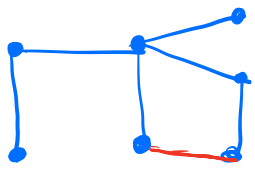
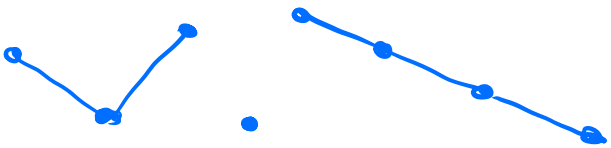
# Trees : introduction
applications
ST / MST

"a connected graph that does not contain a circuit"



tree not-tree

$G=(V, E)$



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

$\Rightarrow$ Assume T is a tree : connected graph
no circuit

a b

connected : there exists a path between a & b

- there is no other path (since it would create a circuit)

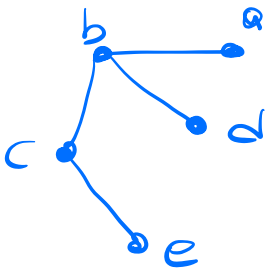
$\Leftarrow$ Assume T is a graph s. t. there is a unique path between any two vertices.

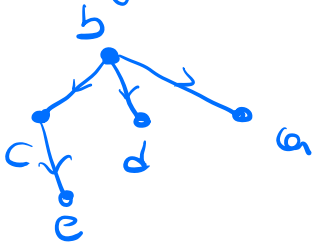
( T is a tree , connected - does not contain any circuit )

- T is connected ( since there exists a path...)
- T does not contain any circuits. .....

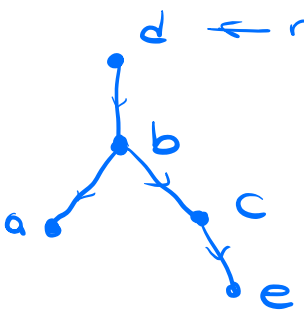
$\Rightarrow$ Thus T is a tree

**rooted tree** One vertex is the root, every edge is directed away from it. (directed graph)




rooted at b




root d

parent of a is b

parent of b is d

c is a child of b

a and c are the children of b

a is the left child of b

descendants of b are a, c, e

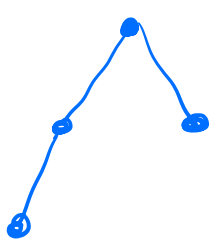
ancestors of e are c, d, d

leaves: a, e

internal nodes: d, b, c

---

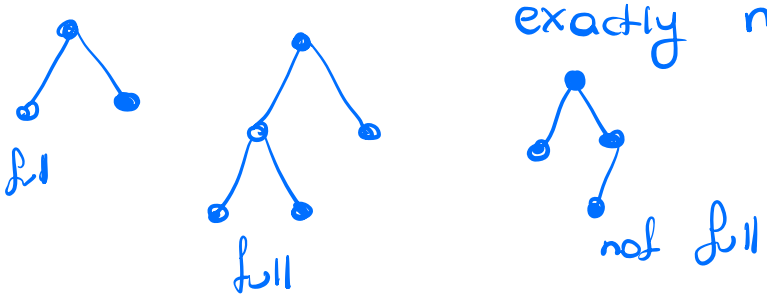
m-ary tree: rooted tree every vertex has no more than m children.



binary tree

ternary tree

Full m-ary tree : every internal (no-leaf) vertex has exactly m children




## Properties of Trees

Thm : A tree with n vertices has n-1 edges.

Proof (induction)

Base case : $k = 1$, edges $\to 0$

•

Inductive hypothesis : Assume that all trees with $k$ vertices has $k-1$ edges.

I. S. : Suppose that $T$ is a tree with $k+1$ vertices.

Let $v$ be a leaf of $T$ (a tree has to have at least 1 leaf)

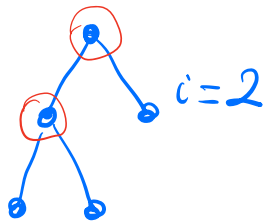

- remove $v$ from $T$ and obtain $T'$
  - remove $v$
  - remove the edge connecting $v$ to $T'$

$T'$ is a tree with $k$ vertices. (• connected, • no circuits since we can not create one by removing an edge.)

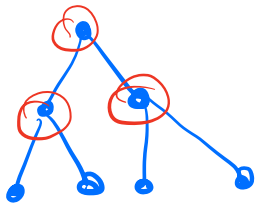
By I.H. it has $k-1$ edges

Thus $T$ has $k$ edges.

**Ex**: How many vertices does a full m-ary tree with i internal vertices have?




n = 5




n = 7

$$n = m \cdot i + 1$$

$m \cdot i$ edges $\rightarrow$ $m \cdot i + 1$ vertices

every vertex except the root is a child of an internal vertex.

$$m \cdot i + 1$$

$n$ vertices $\quad i = \frac{n-1}{m} \quad l = n - i$

$n$ : # vertices
$l$ : # leaves
$i$ : # internal vertices
$m$ : " "

$$l = n - \frac{n-1}{m}$$

$$= \frac{n \cdot m - n + 1}{m}$$

$$n = l + i$$

$$n = m \cdot i + 1$$

$l + i = m \cdot i + 1$ 19/1/22

$l = i(m-1) + 1$ $\quad i = \frac{l-1}{m-1}$

Ex chain-letter example
- Send to 3 other people
- no-one received two letters.

81 people did not send it.
How many people received it? ⇝ 120 (except the root)

$m = 3$
$l = 81$

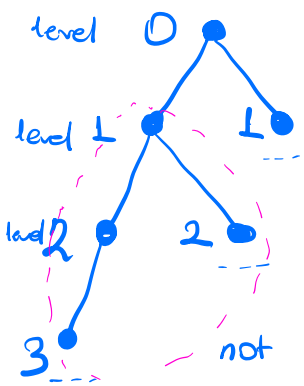
full ternary tree

* $n = l + i$
** $n = mi + 1$

$n = \frac{81 + i}{*} = \frac{3i + 1}{**}$

$i = 40$

$n = 40 + 81 = 121$



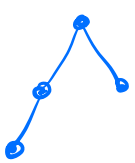

level : the length of the path from the root

height : max level in a tree

height = 3

not balanced

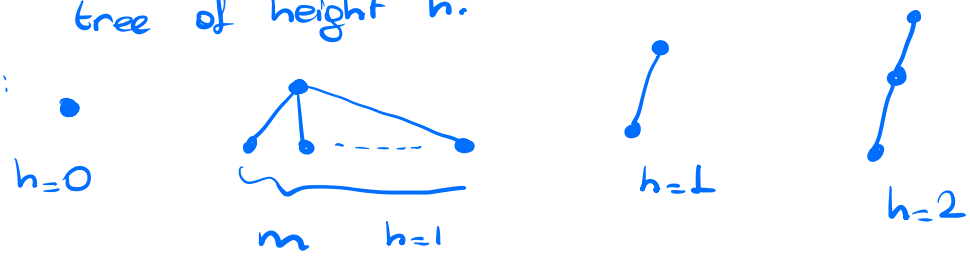
A tree is balanced if all leaves are at levels $h, h-1$.



→ balanced

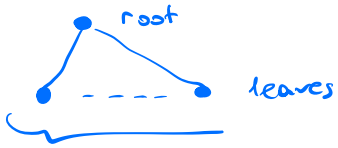
**Thm** There are at most $m^h$ leaves in an $m$-ary tree of height $h$.

examples:




**Proof** By induction.

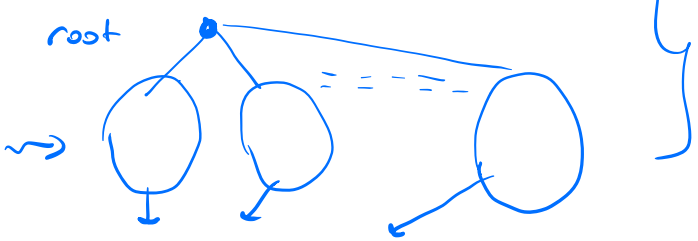
Base case, $h = 1$




at most $m$ leaves. $m^1 = m$

I.S. Assume that the property holds for trees with height $< h$ (i.e. they have at most $m^{h-1}$ leaves)

Construct a tree of height $h$.




$m$-ary trees with height $< h$ (at most $h-1$)

→ leaves of the subtrees are leaves of the new tree $T$

By IH, each has at most $m^{h-1}$ leaves,

$T$ has at most $m$ subtrees, thus it has at most

$$m \cdot m^{h-1} = m^h \text{ leaves.}$$

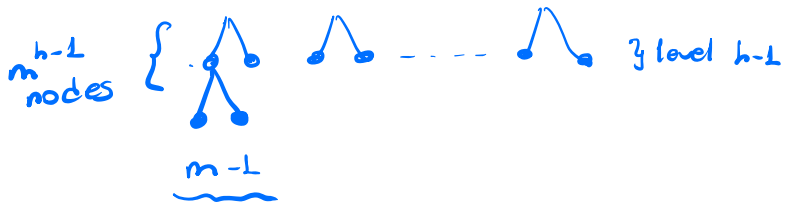
Cor For an $m$-ary tree $l$ leaves, $h \geq \lceil \log_m l \rceil$,

if the tree is full and balanced $h = \lceil \log_m l \rceil$

proof $l \leq m^h$

$\log_m l \leq h$ $\quad \lceil \log_m l \rceil \leq h$ since $h$ is an integer.

full & balanced: $h-1$, $h$




$m^{h-1} < l \leq m^h$

at least 1 leaf in level $h$

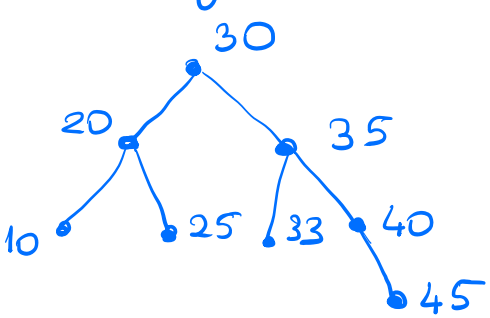
$h-1 < \log_m l \leq h$

$h = \lceil \log_m l \rceil$

---

## Applications of Trees

1) Binary search trees (BST)




- vertices has keys
- the key of a vertex is larger than the keys in its left and smaller than the keys on the right.

Search for 24 : 30, 20, 25, null : 24 is not in the tree

insert ( T , x )

v = root T
→ if v is null create the root with key x
while v ≠ null and v.key ≠ x
if v.key < x
if right child of v is not null
v = right child of v
else add new vertex with key x as the right child of v
else
if left child of v is not null
v = left child of v
else add .....

---

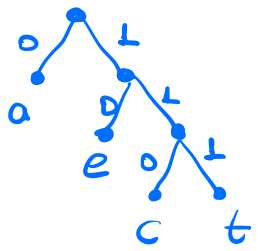
max number of comparisons for insertion
height
in general : n
balanced tree : log n

---

2) Binary decision trees

26 letters, 5 bits ( $2^5 = 32$ )

BDT encoding more effizient than 5 bit-codes
⇒ use shorter bit strings for frequently encountered letters
⇒ unambiguity: make sure a bit string of a letter is not prefix for another one.




a: 0
e: 10
c: 110

111 10 0
t e a

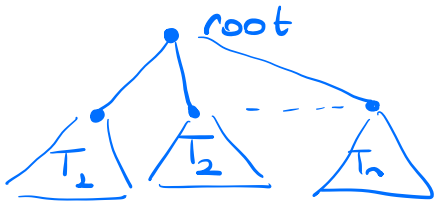
cat ?

3) Game trees : possible actions are given as child nodes

---

## Tree Traversals

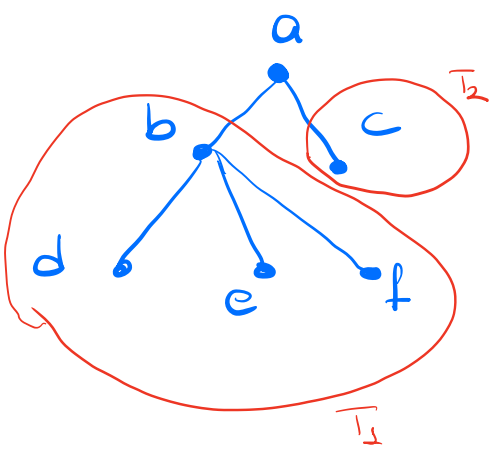
Procedures for visiting every vertex of an ordered tree.




(1) Preorder traversal : root first
root, $T_1$, $T_2$, ... $T_n$

(2) Inorder traversal
$T_1$, root, $T_2$, ... $T_n$

(3) Post traversal
$T_1$, $T_2$, ..., $T_n$, root

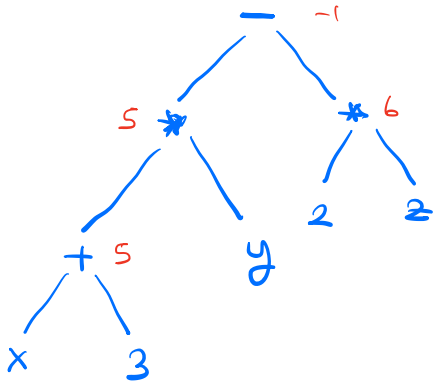



preorder:
a, b, d, e, f, c
(a = root; b, d, e, f = $T_1$; c = $T_2$)

inorder
d, b, e, f, a, c

postorder
d, e, f, b, c, a

$(x+3) * y - 2 * 2$




preorder

$- * + x\ 3\ y\ * 2\ 2$

← Evaluate left to right

inorder

$x + 3 * y - 2 * 2$

$((x+3) * y) - (2 * 2)$

$x = 2$
$y = 1$
$z = 3$

postorder

$x\ 3\ +\ y\ *\ 2\ 2\ *\ -$

$2\ 3\ +\ 1\ *\ 2\ 3\ *\ -$

$5 \qquad 6$

$5$

$-1$

## Spanning trees

Let $G$ be a simple graph.

A spanning tree of $G$ is a subgraph of $G$ that is a tree containing every vertex of $G$.

**Thm** A simple graph is connected iff it has a spanning tree.

$\Rightarrow$ ST $\rightsquigarrow$ connected

$\Leftarrow$ connected $\rightsquigarrow$ ST

Construct a tree

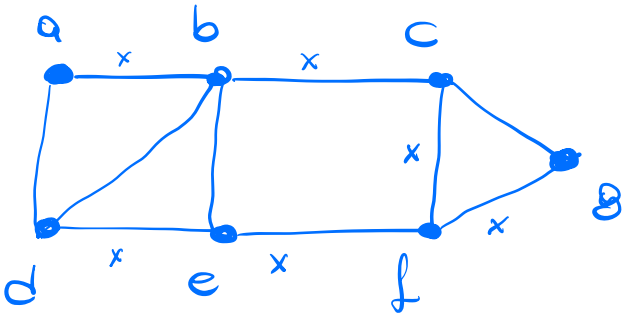
→ depth first

→ breadth first

---

Depth first search

- start from a vertex
- add vertices along a path as long as possible




visit (v)

for each u adjoced to v

if u is not in T

add u to T {u,v}

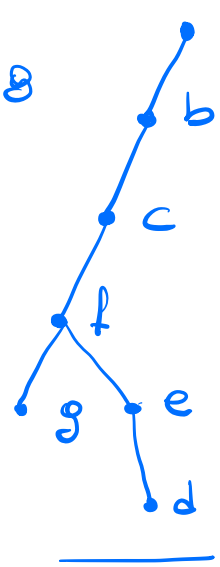
visit (u)

DFS ( G = ( {$v_1$, ..., $v_n$}, E)

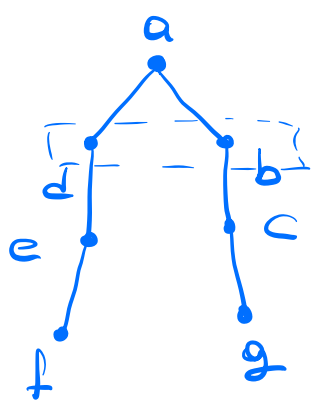
Tree T,

visit ($v_1$)

7 vertices, 6 edges




DFS




BFS

# Breadth First Search

- start from a vertex
- add all the nodes, adjacent to it to the tree (that are not visited)
- arbitrarily order the nodes that are at the highest level
- add all the vertices, adjacent to them that are not visited