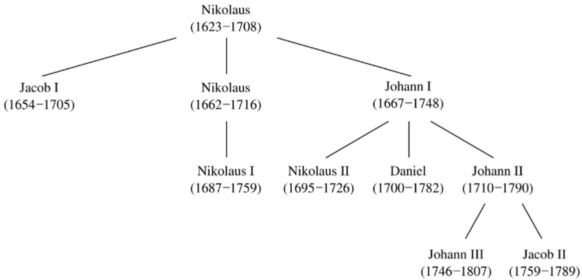
Discrete Mathematics: Lecture 13

Today:

- Chap 11.1: Introduction to trees
- Chap 11.3: Tree traversal

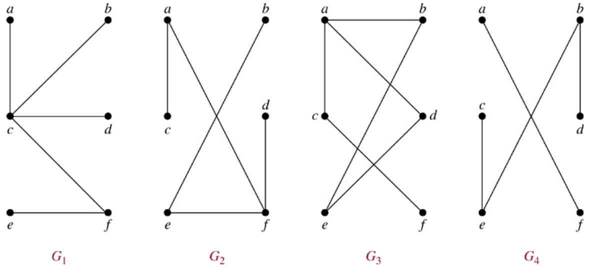
10.1: Introduction to trees

- We will focus on a particular type of graphs called trees.
- Example: the family tree of the male members of the Bernoulli family of Swiss mathematicians



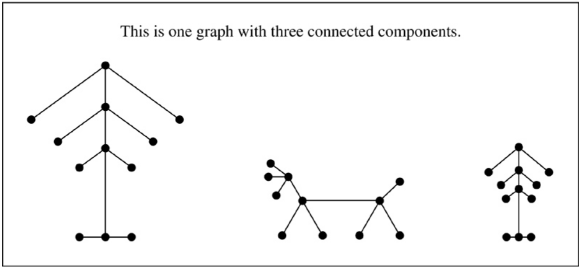
Definition of trees (树)

- Definition: A tree is a connected undirected graph with no simple circuits.
- A tree cannot have multiple edges or loops. Thus any tree must be a simple graph.
- Example: which of the following graphs are trees?



Forests (森林)

- Definition: A forest is an undirected graph with no simple circuits.
- A forest is not necessarily connected. Each connected component of a forest is a tree.
- Example:

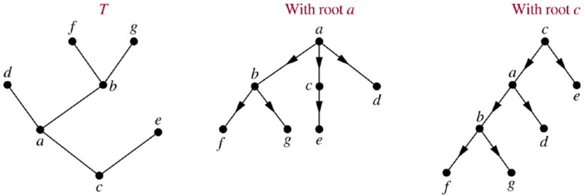


The basic property of trees

- Lemma: Let G be a simple graph. Let P_1 and P_2 be two distinct paths between the two vertices u and v. Then there is a simple circuit in G.
- Theorem: An undirected graph is a tree iff there is a unique simple path between any two of its vertices.

Rooted trees (根树)

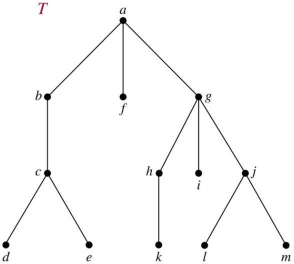
- In many applications of trees, a particular vertex of a tree is designated as the root.
- Then we can direct each edge away from the root.



- Note that different choices of the root produce different rooted trees.

Rooted trees

- Definition: 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.
- Rooted trees can also be defined recursively as in Chap 4.3.
- We usually draw a rooted tree with its root at the top of the graph. Then we omit the arrows on the edges.

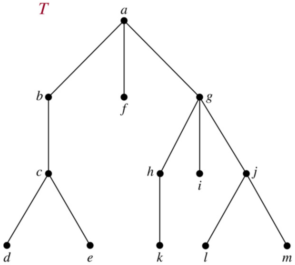


Tree terminology

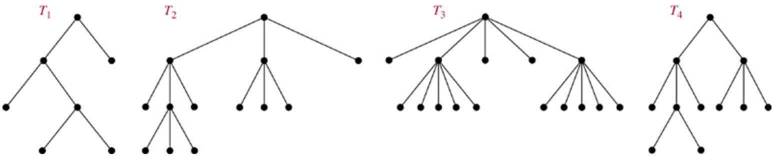
- Tree terminology has botanical (植物学的) and genealogical (家谱的) origins.
- Let T be a rooted tree.
- If v is a vertex in T other than the root, the parent of v is the unique vertex u such that there is a directed edge from u to v.
- When u is the parent of v, v is called a child of u.
- Vertices with the same parent are called siblings.
- The ancestors of a vertex v other than the root are all the vertices on the unique path from the root to v, excluding v itself and including the root.
- The descendants of a vertex v are those vertices with v as an ancestor.

Tree terminology

- A vertex with no children is called a leaf.
- A vertex with children is called an internal vertex.
- The subtree with a vertex v as its root is the subgraph of the tree consisting of v and its descendants and all edges incident to these descendants.
- Example: parent of c, children of g, siblings of h, ancestors of e, descendants of b, all internal vertices, all leaves, the subtree rooted at g

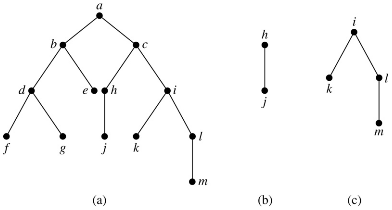


- Definition: A rooted tree is called an m-ary tree if every internal vertex has no more than m children. The tree is called a full m-ary tree if every internal vertex has exactly m children. An m-ary tree with $m = 2$ is called a binary tree.
- Example: Are the following trees full m-ary tree for some m?

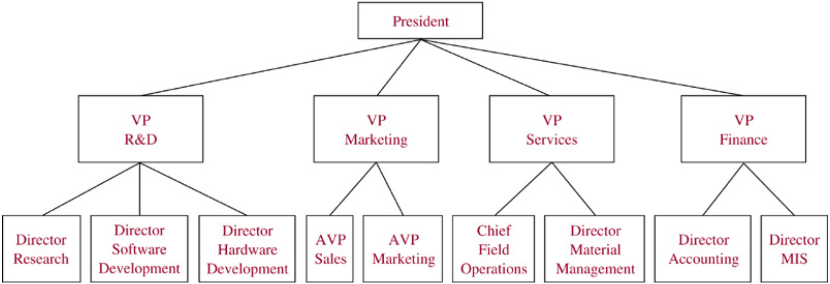


Ordered rooted trees (有序根树)

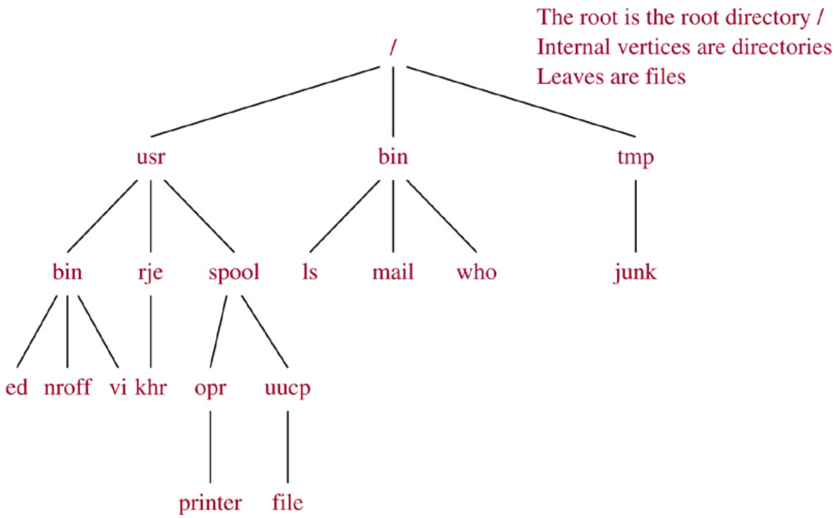
- An ordered rooted tree is a rooted tree where the children of each internal vertex are ordered.
- Ordered rooted trees are drawn so that the children of each internal vertex are shown in order from left to right.
- In an ordered binary tree, if an internal vertex has two children, the first child is called the left child and the second child is called the right child.
- The tree rooted at the left (resp. right) child of a vertex is called the left (resp. right) subtree of the vertex.



Trees as models: Representing organizations



Trees as models: Computer file systems



Properties of trees

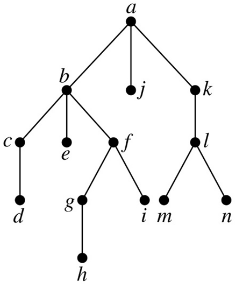
- Theorem 2: A tree with n vertices has $n - 1$ edges.
- Theorem 3: A full m-ary tree with i internal vertices contains $n = mi + 1$ vertices.
- Theorem 4: A full m-ary tree with
  - n vertices has $i = (n - 1)/m$ internal vertices and $l = [(m - 1)n + 1]/m$ leaves,
  - i internal vertices has $n = mi + 1$ vertices and $l = (m - 1)i + 1$ leaves,
  - l leaves has $n = (ml - 1)/(m - 1)$ vertices and $i = (l - 1)/(m - 1)$ internal vertices.

An example

- Someone starts a chain letter.
- Each person who receives the letter is asked to send it on to 4 other people.
- Some people do this, but others do not send out any letters.
- The chain letter ends after there have been 100 people who read it but did not send it out.
- How many people have seen the letter?
- How many people sent out the letter?

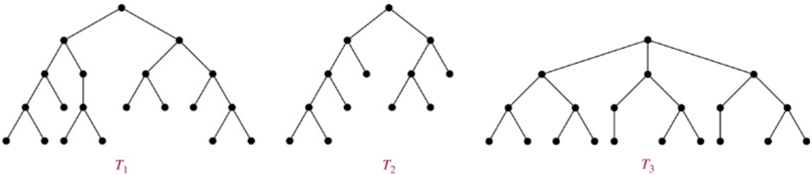
Level of a vertex and height of a tree

- The level of a vertex v in a rooted tree is the length of the unique path from the root to v.
- The height of a rooted tree is the maximum of the levels of vertices. That is, the height of a rooted tree is the length of the longest path from the root to any vertex.
- Example: the level of each vertex, the height of the tree



Balanced trees (平衡树)

- A rooted m-ary tree of height h is balanced if all leaves are at levels h or $h - 1$.
- Example: which trees are balanced?



Properties of trees

- Theorem: There are at most m^h leaves in an m-ary tree of height h.
- Corollary: If an m-ary tree of height h has l leaves, then $h \geq \lceil \log_m l \rceil$. If the m-ary tree is full and balanced, then $h = \lceil \log_m l \rceil$.

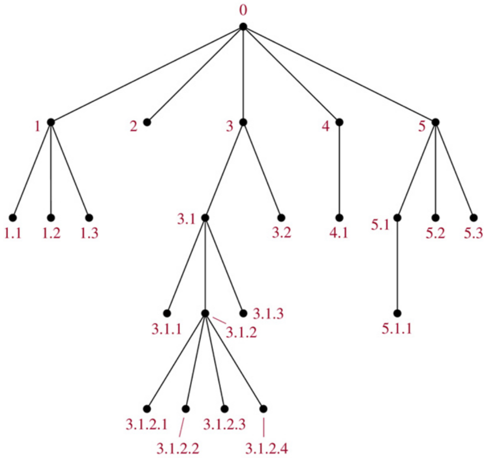
10.3: Tree travesal

- Ordered rooted trees are often used to store information.
- We will discuss several important algorithms for visiting each vertex of an ordered rooted tree.

Universal address systems (通用地址系统)

- We label all the vertices of an ordered rooted tree as follows:
  - Label the root with the integer 0. Then label its k children (at level 1) from left to right with $1, 2, \ldots, k$.
  - For each vertex v with label A, label its k_v children, as they are drawn from left to right, with $A.1, A.2, \ldots, A.k_v$.
- The labeling is called the universal address system of the ordered rooted tree.
- Then a vertex v at level n, is labeled $x_1.x_2.\ldots.x_n$, where the unique path from the root to v goes through the x_1st vertex at level 1, the x_2nd vertex at level 2, and so on.
- We can totally order the vertices using the lexicographic ordering of their labels.

An example

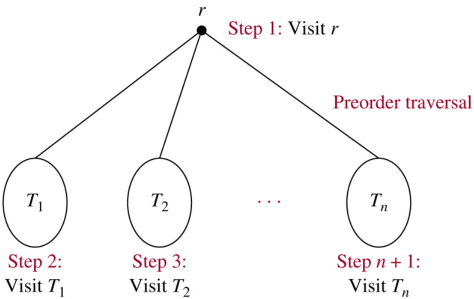


$0 < 1 < 1.1 < 1.2 < 1.3 < 2 < 3 < 3.1 < 3.1.1 < 3.1.2 < 3.1.2.1 < 3.1.2.2 < 3.1.2.3 < 3.1.2.4 < 3.1.3 < 3.2 < 4 < 4.1 < 5 < 5.1 < 5.1.1 < 5.2 < 5.3$

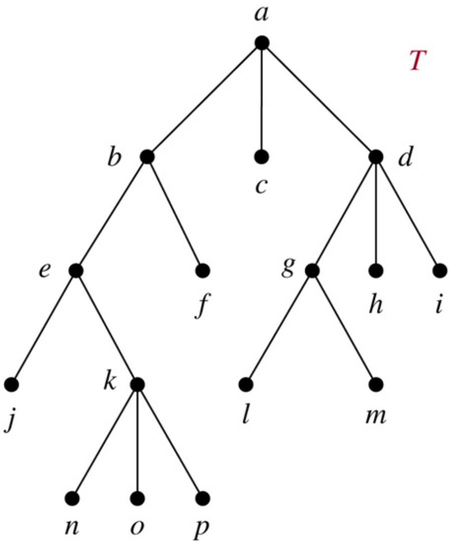
- Ordered rooted trees are often used to store information.
- Traversal (遍历) algorithms are procedures for systematically visiting every vertex of an ordered rooted tree.
- Three commonly used algorithms: preorder (前序) traversal, inorder (中序) traversal, and postorder (后序) traversal

Preorder traversal

- 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 $T_1, T_2, \ldots, T_n$ are the subtrees at r from left to right.
- 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.

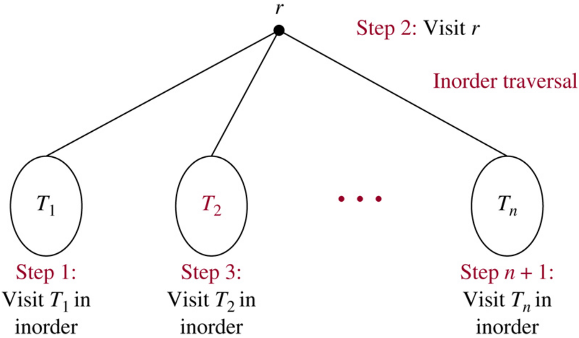


An example



Inorder traversal

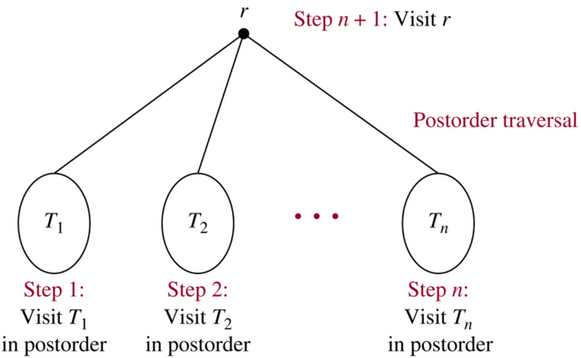
Definition: ... The inorder traversal begins by traversing T_1 in inorder, then visiting r. It continues by traversing T_2 in inorder, and so on, until T_n is traversed in inorder.



Example:

Postorder traversal

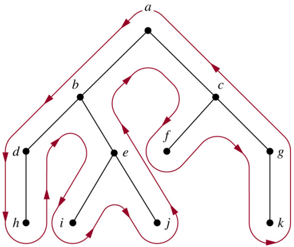
Definition: ... The postorder traversal begins by traversing T_1 in postorder, then T_2 in postorder, $\ldots$, then T_n in postorder, and ends by visiting r.



Example:

A shortcut for traversing an ordered rooted tree

- First draw a curve around the ordered rooted tree starting at the root, moving along the edges.
- Preorder: list each vertex the first time the curve passes it
- Inorder: list a leaf when the curve passes it and list each internal vertex the second time the curve passes it
- Postorder: list a vertex the last time the curve passes it

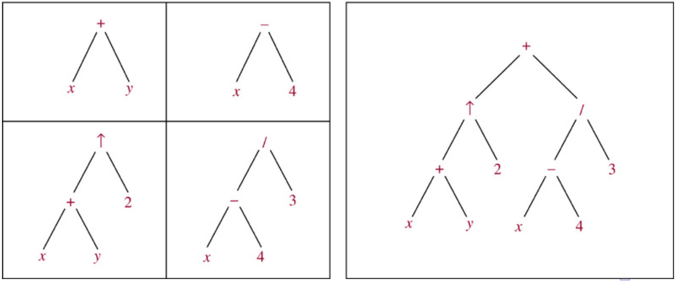


A recursive algorithm for inorder traversal

procedure inorder(T: ordered rooted tree)
r := root of T
if r is a leaf then list r
else
    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
        $T(c)$:= subtree with c as its root
        $inorder(T(c))$

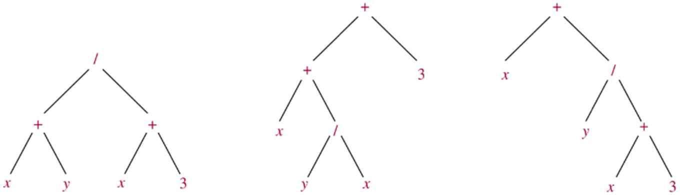
Binary tree representation of expressions

- We can represent expressions using ordered rooted trees.
- The internal vertices represent operations.
- The leaves represent the variables or numbers.
- Each binary operation operates on its left and right subtrees, each unary operation operates on its single subtree.
- Example: $((x + y) \uparrow 2) + ((x - 4)/3)$



Infix (中缀) notation

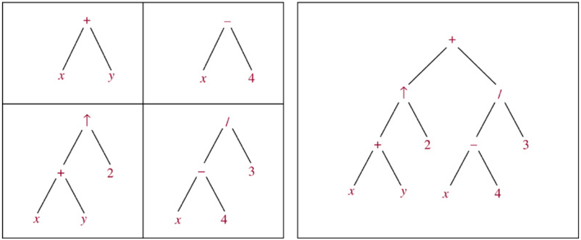
- When there are only binary operations, an inorder traversal of the binary tree representing an expression produces the original expression without parentheses.
- Example: inorder traversals of the following binary trees all lead to $x + y/x + 3$



- To avoid ambiguity, we should include parentheses for operations.
- The fully parenthesized expression is said to be in infix form.

Prefix (前缀) notation

- We obtain the prefix form of an expression when we traverse its rooted tree in preorder.
- Expression written in prefix form is said to be in Polish notation.
- An expression in prefix form is unambiguous, so no parentheses are needed.
- Example: the prefix form for $((x + y) \uparrow 2) + ((x - 4)/3)$



Evaluating an expression in prefix form

- In the prefix form, a binary operator (操作符) precedes its two operands (操作数).
- We can evaluate an expression in prefix form by working from right to left.
- When we see an operator, we perform the operation with the two operands immediately to the right of the operator.
- Whenever an operation is performed, we consider the result a new operand.
- Example: evaluating $+ - * 235 / \uparrow 234$

Postfix (后缀) notation

- We obtain the postfix form of an expression when we traverse its rooted tree in postorder.
- Expression written in postfix form is said to be in reverse Polish notation.
- An expression in postfix form is unambiguous, so no parentheses are needed.
- We can evaluate an expression in postfix form similarly as for prefix form, except that we work from left to right.