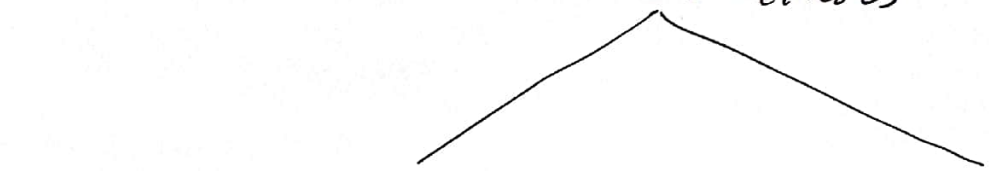
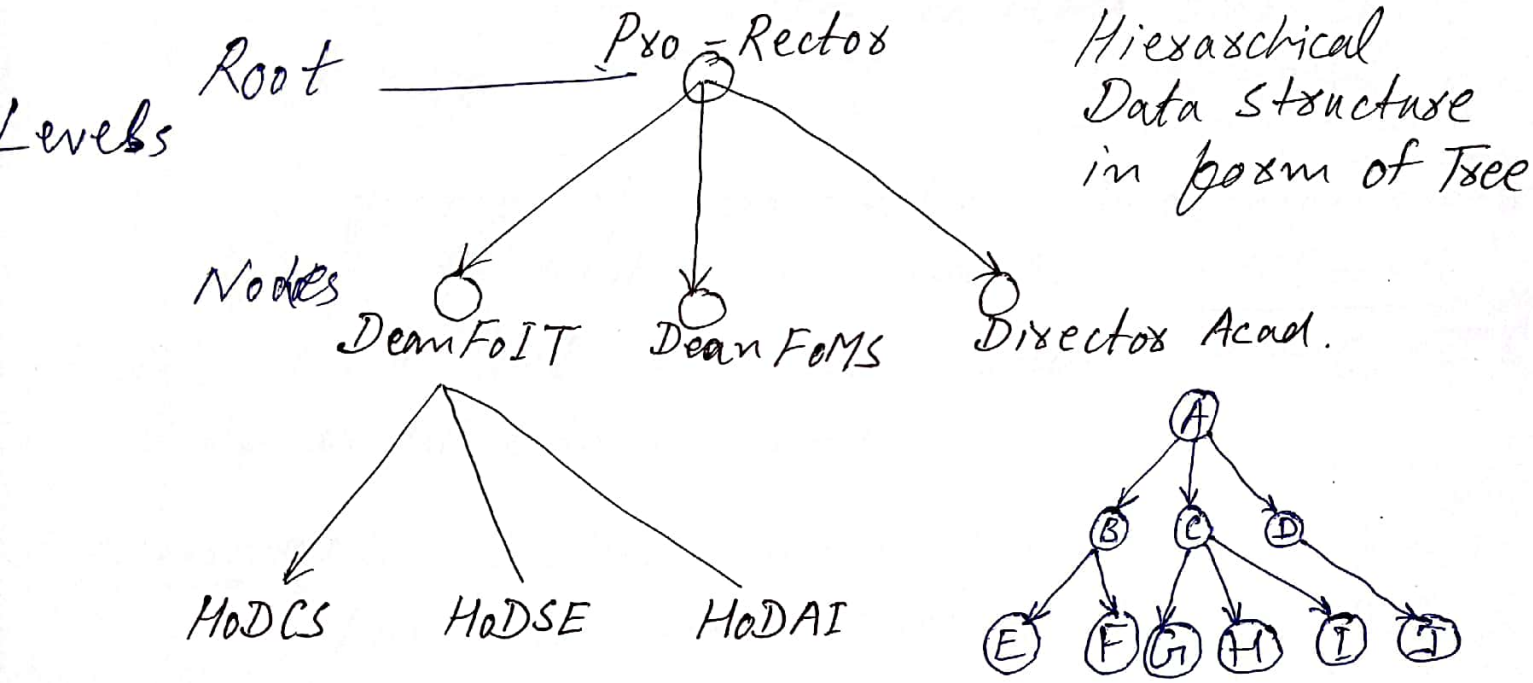
# Data Structures




Hierarchical Data structure in form of Tree

* Tree upside down.
* Collection of nodes containing information linked together in a hierarchy.

**Root** Top most node, without any parent
1st node in the hierarchy. (A)

**Parent Node**

Immediate predecessor of any node is referred to as parent ~~no~~ node.

**Child Node** Immediate successor of any node is its child node.

Parent child relationship. Parents, Grand parents, Child & Grand child.

**Leaf Node** Node without any child node /external node.

**Non-Leaf Node** Node with at least one child. /Internal Node

**Path** Sequence of consecutive edges from source node to destination nodes. A → C → G (Path from A to G)

**Edge** Link between two nodes A → C /directed edge/directed branch/ or simply branch

**Ancestor Nodes** All predecessor nodes from root to a particular node. Ancestor of H → A, C

**Descendant Successor Nodes** All successor nodes on the path from a particular node towards leaf nodes.

**Subtree** Sub part of tree containing a node and all it descendants.

**Sibling** Children of same parent node (E, F), (G, H, I)

**Degree** Total /# of children of a node, not grand children

**Degree of Tree** Maximum degree of any node in tree

**Depth of Node.** Length of path from root to a particular node. (# of edges in the path from root to node of interest). Root has depth 0.

**Height of Node** # of edges in the longest path from node of interest to leaf node.

**Height of Tree** Height of root node.

**Level of Tree** is equal to height of tree.

**Level of Node** is equal to depth of the node.

---

If tree has $n$ nodes then there must be $n-1$ edges in it.

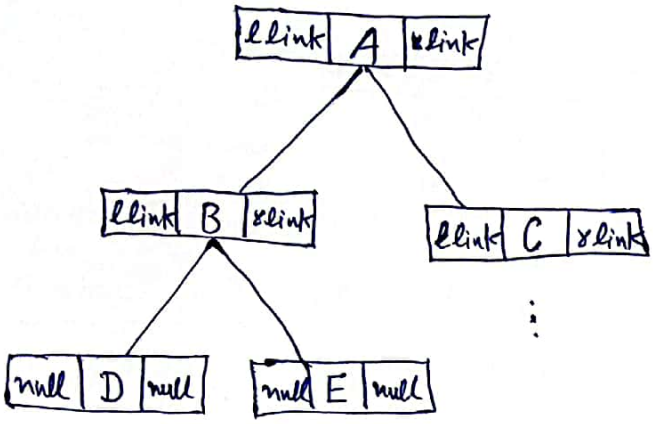
DS

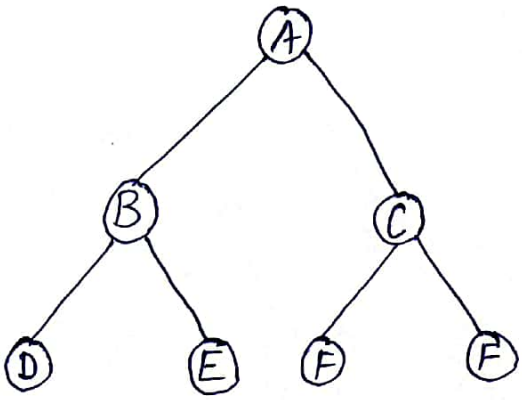
27-12-22

(6)

## Binary Tree

Tree with nodes having at most 2 children. (Max degree = 2)




```
struct node
{
    int data/info;
    struct node * left;
    struct node * right;
};
```

```
class node
{public:
    int data;
    node * left;
    node * right;
};
```

## Applications & Key Features

④ In Compilers (Expression tree)
⑤ Huffman encoding trees used in data compression Algo's
⑥ Priority Queues.

① Hierarchical structures like file system (Directories & subdirectories)

② Routing protocols

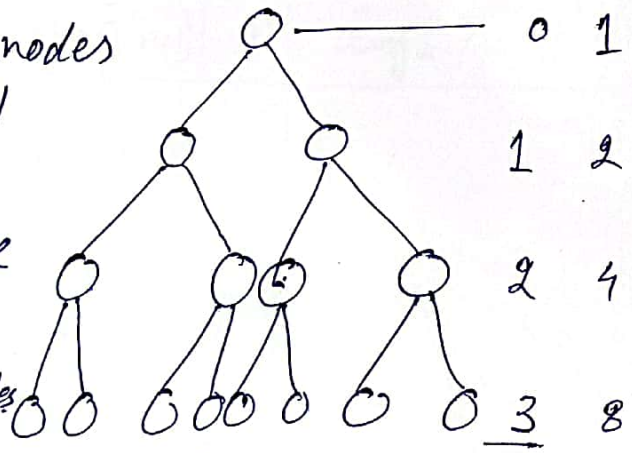
③ Organize data for a quick search/ patter match / features / recognition Face / Thumb (finger) print

* Access/Search is better than linked lists & slower than Arrays.
* Moderate Insertion/Deletion better than Arrays, slower than unordered lists.
* Flexible size.

**Binary Tree** Tree with 0, 1 or 2 children.

**Max # nodes at a level**
* At any level of BT. $2^l$ children are possible.

**Max # of nodes of height h.** $2^{h+1} - 1$ / **Min. nodes** $h+1$




$2^3 + 2^2 + 2^1 + 2^0 = 15$

general case
$2^0 + 2^1 + 2^2 + \dots + 2^h = 2^{h+1} - 1$

/Proper BT/2-tree

**Full BT** A BT with every node having 0 or 2 children or all nodes have 2 children except leaf nodes or BT with every parent node having two or zero children.

**Complete BT**
All levels completely filled except possibly at last level (that too with all nodes as left as possible.

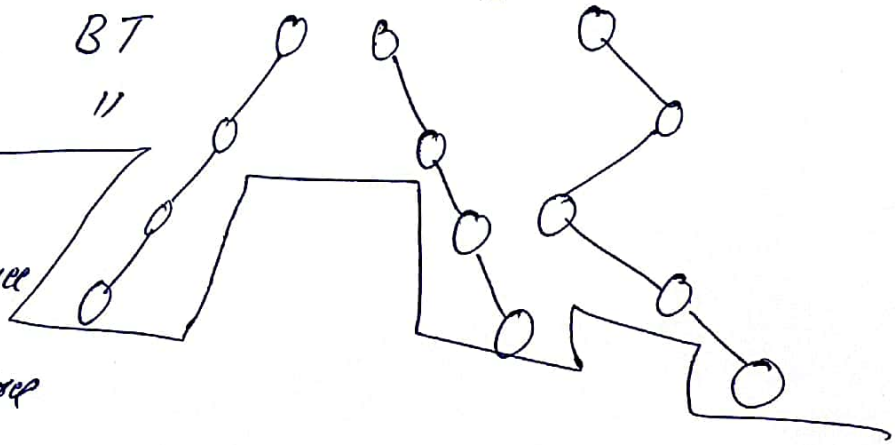
**Perfect BT** BT with all internal nodes has exactly 2 child nodes, all leaf nodes at the same level.

# of leaf nodes = # of internal nodes + 1

# of leaf nodes = (n+1)/2 (n is total number of nodes)

total # of nodes = $2^{h+1} - 1$

**Degenerate Binary Tree**
All internal nodes have only 1 child.
* Left skewed BT
* Right " "



**Balanced Binary Tree**
Height of left & right subtree is differed by at most 1.
This must be valid for each subtree