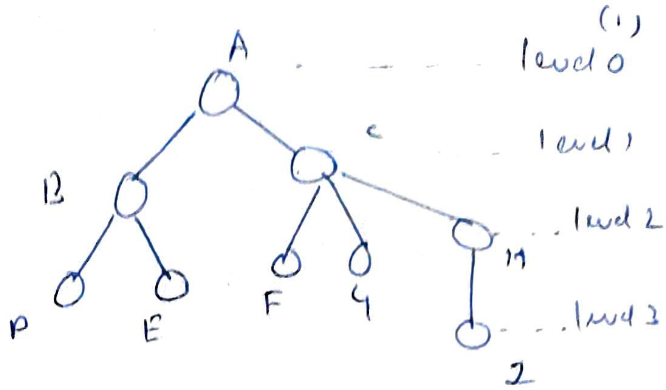
Trees



→ A is root node

→ B is parent of D & E

→ A is ancestor of D & E.

→ D & E are descandants of A

→ C is sibling of B.

→ D & E are children of B.

→ D, E, F, G, H, I are leaves of the tree

↓ nodes with no children.

→ A, B, C, H are internal nodes.

→ The depth (level) of F is 2.

→ height of tree → max. level of any node = 3.

→ Degree of node → no. of children.

↳ degree of node B is 2.

→ Tree can represent heirarchy in an organisation or a file system in an operating system.

Binary tree

ordered tree → Tree in which children of each node are ordered.

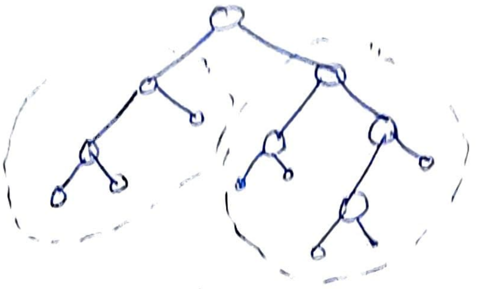
Binary tree → ordered tree with all nodes having atmost 2 children.

Recursive defn of a binary tree

A binary tree is either -

(a) a leaf or

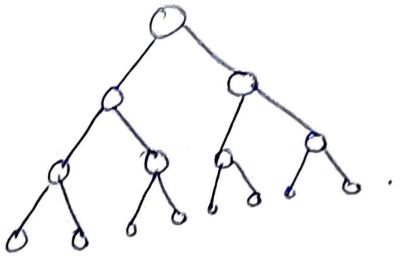
(b) An internal node (the root) and one / two binary trees (left subtree and/or right subtree).



Take a node and attatch a left subtree and/or a right subtree.

Complete binary tree

→ level i has 2^i nodes.



→ In a tree of height h.

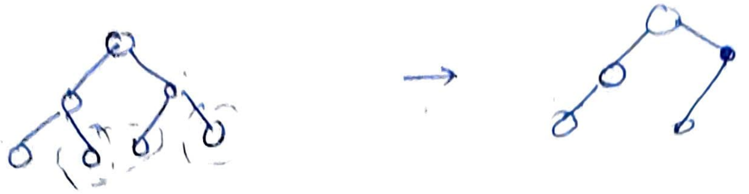
- No of leaves = 2^h
- No of internal nodes $= 1 + 2 + 2^2 \ldots 2^{h-1} = 2^h - 1$
- No of internal nodes = no of leaves − 1
- Total no of nodes $= 2^{h+1} - 1 = n$.

→ In a tree of n nodes.

(1) No of leaves $= \left(\frac{n+1}{2}\right)$

(2) height $= h = \log_2\left(\frac{n+1}{2}\right) = \log_2$(no of leaves)

→ A binary tree can be obtained from an appropriate complete binary tree by pruning.



(every node has atmost 2 children).

Minimum height of a binary tree

→ A binary tree of height h has.

- → At most 2^i nodes at level i
- → At most $1+2+2^2 \dots 2^h = 2^{h+1}-1$ nodes.

→ If the tree has n nodes, then.

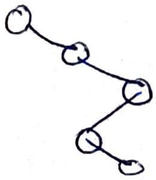
$$n \le 2^{h+1}-1$$

Hence, $$\underbrace{h \ge \log_2\left(\frac{n+1}{2}\right)}_{\text{lower bound on } h}.$$

Maximum height of a binary tree

→ n nodes → max. height = (n−1).

→ obtained when every node (except leaf) has exactly one child.



n = 5

height = 4.

Thus height of a binary tree h.

$$\log\left(\frac{n+1}{2}\right) \leq h \leq (n-1).$$

No of leaves in a binary tree

→ No of leaves ≤ 1 + no of internal nodes.

Proof: by induction on no of internal nodes.

→ Tree with 1 node has a leaf but no internal node.

$$1 \leq 1+0.$$

→ Assume stmt is true for all trees with atmost $(k-1)$ internal nodes.

i.e no of leaves $\leq k$.

→ Now for a tree with k internal nodes. k_1 internal nodes in left subtree, then there are exactly $(k-k_1-1)$ internal nodes in right subtree.

for k_1 internal nodes subtree

$$l_1 \leq (k_1+1)$$

$$l_2 \leq (k-k_1-1)+1.$$

$$\Rightarrow l_1+l_2 \leq \cancel{k_1}+1+k-\cancel{k_1}-\cancel{1}+\cancel{1}$$

$$\leq k+1.$$

Hence, proved.

leaves in a binary tree

For a binary tree on n nodes.

→ No' of leaves + no' of internal nodes = n.

→ No' of leaves ≤ no' of internal nodes + 1.

→ Hence no' of leaves $\le \frac{(n+1)}{2}$.

→ min no° of leaves = 1.

$$1 \le \text{leaves} \le \left(\frac{n+1}{2}\right).$$

ADT's for trees

(i) generic container methods : size ( ), is Empty( ), $\underbrace{\text{elements ( )}}_{\text{lists out elements of a tree}}$

(ii) positional container methods : positions ( ) → gives a seq of all posn in a tree.

$\underbrace{\text{swap Elements (p q)}}_{\text{p, q posn element swapped}}$, replace Elements (p, e) posn p. element e

(iii) query methods : is Root (p) → ~~returns posn of a root~~.

is Internal(p), is External (p).

(iv) accessor methods : root(). → returns posn of root.

parent (p) → return parent of p

children (p) → return children (seq) of p.

left child (p) → give left child

right child (p) → give right child.

sibling (p).

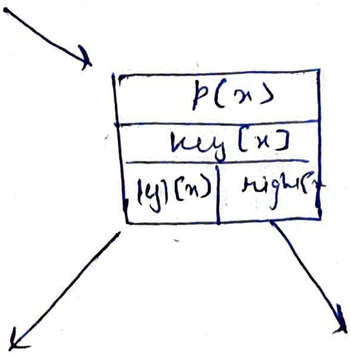
→ Node structure contains -

→ key [x] - key

→ left [x] - pointer to left child

→ right [x] - pt to right child

→ p[x] - pt to parent node.


p(x)
key [x]
left [x]
right[x]