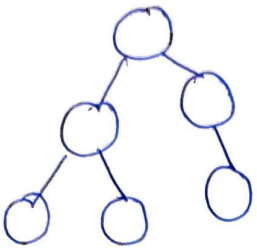
7. Introduction to Tree Algorithms

→ A tree is an undirected graph with no cycles.

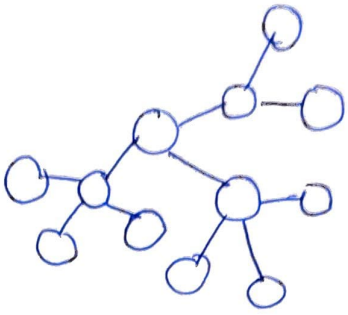
→ equivalently, a tree is a connected graph with n nodes and (n-1) edges.

ex.

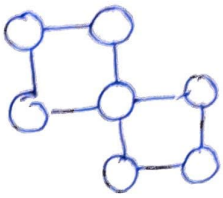
4 nodes
3 edges



6 nodes
5 edges



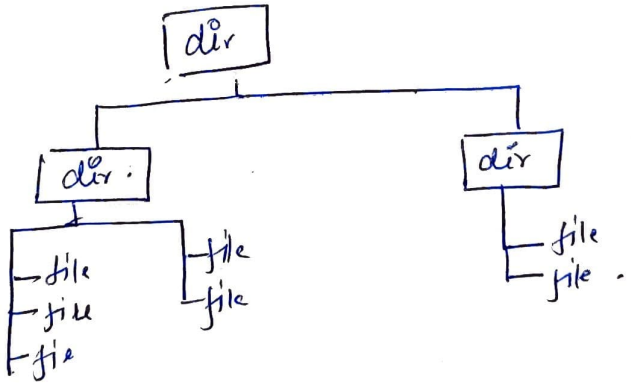
13 nodes
12 edges



7 nodes
8 edges

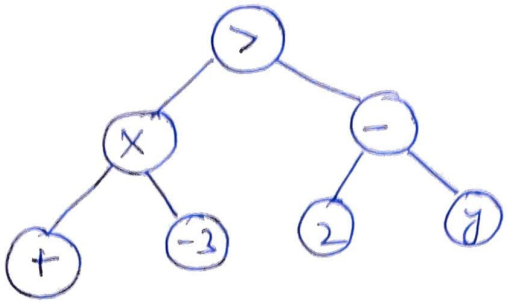
X.

→ Filesystem structures are inherently trees.



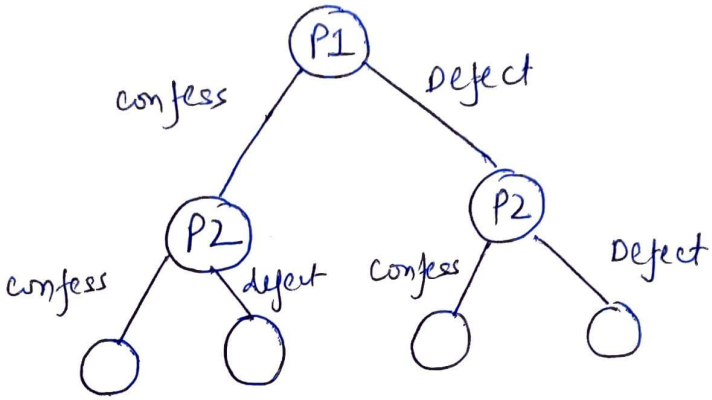
→ Abstract syntax trees to decompose source code and mathematical expressions for easy evaluation.

$$((x+6) * -3) > (2-y)$$



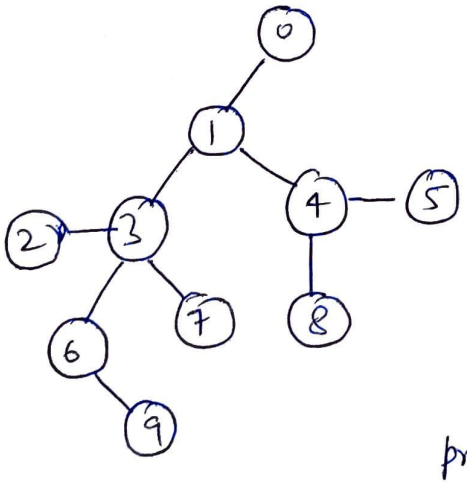
→ Every webpage is a tree as an HTML DOM structure

→ Trees in game theory (Prisoner's dilema)



⇒ storing undirected trees

index all nodes from $[0, n)$.



edge list representation.

[(0,1), (1,4), (4,5), (4,8), (1,3), (3,7), (3,6), (2,3), (6,9)].

pro.

→ super fast and easy to use.

con.

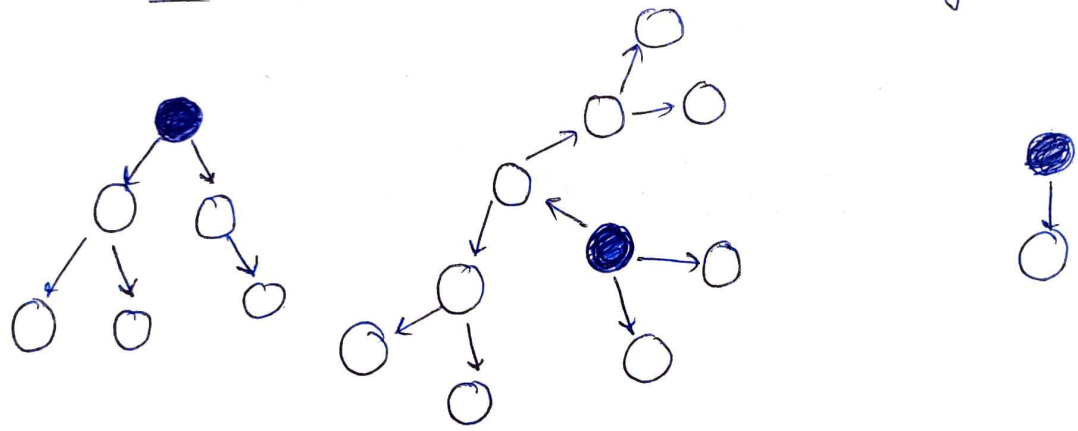
→ storing a tree as a list lacks the structure to do efficiently query all the neighbours of a node.

→ adjacency list representation

0 → [1]
1 → [0, 3, 4]
2 → [3]
3 → [1, 2, 6, 7]
4 → [1, 5, 8]
5 → [4]
6 → [3, 9]
7 → [3]
8 → [4]
9 → [6]

→ can also be represented as adjacency matrix ($O(n^2)$ space). → not used often.

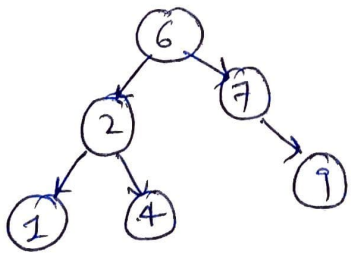
⇒ Rooted trees → Tree with a designated root node.



⇒ Binary trees → every node has almost two child nodes.

⇒ Binary search tree (BST)

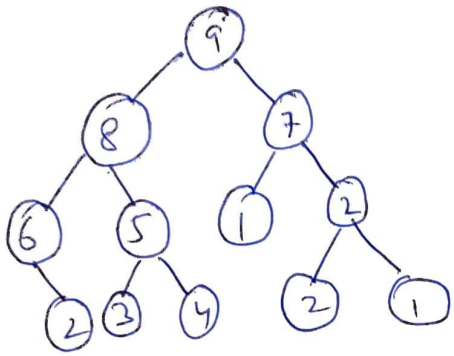
node.left → val ≤ node → val ≤ node.right → val.

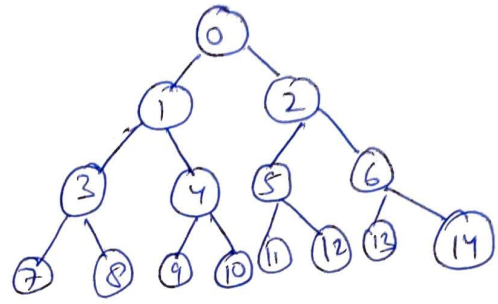


⇒ Storing rooted trees

Maintain a pointer to root node so you can access the tree and its contents. Each node has a list of reference pointers to children.

→ Flattened array representation





0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
9	8	7	6	5	1	2	ϕ	2	3	4	ϕ	ϕ	2	1

Root node is indexed 0

Let i be the index of current node.

left node → $2i + 1$

right node → $2i + 2$

Reciprocally, parent of node i is $\left\lfloor \frac{i-1}{2} \right\rfloor$