Discrete Mathematics (2009 Spring)
Trees (Chapter 10, 5 hours)

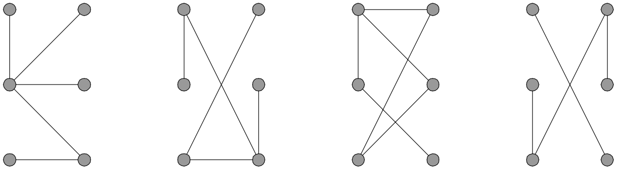
Chih-Wei Yi

Dept. of Computer Science
National Chiao Tung University

June 1, 2009

What's Trees?

- A tree is a connected undirected graph with no simple circuits.

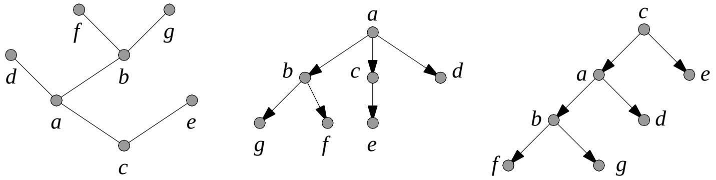


Theorem

An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

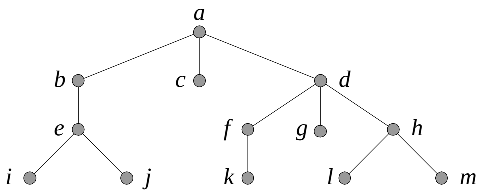
Rooted Trees

- A rooted tree is a tree in which one vertex has been designated as the root and every edge is directed away from the root.



Terminologies of Rooted Trees

- If v is a vertex in T other than the root, the *parent* of v is the unique vertex u such that there is a directed edge from u to v.
- If u is the parent of v, v is called a *child* of u.
- Vertices with the same parent are called *siblings*.



Terminologies of Rooted Trees (Cont.)

- The *ancestors* of a vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root.
- The *descendants* of a vertex v are those vertices that have v as an ancestor.
- A vertex of a tree is called a *leaf* if it has no children.
- Vertices that have children are called *internal vertices*.
- If a is a vertex in a tree, the *subtree* with a as its root is the subgraph of the tree consisting of a and its descendants and all edges incident to these descendants.

m-Ary Trees

- A root tree is called an *m-ary tree* if every internal vertex has no more than m children. The tree is called a *full m-ary tree* if every internal vertex has exactly m children. An m-ary tree with $m = 2$ is called a *binary tree*.
- An *ordered rooted tree* is a rooted tree where the children of each internal vertex are ordered. Ordered rooted trees are drawn so that the children of each internal vertex are shown in order from left to right.
- In an ordered binary tree (usually called just a binary tree), if an internal vertex has two children, the first child is called the *left child* and the second child is called the *right child*. The tree rooted at the left child (or right child, resp.) of a vertex is called the *left subtree* (or *right subtree*, resp.) of this vertex.

Properties of Trees

Theorem

A tree with n vertices has $n-1$ edges.

Theorem

A full m-ary tree with i internal vertices contains $n = mi + 1$ vertices.

Properties of Trees (Cont.)

Theorem

A full m-ary tree with

1. n *vertices has* $i = (n-1)/m$ *internal vertices and* $l = [(m-1)n+1]/m$ *leaves,*
2. i *internal vertices has* $n = mi+1$ *vertices and* $l = (m-1)i+1$ *leaves,*
3. l *leaves has* $n = (ml-1)/(m-1)$ *vertices and* $i = (l-1)/(m-1)$ *internal vertices.*

Theorem

There are at most m^h *leaves in an* m*-ary tree of height* h*.*

Binary Search Trees

Decision Trees

Prefix Codes

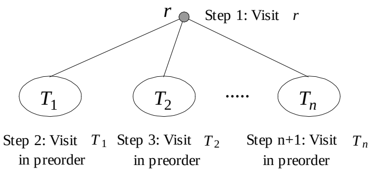
- Huffman coding: a special case of prefix codes

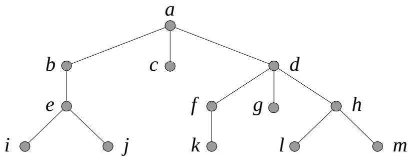
Game Trees

Preorder Traversal

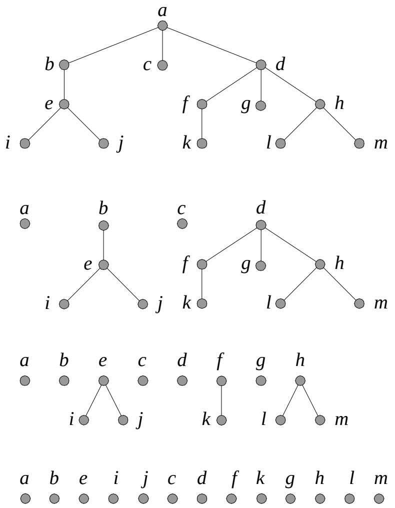
Definition

Let T be an ordered rooted tree with root r. If T consists only of r, then r is the *preorder traversal* of T. Otherwise, suppose that $T_1, T_2, \cdots, T_n$ are the subtrees at r from left to right in T. The *preorder traversal* begins by visiting r. It continues by traversing T_1 in preorder, then T_2 in preorder, and so on, until T_n is traversed in preorder.





Examples of Preorder Traversal



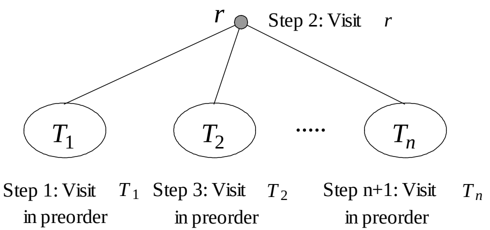
Pseudocode of Preorder Traversal

procedure *preorder* (T : ordered rooted tree)
r = root of T
list r
for each child c of r from left to right
begin
    $T(c)$:= subtree with c as its root
    preorder $(T(c))$
end

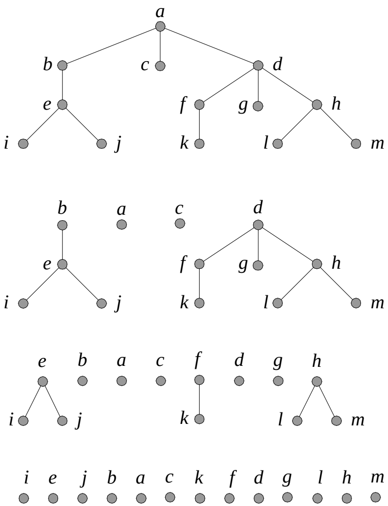
Inorder Traversal

Definition

Let T be an ordered rooted tree with root r. If T consists only of r, then r is the *inorder traversal* of T. Otherwise, suppose that $T_1, T_2, \cdots, T_n$ are the subtrees at r from left to right. The *inorder traversal* begins by traversing T_1 in inorder, then visiting r. It continues by traversing T_2 in inorder, then T_3 in inorder, ..., and finally T_n in inorder.



Examples of Inorder Traversal



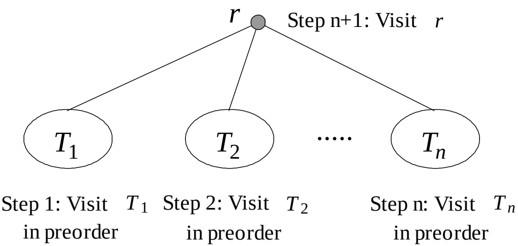
Pseudocode of Inorder Traversal

procedure *inorder* (T : ordered rooted tree)
r = root of T
if r is a leaf **then** list r
else
begin
    $l :=$ first child of r from left to right
    $T(l) :=$ subtree with l as its root
    inorder $(T(l))$
    list r
    for each child c of r except for l from left to right
    begin
        $T(c) :=$ subtree with c as its root
        inorder $(T(c))$
    end
end

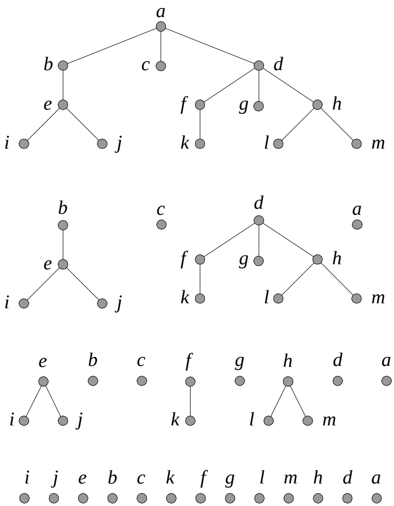
Postorder Traversal

Definition

Let T be an ordered rooted tree with root r. If T consists only of r, then r is the *postorder traversal* of T. Otherwise, suppose that $T_1, T_2, \cdots, T_n$ are the subtrees at r from left to right in T. The *postorder traversal* begins by traversing T_1 in postorder, then T_2 in postorder, ..., then T_n in postorder, and end by visiting r..



Examples of Postorder Traversal

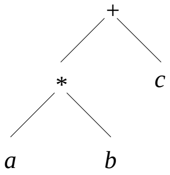


Pseudocode of Postorder Traversal

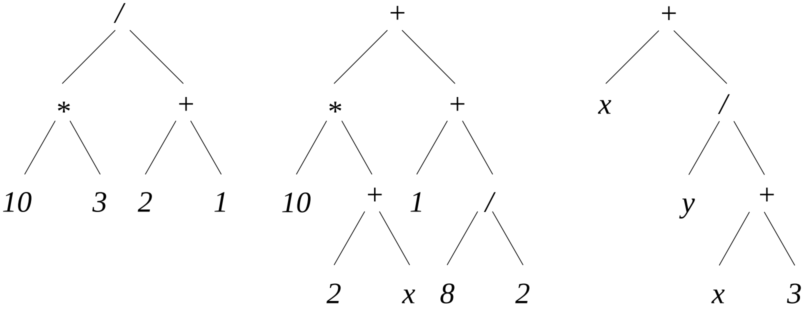
procedure *postorder* (T : ordered rooted tree)
r = root of T
for each child c of r from left to right
begin
    $T(c) :=$ subtree with c as its root
    postorder ($T(c)$)
end
list r

Infix, Prefix, and Postfix Notation

- Examples: infix, prefix, and postfix notations of $a \times b + c$
  - Infix: $a * b + c$
  - Prefix: $+ * abc$ (also called Polish notation)
  - Postfix: $ab * c+$
- Represented by ordered rooted trees.



Examples of Binary Tree Representation



What Is a Spanning Tree?

Definition

Let G be a simple graph. A *spanning tree* of G is a subgraph of G that is a tree containing every vertex of G.

Give Example Here!

Theorem

A simple graph is connected if and only if it has a spanning tree.

Proof.

First, we prove the "IF" part.
Then, we prove the "ONLY IF" part. □

How to Construct Spanning Trees?

- Depth-first search (DFS)
- Breadth-first search (BFS)

Algorithm: Depth-First Search

procedure
DFS $(G:$ connected graph with vertices $v_1, v_2, \cdots, v_n$ $)$
$T :=$ tree consisting only of the vertex v_1
visit (v_1)
procedure *visit* $(v:$ vertex of $G)$
for each vertex w adjacent to v and not yet in T
begin
    add vertex w adn edge $\{v, w\}$ to T
    visit (w)
end

An Example of Depth-First Search

Breadth-First Search

Algorithm: Breadth-First Search

procedure
BFS (G : connected graph with vertices $v_1, v_2, \cdots, v_n$)
$T :=$ tree consisting only of the vertex v_1
$L :=$ empty list
put v_1 in the list L of unprocessed vertices
while L is not empty
begin
    remove the first vertex, v, from L
    for each neighbor w of v and not yet in T
        if w is not in L and not in T **then**
        begin
            add w to the end of the list L
            add w and edge $\{v, w\}$ to T
        end
end

An Example of Breadth-First Search

Minimum Spanning Trees

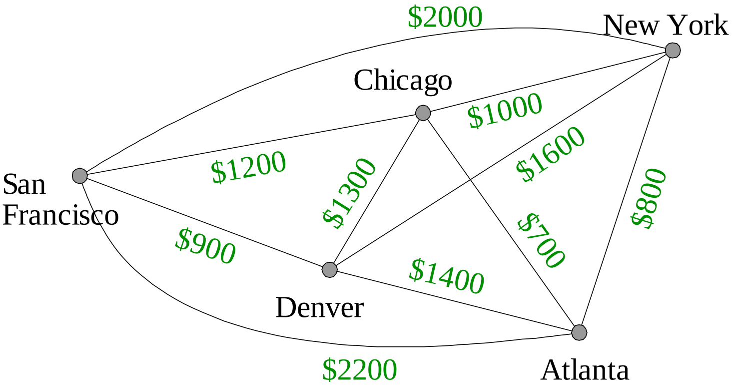
- If T is a spanning tree in a weighted graph $G(V, E, w)$, the weight of T, denoted by $w(T)$, is the sum of weights of edges in T.

$$w(T) = \sum_{e \in T} w(e).$$

- Given a weighted graph $G(V, E, w)$, the minimum spanning tree problem is to find a spanning tree in G that has the smallest wiehgt.

The Cost of a Computer Network

- What is the smallest total cost to maintain a connected network between those five cities?



Some Spanning Trees

- $T_1 = \left\{ \begin{array}{c} \{\text{Chicago,SF}\}, \{\text{Chicago,Denvor}\}, \\ \{\text{Chicago,Atlanta}\}, \{\text{Chicago,NY}\} \end{array} \right\}$

$$\begin{aligned} w(T_1) &= w(\{\text{Chicago,SF}\}) + w(\{\text{Chicago,Denvor}\}) \\ &\quad + w(\{\text{Chicago,Atlanta}\}) + w(\{\text{Chicago,NY}\}) \\ &= \$1200 + \$1300 + \$700 + \$1000 = \$4200. \end{aligned}$$

- $T_2 = \left\{ \begin{array}{c} \{\text{Chicago,SF}\}, \{\text{SF,Denvor}\}, \\ \{\text{Chicago,Atlanta}\}, \{\text{Atlanta,NY}\} \end{array} \right\}$

$$\begin{aligned} w(T_2) &= w(\{\text{Chicago,SF}\}) + w(\{\text{SF,Denvor}\}) \\ &\quad + w(\{\text{Chicago,Atlanta}\}) + w(\{\text{Atlanta,NY}\}) \\ &= \$1200 + \$900 + \$700 + \$800 = \$3600. \end{aligned}$$

Some Spanning Trees (Cont.)

- $T_3 = \left\{ \begin{array}{c} \{\text{Chicago,Denvor}\}, \{\text{Denvor,SF}\}, \\ \{\text{Denvor,Atlanta}\}, \{\text{Atlanta,NY}\} \end{array} \right\}$

$$\begin{aligned} w(T_3) &= w(\{\text{Chicago,Denvor}\}) + w(\{\text{Denvor,SF}\}) \\ &\quad + w(\{\text{Denvor,Atlanta}\}) + w(\{\text{Atlanta,NY}\}) \\ &= \$1300 + \$900 + \$1400 + \$800 = \$4400. \end{aligned}$$

- Problem: Which one is with the smallest weight among all possible spanning trees?

Prim's Algorithm

procedure *Prim* (G : weighted connected undirected graph with n vertices)

$T :=$ a minimum-weighted edge

for $i := 1$ to $n - 2$

begin

  $e :=$ an edge of minimum weight incident to a vertex in T and not forming a simple circuit in T if added to T

  $T := T$ with e added

end (T is a minimum spanning tree of G)

An Example of Prim's Algorithm

Choice	Edge	Cost
1	{Atlanta,Chicago}	\$700
2	{Atlanta,NY}	\$800
3	{Chicago,SF}	\$1200
4	{Denver,SF}	\$900
	Total	\$3600

Kruskal's Algorithm

procedure *Kruskal* (G : weighted connected undirected graph with n vertices)
$T :=$ empty graph
for $i := 1$ to $n-1$
begin
    $e :=$ an edge in G with smallest weight that does not form a simple circuit when added to T
    $T := T$ with e added
end (T is a minimum spanning tree of G)

An Example of Kruskal's Algorithm

- First, sort all edges based on their weight in ascending order.
  - {Atlanta,Chicago}, {Atlanta,NY}, {Denver,SF}, {Chicago,NY}, {Chicago,SF}, {Chicago,Denver}, {Atlanta,Denver}, {Denver,NY}, {NY,SF}, {Atlanta,SF}
- Exam each edge one by one until a spanning tree is constructed.

Choice	Edge	Cost
1	{Atlanta,Chicago}	\$700
2	{Atlanta,NY}	\$800
3	{Denver,SF}	\$900
4	{Chicago,SF}	\$1200
	Total	\$3600

Find a Spanning Tree with Minimum Weight