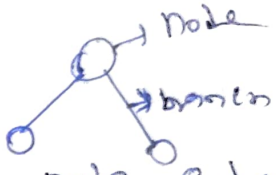
Introduction to Trees

* To Represent algebric formulars.
* Searching, is efficient in large dynamic list.
* For encoding algorithms.

Basic concepts of Trees:

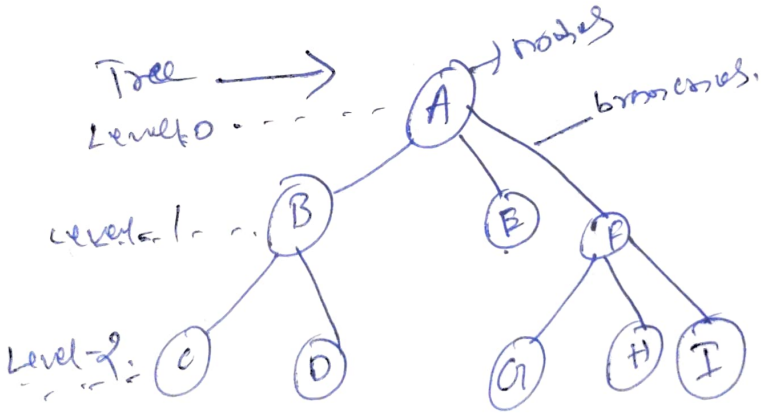
(1) A Tree consists of finite set of elements called Node.

(2) A tree consists of finite set directed lines called branches.



(3) Branch is directed towards the node, indegree, branch.

(3) Branch is directed away from the node, outdegree branch.



Parents: { A, B, F }.

childs: { B, E, F, C, D, G, H, I }.

siblings: { B, E, F } { C, D }, { G, H, I }.

Leaf node: { C, D, G, H, I } (Node does not have childs.) called Leaves.

Internal node: { B, f } parent, one child must be so that it's called Internal node.

Some factors are considered for choosing Data Structures:

* what type of data need to be stored?
* cost of operations. (ex: binary search)
* memory usage.

"

A tree is also one of the data structures that represent hierarchical data.

Properties of Tree:

1. Recursive data structure — minimize the contents.
2. Number of edges
3. Depth of node
4. Height of node.

Applications of tree:

1. Storing naturally hierarchical data.
2. organize data.
3. Trie. — dictionary, dynamic spell checking.
4. Heap
5. B-Tree & B+Tree.
6. Routing table.

Types of Tree data structure:

1. General tree,
2. Binary tree,
3. Binary Search Tree.
4. AVL tree.
5. Red-Black tree
6. Splay tree
7. Treap
8. B-Tree
9. B+ Tree.

Implementation of Trees:

The tree data structure can be created by creating the nodes dynamically with help of the pointers.

Left Data Right.

Address of the Right Child node

Address of left child node