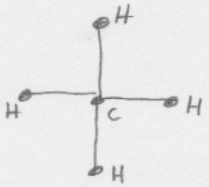
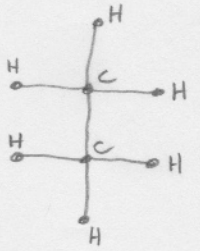
Notes

TREE – CONNECTED GRAPH WITH NO SIMPLE CIRCUITS (undirected)

EXAMPLES

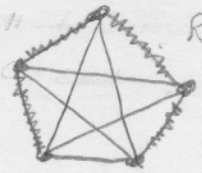


methane gas CH_4



ethane C_2H_6

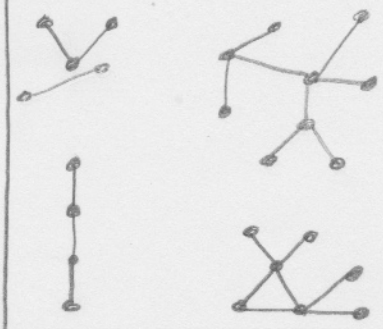
H = hydrogen, valence 1
C = carbon, valence 4



telephone network
5 towns

mm tree - most efficient way to connect all towns

Which are trees?



PROPERTIES OF TREES

↑ examples above

(1) G is a tree ("P") $\iff$ between every pair of vertices ("Q") there is exactly one simple path.

Proof.

$P \Rightarrow Q$ Let G be a tree and U, V be a pair of vertices in G.

- G has at least one U-V path ∵ G is connected (WHY?)
- ∴ G has at least one simple U-V path ∵ Every path contains a simple path.
- G has at most one simple U-V path ∵ "Proof by contradiction" below

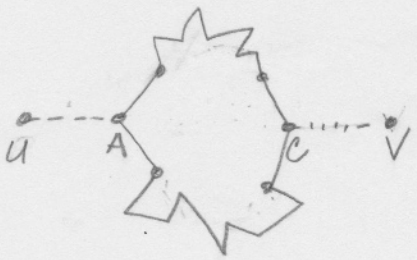
If $\neg Q \Rightarrow \neg P$ (not Q $\Rightarrow$ not P) then $P \Rightarrow Q$.

Statement $\neg Q$ = Suppose there were 2 simple U-V paths

* Start at U and move towards V.
* Let A be the first vertex you come to where the paths part – A could be U.
* Follow both paths until you come to the first vertex C where the paths meet again – C could be V.
* Then there is a cycle in G

$\neg P$ * G is not a tree

∴ $\neg Q \Rightarrow \neg P$ ∴ $P \Rightarrow Q$.



$Q \Rightarrow P$ We will also show $\neg P \Rightarrow \neg Q$.

$\neg P$ = G is not a tree = G is not connected or G contains a simple circuit $V_1 V_2 \cdots V_n V_1$

↓ There are two vertices with no path between them $\Rightarrow \neg Q$

↓ There are two different simple paths between V_1 and V_2. $V_1 V_2$ and $V_1 V_n \cdots V_2$

$\Rightarrow \neg Q$

$\therefore Q \Rightarrow P$.

Q.E.D.

(2) Every tree G with more than one vertex has at least 2 vertices of degree 1.

examples on p.1.

Proof

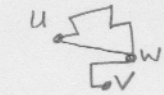
- There is at least one simple path in G.
- Of all simple paths there will be at least one path P with the maximal number of edges. Let U and V be the first and last vertices of P.

with $\therefore$ G has at least 2 vertices

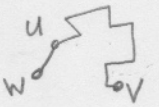
$\therefore$ there are finitely many simple paths in a finite graph, so there must be a longest one.

- $\deg U = 1$

$\therefore$
- if $\deg U > 1$ there would be an edge from U to another vertex W
- W is not in P $\therefore$ if W in P then P would have a simple circuit, which would also be a simple circuit in $G - C$!



- $WU \cdots V$ would be a simple path in G that is longer than P. C!



- $\deg V = 1$ — similar argument

Q.E.D.

(3) **A tree with n vertices has exactly $n-1$ edges**

Proof by induction on the number n of vertices.

Let $S(n)$ be a statement concerning n.
If (1) $S(1)$ is true
and (2) $S(k) \Rightarrow S(k+1)$ for every $k \geq 1$
Then (3) $S(n)$ is true for every n.

Step 1 $n=1$: $S(1)$ says "a tree with 1 vertex has no edges". TRUE

Step 2 Suppose $S(k)$ is TRUE. $S(k)$ says "a tree with k vertices has $k-1$ edges"

We will show $S(k+1)$ is TRUE

$S(k+1)$ says "every tree with $k+1$ vertices has k edges."

- Let G be a tree with $k+1$ vertices.
- Choose a vertex V in G with degree 1 ∵ (there are at least 2 to choose from by Theorem (2) above.)
- Remove V and the edge incident with it from V.
- The result is a tree G^* with k edges
- G^* has $k-1$ edges by $S(k)$.
- ∴ G has k edges.

∴ $S(k) \Rightarrow S(k+1)$ for any $k \geq 1$.

Step 3 $S(n)$ is true for every n. = any tree with n vertices has exactly $n-1$ edges.

---

(4) **(a) Removing an edge from a tree (keeping the vertices) disconnects the tree.**
(b) Adding an edge to a tree (without adding more vertices) creates a simple circuit

Proof "G is not a tree" = "either G is disconnected or G has a simple circuit."

(a) Let G be the result of removing an edge from a tree with k vertices.

Then
- G has k vertices and $k-2$ edges
- G is not a tree
- Removing an edge cannot create a simple circuit, so G is disconnected.

(b) Let H be the result of adding an edge to a tree with k vertices

- H has k vertices and k edges
- H is not a tree
- Adding an edge cannot disconnect a graph, so H has a simple circuit.

(5) ALTOGETHER.

All of the following statements are equivalent.

= "each statement implies all the others and each statement is implied by all the others.

(a) T is a tree

(b) T is connected and the number of vertices is one more than the number of edges.

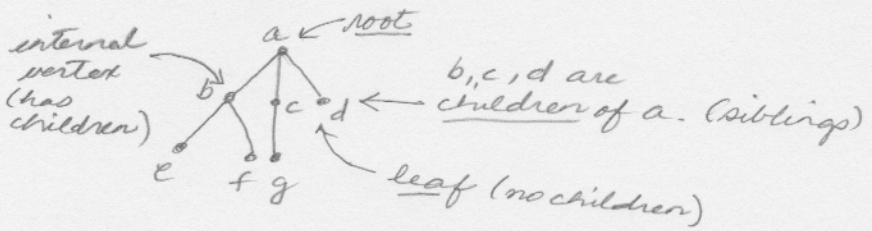
(c) T has no simple circuits and the number of vertices is one more than the number of edges.

(d) There is exactly one simple path between each pair of vertices.

(e) T is connected and removing any edge disconnects T.

(f) T has no simple circuits, and adding any edge creates a simple circuit.

A rooted tree



m-ary tree - each vertex has no more than m children

full m-ary tree - every vertex that is not a leaf has exactly m children

binary tree - m-ary tree with m=2.

A rooted tree can be thought of as a directed graph with edges to children directed away from the parent.

Internal vertex - a vertex with at least one child.

Leaf - a vertex with no children

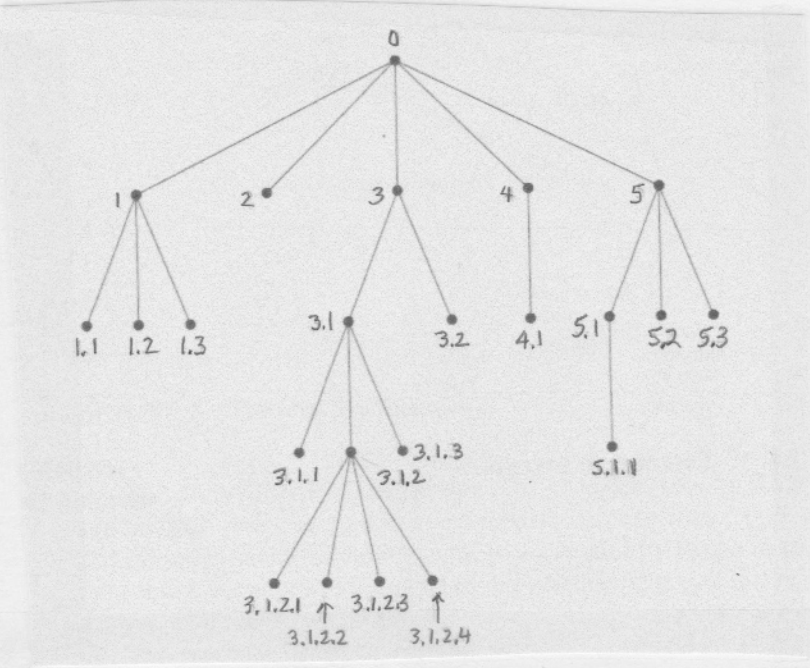
Level of a vertex - length of path from root to the vertex

Height of a rooted tree - maximal length of path from root to any vertex.

Balanced tree - if h is the height of a rooted tree, then all leaves are at levels h or h-1.

Tree Traversal

Universal address system for a rooted tree.



Total ordering in lexicographic (dictionary) order.

Each subtree at 0 gets counted completely first from left to right.

$0 < 1 < 1.1 < 1.2 < 1.3$

< 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

Write out as one long ordering:

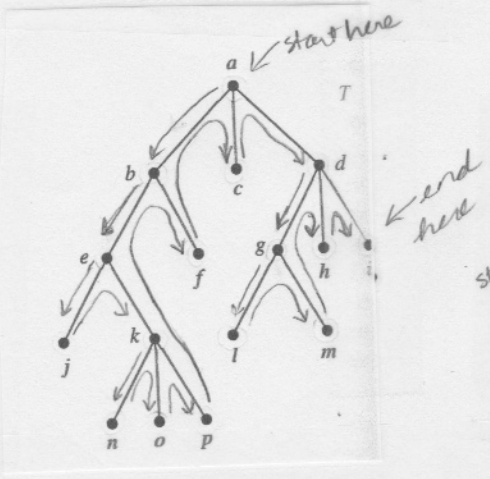
$0 < 1 < 1.1 < 1.2 < 1.3 < 2 < 3 < 3.1 < 3.1.1 < 3.1.2 < 3.1.2.1 < 3.1.2.2 < 3.1.2.3 < 3.1.2.4 <$

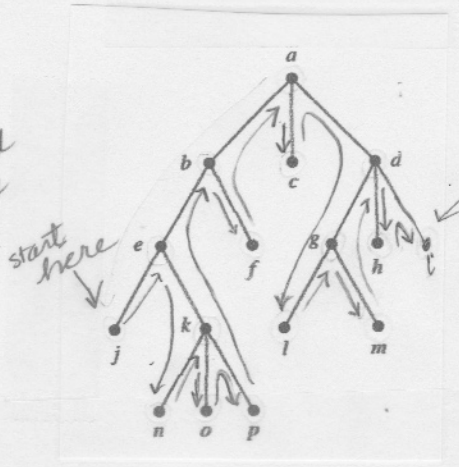
$< 3.1.3 < 3.2 < 4 < 4.1 < 5 < 5.1 < 5.1.1 < 5.2 < 5.3$

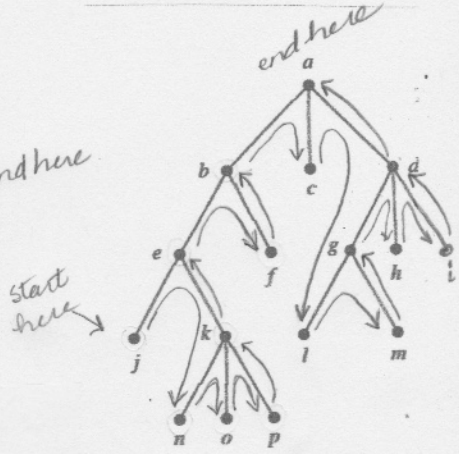
Using binary trees to do arithmetic

PREORDER TRAVERSAL	INORDER TRAVERSAL	POSTORDER TRAVERSAL
PREFIX NOTATION Polish Notation	INFIX NOTATION Our usual notation	POST FIX NOTATION Reverse Polish Notation
Puts the operation first, then the two numbers +34 parentheses not needed	3+4 needs parentheses	Puts the two numbers first and then the operation 34+ parentheses not needed

Work from left to right.







Preorder traversal

a b e j k n o p f c d g l m h i

Parents before children

Inorder traversal

j e n k o p b f a c l g m d h i

Parents between children

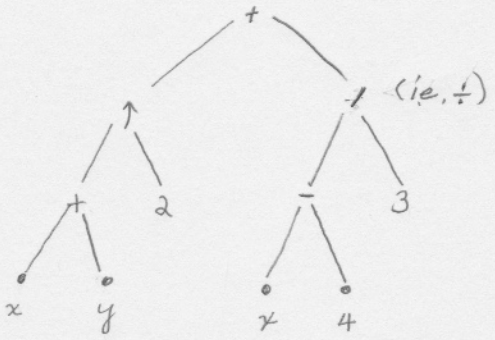
Postorder traversal

j n o p k e f b c l m g h i d

Parents after children

Rooted trees representing arithmetic

$((x+y)\uparrow 2) + ((x-4)/3)$ parentheses essential



Preorder traversal
⇒ Prefix Notation

Operation before numbers

$+ \uparrow + x\ y\ 2\ /\ -\ x\ 4\ 3$

no parentheses needed

Inorder traversal
⇒ Infix notation

Operations between numbers

$((x+y)\uparrow 2) + ((x-4)/3)$

parentheses needed

Postorder traversal
⇒ postfix notation

Operations after numbers

$x\ y + 2 \uparrow x\ 4 - 3 / +$

no parentheses needed.

How to evaluate

$+ \uparrow + x\ y\ 2\ /\ -\ x\ 4\ 3$

$\uparrow (x+y)\ 2 \quad /\ (x-4)\ 3$

$+ \quad (x+y)^2 \quad (x+y)/3$

$(x+y)^2 + ((x+y)/3)$

As usual using PEMDAS

$x\ y + 2 \uparrow x\ 4 - 3 / +$

$= (x+y) \quad = (x-4)$

$(x+y)\ 2 \uparrow \quad (x-4)\ 3 /$

$= (x+y)^2 \quad = ((x-4)/3)$

$(x+y)^2 \ ((x-4)/3) +$

$= (x+y)^2 + ((x-4)/3)$.

Examples on pages 780 – 781.