put(q,h);
    while (!empty(Q)) {
        (*visit)(h = get(Q)); //gets first node
        if (h->left != NULL) put(Q,h->left);
        if (h->right != NULL) put(Q,h->right);
    }
}
General Trees

• In a general tree, a node can have any number of children.

• How would you implement a general tree?
General Trees

• In a general tree, a node can have any number of children.

• Left-child - right-sibling implementation
  – Draw tree and show example
  – (There is a one-to-one correspondence between ordered trees and binary trees)