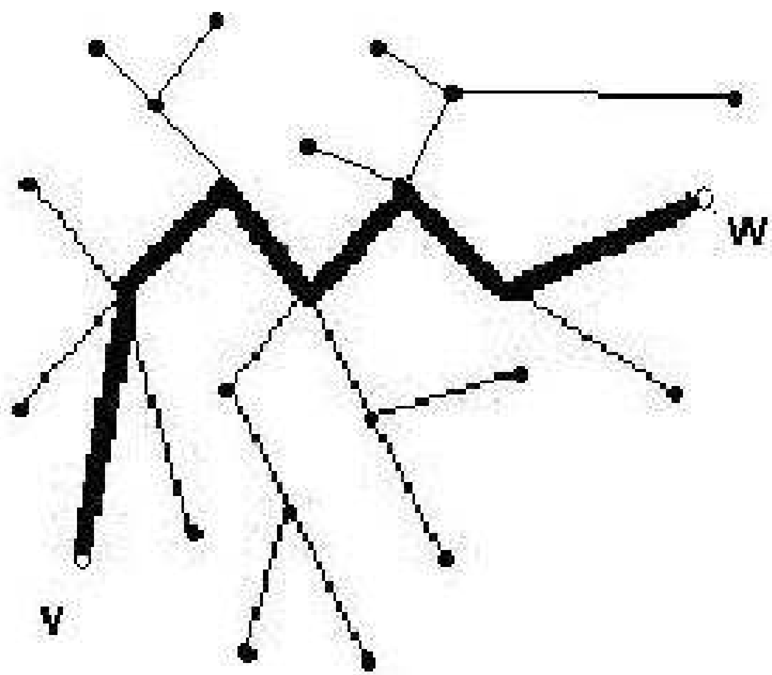
Data Structure

TREES

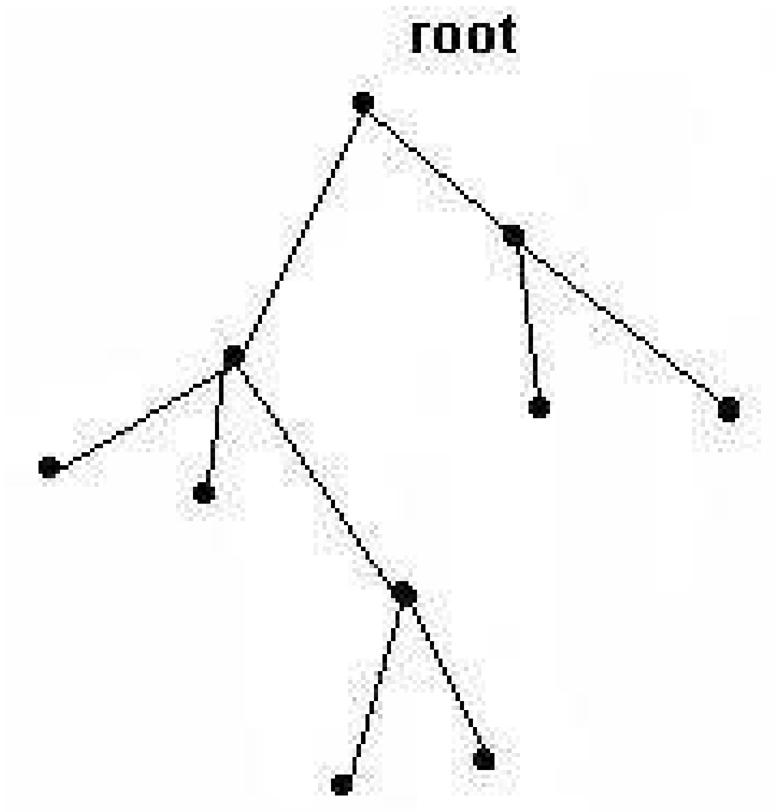
Introduction



A *(free) tree* T is

- A simple graph such that for every pair of vertices v and w
- there is a unique path from v to w

Rooted tree



A *rooted tree* is a tree where one of its vertices is designated the *root*

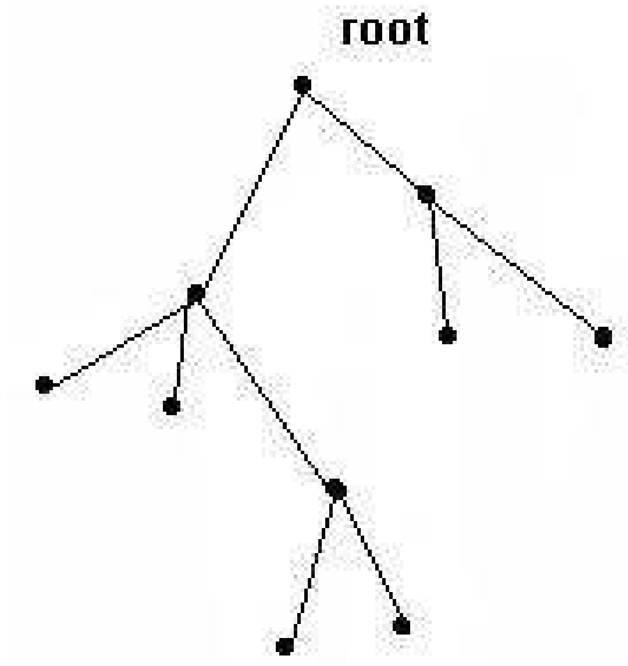
Level of a vertex and tree height

Let T be a rooted tree:

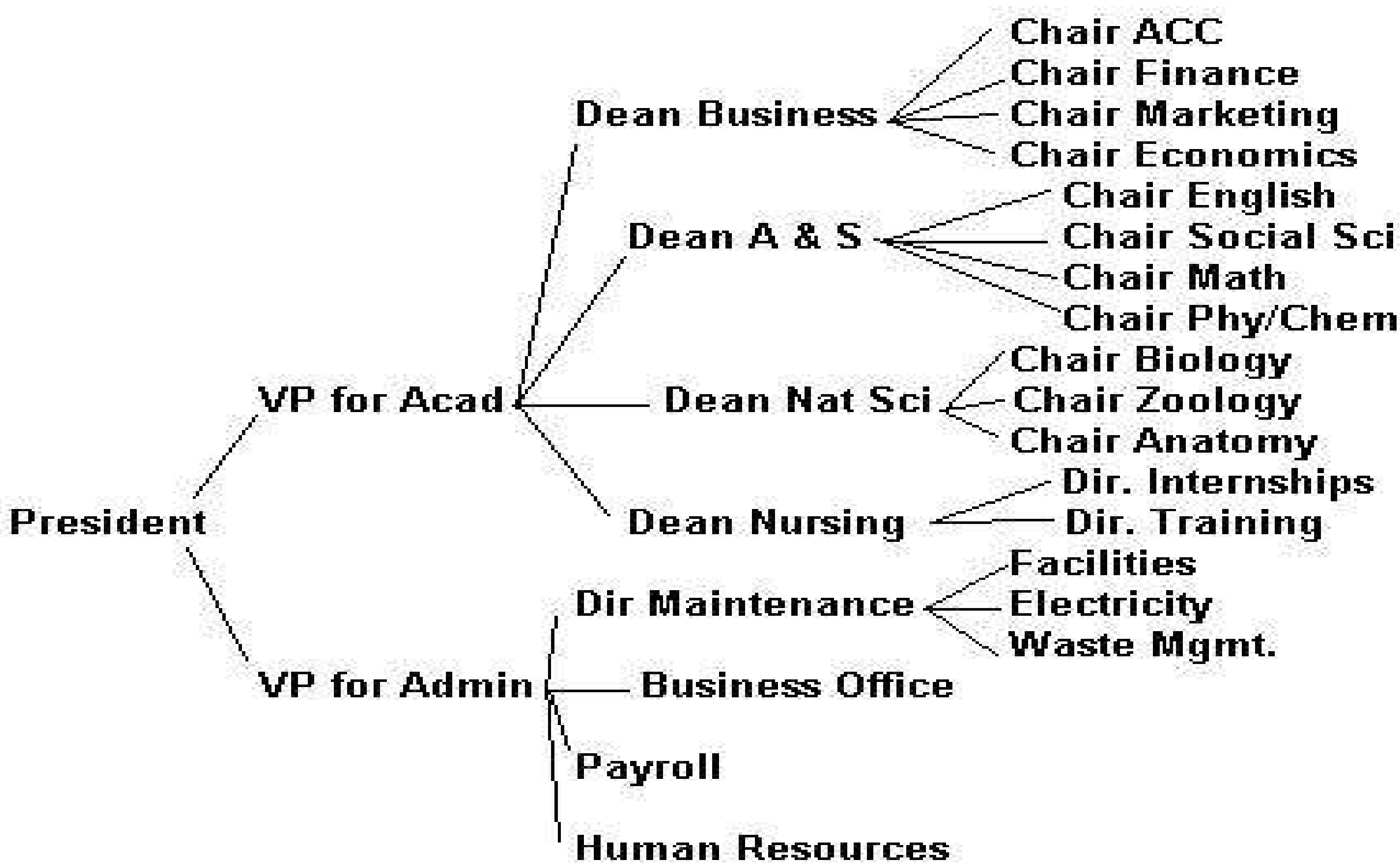
- The *level* *l(v)* of a vertex **v** is the length of the simple path from **v** to the root of the tree
- The *height* *h* of a rooted tree T is the maximum of all level numbers of its vertices:

$$h = \max_{v \in V(T)} \{ l(v) \}$$

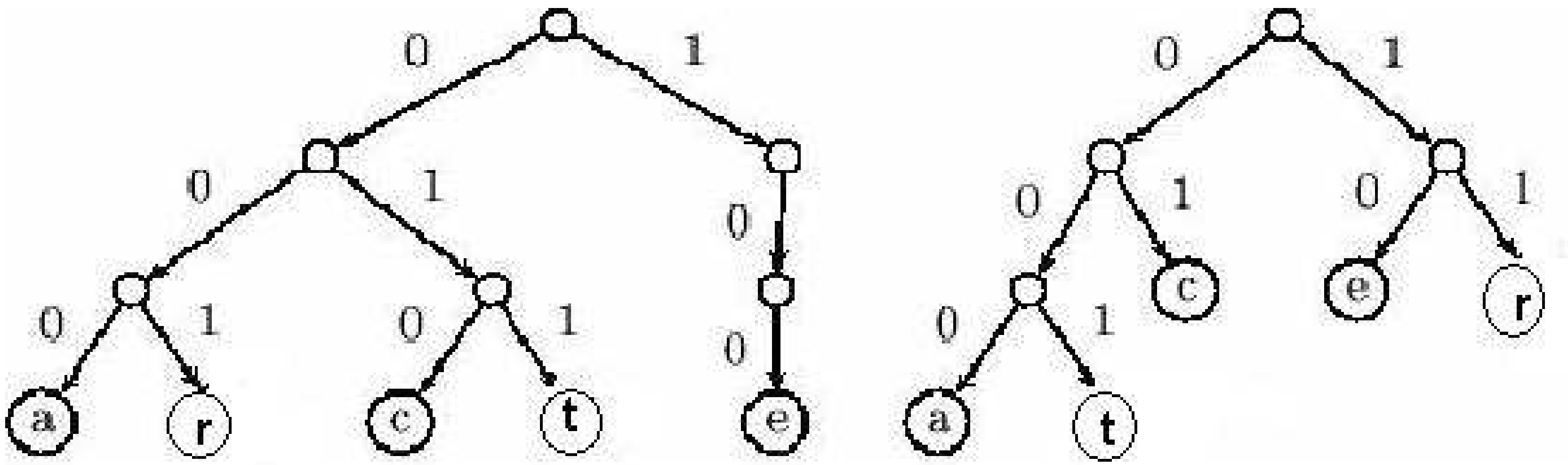
- Example:
  - the tree on the right has height 3



Organizational charts



Huffman codes



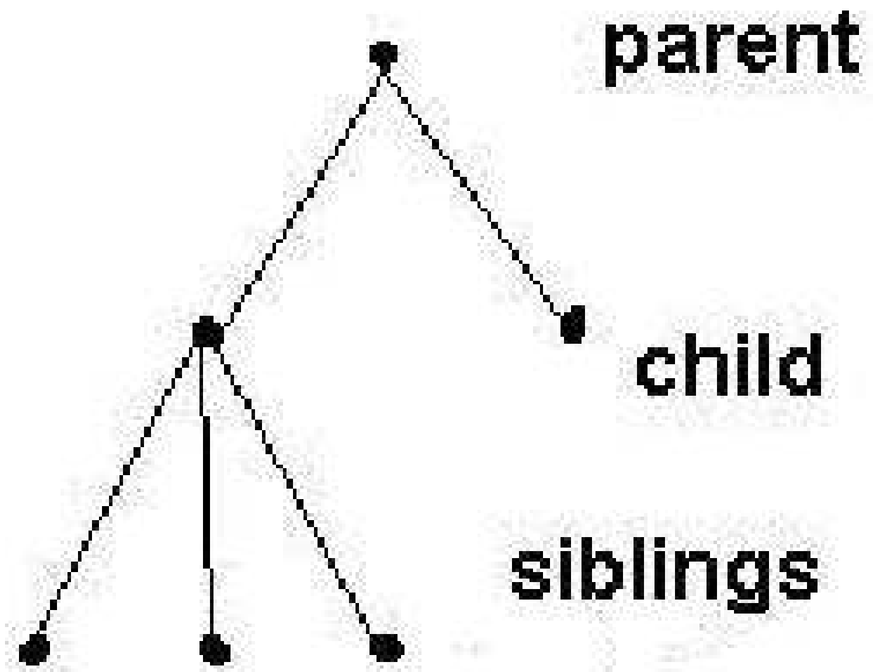
- On the left tree the word **rate** is encoded

  001 000 011 100
- On the right tree, the same word **rate** is encoded

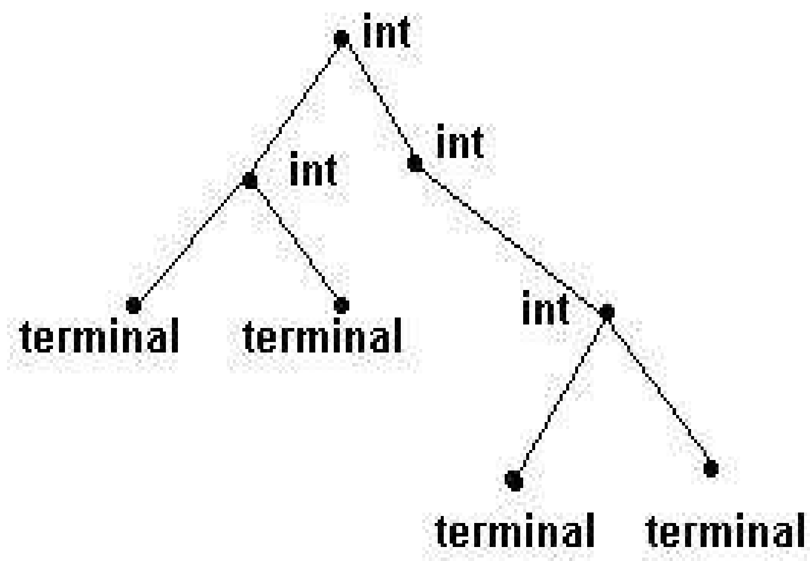
  11 000 001 10

Terminology

- Parent
- Ancestor
- Child
- Descendant
- Siblings
- Terminal vertices
- Internal vertices
- Subtrees



Internal and external vertices

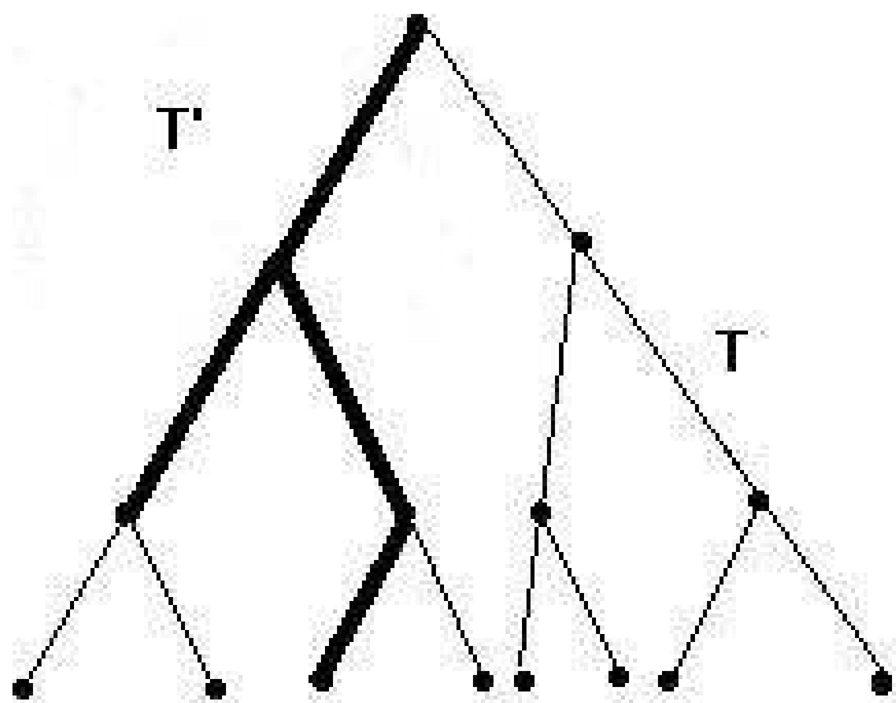


- An *internal vertex* is a vertex that has at least one child
- A *terminal* vertex is a vertex that has no children
- The tree in the example has 4 internal vertices and 4 terminal vertices

Subtrees

A subtree of a tree T is a tree T' such that

- $V(T') \subseteq V(T)$ and
- $E(T') \subseteq E(T)$



Characterization of trees

<u>Theorem</u>

If T is a graph with n vertices, the following are equivalent:

a) T is a tree

b) T is connected and acyclic

- (“acyclic” = having no cycles)

c) T is connected and has n-1 edges

d) T is acyclic and has n-1 edges

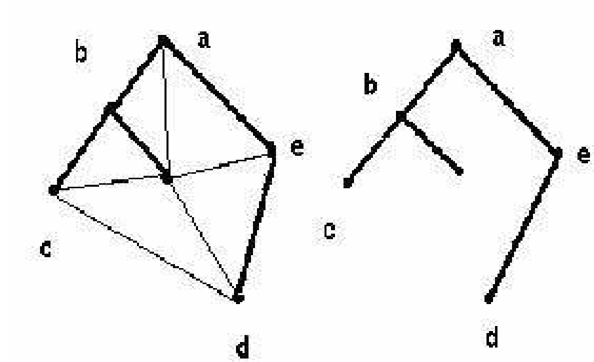
Spanning trees

Given a graph G, a tree T is a *spanning tree* of G if:

- T is a subgraph of G

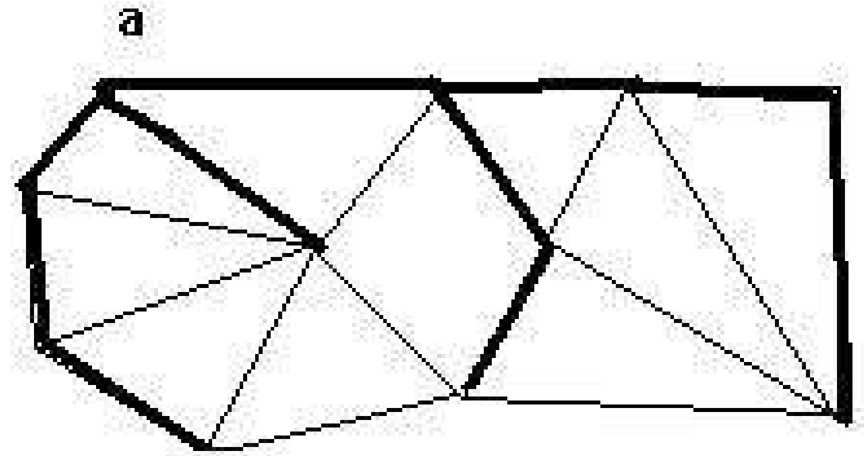
  and

- T contains all the vertices of G

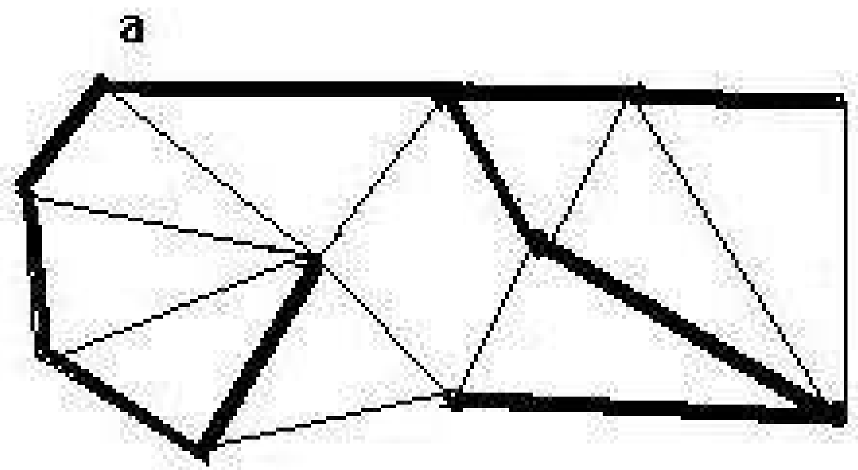


Spanning tree search

- Breadth-first search method



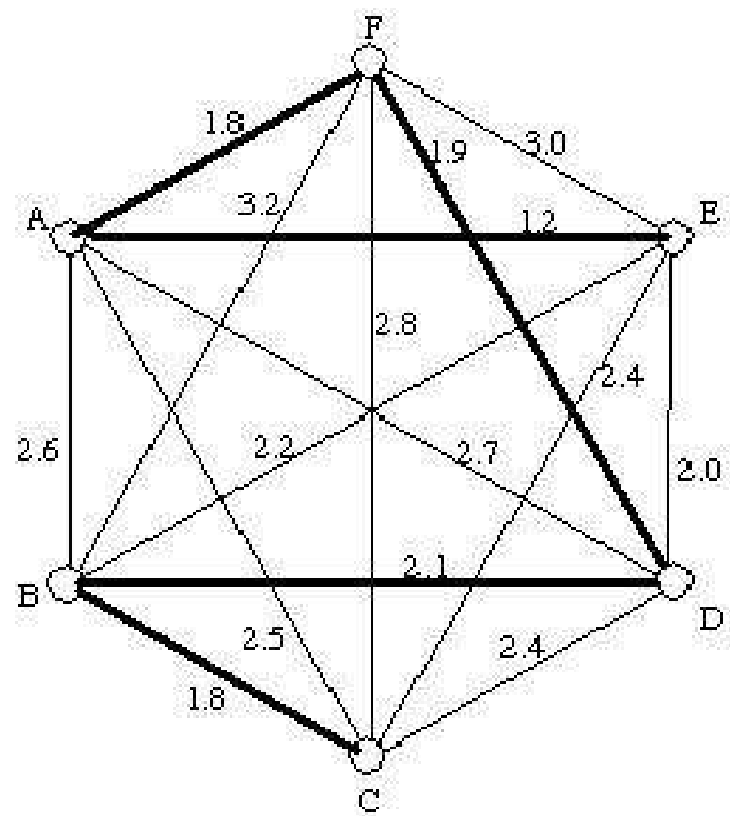
- Depth-first search method (backtracking)



Minimal spanning trees

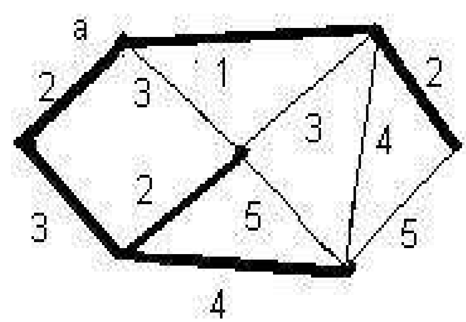
Given a weighted graph G, a *minimum spanning tree* is

- a spanning tree of G
- that has minimum "weight"



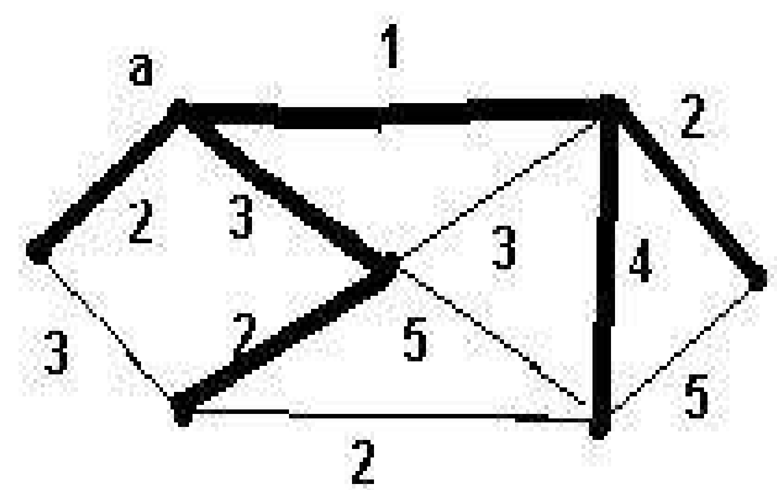
1. Prim’s algorithm

- Step 0: Pick any vertex as a starting vertex (call it *a*). T = {a}.
- Step 1: Find the edge with smallest weight incident to *a*. Add it to T Also include in T the next vertex and call it *b*.
- Step 2: Find the edge of smallest weight incident to either *a* or *b.* Include in T that edge and the next incident vertex. Call that vertex *c*.
- Step 3: Repeat Step 2, choosing the edge of smallest weight that does not form a cycle until all vertices are in T. The resulting subgraph T is a minimum spanning tree.



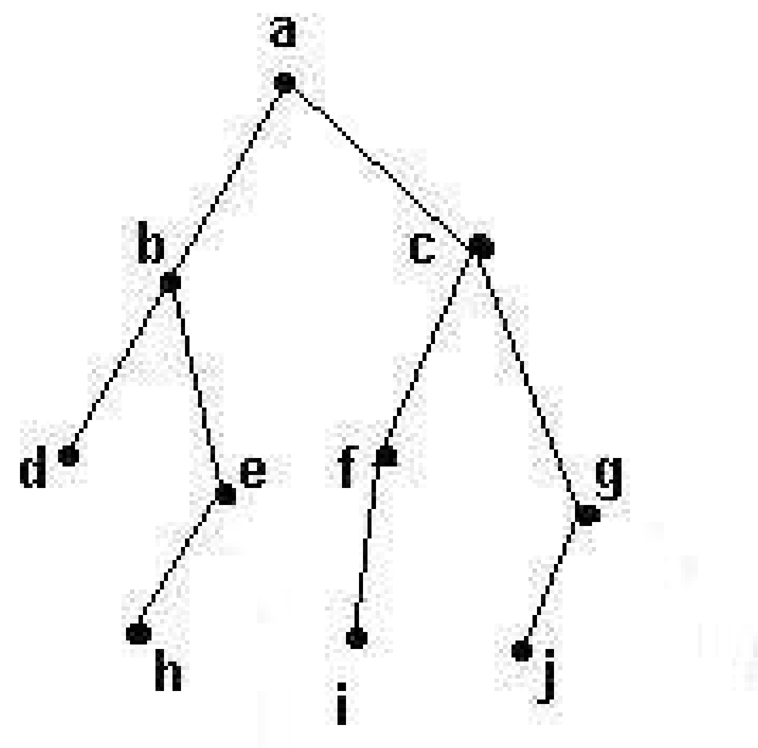
2. Kruskal's algorithm

- Step 1: Find the edge in the graph with smallest weight (if there is more than one, pick one at random). Mark it with any given color, say red.
- Step 2: Find the next edge in the graph with smallest weight that doesn't close a cycle. Color that edge and the next incident vertex.
- Step 3: Repeat Step 2 until you reach out to every vertex of the graph. The chosen edges form the desired minimum spanning tree.

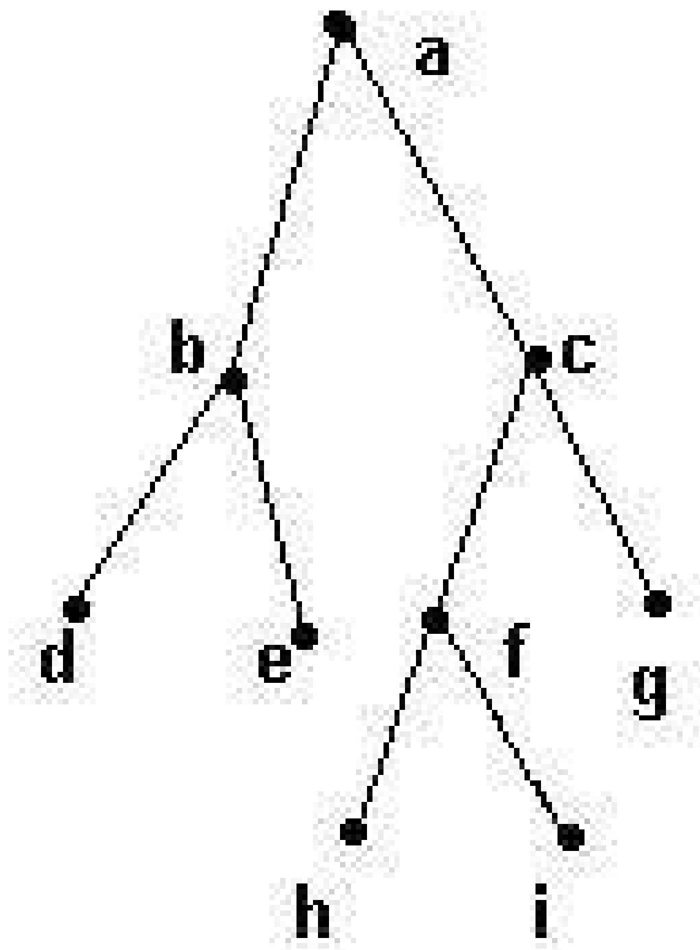


Binary trees

A *binary tree* is a tree where each vertex has zero, one or two children

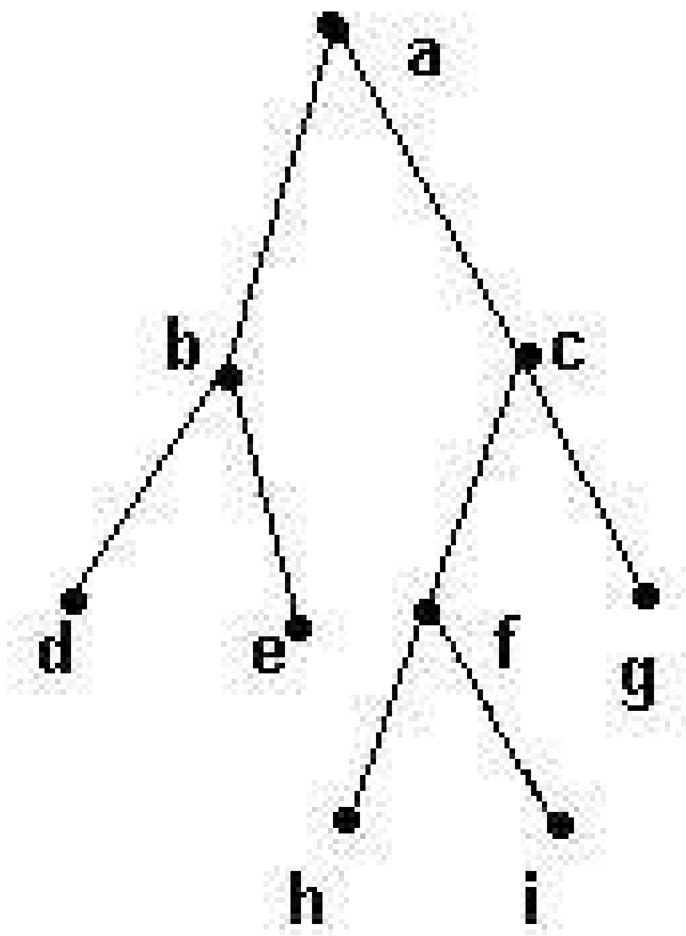


Full binary tree



A *full* binary tree is a binary tree in which each vertex has two or no children.

Full binary tree



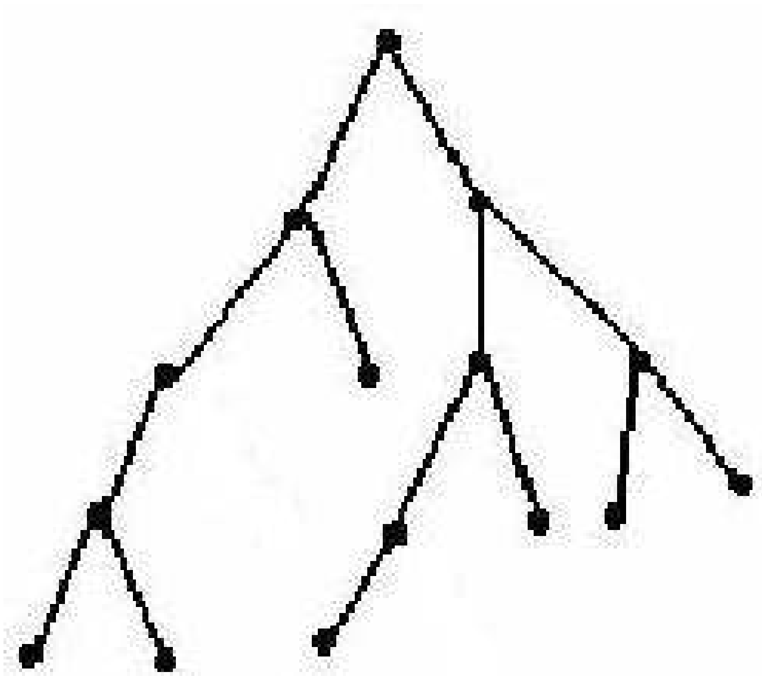
Theorem : If T is a *full binary tree* with k internal vertices, then

- T has k + 1 terminal vertices and
- the total number of vertices is 2k + 1.
  - Example: there are k = 4 internal vertices (a, b, c and f) and 5 terminal vertices (d, e, g, h and i) for a total of 9 vertices.

Height and terminal vertices

- Theorem : If a binary tree of height h has t terminal vertices, then $\lg t \leq h$, where lg is logarithm base 2.

  Equivalently, $t \leq 2^h$.
  - Example, $h = 4$ and $t = 7$. Then: $t = 7 < 16 = 2^4 = 2^h$

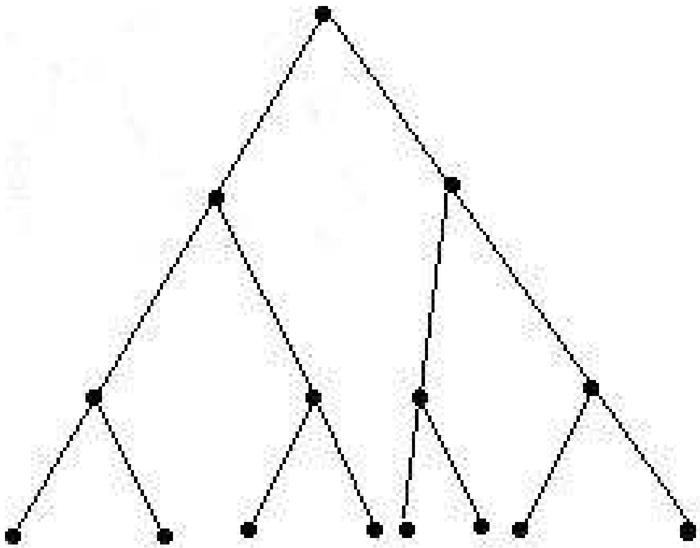


A case of equality

- If all t terminal vertices of a full binary tree T have the same level h = height of T, then

$$t = 2^h.$$

- Example:
  - The height is $h = 3$,
  - and the number of terminal vertices is $t = 8$
  - $t = 8 = 2^3 = 2^h$



Alphabetical order

Alphabetical or *lexicographic order* is the order of the dictionary:

a) start with an ordered set of symbols X = {a,b,c, …}. X can be infinite or finite.

b) Let $\alpha = x_1x_2\ldots x_m$ and $\beta = y_1y_2\ldots y_n$ be strings over X. Then define $\alpha < \beta$ if

- $x_1 < y_1$
- or if $x_j = y_j$ for all j, $1 \leq j \leq k$, for some k such that $1 \leq k \leq \min\{m,n\}$ and $x_{j+1} < y_{j+1}$
- or if $m \leq n$ and $x_j = y_j$ for all j, $1 \leq j \leq m$

Example of alphabetical order

- Let X = set of letters of the alphabet ordered according to precedence, i.e.

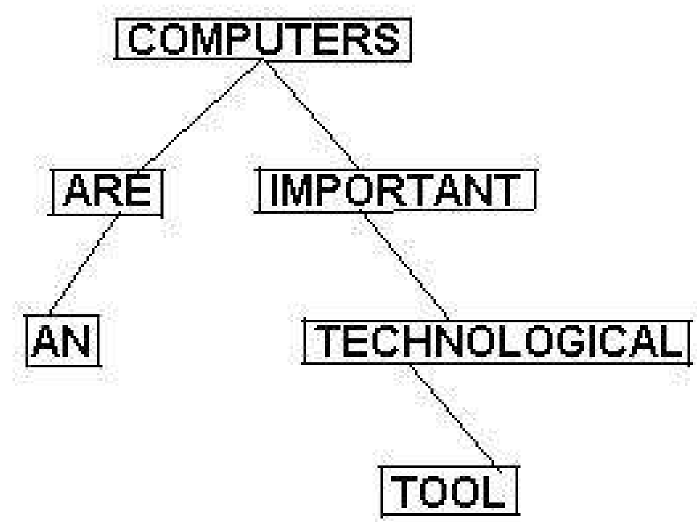
$$a < b < c < \ldots < x < y < z$$

- Let α = *arboreal* and β = *arbiter*.
- In this case,
  - $x_1 = y_1 = a$,
  - $x_2 = y_2 = r$
  - $x_3 = y_3 = b$.
- So, we go the fourth letter: $x_4 = o$ and $y_4 = i$.
- Since $i < o$ we have that $\beta < \alpha$.

Binary search trees

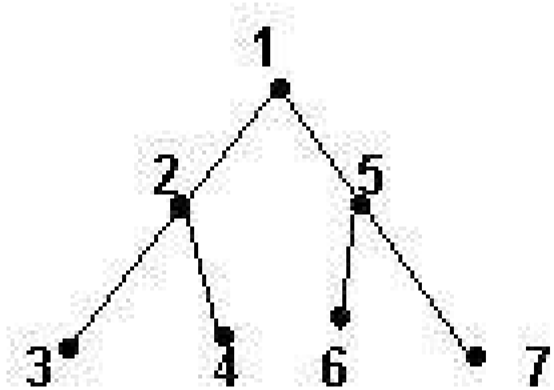
- Data are associated to each vertex
- Order data alphabetically, so that for each vertex v, data to the left of v are less than data in v
- and data to the right of v are greater than data in v

- Example: "Computers are an important technological tool"

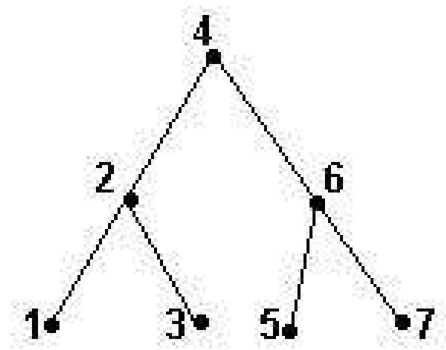


Tree Traversals

- 1: Pre-order traversal

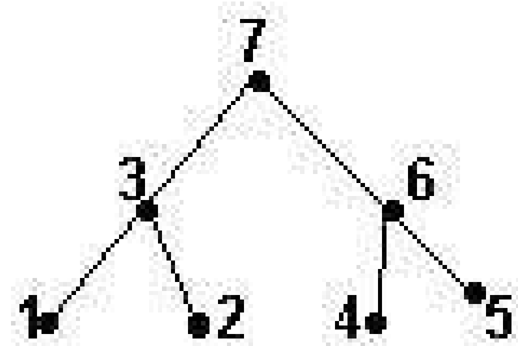


- 2: In-order traversal

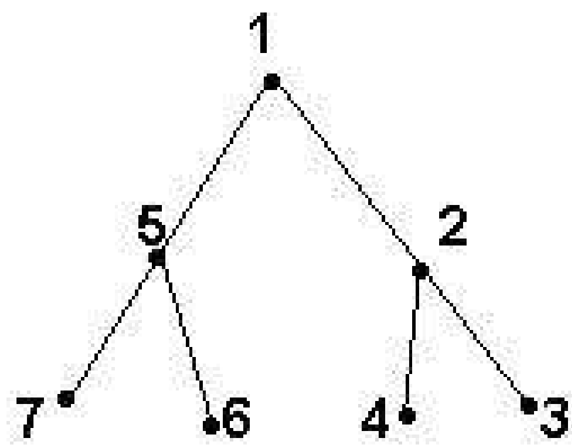


More on tree traversals

- 3: Post-order traversal



- 4: Reverse post-order traversal



Arithmetic expressions

- Standard: *infix* form

  $(A+B) * C - D/ E$

- Fully parenthesized form (in-order & parenthesis):

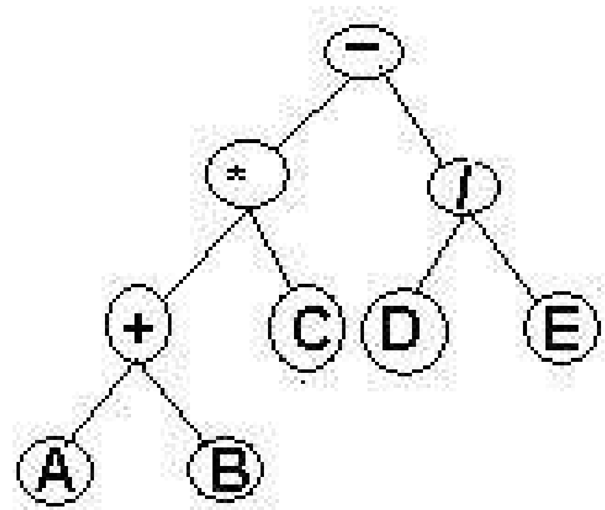
  $(((A + B) * C) - (D / E))$

- *Postfix* form (reverse Polish notation):

  A B + C $*$ D E / -

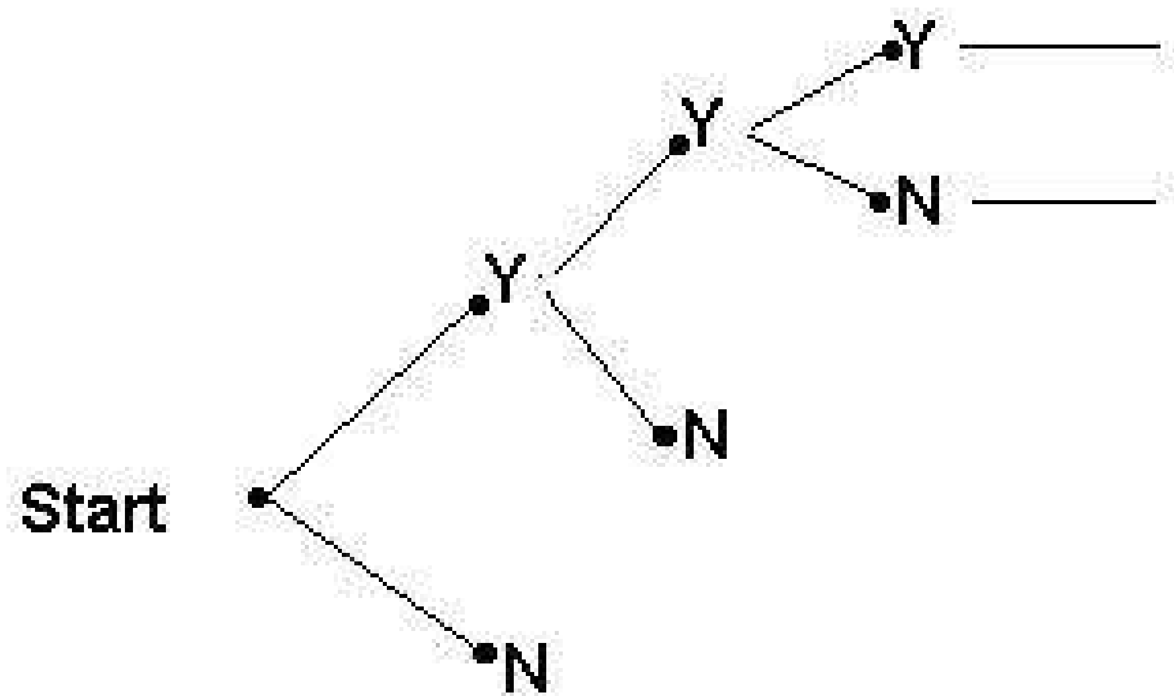
- *Prefix* form (Polish notation):

  - $*$ + A B C / D E



Decision trees

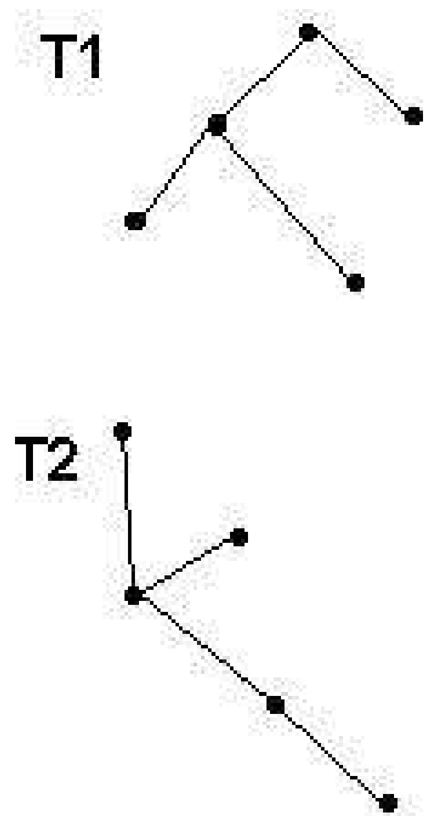
A *decision tree* is a binary tree containing an algorithm to decide which course of action to take.



Isomorphism of trees

Given two trees T_1 and T_2

- T_1 is *isomorphic* to T_2
- if we can find a one-to-one and onto function $f : T_1 \rightarrow T_2$
- that preserves the adjacency relation
  - i.e. if $v, w \in V(T_1)$ and $e = (v, w)$ is an edge in T_1, then $e' = (f(v), f(w))$ is an edge in T_2.



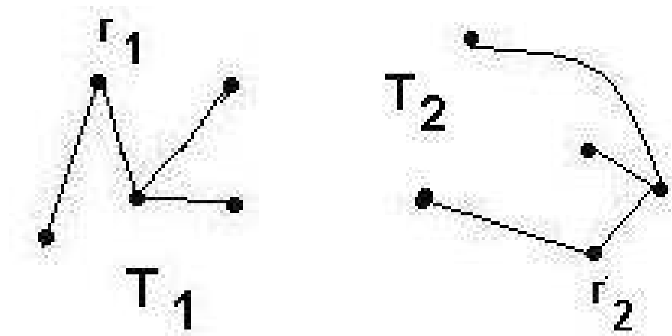
Isomorphism of rooted trees

Let T_1 and T_2 be rooted trees with roots r_1 and r_2, respectively. T_1 and T_2 are *isomorphic as rooted trees* if

- there is a one-to-one function $f: V(T_1) \rightarrow V(T_2)$ such that vertices v and w are adjacent in T_1 if and only if f(v) and f(w) are adjacent in T_2
- $f(r_1) = r_2$

  Example:

  - T_1 and T_2 are isomorphic as rooted trees



Isomorphism of binary trees

Let T_1 and T_2 be binary trees with roots r_1 and r_2, respectively. T_1 and T_2 are <u>isomorphic as binary trees</u> if

a) T_1 and T_2 are isomorphic <u>as rooted trees</u> through an isomorphism f, and

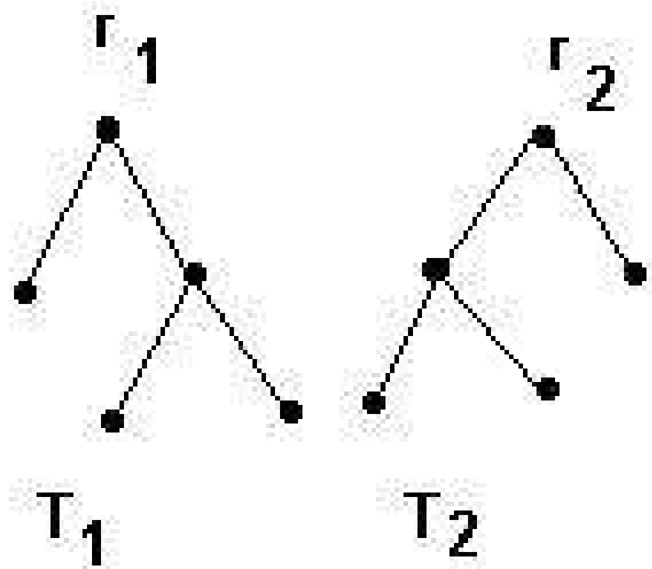
b) v is a left (right) child in T_1 if and only if f(v) is a left (right) child in T_2

- Note: This condition is more restrictive that isomorphism only as rooted trees. Left children must be mapped onto left children and right children must be mapped onto right children.

Binary tree isomorphism

Example: the following two trees are

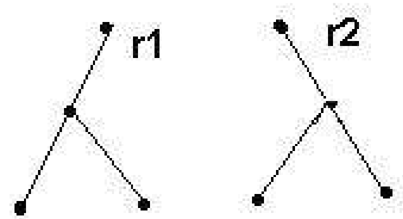
- isomorphic as rooted trees, but
- not isomorphic as binary trees



Summary of tree isomorphism

There are 3 kinds of tree isomorphism

- Isomorphism of trees
- Isomorphism of rooted trees
  - (root goes to root)
- Isomorphism of binary trees
  - (left children go to left children, right children go to right children)



Two binary trees isomorphic as rooted trees, not as binary trees

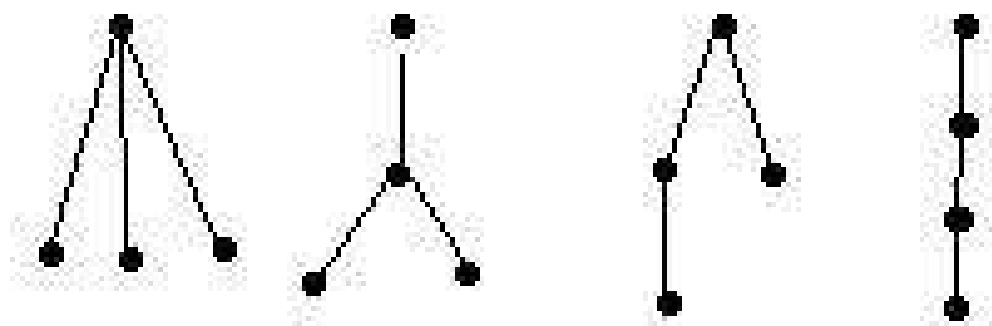
Non-isomorphism of trees

- Many times it may be easier to determine when two trees are <u>not</u> isomorphic rather than to show their isomorphism.
- A tree isomorphism must respect certain properties, such as
  - the number of vertices
  - the number of edges
  - the degrees of corresponding vertices
  - roots must go to roots
  - position of children, etc.

Non-isomorphism of rooted trees

Theorem: There are four non-isomorphic rooted trees with four vertices.

- The root is the top vertex in each tree.
- The degrees of the vertices appear in parenthesis

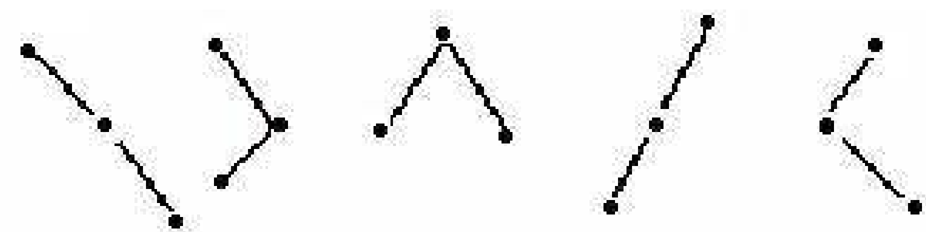


(1,1,1,3) (1,1,1,3) (1,1,2,2) (1,1,2,2)

Non-isomorphic binary trees

Theorem:

- There are C(2n,n) / (n+1) non-isomorphic binary trees with n vertices, n $\geq$ 0, where
- C(2n,n) / (n+1) are the Catalan numbers C_n



5 nonisomorphic binary trees
with 3 vertices

Catalan numbers (1)

- **Eugene Charles Catalan**
  - Belgian mathematician, 1814-1894
- Catalan numbers can be computed using this formula:

$$C_n = C(2n,n) / (n+1) \quad \text{for } n \geq 0$$

- The first few Catalan numbers are:

n	= 0	1	2	3	4	5	6	7	8	9	10
C_n	= 1	1	2	5	14	42	132	429	1430	4862	16796

Applications of Catalan numbers

- The number of ways in which a polygon with n+2 sides can be cut into n triangles
- The number of ways in which parentheses can be placed in a sequence of numbers, to be multiplied two at a time
- The number of rooted trivalent trees with n+1 vertices
- The number of paths of length 2n through an n by n grid that do not rise above the main diagonal
- The number of nonisomorphic binary trees with n vertices

Isomorphism of binary trees

There is an algorithm to test whether two binary trees are isomorphic or not.

- If the number of vertices in the two trees is n,
- and if the number of comparisons needed is a_n, it can be shown that $a_n \leq 3n + 2$.

Theorem : The worst case of this algorithm is $\Theta(n)$.

Game trees

Trees can be used to analyze all possible move sequences in a game:

- Vertices are positions:
  - a square represents one player and a circle represents another player
- An edge represents a move
- A path represents a sequence of moves


thank you