

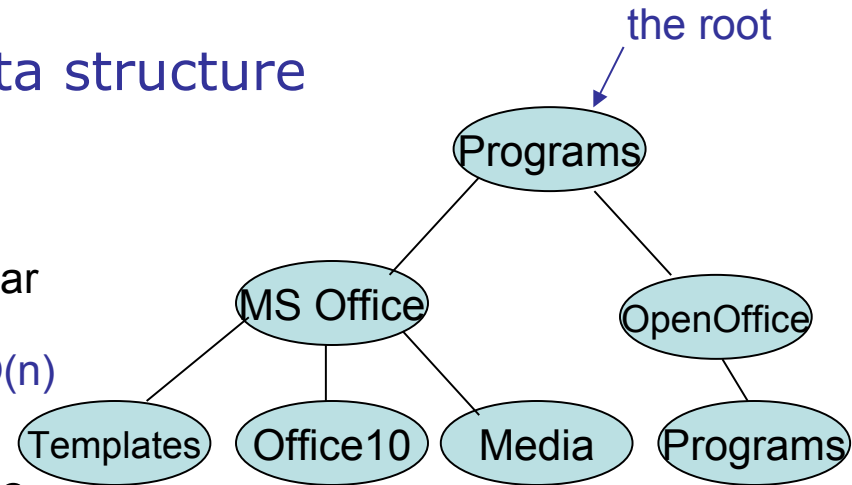
Tree Data Structures

Goodrich et al chapter 6

What is a tree?

A. another type of linked data structure

- The linked list we studied had one draw back
 - It's only possible to access data in a linear fashion
 - So algorithms based on it are typically $O(n)$ (linear)
- A tree is a **hierarchical** data structure
 - Each node can have many branches
 - One 'root node' (cp the trunk of a tree), many main branches which branch from this
 - Secondary branches which branch from the main ones ...
 - ... and so on
 - Use to represent hierarchical organisation of information
- In computer science, we draw a tree from the top down
 - so the root is at the head!



You can see in this diagram that by making a decision at each 'branch' it is possible to get from the root to any node in only 2 steps

- This will give scope for algorithms that do better than $O(n)$

Uses of Trees?

- **Trees have many applications in computer science:**
 - Decision trees are used for programs and Artificial Intelligence
 - A compiler will use an *expression tree* to evaluate arithmetic and other expressions in our code
 - A *Huffman tree* is a tree which is used to represent *Huffman codes* for characters that might appear in a text file
 - This provides a very efficient way to compress text files
 - We will introduce '*binary search trees*' which provide an efficient searching mechanism
 - Heaps (a type of tree) are used for sorting and finding max and min
- **And in OR and engineering:**
 - Spanning trees are used in electrical communications and transport networks
 - Molecule trees are used in chemical engineering
 - Shortest path trees for road networks and circuit design
 - Fault trees for reliability and fault finding – eg as diagnostic tool in car maintenance books.

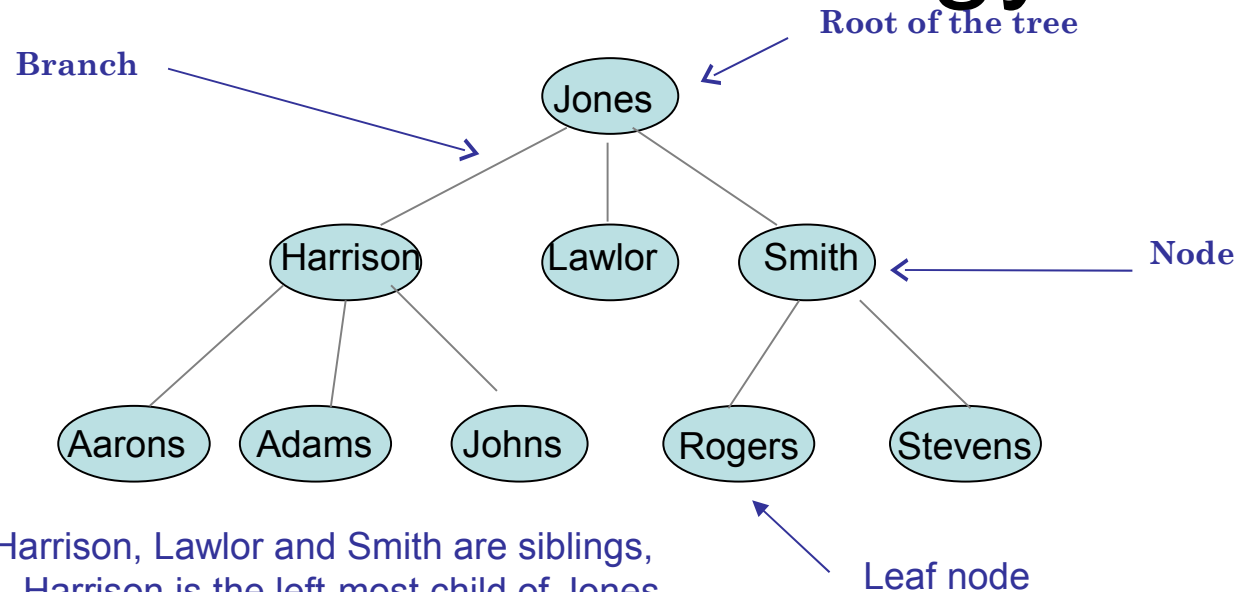
Some tree terminology

diagram

The elements of a tree are called *nodes*. A rooted-tree has exactly one *root node*, and many *branches* which each lead to other nodes of the tree (called *branch nodes*)

- A branch determines a relationship between nodes. The node nearer to the root is called the *parent node* of the other, which is a *child node*. Children of the same parent node are called *siblings*.
 - *Harrison is the left-most child of Jones*
 - *Harrison, Lawlor and Smith are siblings*
- A node with no child nodes is called a *leaf-node of the tree*
 - *Aarons and Lawlor are two examples of leaf nodes on the tree*
- A tree rooted at a child node is a *sub-tree* of the tree rooted at the parent.
 - *the sub-tree rooted at Harrison is the left-most subtree of the tree rooted at Jones.*
 - *Similarly, Smith is the right-most child of Jones, and the sub-tree rooted at Smith is the right-most sub-tree of the tree rooted at Jones.*
- An *ancestor* of node X is any node of a higher level that can be traced to node X. A *descendant* of node X is any node that can be traced from node X
 - *Jones is an ancestor of Adams on this tree*
 - *Rogers is a descendant of Jones on this tree.*
- The *level of a node* refers to its distance from the root. The root node is at level 0, and any child is one level more than its parent.
 - *In the tree shown, Jones is at level 0. Harrison Lawlor and Smith are level 1, and Aarons, Adams, Johns, Rogers and Stevens are level 2.*
- The *height of a tree* is determined by the maximum number of branches from the root to a leaf node, so it will be equal to the level of the leaf node farthest from the root
 - *The height of the tree shown is 2*

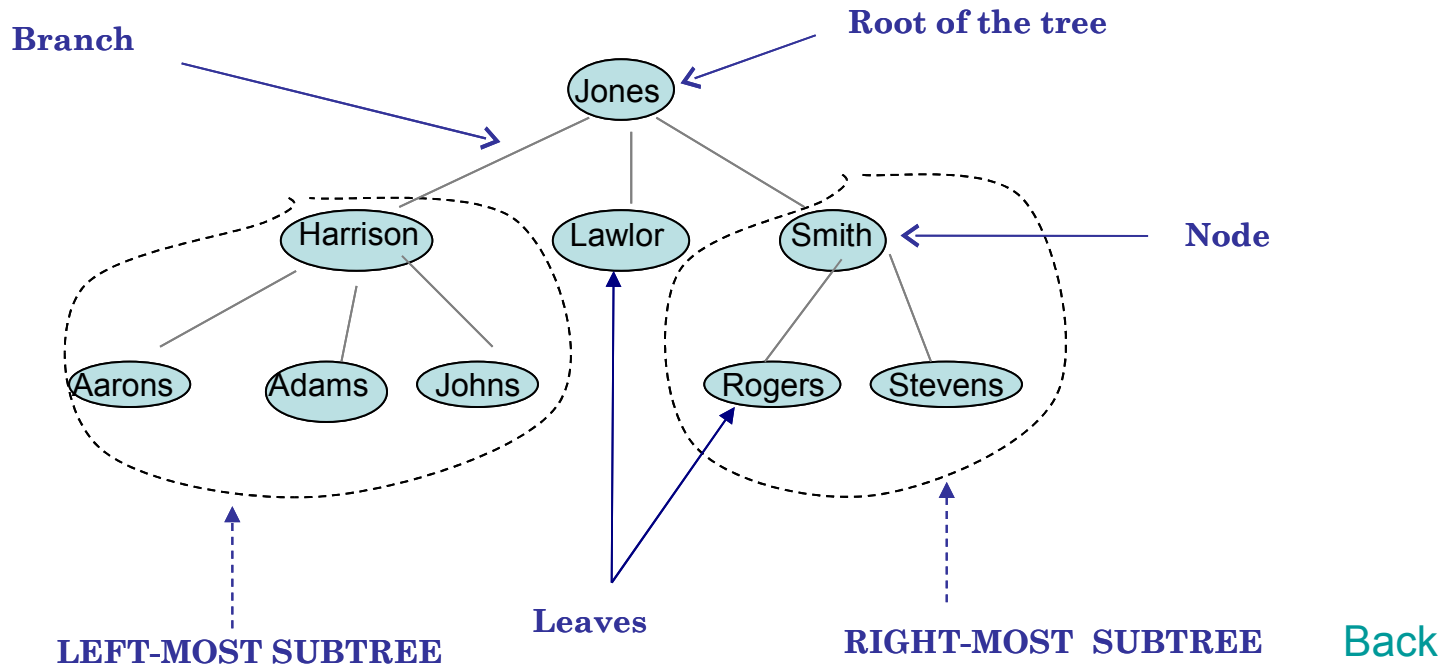
Tree terminology example



[Back](#)

1. Harrison, Lawlor and Smith are siblings,
Harrison is the left-most child of Jones
1. Aarons and Lawlor are two examples of leaf nodes on the tree.
2. subtrees
 1. Jones is an ancestor of Adams on this tree
Rogers is a descendant of Jones on this tree.
 1. in the tree shown Jones is at level 0. Harrison Lawlor and Smith are level 1, and Aarons, Adams, Johns, Rogers and Stevens are level 2.
 2. The height of the tree shown here is 2.

More tree terminology



The sub-tree rooted at Harrison is the left-most subtree of the tree rooted at Jones.

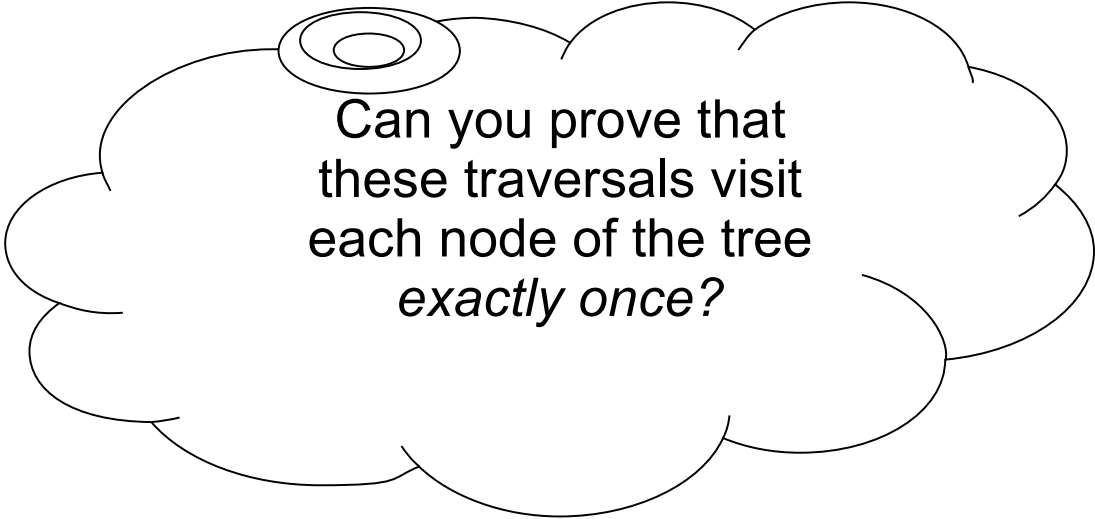
Smith is the right-most child of Jones, and the sub-tree rooted at Smith is the right-most sub-tree of the tree rooted at Jones

Recursive Tree Traversals

- Tree traversals provide systematic methods of ‘visiting’ each node of a tree (*exactly once for the traversals we are considering*)
- Tree traversals can easily be understood using recursive implementations.
 - **Pre-order traversal:** visits the root of the tree first , then traverses all sub-trees rooted at its children.
 - **Post-order traversal:** traverses all subtrees rooted at the children of the tree first, then visits its root.

What happens when you ‘visit’ a node?

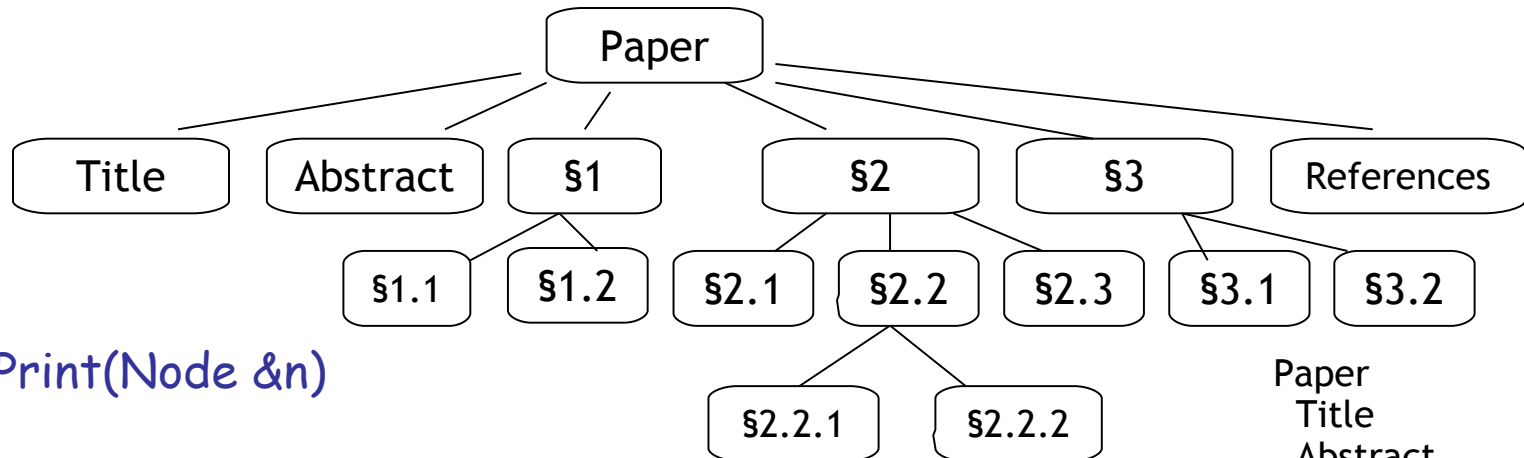
Maybe you visit to print the details of the node or to do something with the value at a node .. Compare it with a target value, or perform a calculation with it or ...



Can you prove that these traversals visit each node of the tree *exactly once*?

Application for pre-order traversal

- USE PRE-ORDER WHEN WE WANT TO PERFORM A TASK FOR A NODE, AND THEN RECURSIVELY PERFORM The TASK FOR EACH OF ITS CHILDREN
- Example1: print the table of contents for a book or paper



preorderPrint(Node &n)

BEGIN

print n

get an iterator over the children of n

FOR EACH of children

preorderPrint(c)

END FOR EACH

END

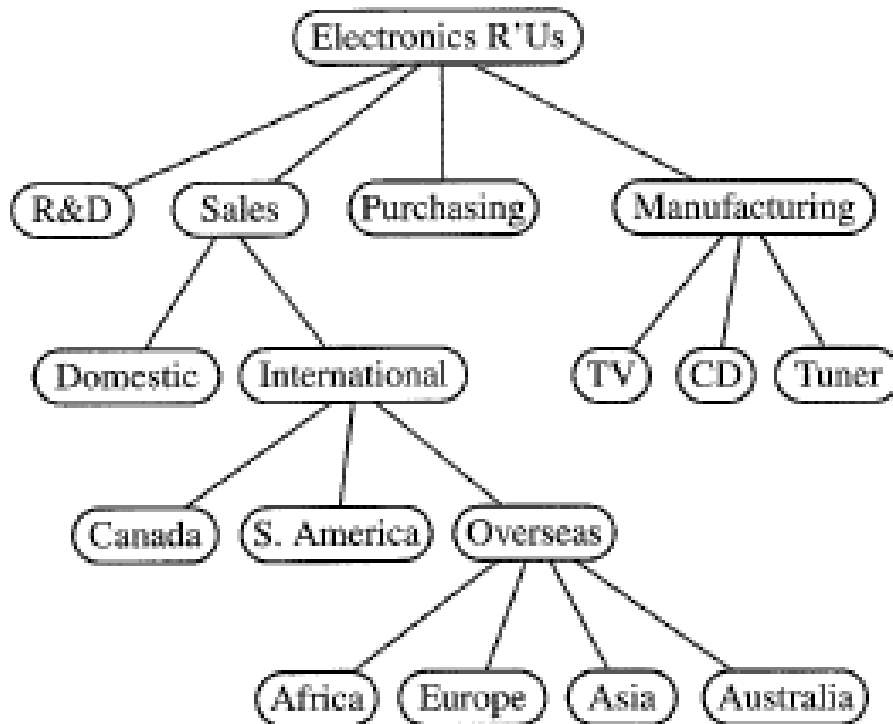
Paper
 Title
 Abstract
 \$1
 \$1.1
 \$1.2
 \$2
 \$2.1
 \$2.2
 \$2.2.1
 \$2.2.2
 \$2.3
 \$3
 \$3.1
 \$3.2
 References

Applications for post-order traversals

- Post-order traversal is useful for solving problems where we wish to compute some property for each node of a tree, but computing it for node v requires that we have already computed it for all of the children of v

Example1 (from Goodrich page 255)

- to print the name and total expenditure of each sub-division by adding its direct expenditure and any department expenditures



`void divTotal(Node &n)`

`BEGIN`

`double total = direct-expend of n`

`get iterator over children of n`

`FOR EACH of children`

`total += divTotal(child-node)`

`END FOR EACH`

`Print name and total for node n`

`Return total`

`END`

Applications for post-order traversals

- Example2: A file system tree
 - to compute the disk space used for a directory, which is recursively given by:
 - the size of the directory itself
 - The size of the files it contains
 - The space required for each sub-directory

Recap on Trees and Recursion

- Why are recursive algorithms so useful with tree data structures?
 - We can visit each node of the tree by recursively visiting each sub-tree
 - A simple case occurs when the 'sub-tree' turns out to be a leaf node
 - We know that we will reach such a simple base case because eventually we must always reach a leaf node
- What is meant by a pre-order traversal of a tree structure?
 - Visit the root before visiting its sub-trees
- A post-order traversal of a tree structure?
 - Visit the sub-trees before visiting the root