Today we will discuss:

Section 11.1 *Introduction to trees*

11.1 *Introduction to trees*

A *tree* is an undirected graph that is connected and has no simple circuits (cycles).

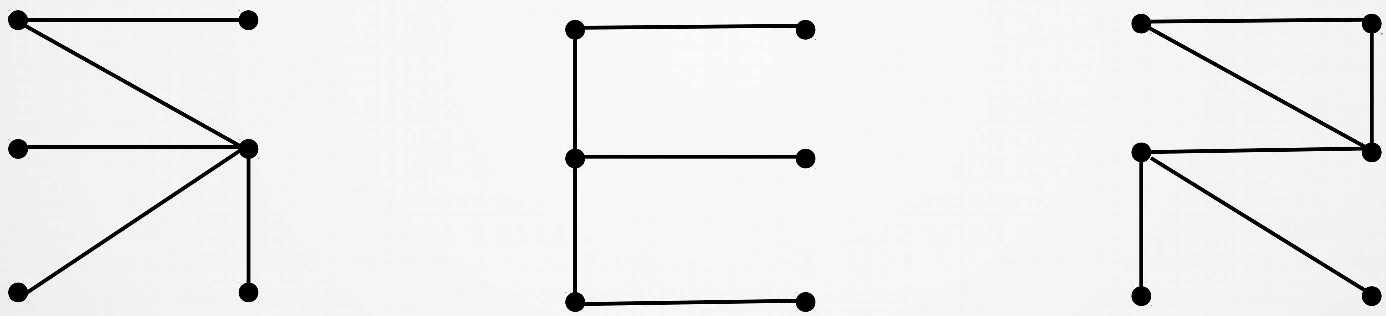
Trees can be used to:

- construct efficient algorithms for location items in a list
- study games (checkers, chess) and determine winning strategies
- model procedures carried our using a sequence of decisions, which helps to determine the computational complexity of the algorithm
- in data compression (Huffman coding)

...

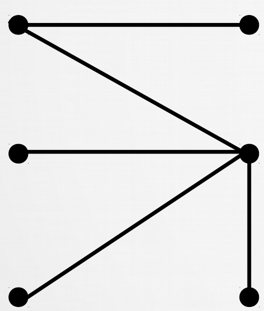
11.1 *Introduction to trees*

A *tree* is an undirected graph that is connected and has no simple circuits (cycles).

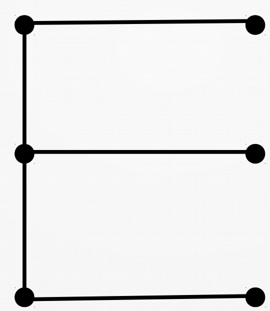


11.1 *Introduction to trees*

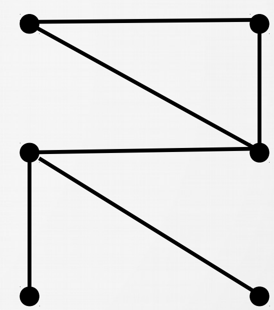
A *tree* is an undirected graph that is connected and has no simple circuits (cycles).



a *tree*



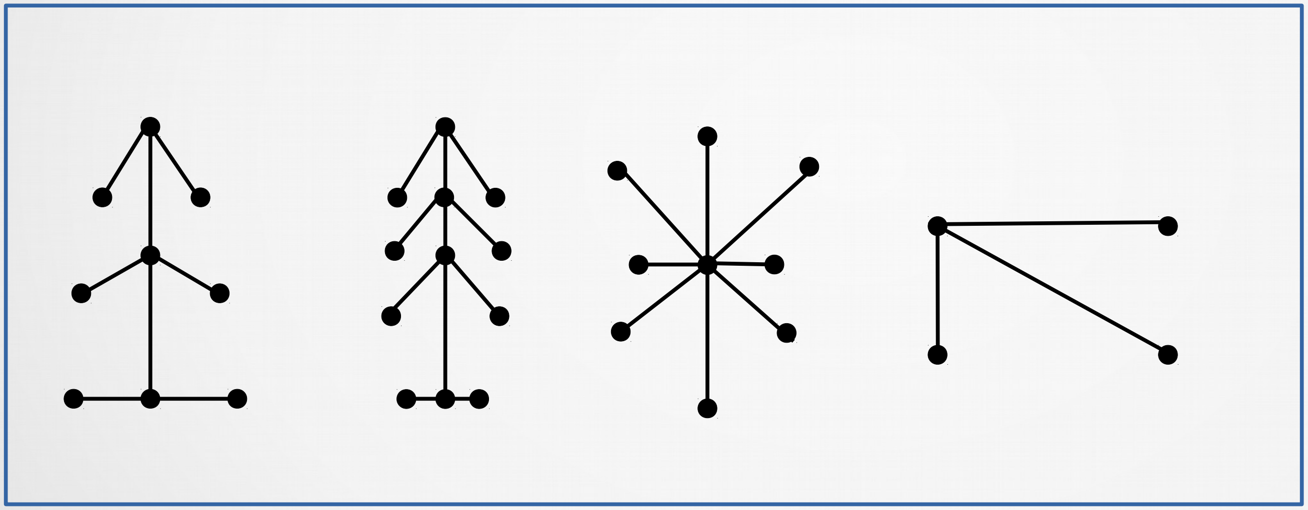
a *tree*



not a *tree*

11.1 *Introduction to trees*

A *forest* is an undirected graph that has no simple circuits (cycles).
Each of its connected components is a *tree*.



a *forest*

11.1 *Introduction to trees*

A *tree* is an undirected graph that is <u>connected</u> and has <u>no simple circuits</u> (<u>cycles</u>).

alternative definition:
a *tree* is an undirected graph such that there is a <u>unique simple path</u> between every pair of its vertices.

11.1 *Introduction to trees*

A *tree* is an undirected graph that is connected and has no simple circuits (cycles).

alternative definition:
a *tree* is an undirected graph such that there is a unique simple path between every pair of its vertices.

[Theorem]
An undirected graph is a tree iff there is a unique simple path between any two of its vertices.

11.1 *Introduction to trees*

[Theorem]

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

Proof:

1) (→) assume that T is a tree

11.1 *Introduction to trees*

[Theorem]

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

Proof:

1) (→) assume that T is a tree, then
T is *connected* and has *no simple circuits*.

11.1 *Introduction to trees*

[Theorem]
An undirected graph is a tree iff there is a unique simple path between any two of its vertices.

Proof:
1) (→) assume that T is a tree, then
T is *connected* and has *no simple circuits*.
Let *x* and *y* be any two vertices of T.

11.1 *Introduction to trees*

[Theorem]

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

Proof:

1) (→) assume that T is a tree, then
T is *connected* and has *no simple circuits*.
Let *x* and *y* be any two vertices of T.
By **Theorem** 1 from *Section 10.4* there is a simple path between *x* and *y* because T is connected.

11.1 *Introduction to trees*

[Theorem]

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

Proof:

1) (→) assume that T is a tree, then
T is *connected* and has *no simple circuits*.
Let *x* and *y* be any two vertices of T.
By **Theorem** 1 from *Section 10.4* there is a simple path between *x* and *y* because T is connected.
Assume that there exists another simple path between *x* and *y*.

11.1 *Introduction to trees*

[Theorem]

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

Proof:

1) (→) assume that T is a tree, then
T is *connected* and has *no simple circuits*.
Let *x* and *y* be any two vertices of T.
By **Theorem** 1 from *Section 10.4* there is a simple path between *x* and *y* because T is connected.
Assume that there exists another simple path between *x* and *y*. In this case we will be able to form a circuit (*not necessarily simple*) by combining two paths.

11.1 *Introduction to trees*

[Theorem]

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

Proof:

1) (→) assume that T is a tree, then
T is *connected* and has *no simple circuits*.
Let *x* and *y* be any two vertices of T.
By **Theorem** 1 from *Section 10.4* there is a simple path between *x* and *y* because T is connected.
Assume that there exists another simple path between *x* and *y*. In this case we will be able to form a circuit (*not necessarily simple*) by combining two paths. This implies that a simple circuit can be built (exercise 50 from Section 10.4).

11.1 *Introduction to trees*

[Theorem]

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

Proof:

1) (→) assume that T is a tree, then
T is *connected* and has *no simple circuits*.
Let *x* and *y* be any two vertices of T.
By **Theorem** 1 from *Section 10.4* there is a simple path between *x* and *y* because T is connected.
Assume that there exists another simple path between *x* and *y*. In this case we will be able to form a circuit (*not necessarily simple*) by combining two paths. This implies that a simple circuit can be built (exercise 50 from Section 10.4).
This contradicts to the statement that T is a tree.

11.1 *Introduction to trees*

[Theorem]

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

Proof:

1) (→) assume that T is a tree, then
T is *connected* and has *no simple circuits*.
Let *x* and *y* be any two vertices of T.
By **Theorem** 1 from *Section 10.4* there is a simple path between *x* and *y* because T is connected.
Assume that there exists another simple path between *x* and *y*. In this case we will be able to form a circuit (*not necessarily simple*) by combining two paths. This implies that a simple circuit can be built (exercise 50 from Section 10.4).
This contradicts to the statement that T is a tree.
Therefore the simple path is unique.

11.1 *Introduction to trees*

[Theorem]

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

Proof:

2) (←) assume that there is a unique path between any two vertices of T.

11.1 *Introduction to trees*

[Theorem]

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

Proof:

2) (←) assume that there is a unique path between any two vertices of T.

Then T is connected.

11.1 *Introduction to trees*

[Theorem]

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

Proof:

2) (←) assume that there is a unique path between any two vertices of T.

Then T is connected.

Assume that T has a simple circuit that contains vertices x and y.

11.1 *Introduction to trees*

[Theorem]

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

Proof:

2) (←) assume that there is a unique path between any two vertices of T.

Then T is connected.

Assume that T has a simple circuit that contains vertices *x* and *y*. Then we can form two paths between these vertices. This contradicts to the assumption about uniqueness of a path.

11.1 *Introduction to trees*

[Theorem]

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

Proof:

2) (←) assume that there is a unique path between any two vertices of T.

Then T is connected.

Assume that T has a simple circuit that contains vertices *x* and *y*. Then we can form two paths between these vertices. This contradicts to the assumption about uniqueness of a path.

Therefore, T has no simple circuits.

11.1 *Introduction to trees*

[Theorem]
An undirected graph is a tree iff there is a unique simple path between any two of its vertices.

Proof:
2) (←) assume that there is a unique path between any two vertices of T.
Then T is connected.
Assume that T has a simple circuit that contains vertices *x* and *y*. Then we can form two paths between these vertices. This contradicts to the assumption about uniqueness of a path.
Therefore, T has no simple circuits.
By definition, T is a tree.

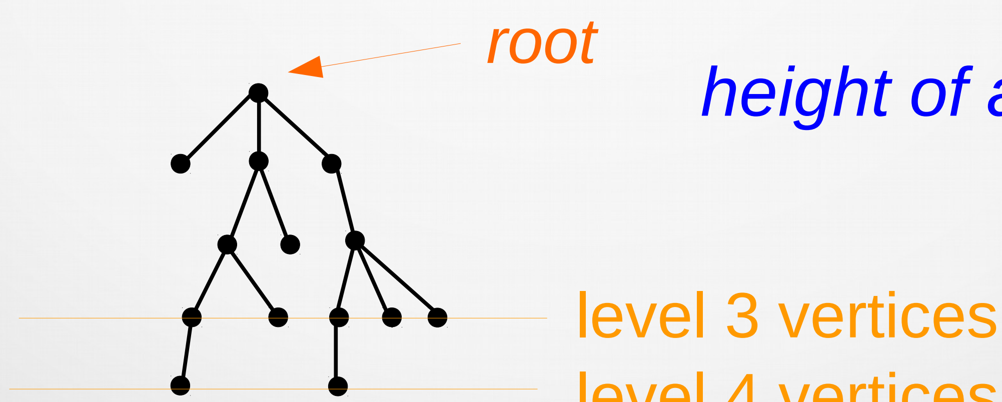
q.e.d.

11.1 *Introduction to trees*

[Def] A *rooted tree* is a tree in which one vertex has been designated as the *root* and <u>every edge is directed away</u> from the root.

[Def] The *level of a vertex* is its distance from to the root.

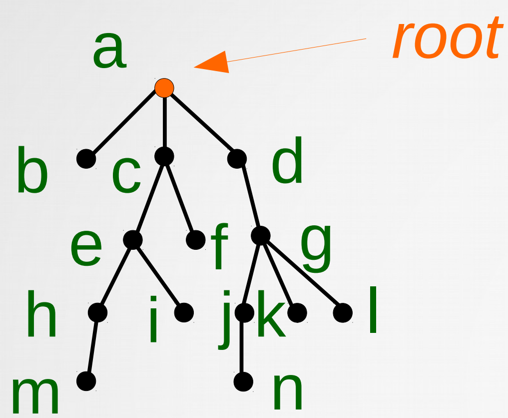
[Def] The *height of a tree* is the highest level of any vertex.



height of a tree = *4*

a *rooted tree*

11.1 *Introduction to trees*

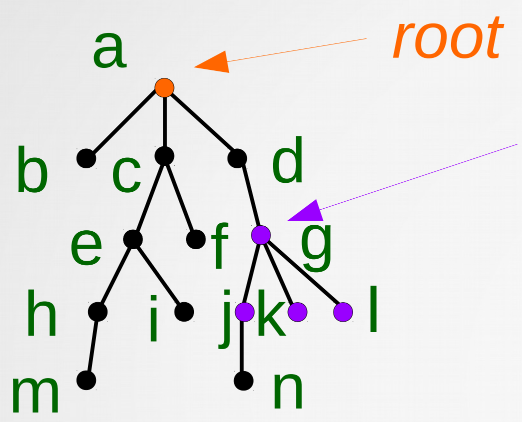


a *rooted tree*

[Corollary] There is a unique path from the root of the tree to each vertex of the tree.

This follows from Theorem we just proved.

11.1 *Introduction to trees*

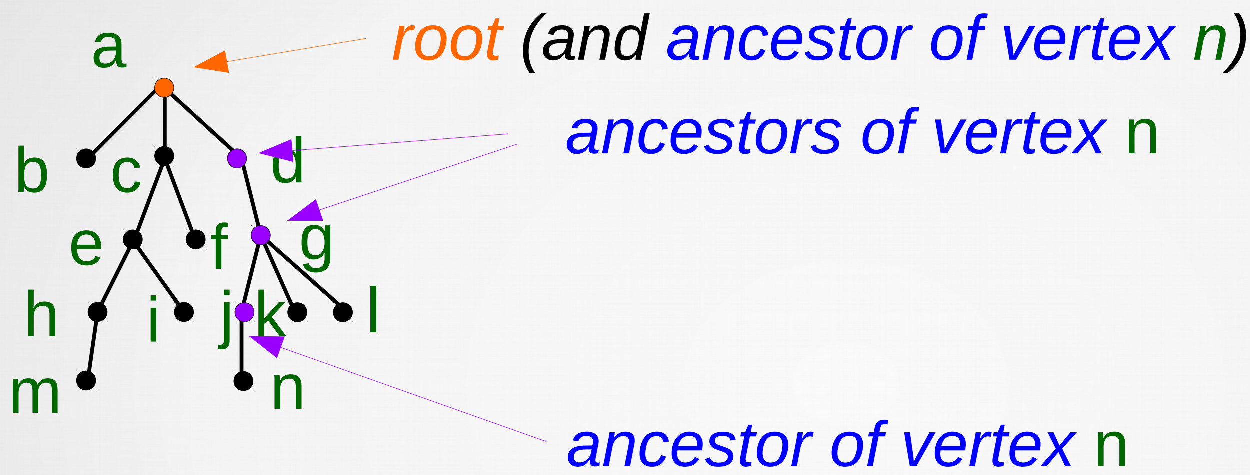


vertex g is the *parent*
of vertices j, k, and l.

a *rooted tree*

- Every vertex in a rooted tree T has a unique parent, except for the root which does not have a parent. The *parent of vertex* v is the first vertex after v encountered along the path from v to the *root*.

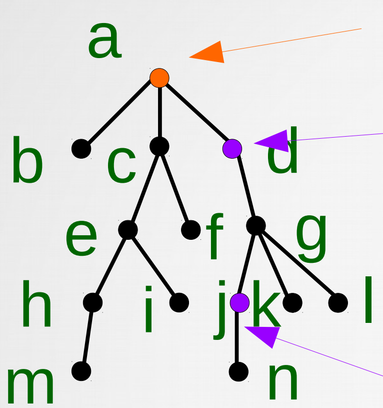
11.1 *Introduction to trees*



a *rooted tree*

- Every vertex along the path from v to the *root* (except for the vertex v itself, but including the *root*) is an *ancestor of vertex* v.

11.1 *Introduction to trees*



root

descendants of vertex d are vertices g, j, k, l and n

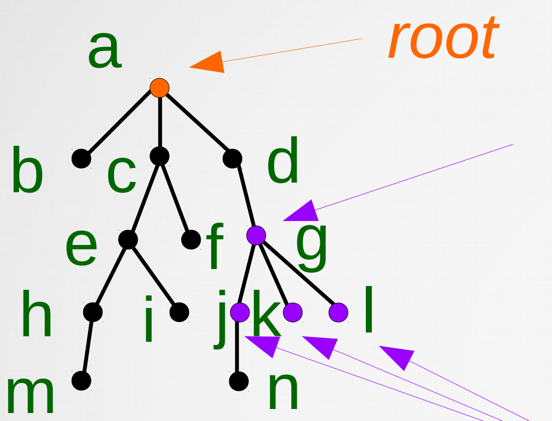
descendant of vertex k is n

a *rooted tree*

- Every vertex along the path from v to the *root* (except for the vertex v itself, but including the *root*) is an *ancestor of vertex* v.

- If u is an ancestor of v, then v is a *descendant* of u.

11.1 *Introduction to trees*



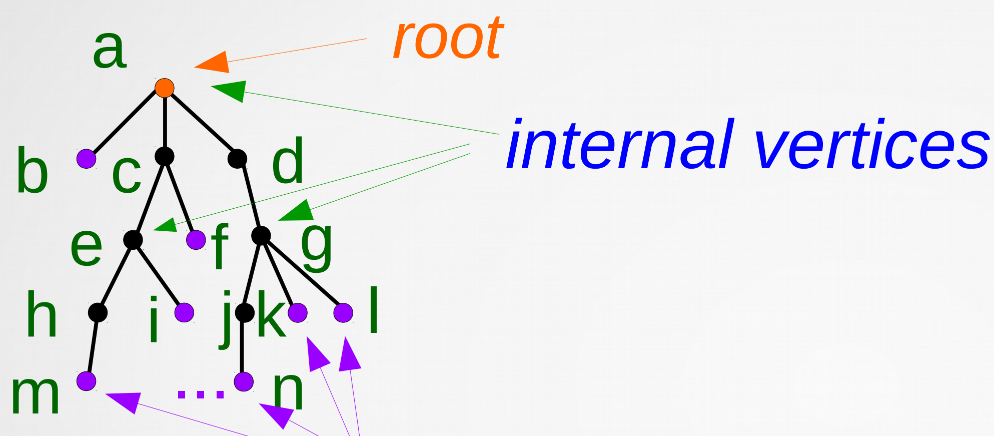
vertex g is the *parent* of vertices j, k, and l.

vertices j, k, and l are *siblings* and are *children* of vertex g.

a *rooted tree*

- If v is the parent of vertex u, then u is a *child of vertex* v.
- Two or more vertices are *siblings* if they have the same parent.

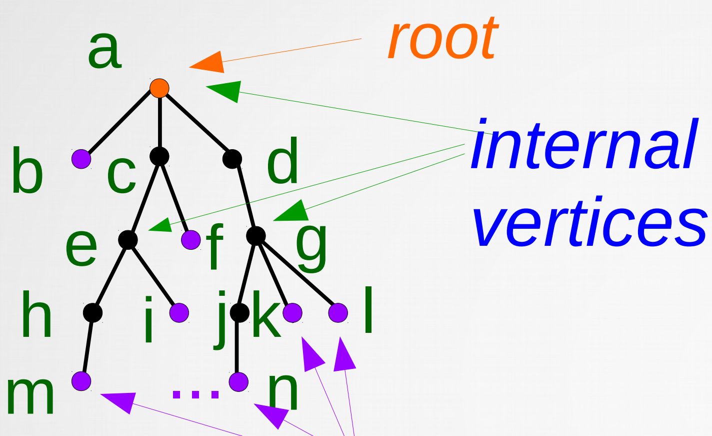
11.1 *Introduction to trees*



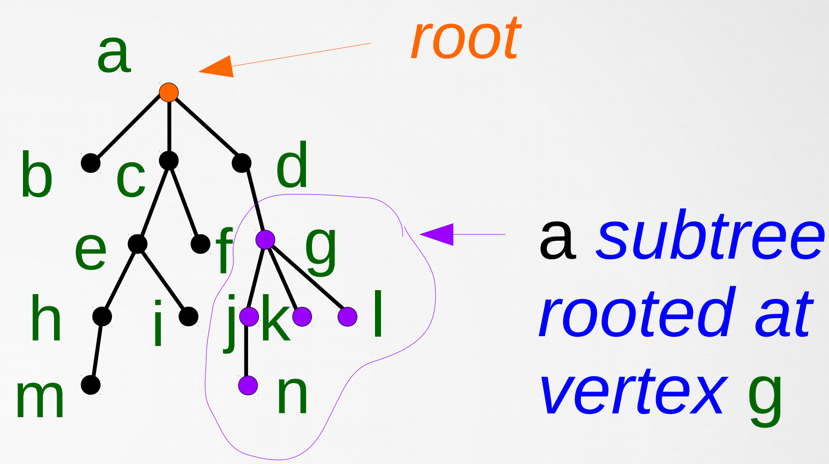
b, f, l, k, l, m and n are leaves

- a *leaf* is a vertex which has no children.
- vertices that have children are called *internal vertices*

11.1 *Introduction to trees*



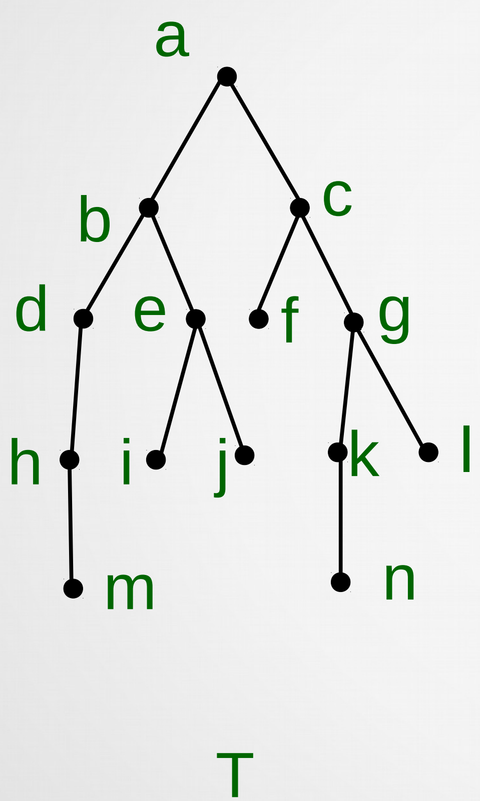
b, f, I, k, l, m and n are leaves



- a *leaf* is a vertex which has no children.
- vertices that have children are called *internal vertices*
- a *subtree rooted at vertex* v is the tree consisting of v and all v's descendants and all the edges incident to the descendants.

11.1 *Introduction to trees*

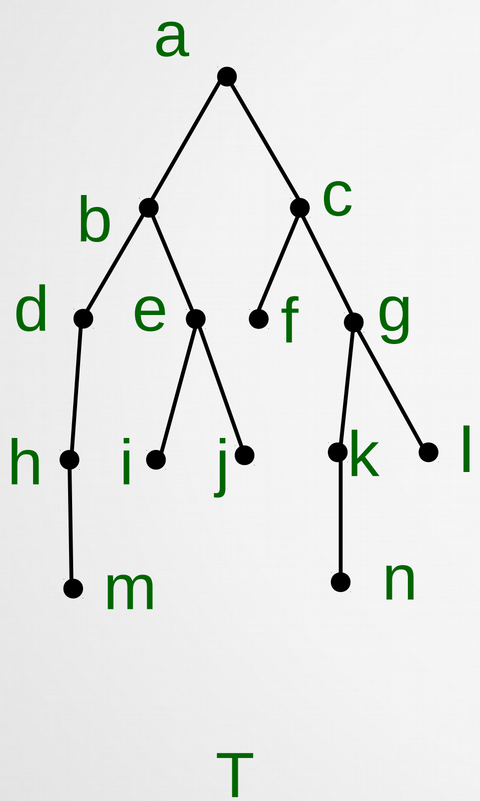
Practice: for the given tree T find



- the root of the tree T
- all leaves
- all internal vertices
- the parent of g
- the ancestors of h
- the descendants of b
- the height of the tree
- the level of vertex i

11.1 *Introduction to trees*

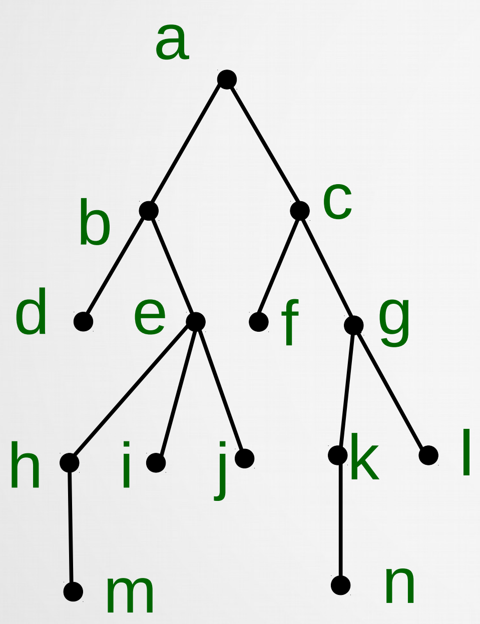
Practice: for the given tree T find



- the root of the tree T : a
- all leaves : m, l, j, f, n, l
- all internal vertices :
  a, b, c, d, e, g, h, k
- the parent of g : c
- the ancestors of h : d, b, a
- the descendants of b :
  d, e, h, i, j, m
- the height of the tree : 4
- the level of vertex i : 3

11.1 *Introduction to trees*

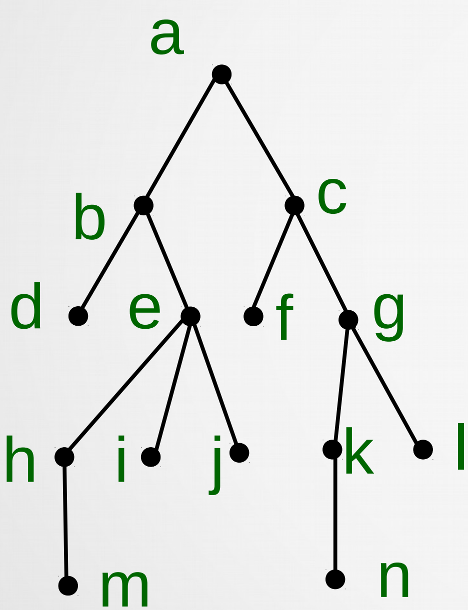
[Def] a rooted tree is called *m-ary tree* if every internal vertex has no more than m children.



3-ary tree

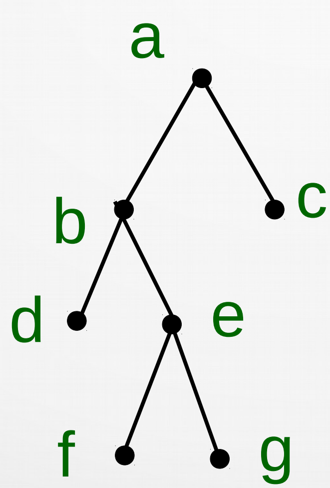
11.1 *Introduction to trees*

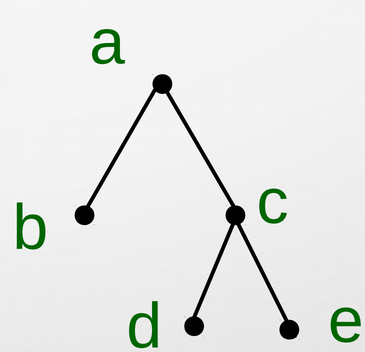
[Def] a rooted tree is called *m-ary tree* if every internal vertex has no more than m children.

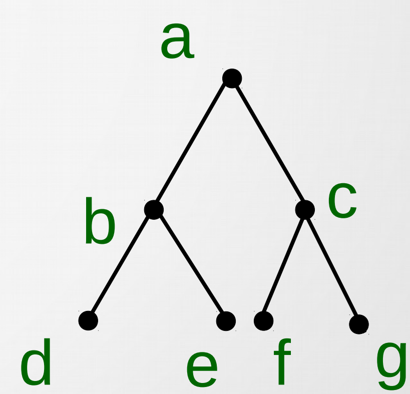


3-ary tree

[Def] The tree is call a *full m-ary tree* if every internal vertex has exactly m children.

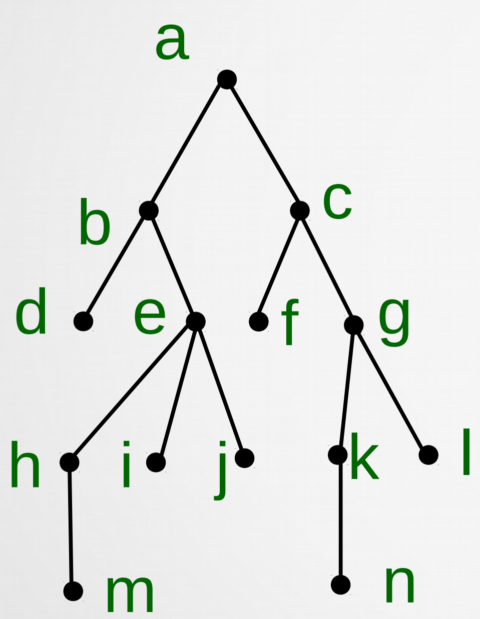






11.1 *Introduction to trees*

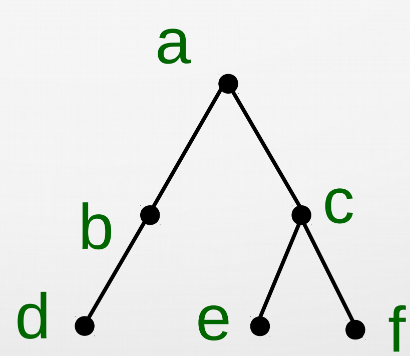
[Def] a rooted tree is called *m-ary tree* if every internal vertex has no more than m children.



3-ary tree

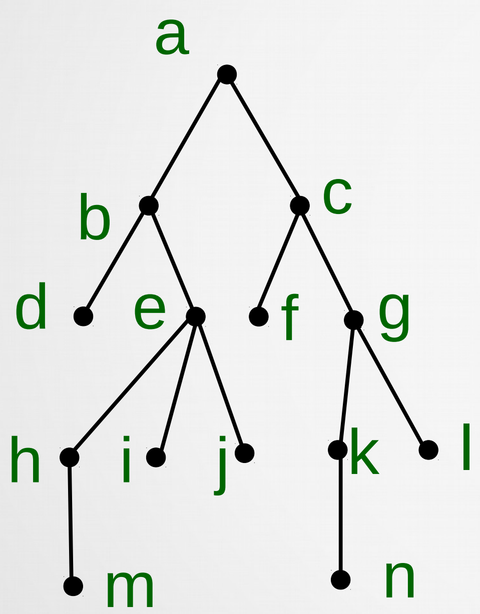
[Def] The tree is call a *full m-ary tree* if every internal vertex has exactly m children.

[Def] An *m-ary tree* with m = 2 is called a *binary tree*.



11.1 *Introduction to trees*

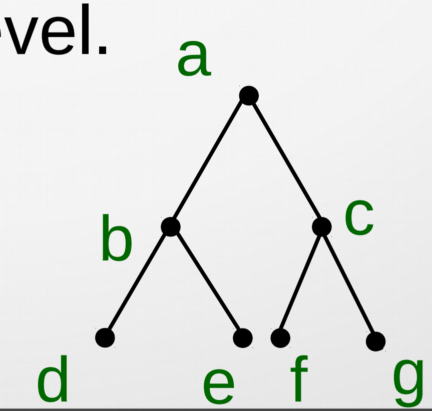
[Def] a rooted tree is called *m-ary tree* if every internal vertex has no more than m children.



3-ary tree

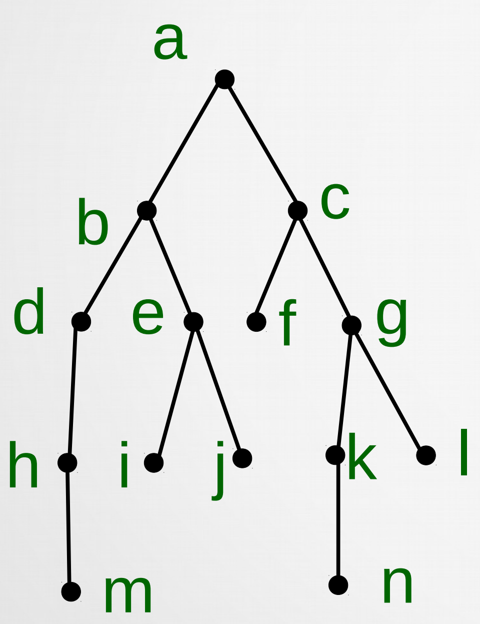
[Def] The tree is call a *full m-ary tree* if every internal vertex has exactly m children.

[Def] A *complete m-ary tree* is a *full m-ary tree* in which each leaf is at the same level.



11.1 *Introduction to trees*

We will be “ordering rooted trees” so that the children of each internal vertex are shown in order from left to right.



This is a *binary tree*.

Vertex b has the *left child* d and the *right child* e.

The subtree rooted at the *left child* is called *left subtree*.

The subtree rooted at the *right child* is called *right subtree*.

11.1 *Introduction to trees*

Trees are used as models in Computer Science, Geology, Biology, Chemistry, Botany and Psycology.

11.1 *Introduction to trees*

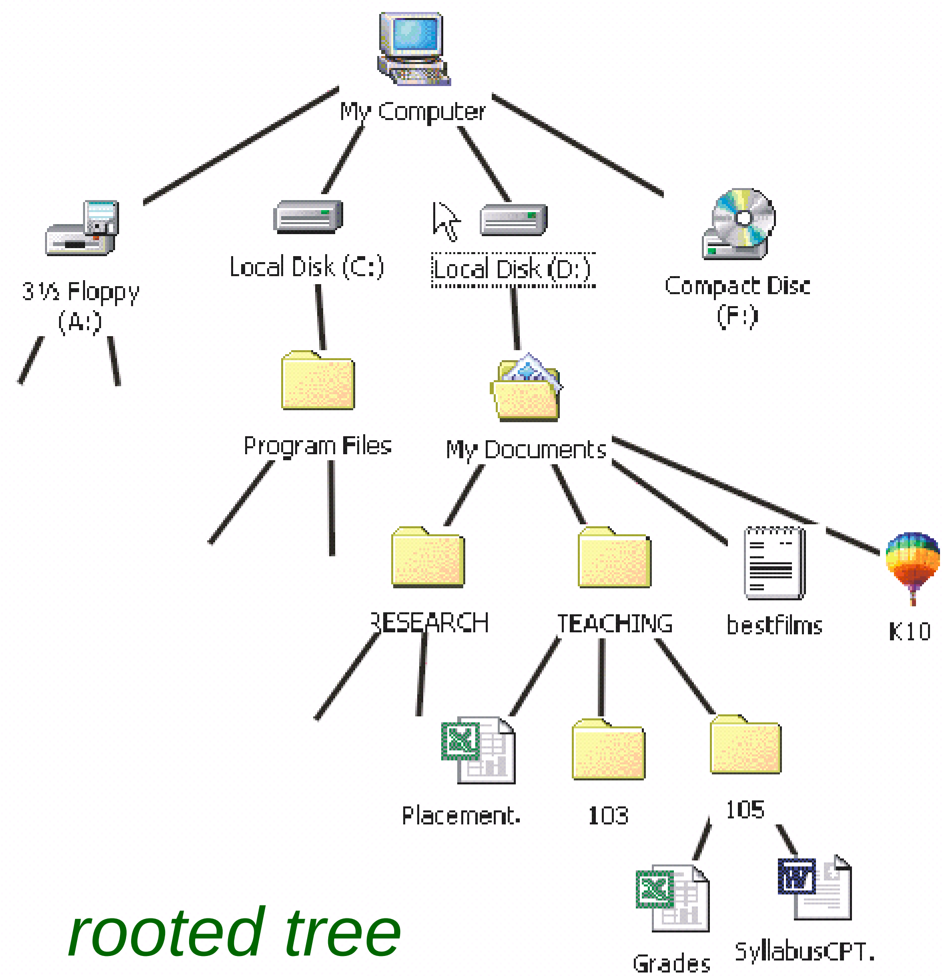
Trees are used as models in Computer Science, Geology, Biology, Chemistry, Botany and Psycology.

Example 1: File trees

Files can are organized into *directories/folders*.

A *directory/folder* can Contain both files and *subdirectories/subfolders*

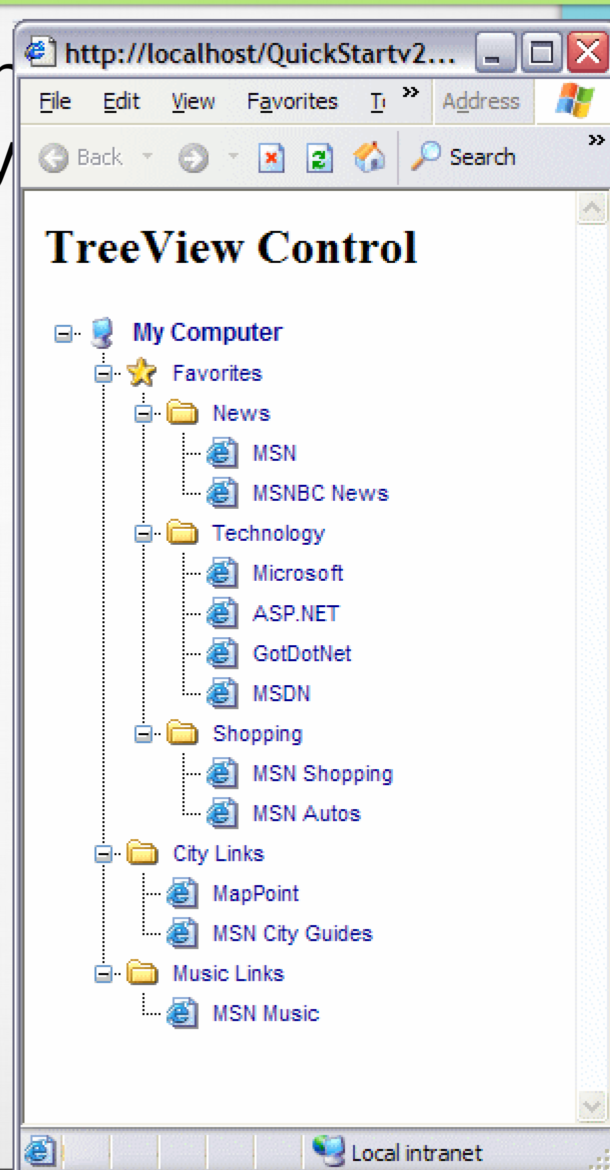
The *root directory/folder* contains the entire *file system.*



rooted tree

11.1 *Introduction to trees*

Trees are used as models in ,
Geology, Biology, Chemistry logy.



Example 1: File trees

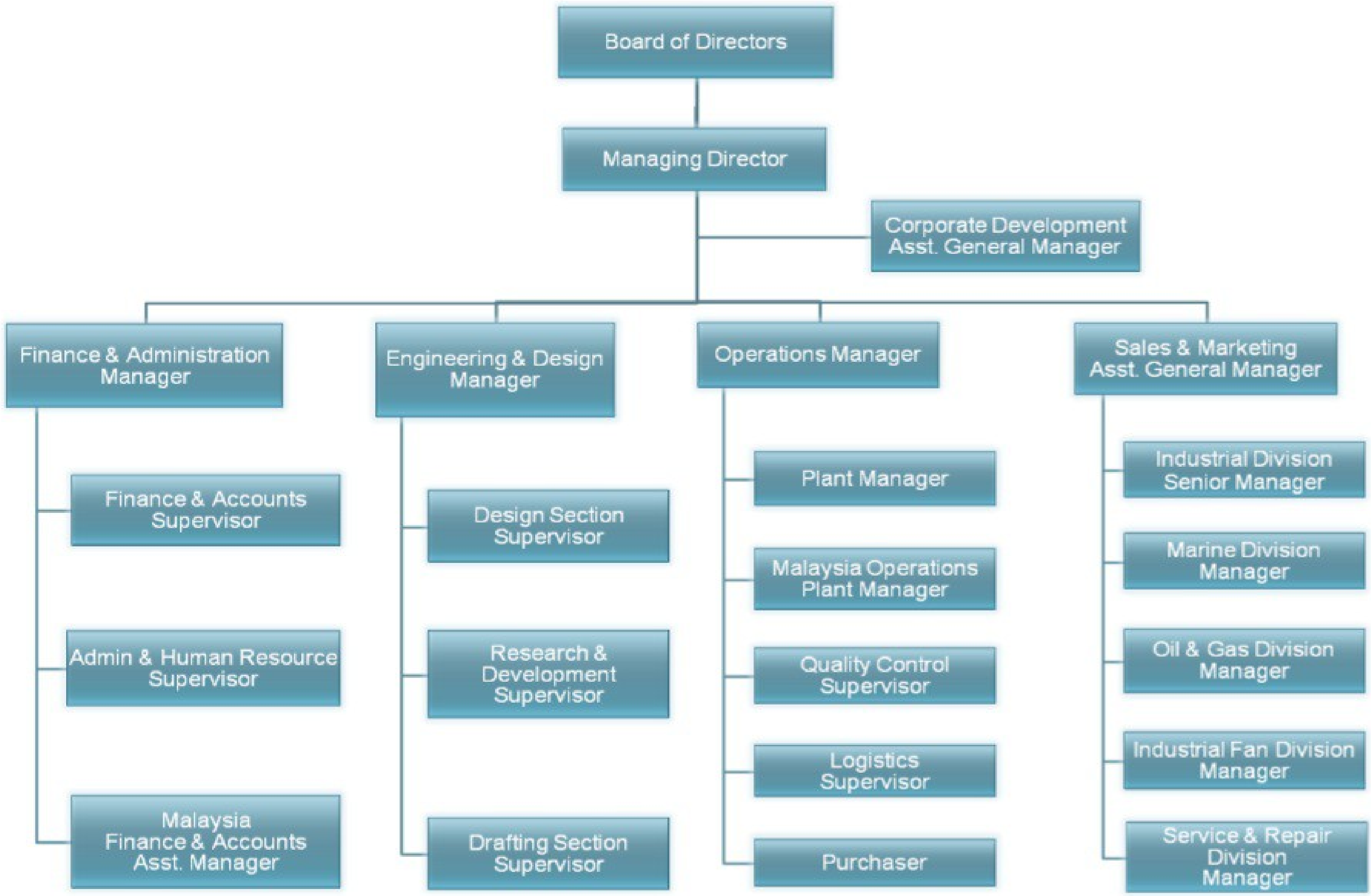
Files can are organized into *directories/folders*.

A *directory/folder* can Contain both files and *subdirectories/subfolders*

The *root directory/folder* contains the entire *file system.*

11.1 *Introduction to trees*

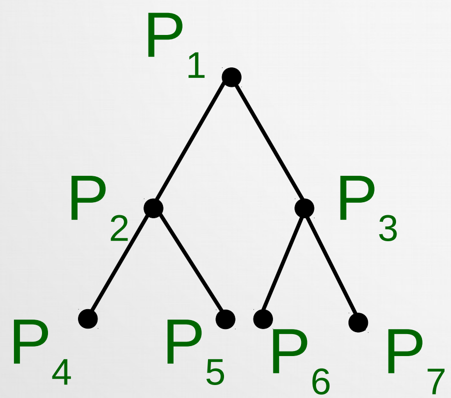
Example 2: the structure of a large organization can be modeled using a rooted tree



11.1 *Introduction to trees*

Example 3: tree-connected parallel processors
A tree-connected network is one of the ways to interconnect processors.

Consider a *complete binary tree* of height 2:
7 processors are interconnected with each other.
Each edge is a two-way connection.

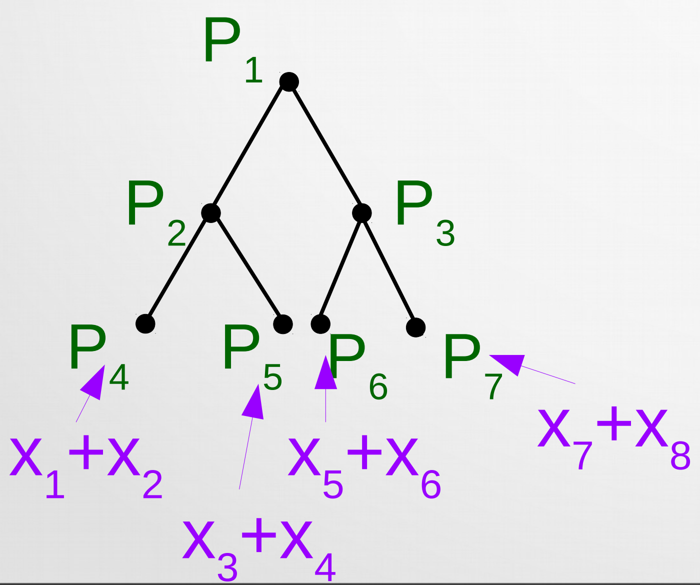


Let's add 8 numbers using 3 steps:

11.1 *Introduction to trees*

Example 3: tree-connected parallel processors
A tree-connected network is one of the ways to interconnect processors.

Consider a *complete binary tree* of height 2:
7 processors are interconnected with each other.
Each edge is a two-way connection.

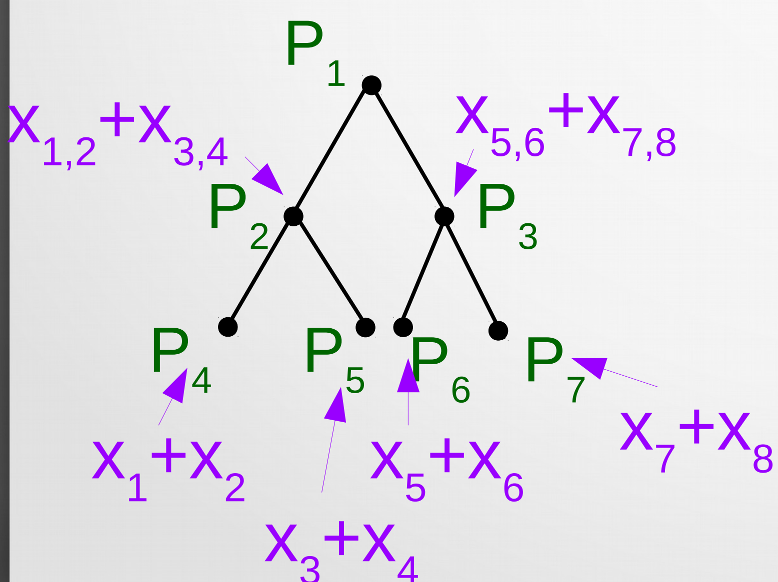


Let's add 8 numbers using 3 steps:

11.1 *Introduction to trees*

Example 3: tree-connected parallel processors
A tree-connected network is one of the ways to interconnect processors.

Consider a *complete binary tree* of height 2:
7 processors are interconnected with each other.
Each edge is a two-way connection.

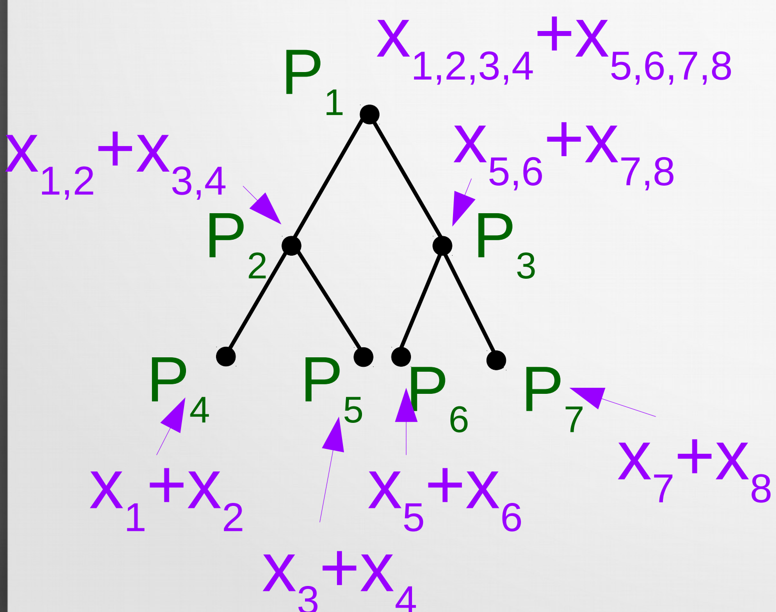


Let's add 8 numbers using 3 steps:

11.1 *Introduction to trees*

Example 3: tree-connected parallel processors
A tree-connected network is one of the ways to interconnect processors.

Consider a *complete binary tree* of height 2:
7 processors are interconnected with each other.
Each edge is a two-way connection.

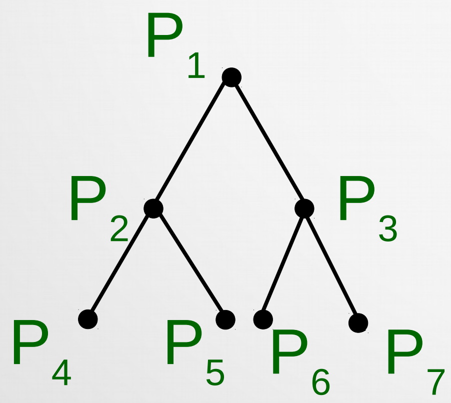


Let's add 8 numbers using 3 steps:

11.1 *Introduction to trees*

Properties of trees

[Theorem 2] A rooted tree with *n* vertices has *n-1* edges



11.1 *Introduction to trees*

Properties of trees

[Theorem 2] A rooted tree with *n* vertices has *n-1* edges

Proof: by mathematical induction

11.1 *Introduction to trees*

Properties of trees

[Theorem 2] A rooted tree with n vertices has $n-1$ edges

Proof: by mathematical induction

Basis step: when $n = 1$ (1 vertex) •

The number of edges is 0. $1-1 = 0$

11.1 *Introduction to trees*

Properties of trees

[Theorem 2] A rooted tree with *n* vertices has *n-1* edges

Proof: by mathematical induction

Basis step: when $n = 1$ (1 vertex) •

The number of edges is *0*. $1-1 = 0$

Inductive step: Assume that any arbitrary tree with *k* vertices has *k-1* edges (IH).

11.1 *Introduction to trees*

Properties of trees

[Theorem 2] A rooted tree with n vertices has $n-1$ edges

Proof: by mathematical induction

Basis step: when $n = 1$ (1 vertex)

The number of edges is 0. $1-1 = 0$

Inductive step: Assume that any arbitrary tree with k vertices has $k-1$ edges (IH).

Consider a tree T with $k+1$ vertices. Let v be a leaf of T (the tree is finite, therefore such a vertex exists).

11.1 *Introduction to trees*

Properties of trees

[Theorem 2] A rooted tree with n vertices has n-1 edges

Proof: by mathematical induction

Basis step: when $n = 1$ (1 vertex) •
The number of edges is 0. 1-$1 = 0$

Inductive step: Assume that any arbitrary tree with k vertices has k-1 edges (IH).
Consider a tree T with k+1 vertices. Let v be a leaf of T (the tree is finite, therefore such a vertex exists).
Let vertex w be a parent of v. If we remove vertex v and edge (w,v) from T, then we will get a tree T' with k vertices (for which IH holds).
Therefore, tree T has $(k$-$1)$+1 edges. This completes I.s.

11.1 *Introduction to trees*

Properties of trees

[Theorem 2] A rooted tree with n vertices has $n-1$ edges

Proof: by mathematical induction

Basis step: when $n = 1$ (1 vertex) •
The number of edges is 0. $1-1 = 0$

Inductive step: Assume that any arbitrary tree with k vertices has $k-1$ edges (IH).
Consider a tree T with $k+1$ vertices. Let v be a leaf of T (the tree is finite, therefore such a vertex exists).
Let vertex w be a parent of v. If we remove vertex v and edge (w,v) from T, then we will get a tree T' with k vertices (for which IH holds).
Therefore, tree T has $(k-1)+1$ edges. This completes I.s.
By math. induction we proved the statement true.

q.e.d.

11.1 *Introduction to trees*

Properties of trees

[Theorem 3] A *full m-ary tree* with i internal vertices contains $n = mi+1$ vertices.

11.1 *Introduction to trees*

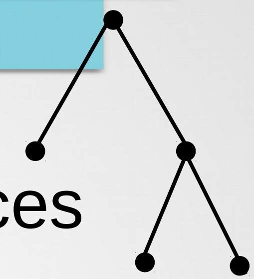
Properties of trees

[Theorem 3] A *full m-ary tree* with i internal vertices contains $n = mi+1$ vertices.

Proof:
Every vertex except the root is the child of internal vertex.

11.1 *Introduction to trees*

Properties of trees



[Theorem 3] A *full m-ary tree* with i internal vertices contains $n = mi+1$ vertices.

Proof:

Every vertex except the root is the child of internal vertex.

Each of the i internal vertices have m children, hence there are mi vertices in the tree (other than the root).

11.1 *Introduction to trees*

Properties of trees

[Theorem 3] A *full m-ary tree* with i internal vertices contains $n = mi+1$ vertices.

Proof:

Every vertex except the root is the child of internal vertex.

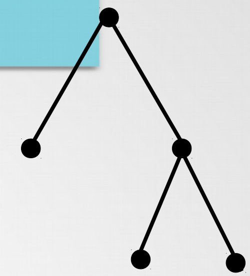
Each of the i internal vertices have m children, hence there are mi vertices in the tree (other than the root).

Therefore, there are $mi+1$ vertices (we include the root).

q.e.d.

11.1 *Introduction to trees*

Properties of trees



[Theorem 4] 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;

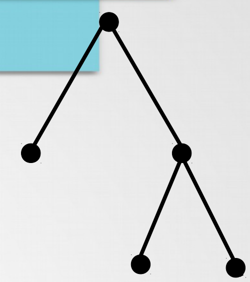
(3) l leaves has

$n = (ml-1) / (m-1)$ vertices, and

$i = (l-1) / (m-1)$ internal vertices.

11.1 *Introduction to trees*

Properties of trees



[Theorem 4] 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.

from Theorem 3.
A *full m-ary tree* with i internal vertices contains $n = mi+1$ vertices.

In addition, $n = l+i$

11.1 *Introduction to trees*

Properties of trees

Practice:

1) How many edges does a tree with 10,000 vertices have?

2) How many vertices does a full 5-ary tree with 100 internal vertices have?

3) How many leaves does a full 5-ary tree with 100 internal vertices have?

11.1 *Introduction to trees*

Properties of trees

Practice:

1) How many edges does a tree with 10,000 vertices have?
10,000 – 1 = 9,999

2) How many vertices does a full 5-ary tree with 100 internal vertices have?
501

3) How many leaves does a full 5-ary tree with 100 internal vertices have?
401

11.1 *Introduction to trees*

Properties of trees

Example: *chain letter*

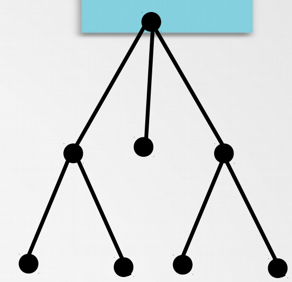
Somebody starts a chain letter. Each person who receives a letter is asked to send it on to *four other people*. Some people do this, some don't.

How many people have seen the letter, including the first person if no one receives more than one letter and if the chain letter ends after there have been 100 people who read it but did not send it out?

How many people send out the letter?

11.1 *Introduction to trees*

Properties of trees



Example: *chain letter*

Use *4-ary tree* to model the situation.

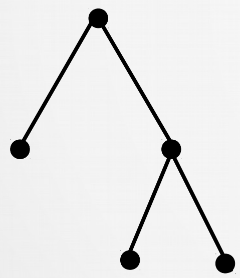
The chain letter stops when there are 100 leaves (people who did not send out the letter). $l = 100$

From **Theorem 4**:
(3) l leaves has
$n = (ml-1) / (m-1)$ vertices, and
$i = (l-1) / (m-1)$ internal vertices.
n = (4*100-1) / (4-1) = 399 / 3 = 133 people saw the letter
i = (100-1) / (4-1) = 99 / 3 = 33 people sent out the letter
or i = n – l = 133 – 100 - 30

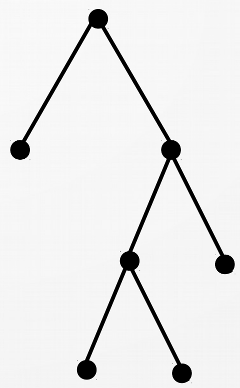
11.1 *Introduction to trees*

Balanced trees

A rooted *m-ary tree* of height *h* is *balanced* if all leaves are at levels *h* or *h-1*.



balanced binary tree



not a balanced binary tree

11.1 *Introduction to trees*

Balanced trees

A rooted *m-ary tree* of height *h* is *balanced* if all leaves are at levels *h* or *h-1*.

[Theorem 5] There are at most m^h leaves in an *m-ary tree* of height *h*, i.e. $l \leq m^h$

- the theorem provides an upper bound for the number of leaves

11.1 *Introduction to trees*

Balanced trees

A rooted *m-ary tree* of height h is *balanced* if all leaves are at levels h or $h-1$.

[Theorem 5] There are at most m^h leaves in an *m-ary tree* of height h, i.e. $l \leq m^h$

- the theorem provides an upper bound for the number of leaves

[Corollary]

1) For a *full* and *balanced m-ary tree* of height h with l leaves, $h = \lceil \log_m l \rceil$

2) For an *m-ary tree* of height h with l leaves,

$h \geq \lceil \log_m l \rceil$.

11.1 *Introduction to trees*

Balanced trees

[Theorem 5] There are at most m^h leaves in an *m-ary tree* of height h, i.e. $l \leq m^h$

[Corollary]

1) For a *full* and *balanced m-ary tree* of height h with l leaves, $h = \lceil \log_m l \rceil$
2) For an *m-ary tree* of height h with l leaves, $h \geq \lceil \log_m l \rceil$.

11.1 *Introduction to trees*

Balanced trees

[Theorem 5] There are at most m^h leaves in an *m-ary tree* of height h, i.e. $l \leq m^h$

[Corollary]

1) For a *full* and *balanced m-ary tree* of height h with l leaves, $h = \lceil \log_m l \rceil$
2) For an *m-ary tree* of height h with l leaves,
   $h \geq \lceil \log_m l \rceil$.

Proof:

2) from **Theorem 5** we have $l \leq m^h$

11.1 *Introduction to trees*

Balanced trees

[Theorem 5] There are at most m^h leaves in an *m-ary tree* of height h, i.e. $l \le m^h$

[Corollary]

1) For a *full* and *balanced m-ary tree* of height h with l leaves, $h = \lceil \log_m l \rceil$
2) For an *m-ary tree* of height h with l leaves,
   $h \ge \lceil \log_m l \rceil$.

Proof:

2) from **Theorem 5** we have $l \le m^h$

Take logarithms to the base m of both sides: $\log_m l \le h$

11.1 *Introduction to trees*

Balanced trees

[Theorem 5] There are at most m^h leaves in an *m-ary tree* of height h, i.e. $l \leq m^h$

[Corollary]

1) For a *full* and *balanced m-ary tree* of height h with l leaves, $h = \lceil \log_m l \rceil$
2) For an *m-ary tree* of height h with l leaves, $h \geq \lceil \log_m l \rceil$.

Proof:

2) from **Theorem 5** we have $l \leq m^h$

Take logarithms to the base m of both sides: $\log_m l \leq h$

Since h is integer, let's apply ceiling function: $h \geq \lceil \log_m l \rceil$

11.1 *Introduction to trees*

Balanced trees

[Theorem 5] There are at most m^h leaves in an *m-ary tree* of height h, i.e. $l \leq m^h$

[Corollary]
1) For a *full* and *balanced m-ary tree* of height h with l leaves, $h = \lceil \log_m l \rceil$

Proof:
1) If the tree is balanced, then each leaf is at level h or $h-1$.

11.1 *Introduction to trees*

Balanced trees

[Theorem 5] There are at most m^h leaves in an *m-ary tree* of height h, i.e. $l \le m^h$

[Corollary]
1) For a *full* and *balanced m-ary tree* of height h with l leaves, $h = \lceil \log_m l \rceil$

Proof:
1) If the tree is balanced, then each leaf is at level h or $h-1$. The height of the tree is h, hence there is at least one leaf at level h.

11.1 *Introduction to trees*

Balanced trees

[Theorem 5] There are at most m^h leaves in an *m-ary tree* of height h, i.e. $l \leq m^h$

[Corollary]
1) For a *full* and *balanced m-ary tree* of height h with l leaves, $h = \lceil \log_m l \rceil$

Proof:
1) If the tree is balanced, then each leaf is at level h or $h-1$. The height of the tree is h, hence there is at least one leaf at level h. Therefore, there must be more than m^{h-1} leaves (exercise 30).

11.1 *Introduction to trees*

Balanced trees

[Theorem 5] There are at most m^h leaves in an *m-ary tree* of height h, i.e. $l \leq m^h$

[Corollary]
1) For a *full* and *balanced m-ary tree* of height h with l leaves, $h = \lceil \log_m l \rceil$

Proof:
1) If the tree is balanced, then each leaf is at level h or $h-1$. The height of the tree is h, hence there is at least one leaf at level h. Therefore, there must be more than m^{h-1} leaves (exercise 30). We get: $m^{h-1} \leq l \leq m^h$

11.1 *Introduction to trees*

Balanced trees

[Theorem 5] There are at most m^h leaves in an *m-ary tree* of height h, i.e. $l \leq m^h$

[Corollary]
1) For a *full* and *balanced m-ary tree* of height h with l leaves, $h = \lceil \log_m l \rceil$

Proof:
1) If the tree is balanced, then each leaf is at level h or $h-1$. The height of the tree is h, hence there is at least one leaf at level h. Therefore, there must be more than m^{h-1} leaves (exercise 30). We get: $m^{h-1} \leq l \leq m^h$
Taking logarithms to the base m: $h-1 \leq \log_m l \leq h$

11.1 *Introduction to trees*

Balanced trees

[Theorem 5] There are at most m^h leaves in an *m-ary tree* of height h, i.e. $l \le m^h$

[Corollary]
1) For a *full* and *balanced m-ary tree* of height h with l leaves, $h = \lceil \log_m l \rceil$

Proof:
1) If the tree is balanced, then each leaf is at level h or $h-1$. The height of the tree is h, hence there is at least one leaf at level h. Therefore, there must be more than m^{h-1} leaves (exercise 30). We get: $m^{h-1} \le l \le m^h$
Taking logarithms to the base m: $h-1 \le \log_m l \le h$
Hence $h = \lceil \log_m l \rceil$

q.e.d.

11.1 *Introduction to trees*

Balanced trees

[Theorem 5] There are at most m^h leaves in an *m-ary tree* of height h, i.e. $l \leq m^h$

[Corollary]
1) For a *full* and *balanced m-ary tree* of height h with l leaves, $h = \lceil \log_m l \rceil$

Why is it important to us?

11.1 *Introduction to trees*

Balanced trees

[Theorem 5] There are at most m^h leaves in an *m-ary tree* of height h, i.e. $l \le m^h$

[Corollary]
1) For a *full* and *balanced m-ary tree* of height h with l leaves, $h = \lceil \log_m l \rceil$

Why is it important to us?
- easy location