Lecture 5-6

Wednesday, October 6, 2021 8:00 AM

Queue - FIFO

front , rear

↓ ↓

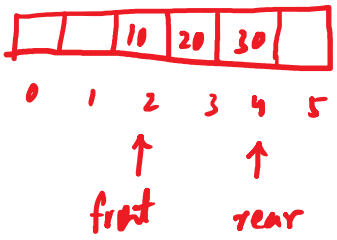
deletion insertion

dequeue enqueue

Initial values

front = -1 or 0 ✓

rear = -1 or -1 ✓



#elements = rear - front + 1

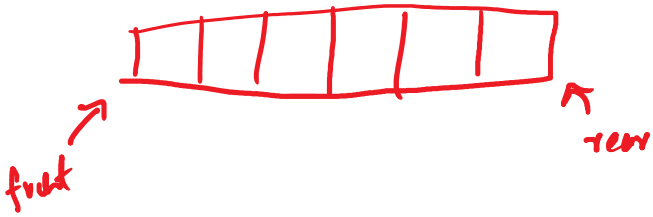
Underflow condition : front > rear

Circular Queue : front = 1 and rear = 4 in above example

Underflow condition : front == rear

Overflow condition : front == rear

Deque : Doubly Ended Queue



Insertion Rear(x)

Insertion Front(x)

Deletion Front()

Deletion Rear()

→ Stack from Queue and vice versa

Non-Linear Data Structure :-

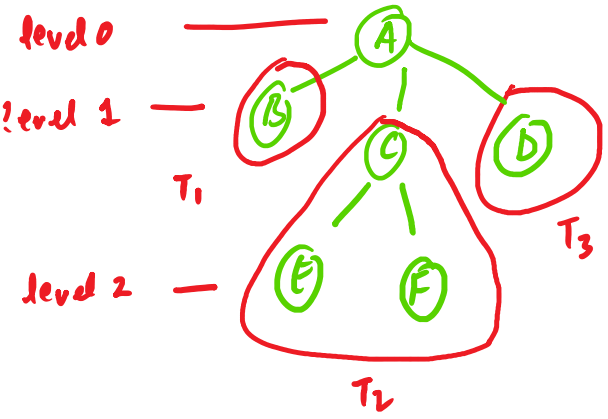
Trees :- Hierarchical DS

Donald Knuth

Tree is a finite set of node, where

1) There is a designated node called the root.

2) The remaining nodes (except the root) can be partitioned into m disjoint subsets $T_1, T_2, \ldots, T_m$; $m \geq 0$ called the subtrees.



node, edge, degree of a node = # children

degree of a tree = max. degree of any node
= 3 in this example

order = # nodes

size = # edges

Leaf node / Terminal node / External node

Non-Leaf nodes / Branch nodes / Internal nodes

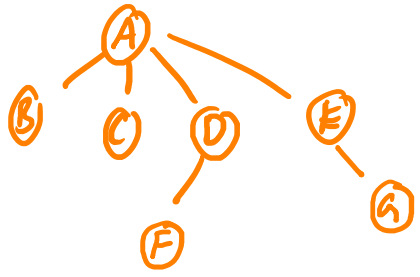
Parent, Child, Ancestor, Descendants

Predecessor, Successor

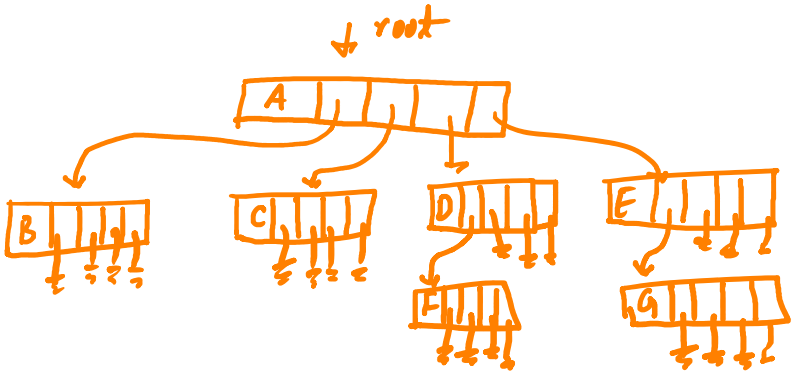
Path, Level

Height Depth

General Tree : K-ary tree — A tree with degree K



4-ary tree

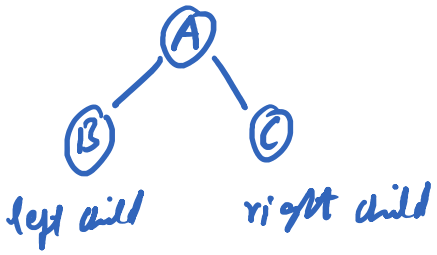


Wastage of memory

Binary Tree :- Donald Knuth

Binary Tree is a finite set of nodes which is either empty or

Can be partitioned into a root, a left subtree and a right subtree.

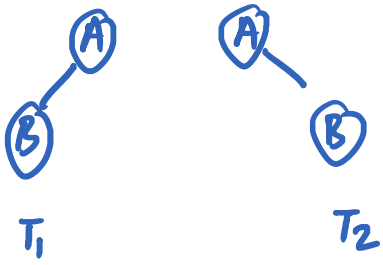


Q- Is Binary Tree is a Tree?

Ans - No

Binary Tree can be empty but a tree cannot

Binary Tree vs 2-ary tree



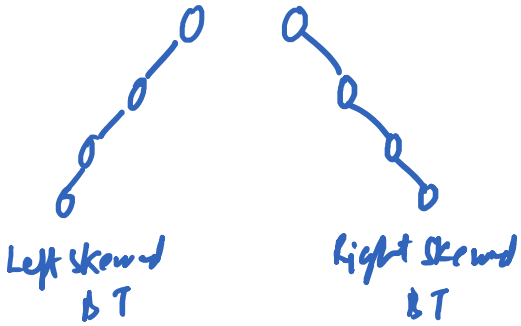
T_1 & T_2 are different binary trees

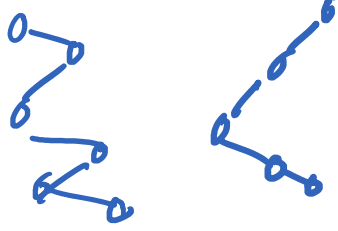
T_1 & T_2 are same 2-ary tree


```
struct btnode
{
   int data;
   struct btnode *lchild, *rchild;
};
typedef struct btnode BTNODE;
```

Special cases of a Binary Tree :-

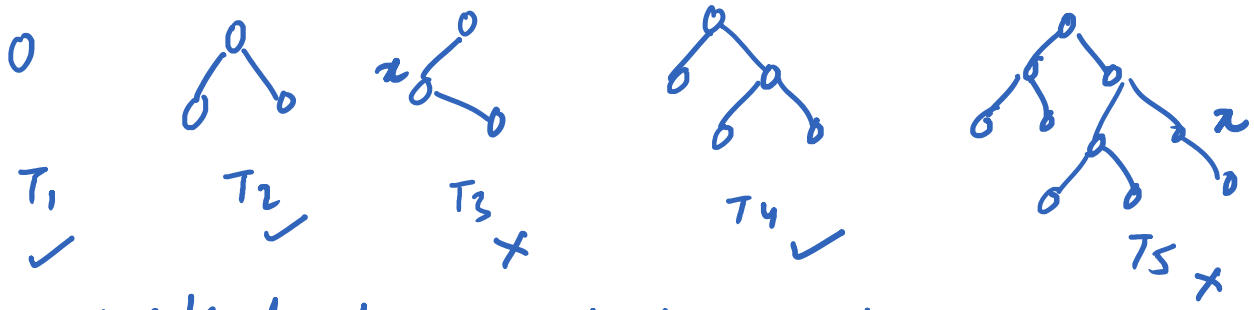
1) Skewed binary Tree - Degree of every internal node is 1





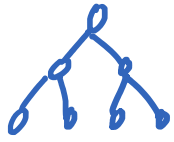
Worst case complexity

2) Strict Binary Tree – Degree of every node is either 0 or 2



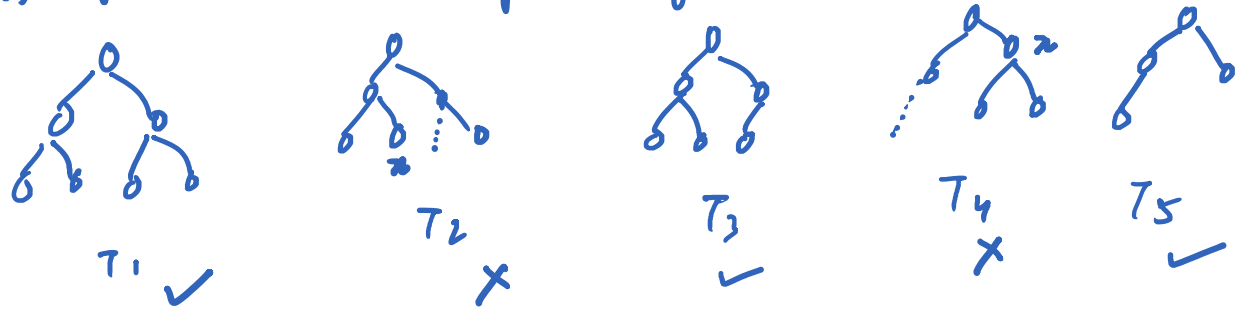
external nodes = # internal nodes + 1

3) Full Binary Tree – All the levels are filled



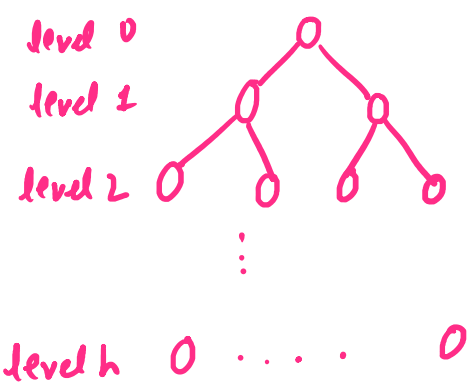
A binary tree with least height & max. no. of nodes.

4) Complete Binary Tree – A binary tree of height 'h' such that all the levels till h-1 are filled and the last level 'h' is filled from left to right.



Full binarytree is a Complete binary tree Heaps

Height of a Binary Tree :-



—	1	2^0
—	2	2^1
—	4	2^2
⋮	⋮	⋮
		2^h

Let n be the no. of nodes

$2^0 + 2^1 + \dots + 2^h = n$

$2^{h+1} - 1 = n$

$2^{h+1} = n+1$

level h 0 . . . . 0 2^h

$2^{h+1} = n+1$

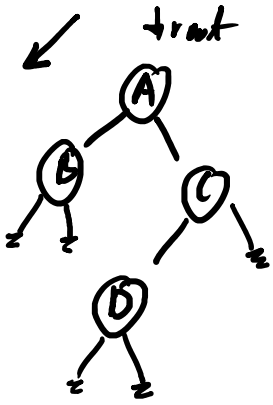
$h+1 = \log_2(n+1)$

$$h = \lceil \log_2(n+1) \rceil - 1$$

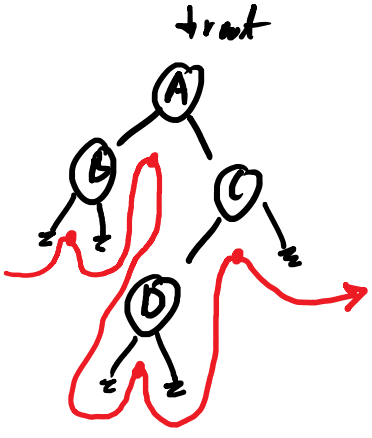
Binary Tree Traversals :-

1) Preorder - Root LST RST
2) Inorder - LST Root RST
3) Postorder - LST RST Root

```
void Preorder (BTNODE *root)
{
    if (root != NULL)
    {
        printf ("%d", root -> data);
        preorder (root -> lchild);
        preorder (root -> rchild);
    }
}
```



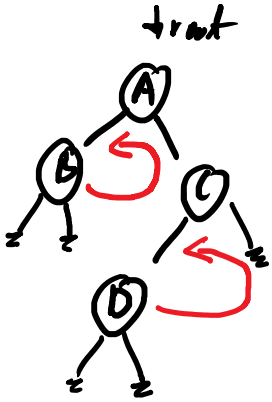
A B C D



Inorder

B A D C

Postorder



B D C A

Construction of Binary Tree :-

Preorder - A B C D

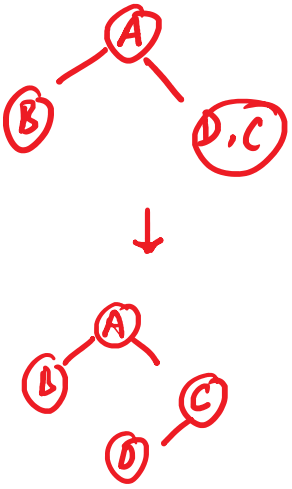
Inorder - B A D C

Preorder Inorder

Preorder — A B C D

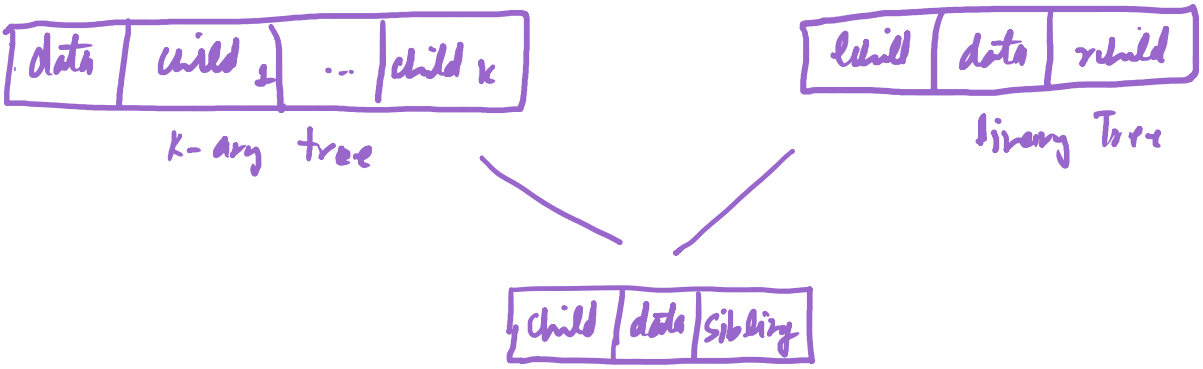
Inorder — B A D C

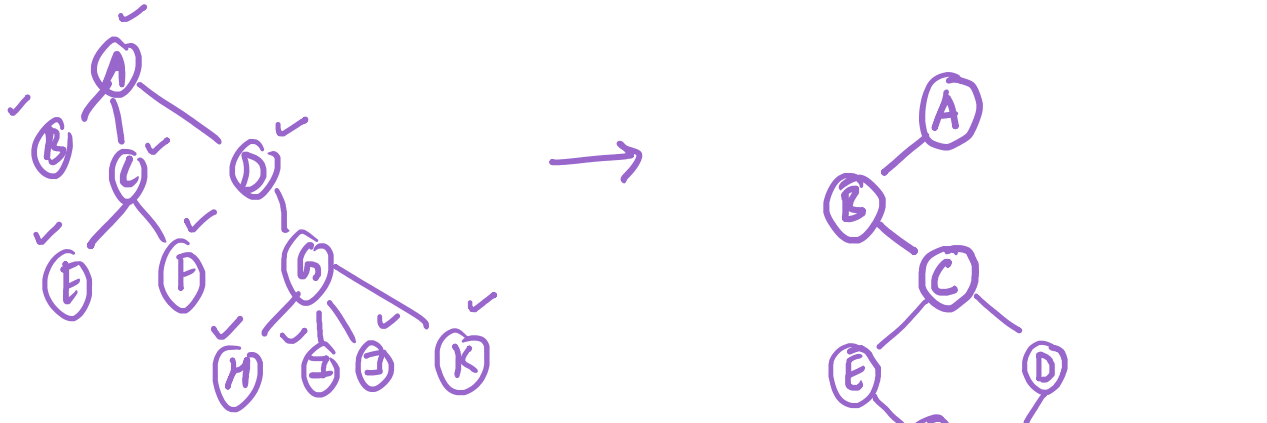
Postorder — B D C A



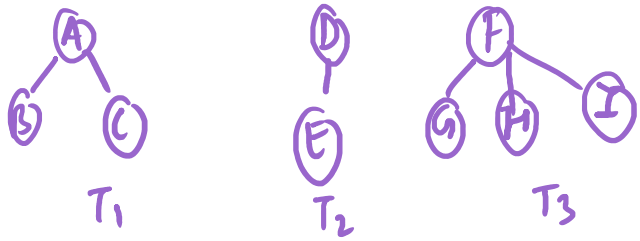
2 traversals are required in which Inorder is mandatory to construct a unique binary tree

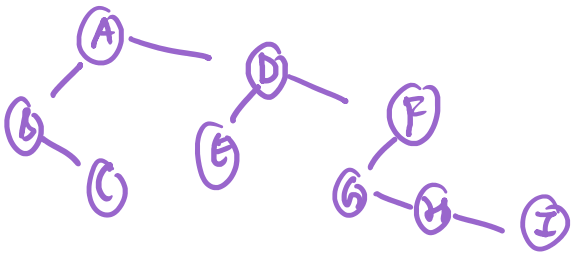
K-ary representation using a Binary Tree





Forest - Collection of Trees




A
B
C
D
E
F
G
H
I