Tree Overview (1A)

Copyright (c) 2015 - 2018 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice and Octave.

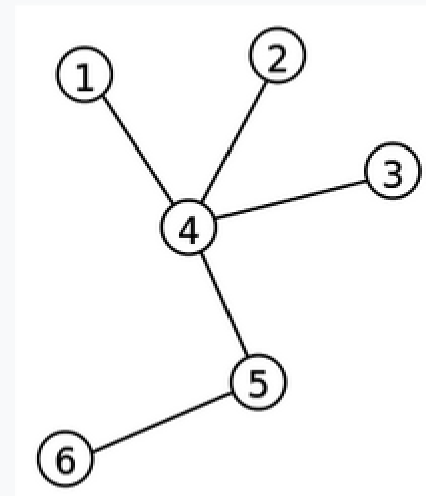
Tree

a tree is an **undirected** graph in which
any two **vertices** are **connected**
by exactly **one path**.

any **acyclic connected** graph is a **tree**.

A **forest** is a disjoint union of trees.

Trees



A labeled tree with 6 vertices and 5 edges.

Vertices	v
Edges	$v - 1$
Chromatic number	2 if $v > 1$

https://en.wikipedia.org/wiki/Tree_(graph_theory)

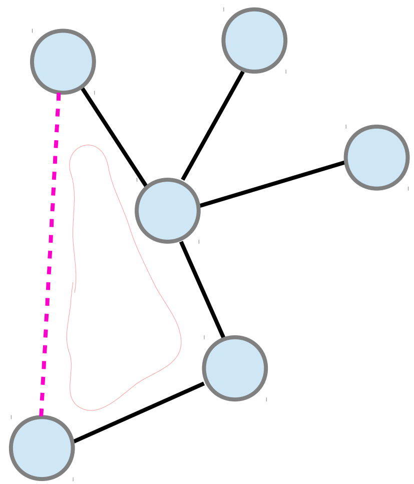
Tree Condition (1)

A **tree** is an **undirected** graph G
that satisfies any of the following equivalent conditions:

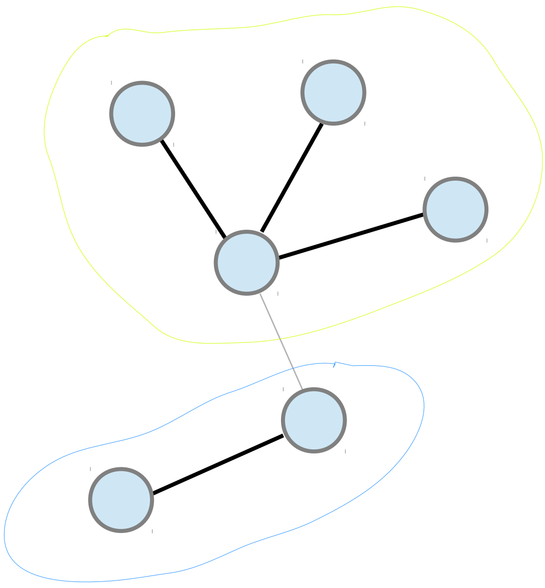
G is **connected** and has <u>no</u> **cycles**.
G is **acyclic**, and a **simple cycle** is formed if any **edge** is <u>added</u> to G.
G is **connected**, but is <u>not</u> **connected** if any single **edge** is <u>removed</u> from G.
G is **connected** and the 3-vertex complete graph $\mathbf{K_3}$ is not a **minor** of G.
Any **two vertices** in G can be **connected** by a <u>unique</u> **simple path**.

https://en.wikipedia.org/wiki/Tree_(graph_theory)

G is **<u>acyclic</u>**, and a **simple cycle** is formed if any **edge** is <u>added</u> to G.



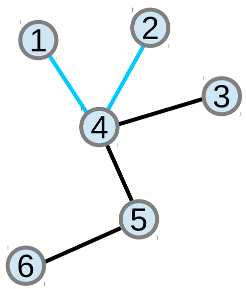
G is **<u>connected</u>**, but is <u>not</u> **connected** if any single **edge** is <u>removed</u> from G.



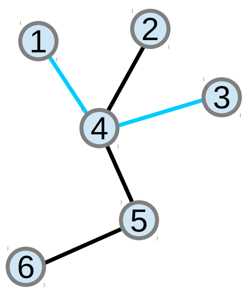
https://en.wikipedia.org/wiki/Tree_(graph_theory)

Tree Condition (3)

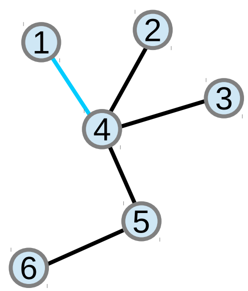
$p_{1,2}$



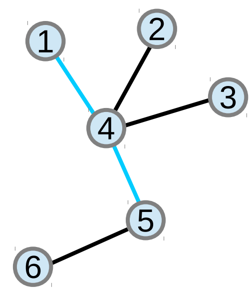
$p_{1,3}$



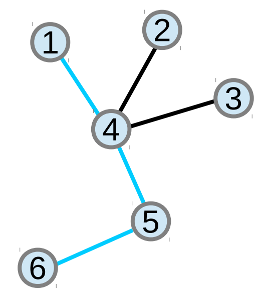
$p_{1,4}$



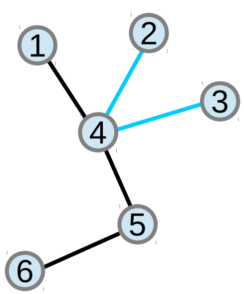
$p_{1,5}$



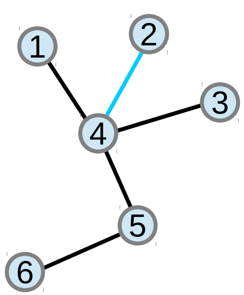
$p_{1,6}$



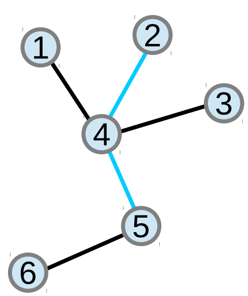
$p_{2,3}$



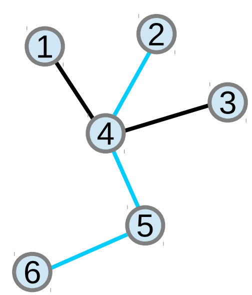
$p_{2,4}$



$p_{2,5}$



$p_{2,6}$

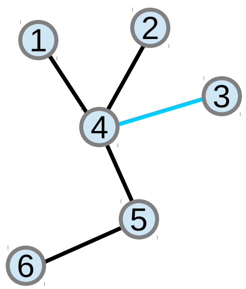


Any **two vertices** in G can be **connected** by a <u>unique</u> **simple path**.

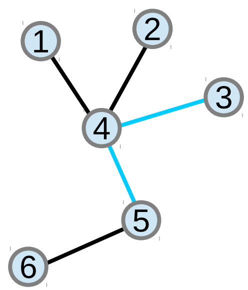
https://en.wikipedia.org/wiki/Tree_(graph_theory)

Tree Condition (4)

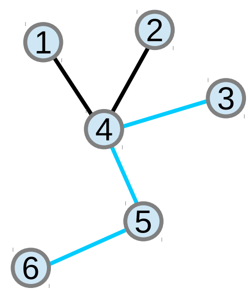
$p_{3,4}$



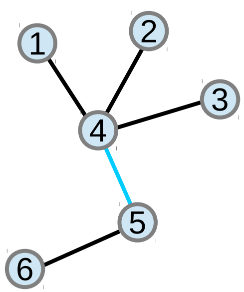
$p_{3,5}$



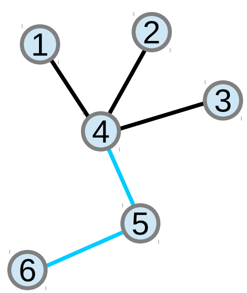
$p_{3,6}$



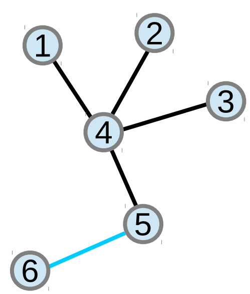
$p_{4,5}$



$p_{4,6}$



$p_{5,6}$



Any **two vertices** in G can be **connected** by a <u>unique</u> **simple path**.

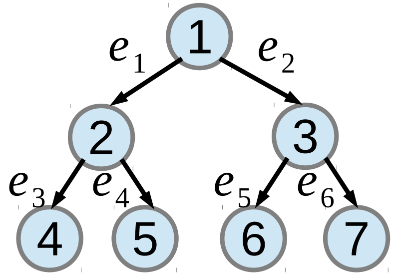
https://en.wikipedia.org/wiki/Tree_(graph_theory)

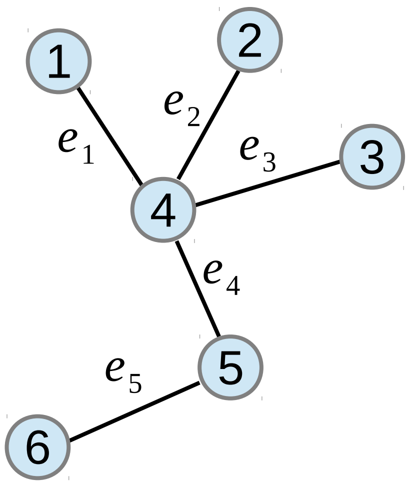
Tree Condition (5)

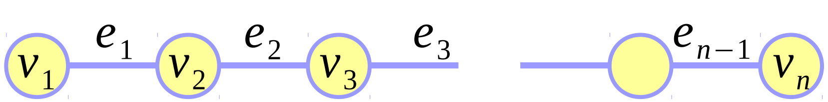
If G has finitely many **vertices**,
say **n vertices**, then the above statements
are also equivalent to any of the following conditions:

G is **connected** and has **n − 1 edges**.

G has **no simple cycles** and has **n − 1 edges**.



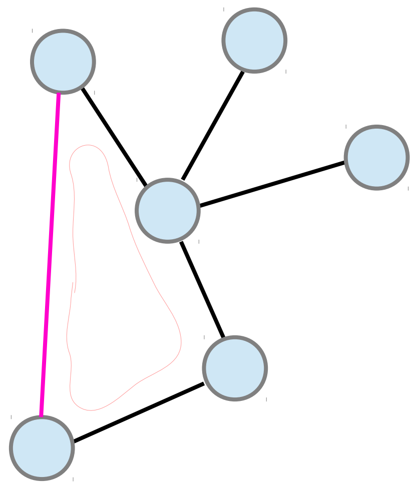


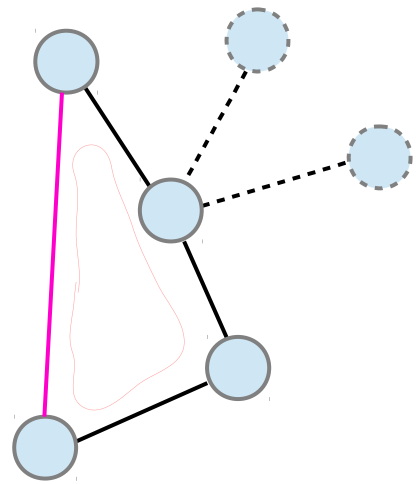


https://en.wikipedia.org/wiki/Tree_(graph_theory)

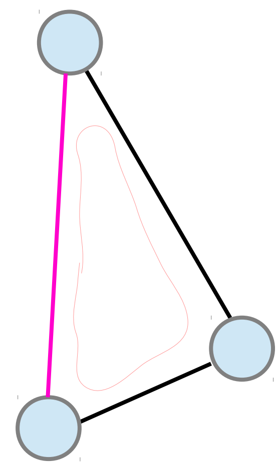
Tree Condition (6)

G is **connected** and the 3-vertex complete graph $\mathbf{K_3}$ is not a **minor** of G.





deleting edges
deleting vertices

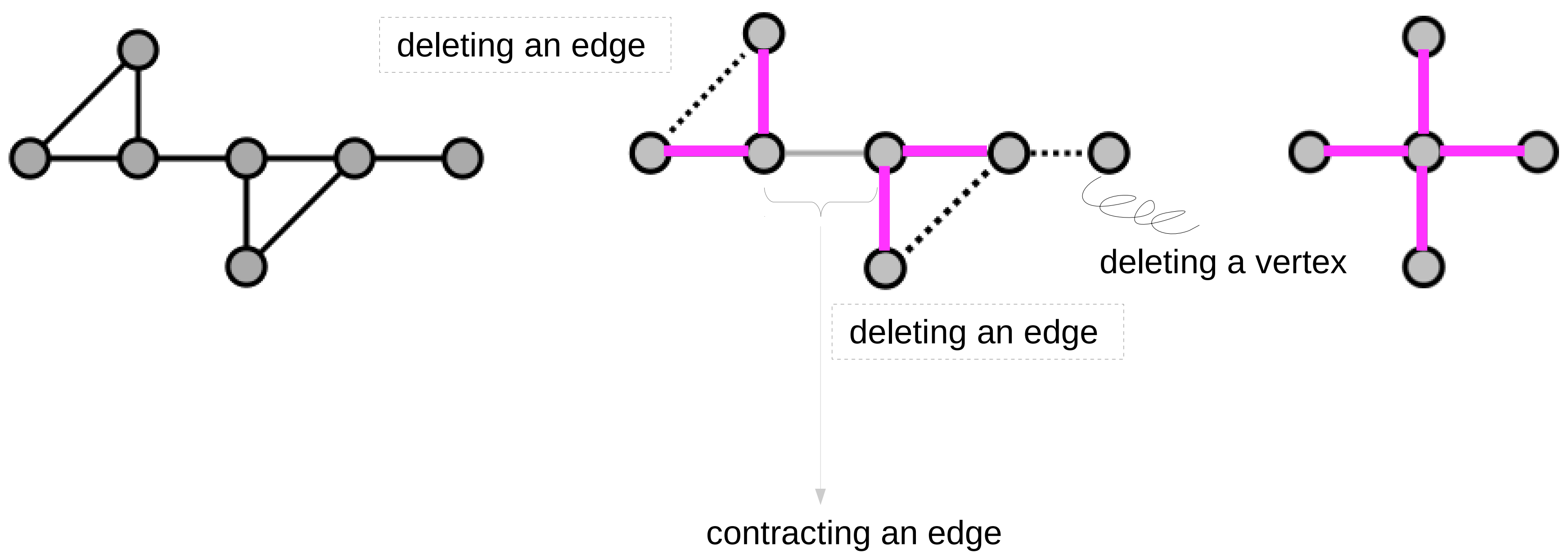


contracting edges

https://en.wikipedia.org/wiki/Tree_(graph_theory)

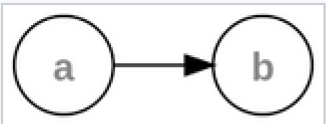
Graph Minor

In graph theory, an undirected graph H
is called a minor of the graph G
if H can be formed from G
by **deleting edges** and **vertices** and
by **contracting edges**.



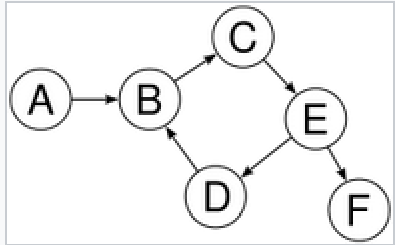
https://en.wikipedia.org/wiki/Graph_minor

Tree Examples

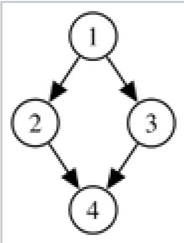


Each linear list is trivially a tree

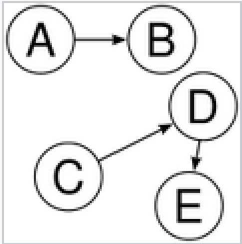
Not a tree: cycle A→A. A is the root but it also has a parent.



Not a tree: cycle B→C→E→D→B. B has more than one parent (inbound edge).



Not a tree: undirected cycle 1-2-4-3. 4 has more than one parent (inbound edge).



Not a tree: two non-connected parts, A→B and C→D→E. There is more than one root.

https://en.wikipedia.org/wiki/Tree_(data_structure)

Terminology used in trees (1)

Root

The top node in a tree.

Child

A node directly connected to another node when moving away from the Root.

Parent

The converse notion of a child.

Siblings

A group of nodes with the same parent.

Descendant

A node reachable by repeated proceeding from parent to child.

Ancestor

A node reachable by repeated proceeding from child to parent.

https://en.wikipedia.org/wiki/Tree_(data_structure)

Terminology used in trees (2)

Leaf (less commonly called **External node**)

A node with no children.

Branch (**Internal node**)

A node with at least one child.

Degree

The number of subtrees of a node.

Edge

The connection between one node and another.

Path

A sequence of nodes and edges connecting a node with a descendant.

https://en.wikipedia.org/wiki/Tree_(data_structure)

Terminology used in trees (3)

Level

The level of a node is defined
by 1 + (the number of connections between the node and the root).

Height of node — **Depth**

The height of a node is the number of edges
on the longest path between that node and a leaf.

Height of tree — **Depth**

The height of a tree is the height of its root node.

Depth — **Height**

The depth of a node is the number of edges
from the tree's root node to the node.

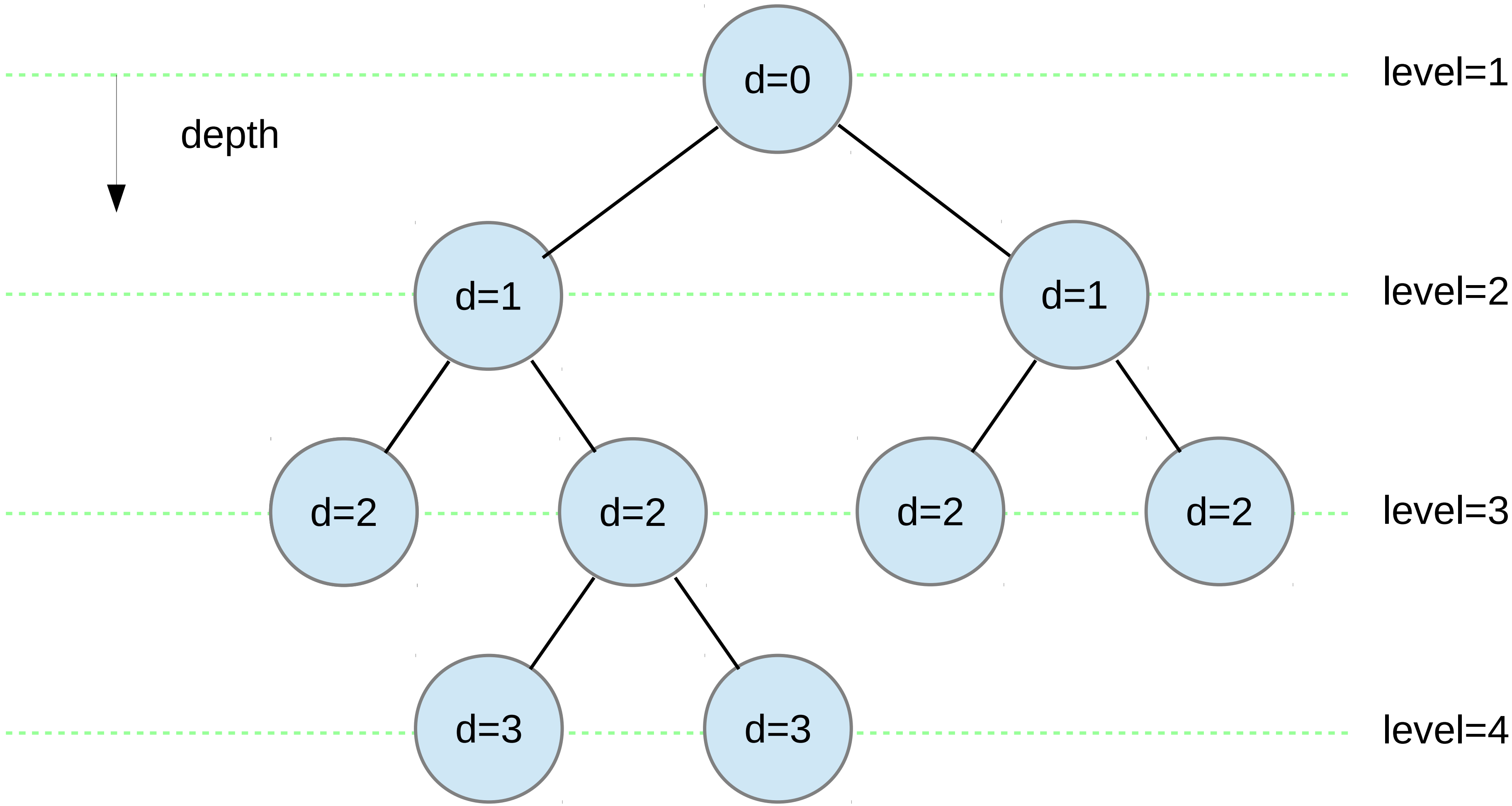
Forest

A forest is a set of $n \geq 0$ disjoint trees.

Some literatures have the reversed definitions of height and depth

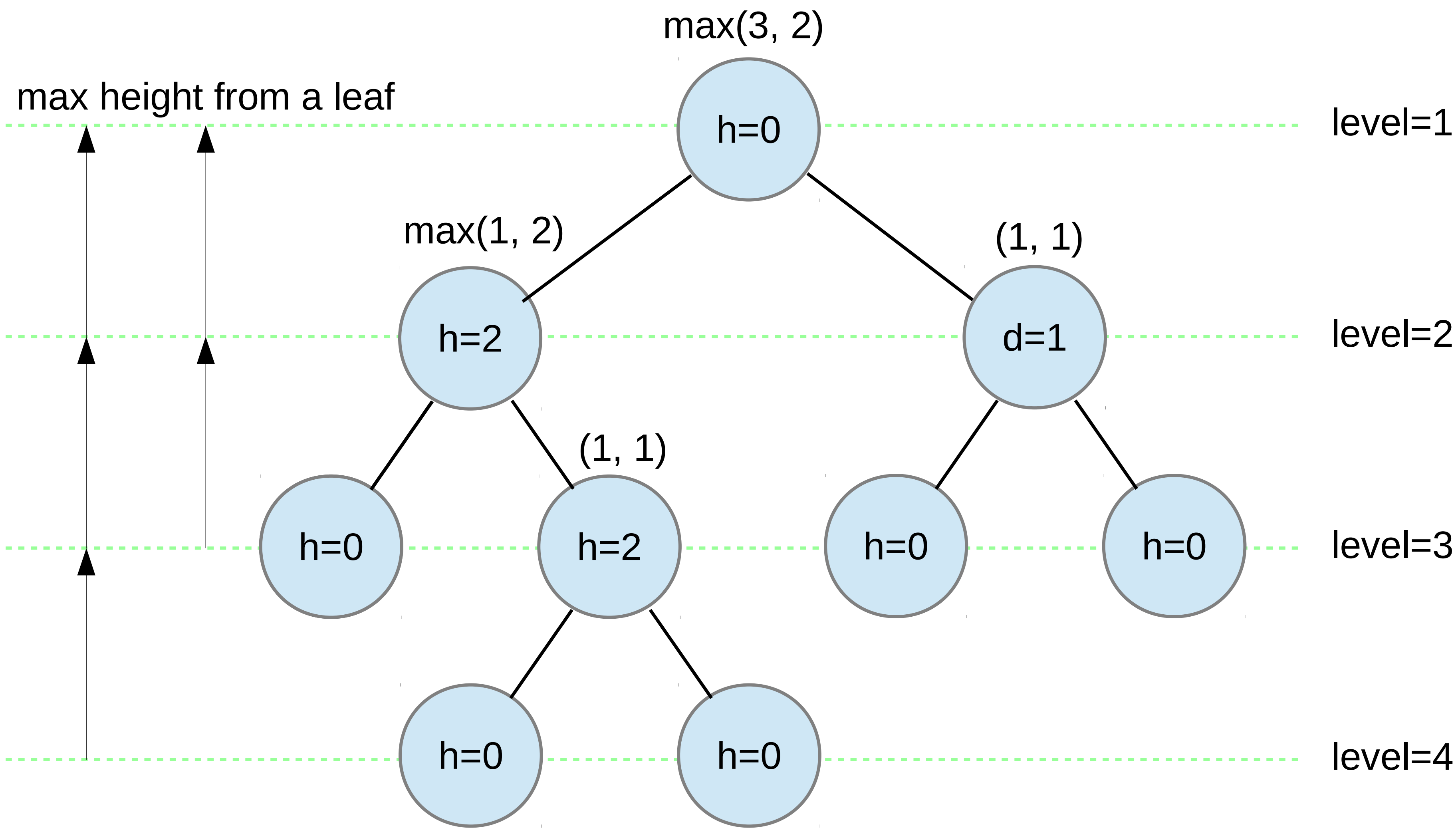
https://en.wikipedia.org/wiki/Tree_(data_structure)

Depth



https://en.wikipedia.org/wiki/Tree_(data_structure)

Height



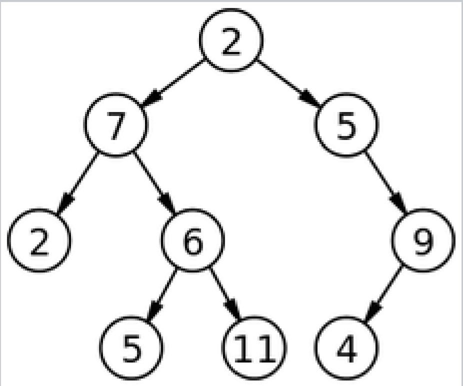
https://en.wikipedia.org/wiki/Tree_(data_structure)

Binary Tree

a **binary tree** is a tree data structure in which
each **node** has <u>at most</u> <u>two</u> **children**,
(the **left child**, the **right child**)

A **recursive definition** using just set theory notions
is that a (non-empty) binary tree is a tuple **(L, S, R)**,
where **L** and **R** are **binary trees** or the **empty set** and
S is a **singleton set**.

Some authors allow the binary tree
to be the empty set as well.



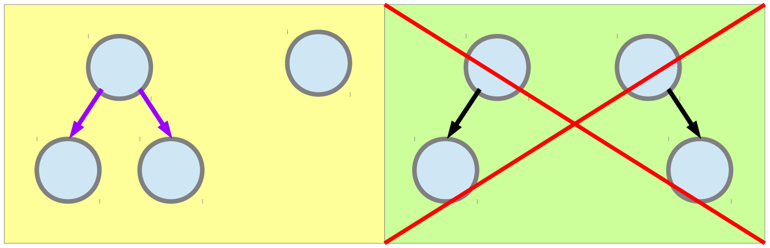
A labeled binary tree of size 9 and height 3, with a root node whose value is 2. The above tree is unbalanced and not sorted.

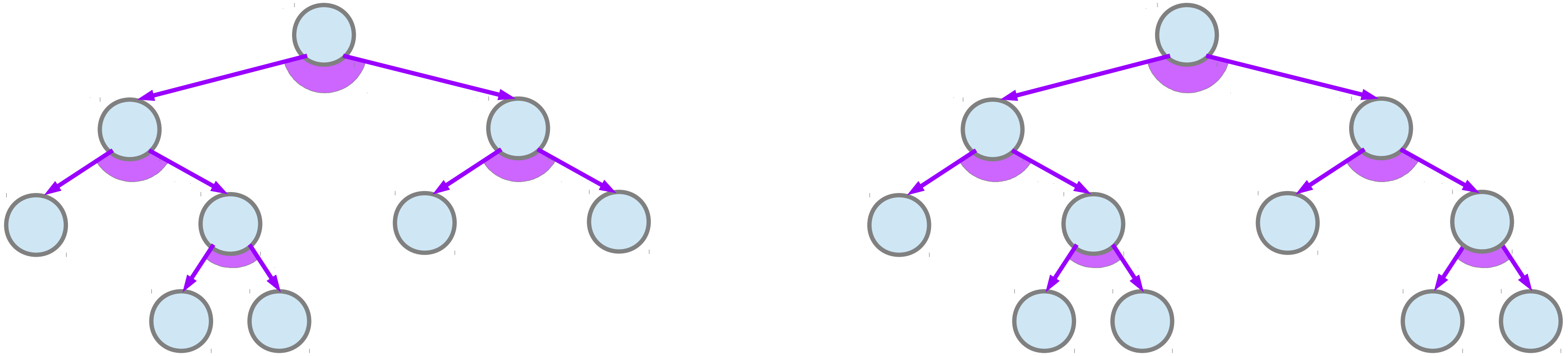
https://en.wikipedia.org/wiki/Binary_tree

Full Binary Tree

A **rooted binary tree** has a **root node** and every **node** has <u>at most</u> <u>two</u> **children**.

A **full binary tree** is
(**proper, plane binary tree**)
a **tree** in which every **node**
has either **0** or **2 children**.





https://en.wikipedia.org/wiki/Tree_(graph_theory)

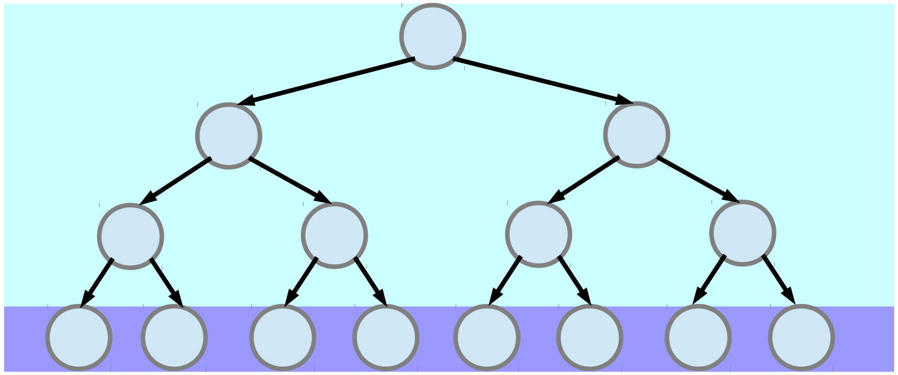
Perfect Binary Trees

A **perfect** **binary tree** is a binary tree in which
all **interior nodes** have two **children** and
all **leaves** have the same **depth** or same **level**.

also called a **complete binary tree**

two **children**

the same **depth** (**level**).

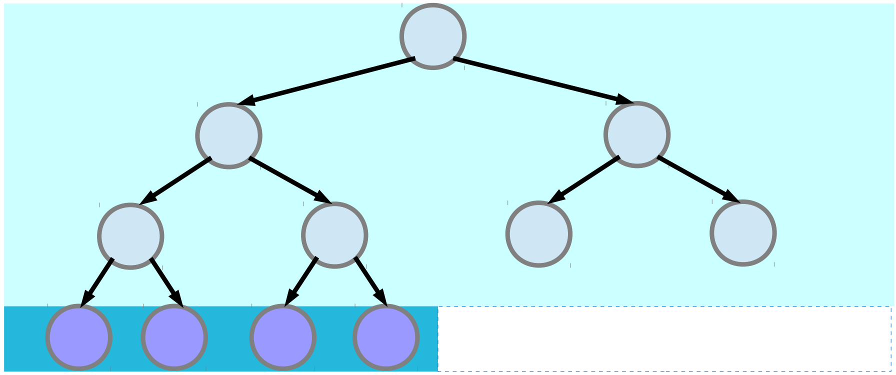


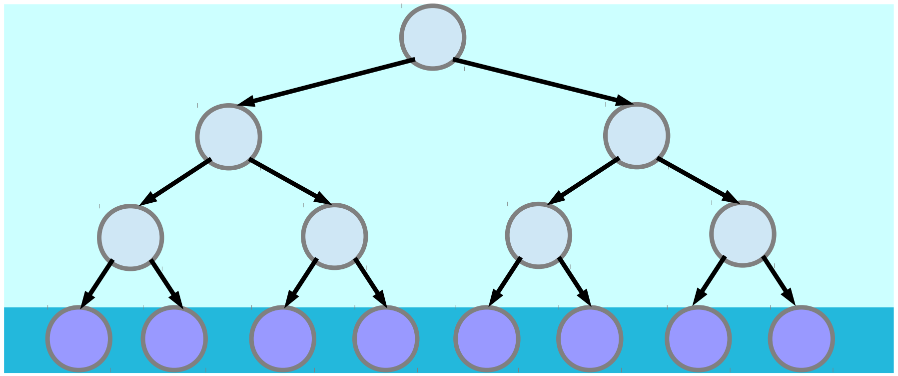
https://en.wikipedia.org/wiki/Tree_(graph_theory)

Complete Binary Trees

In a **complete** **binary tree**
every level, except possibly the last,
is completely filled,
and all nodes in the last level are
as far left as possible.

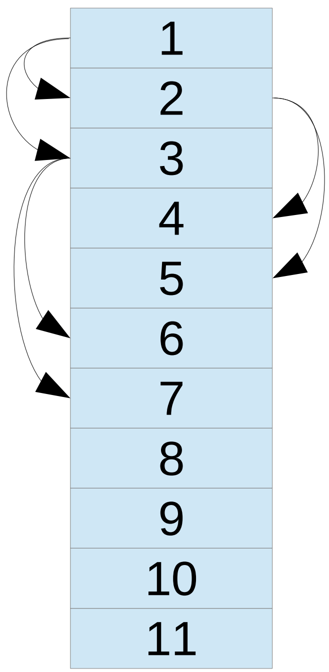
An alternative definition is a **perfect** **tree**
whose rightmost leaves (perhaps all)
have been removed.





https://en.wikipedia.org/wiki/Tree_(graph_theory)

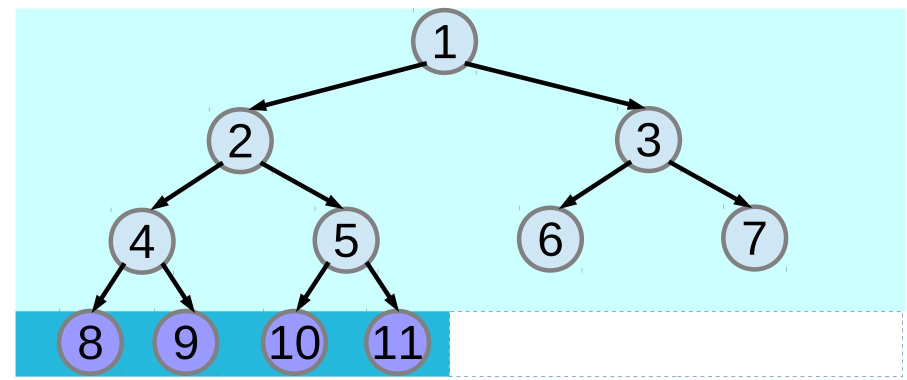
Complete Binary Trees and Linear Arrays

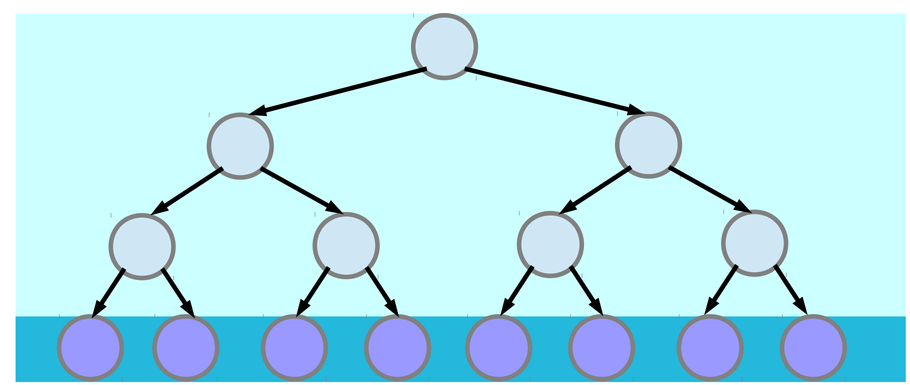


$2 \cdot i$ Left child

$2 \cdot i + 1$ Right child

A complete binary tree can be efficiently represented using an array.





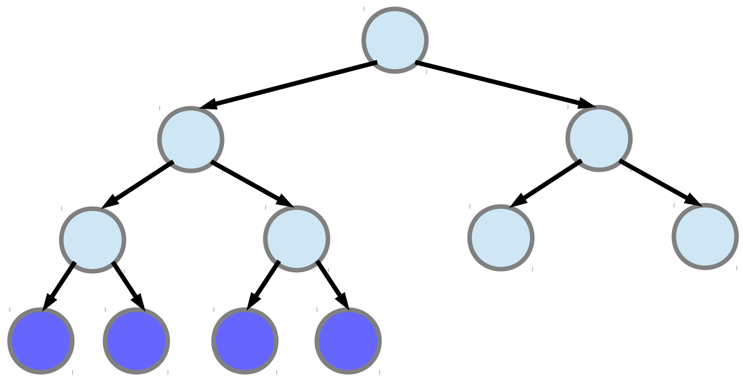
contiguous
no blanks
→ complete

https://en.wikipedia.org/wiki/Tree_(graph_theory)

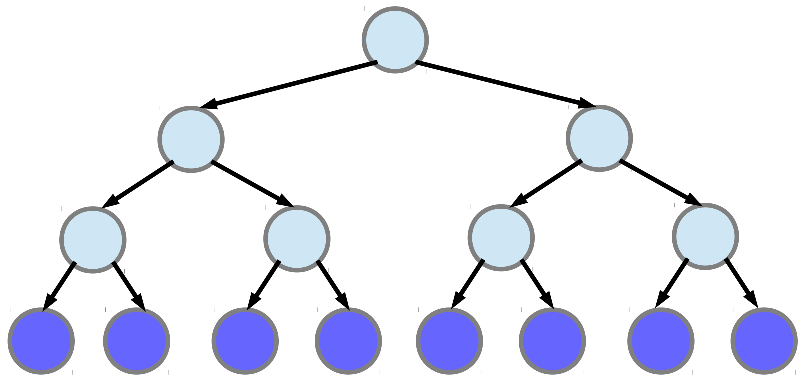
Different use of compute binary trees

Some authors use the term **complete** to refer instead to a **perfect** binary tree as defined above,
in which case they call this type of tree an **almost complete binary tree** or **nearly complete binary tree**.

complete	nearly complete



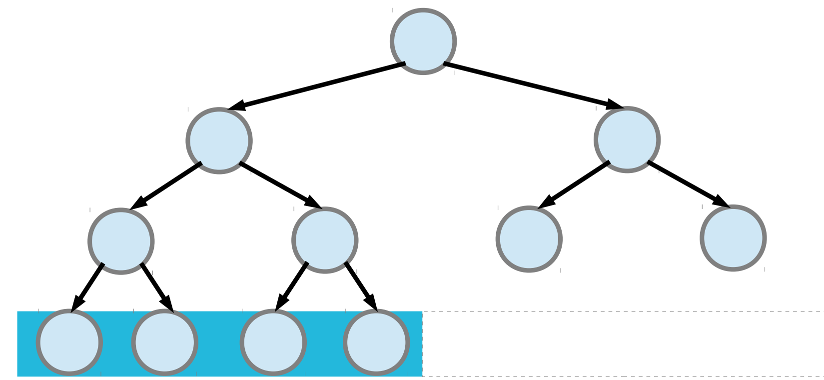
perfect	complete



https://en.wikipedia.org/wiki/Tree_(graph_theory)

Properties of Binary Trees (1)

A **complete binary tree**
can have between **1** and **2^{m-1}** nodes
at the last level **m**.

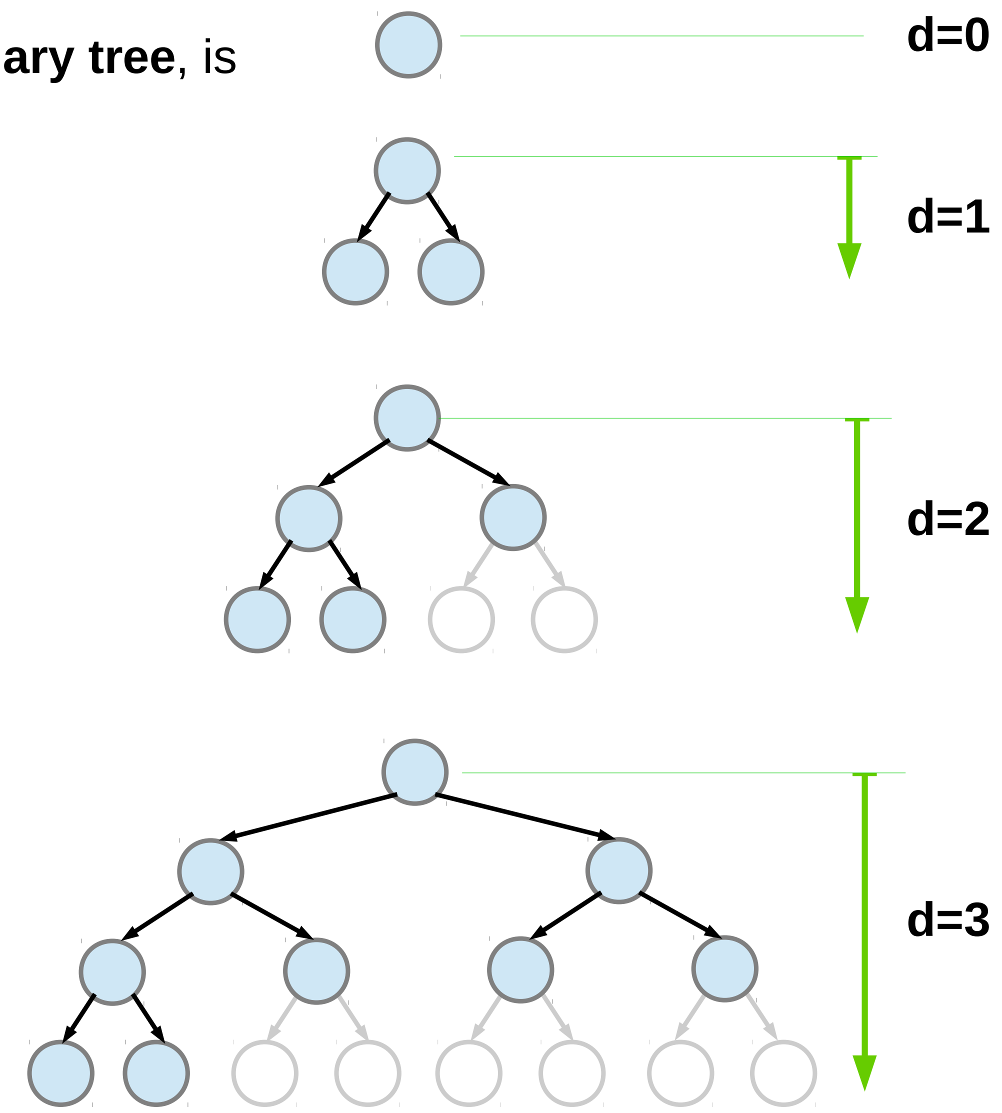


https://en.wikipedia.org/wiki/Tree_(graph_theory)

Properties of Binary Trees (2)

The **number** of **nodes n** in a **full binary tree**, is at least **$n = 2^d + 1$** and at most **$n = 2^{d+1} - 1$**, where **d** is the **detph** of the tree.

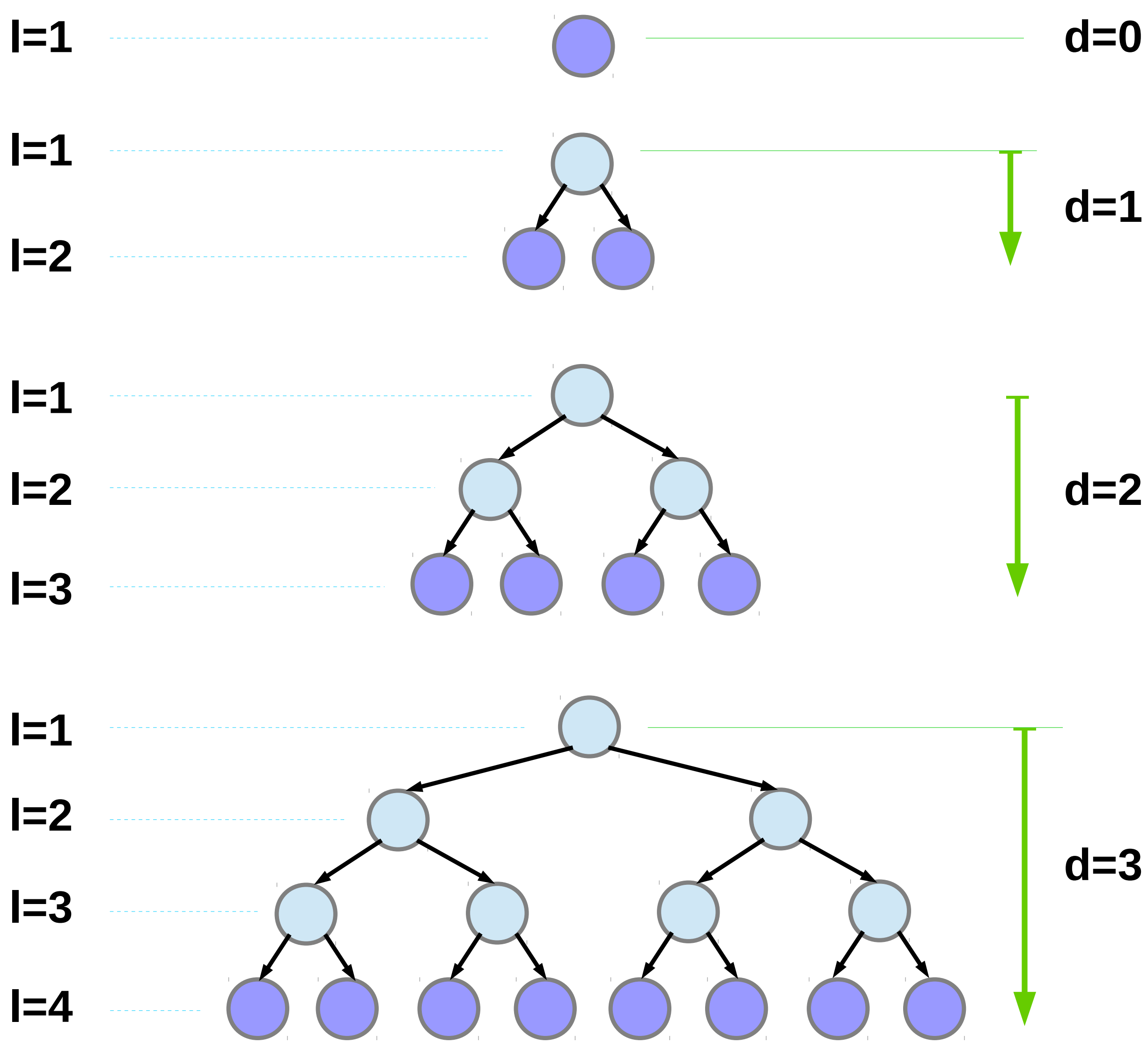
A tree consisting of only a **root node** has a **depth** of **0**.



https://en.wikipedia.org/wiki/Tree_(graph_theory)

Properties of Binary Trees (3)

$2^{d+1}-1$	2^d	2^0
$2^{d+1}-1$	2^d-1	2^1-1
	2^d	2^1
$2^{d+1}-1$	2^d-1	2^2-1
	2^d	2^2
$2^{d+1}-1$	2^d-1	2^3-1
	2^d	2^3



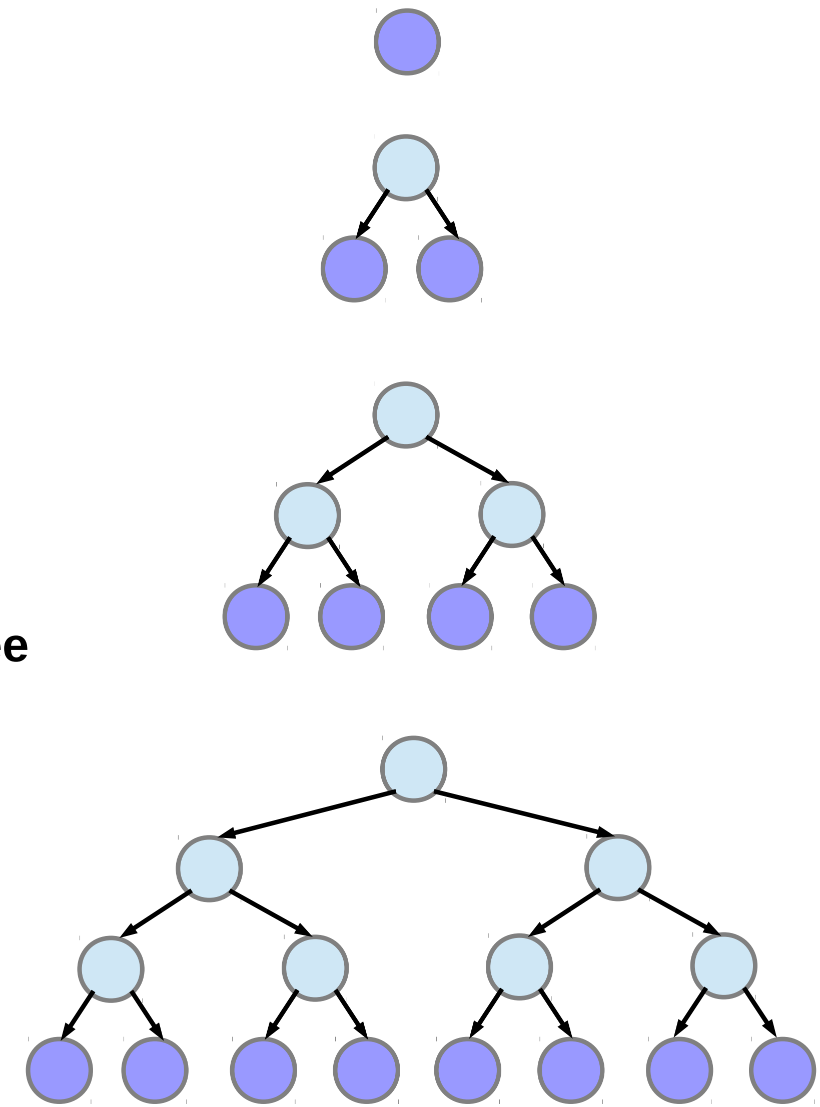
https://en.wikipedia.org/wiki/Tree_(graph_theory)

Properties of Binary Trees (4)

The number of **leaf nodes** is **m**
in a **perfect binary tree**,
is **m=(n+1)/2**

because the number of **non-leaf** (**internal**) **nodes** is **m–1**

This means that a **perfect binary tree** with **m leaves** has
n = 2m–1 nodes.



m	2^0

m-1	2^1-1
m	2^1

m-1	2^2-1
m	2^2

m-1	2^3-1
m	2^3

https://en.wikipedia.org/wiki/Tree_(graph_theory)

References

[1] http://en.wikipedia.org/
[2]