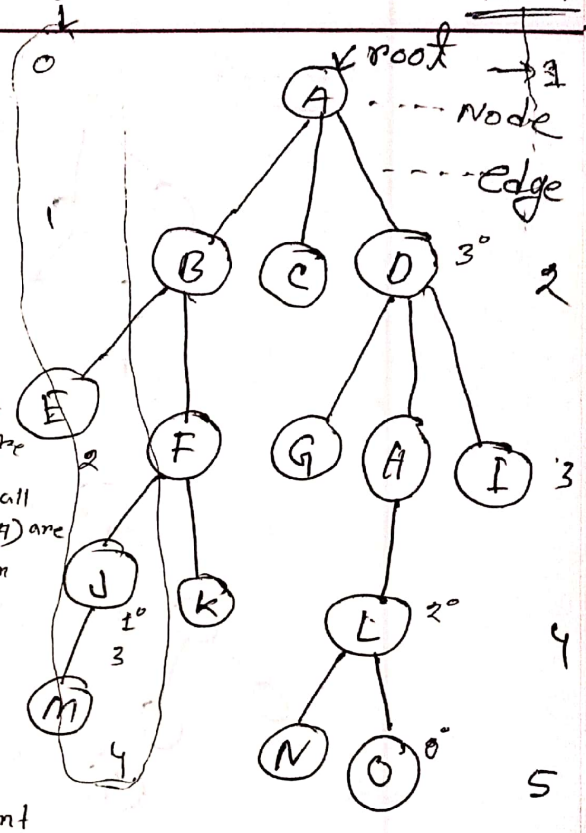
Tree

Lecture 1

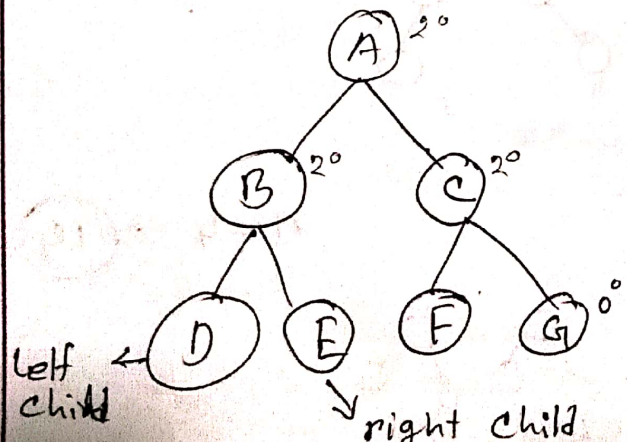
1. Root
2. parent
3. child → D's (G H I)
4. Siblings → (G H I) same parent
5. Descendents → any node rich the tail node → (E, F, J, K, M) are the descendents of B
6. Ancentors → any node to go root nod all are Ancentors (M, J, E, B, A) are
7. Degree a Node → direct children (L) degree 2°
8. Internal (getter than 00) / External node (lef node)
9. levels → (root=1, 2, 3, 4, 5) → Node count
10. Height → (0, 1, 2, 3, 4) → count edge
11. forest → collection of tree is forest, if we remove root (A) the we get (B, C, D) sub tree then it is collection of ~~forest~~ tree



n (vertices/Node)
(n-1) edge

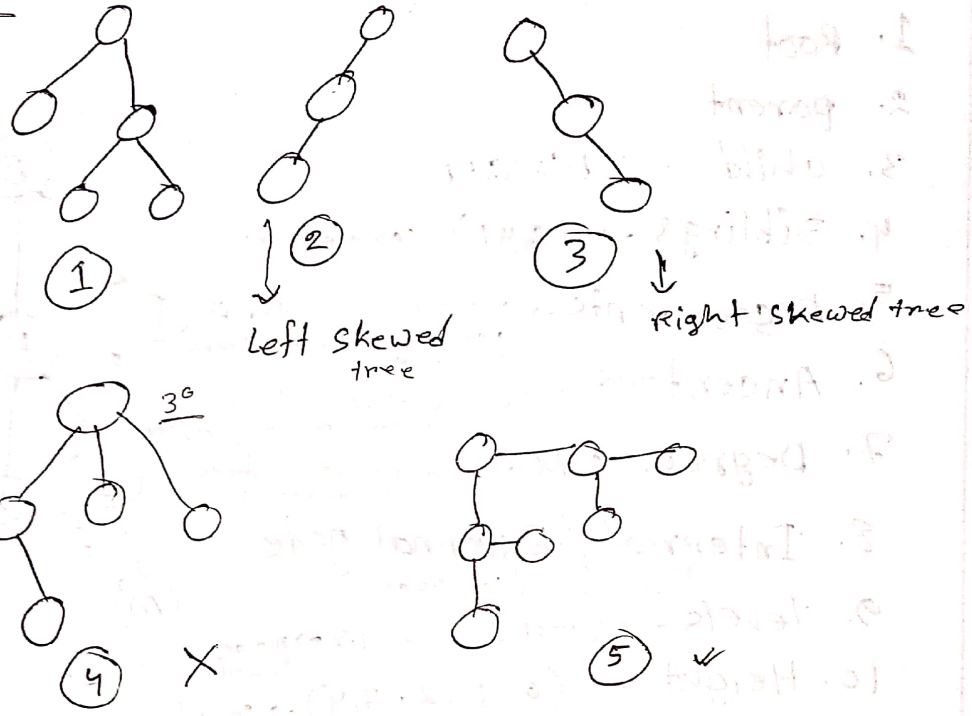
Binary tree

2° degree 2° means binary tree
(maximum 2 chindren)
≤2



(2) degree of ~~for~~ Binary tree = {0, 1, 2}

Example:

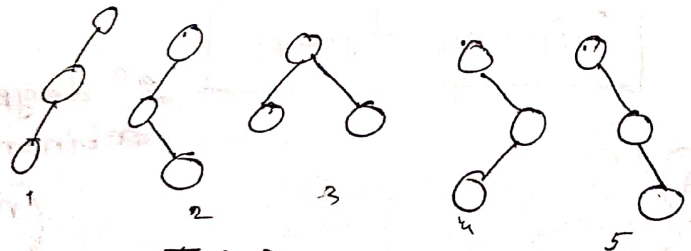


Lecture 2 → Number of Binary tree

n = 3 ← 3 Node

(1) unlabelled node

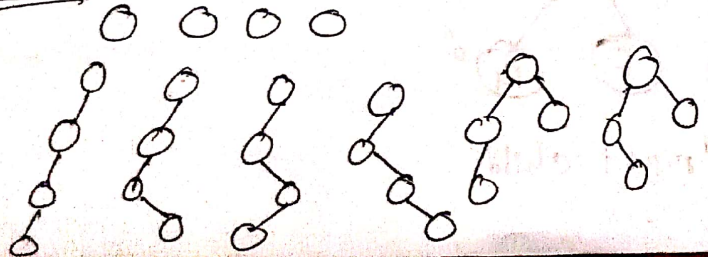
(2) Labelled Nodes



T(3) = 5

n = 4

n = 4 → (14)



(3)

Catalan number $T(n) = \frac{2nC_n}{n+1}$

n number of tree,

$T(3) = 5$

$T(4) = 14$

$T(5) = 42$

To given node

$T(n) = \frac{2nC_n}{n+1}$

$T(5) = \frac{2*5C_5}{5+1} = \frac{10C_5}{6}$

$= \frac{10*9*8*7*6}{5*4*3*2*1} \Big/ 6$

$= 42$

maximum height

$n=3$ max. height $= 4 = 2^2$

$n=4$ " " $= 8 = 2^3$

$n=5$ " " $= 16 = 2^4$

n - - - - - - 2^{n-1}

generate Number of tree using another method.

n	0	1	2	3	4	5	6
$T(n) = \frac{2nC_n}{n+1}$	1	1	2	5	14	42	

$T(6) = 1*42 + 1*14 + 2*5 + 5*2 + 14*1$

$42*1$

$= 132$

$T(6) = \frac{2*6C_6}{6+1} = \frac{12C_6}{7} = 132$

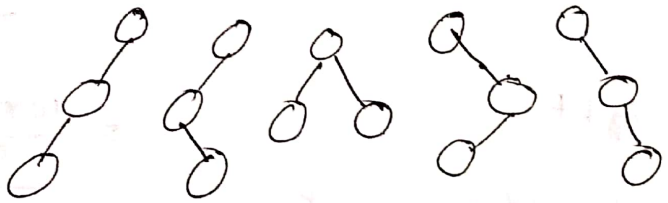
formula $T(n) = \sum_{i=1}^{i=n} T(i-1) * T(n-i)$

(4) Label Node → (n!) way

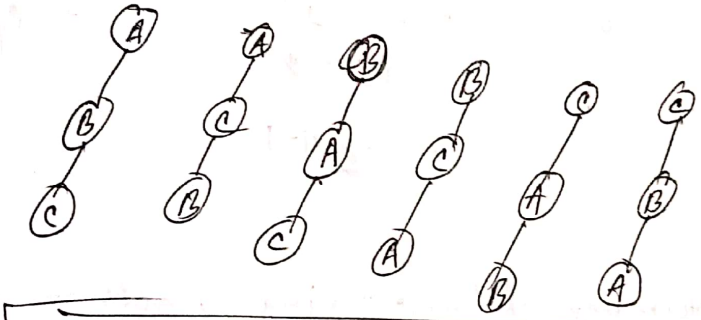
Labelş Node

n = 3

(A), (B), (C)



→ untabells



→ Label node

$$T(n) = \frac{^{2n}C_n}{n+1} * n!$$

↓ shapes ↓ filling data

Lecture: 3

Heigh vs Nodes formula

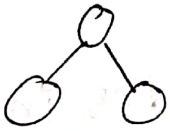
given height

height h=1

Node → min

n=2

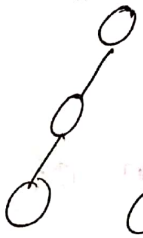
max



n=3

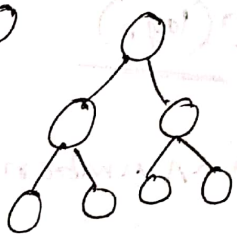
height h=2

min



n=3

max

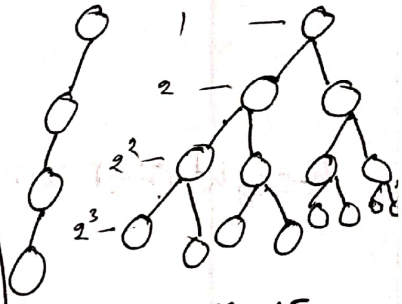


n=7

height h=3

mim

max



n=4 n=15

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

$= 2^{3+1}-1$

$= 16-1$

Gp series $= a+ar+ar^2+\cdots$

$= \dfrac{ar^{k+1}-1}{r-1}$

min Nodes $n = h+1$

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

Node is given

n=3

min

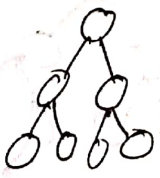
h=1

max

n=2

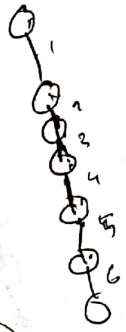
n=7

min



n=2

max



n=6

$1+2+2^2+2^3+\cdots 2^h =$

$a=1$

$r=2$

$= \dfrac{2^{h+1}-1}{2-1}$

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

max height $h = n-1$

min height $h = \log_2(n+1)-1$

$n=7$

$h = \log_2(7+1)-1$

$= \log_2 8 - 1$

$= 3-1 = 2$

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

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

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

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

Range check

Node দেয়া থাকলে → Height of Binary tree

$$\log_2(n+1) - 1 \leq h \leq n-1$$

$O(\log n)$ ——— ——→ $O(n)$

height দেয়া থাকলে → Number of Nodes in a Binary tree

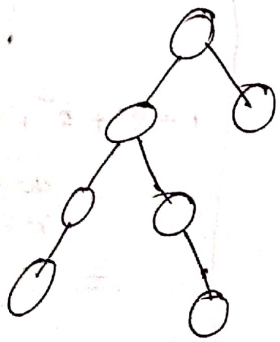
$$h+1 \leq n \leq 2^{h+1} - 1$$

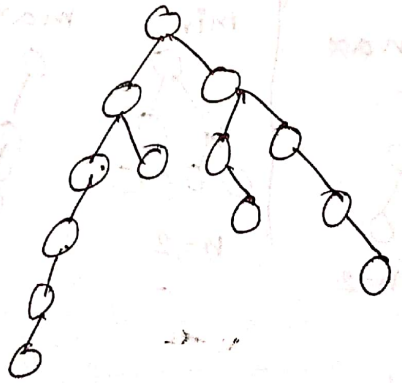
Lecture 4

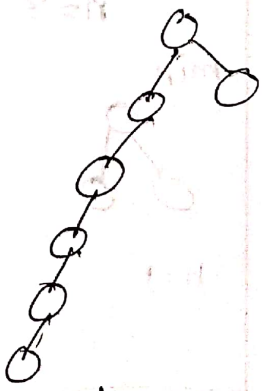
Height vs Node

Internal and external

Non-left







relη

$\deg(2) = 2$
$\deg(1) = 2$
$\deg(0) = 3$

$\deg(0) = \deg(2) + 1$

$\deg(2) = 3$
$\deg(1) = 5$
$\deg(0) = 4$

external

$\deg(2) = 1$
$\deg(1) = 4$
$\deg(0) = 2$

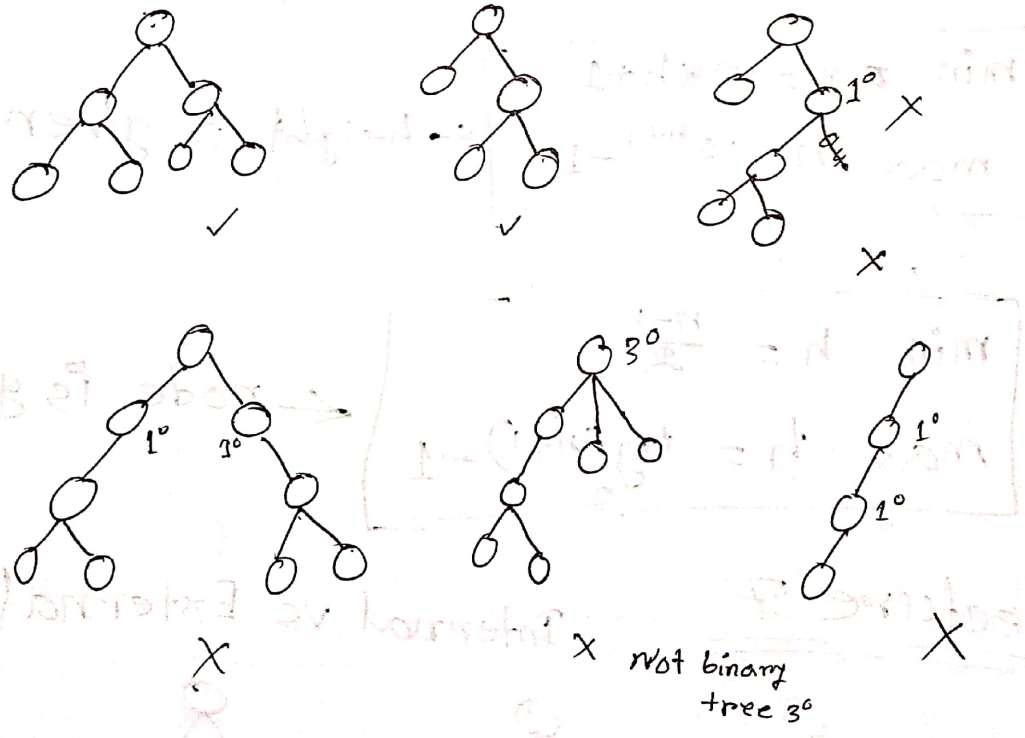
Strict Binary tree

Lecture: 5

1. Strict / proper / complete
2. Height vs Node
3. Internal vs External Nodes

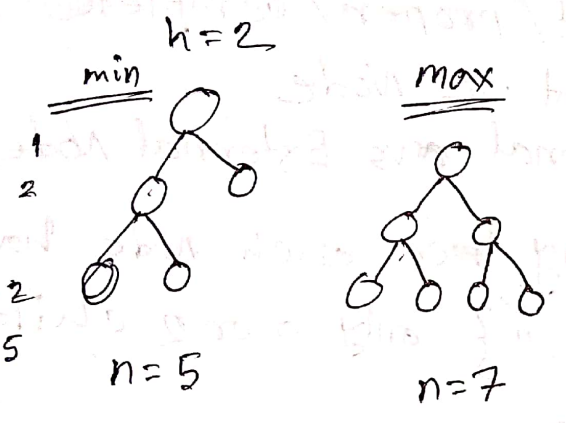
in strict binary tree each Node have {0, 1, 2} only 0 or 2 children

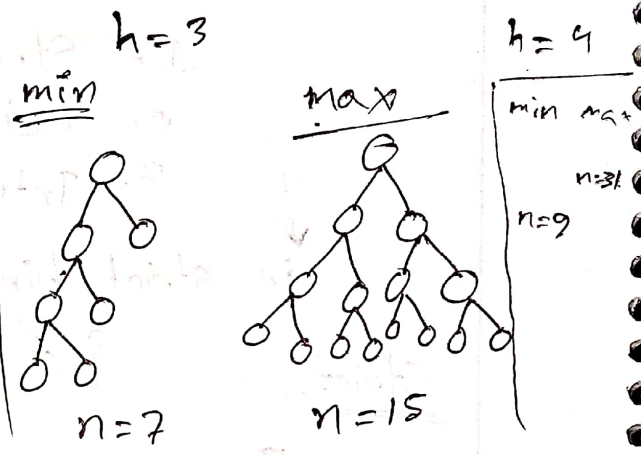
Strict



strict binary tree

Lecture 6 — Heigh vs Node





min $n = 2*h+1$

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

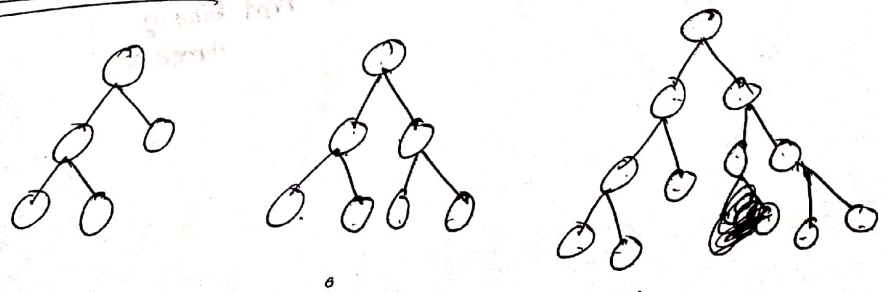
← height is given

min $h = \frac{n-1}{2}$

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

← node is given

Lecture 7 — Internal vs External Nodes



In = 2
Ex = 3

i = 3
E = 4

i = 5
E = 6

$E = i + 1$

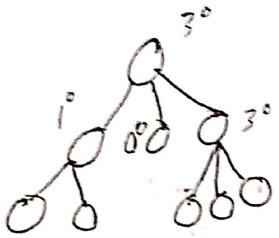
m-ary tree

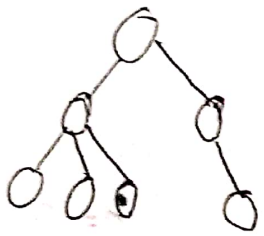
Lecture 8

1. What are n-ary tree.
2. Strict n-arry tree
3. Height vs Node
4. Internal vs External Nodes.

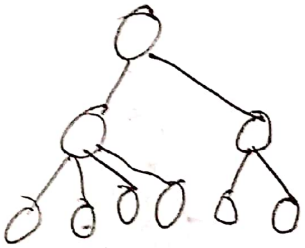
⇒ n-ary means degree of a node, at most n children

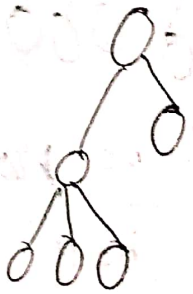
3-ary tree {0,1,2,3}.

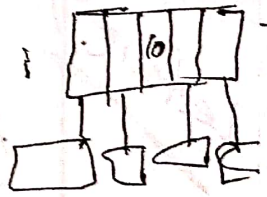


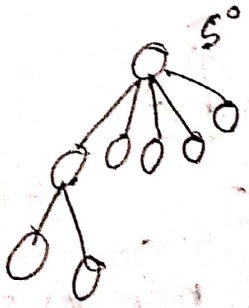


4-ary tree





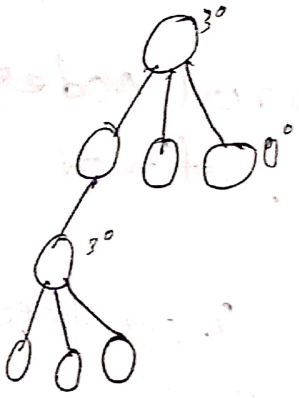


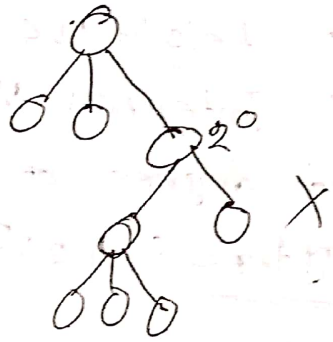


X

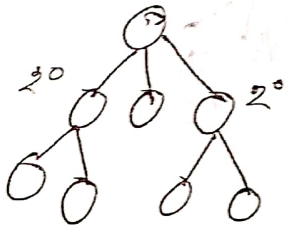
m-ary tree

Strict 3-ary tree {0, 1, 2, 3}

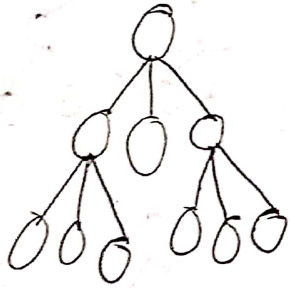




×



×

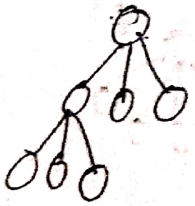


Lecture 9

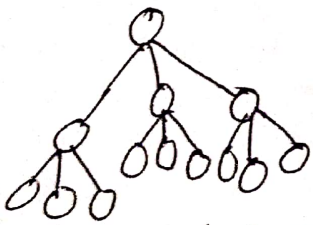
Height vs Node

h = 2

min

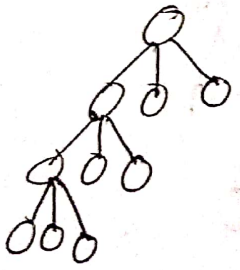


max

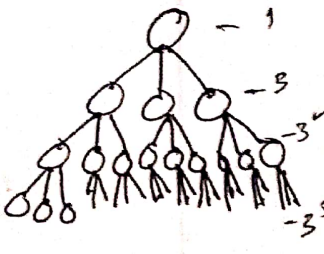


h = 3

min



max



$1 + 3 + 3^2 + 3^3 + \cdots = m^h$

$1 + m + m^2 + m^3$

$\left(\frac{m^{h+1} - 1}{m - 1}\right) \quad \frac{a(r^{k+1} - 1)}{a - 1}$

degree 3°

$\min n = m * h + 1$

$\max n = \frac{m^{h+1} - 1}{m - 1}$

is nodes is given

min Height $h = \log_m [m(m-1)+1] - 1$

max height $h = \frac{n-1}{m}$

Internal and External nodes

h = 2		h = 3	
min	max	min	max
i = 2	i = 4	i = 3	i = 13
E = 5	E = 9	E = 7	E = 27

2*2+1=5 2*4+1=9 2*3+1=7 2*3+1=27

3-ary tree ⇒ $e = 2i + 1$

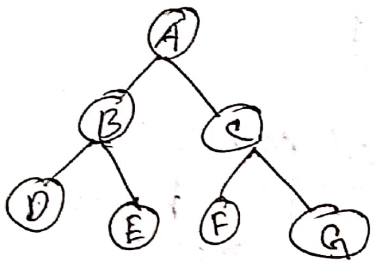
m-ary or m ⇒ $e = (m-1)*i + 1$

Lecture 10 Representation of Binary tree

1. Array
2. Link List

Lecture 10 — Representation of Binary tree

1. Array Re
2. Linked Re



maintain parent child Relationship

T	A	B	C	D	E	F	G	*
	1	2	3	4	5	6	7	

element	index	Lchild	R.child.
A	1	2	3
B	2	4	5
C	3	6	7
	i	2*i	2*i+1

Lchild multiply by 2
parent get divided by 2

$7/2 = 3$
$5/2 = 2$

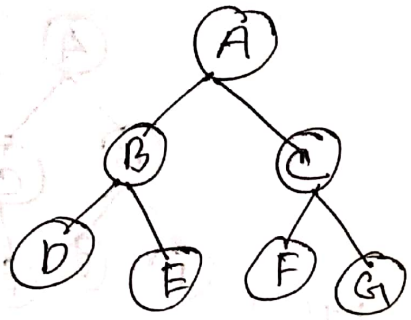
follow value

element i
Lchild $2*i$
Rchild $2*i+1$
Parent from child $\lfloor i/2 \rfloor$

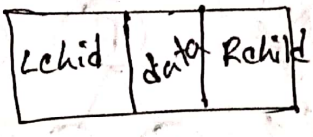
Representation of Binary tree

Lecture 11

(Link-List)

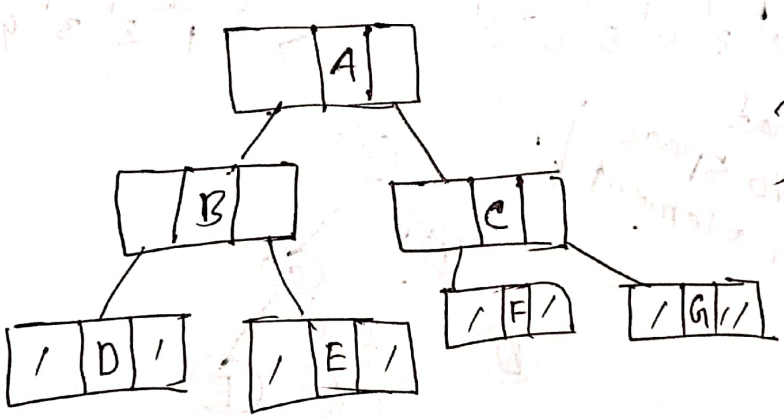


Node



~~structure~~
struct Node
{
Node *Lchild;
int data;
Node Rchild;
};

root

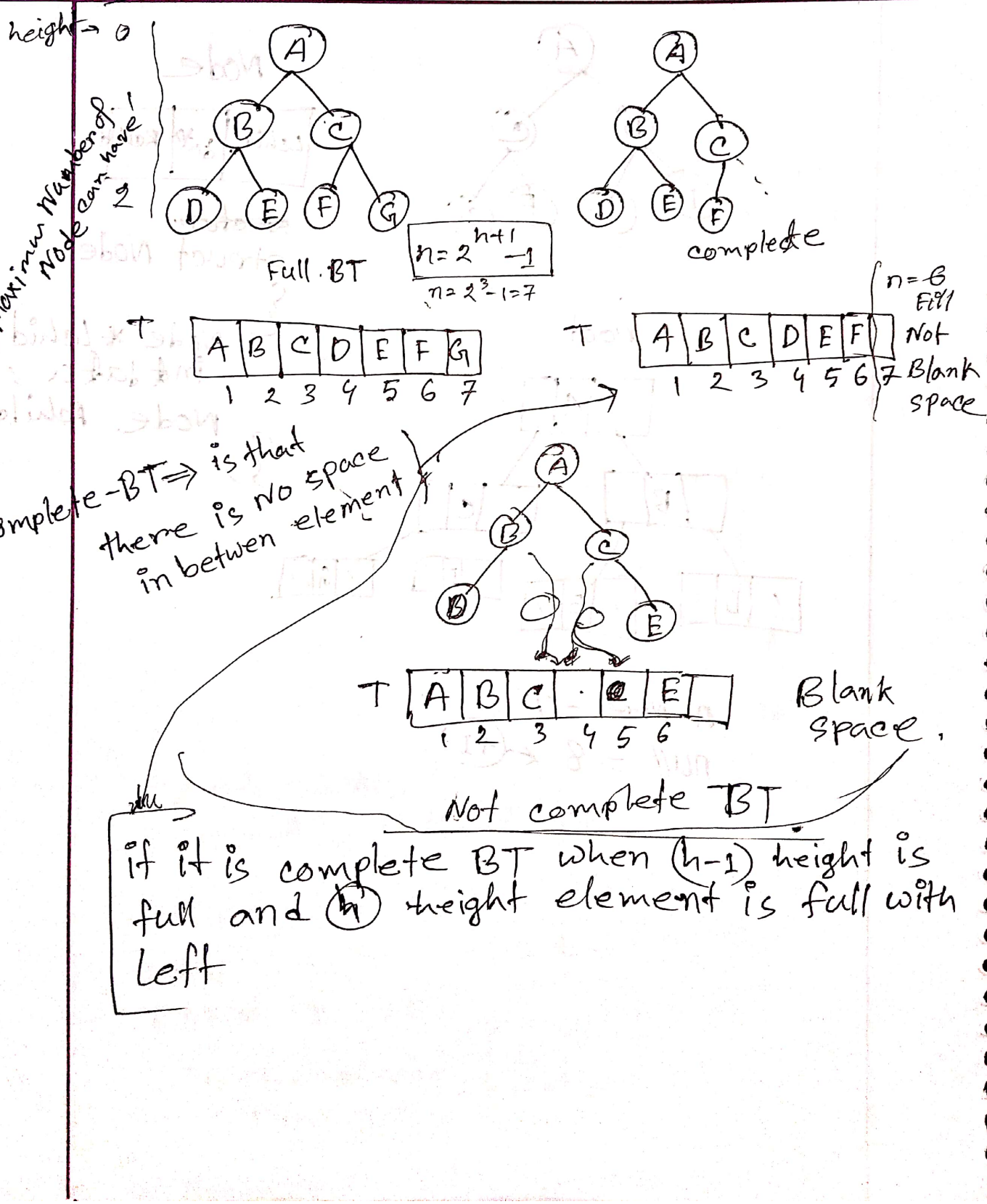


n-Node = 7

null = 8 ← (+1)

Full vs complete Binary tree

Lecture 12



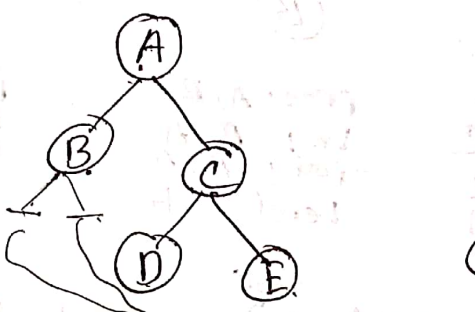
if it is complete BT when $(h-1)$ height is full and (h) height element is full with Left

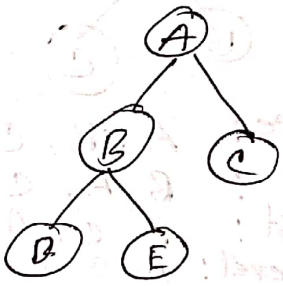
strict {0, 2} V/S complete

↓ complete ↓ almost complete

Lecture 13

strict is when degree (2° or 0°) then it is called strict BT.





T

A	B	C	-	-	D	E
1	2	3	4	5	6	

strict ✓
complete ✗

T

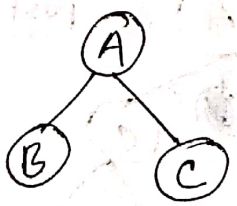
A	B	C	D	E

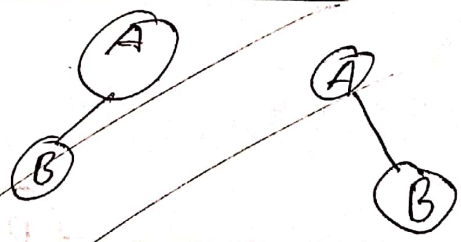
left to complete.

strict ✓
complete ✓

Lecture 14

Traversals



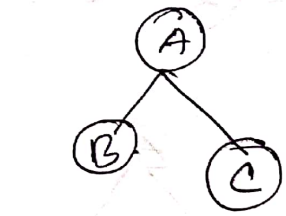


preorder : visit(Node), Preoder(Lsubtree) preoder(Rsubtre)

Inodd

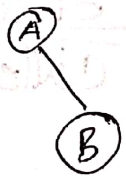
Tree traversals

Lecture 14



pre: A, B, C
In : B, A, C
post : B, C, A
Level: A, B, C

pree: A, B,
In : B, A
post : B, A
Level A, B



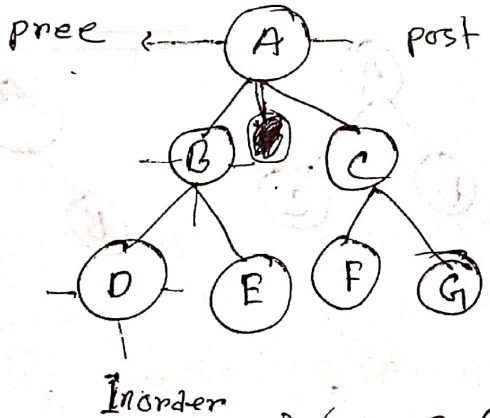
Pre: A, B
In : A, B
post : B, A
level: A, B

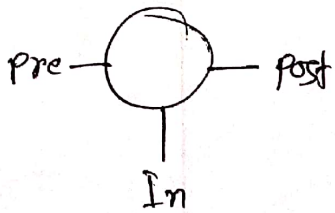
preorder: visit(Node), Preorder(L subtree), Pred(R subtree)

Inorder: Inorder(left), visit(Node), Inorder(right)

post order: Postorder(left), postorder(right), visit(Node)

Level order: level by level





pre: A, (B, D, E), (C, F, G)

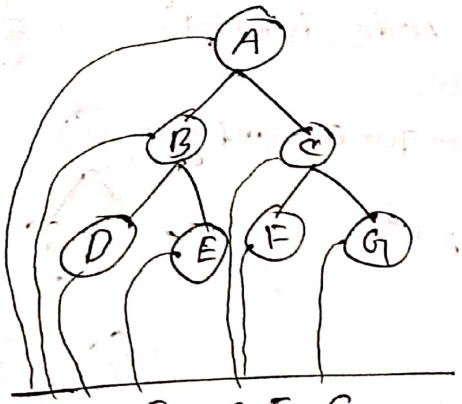
In: (D, B, E), A, (F, C, G)

post: (D, E, B), (F, G, C), A

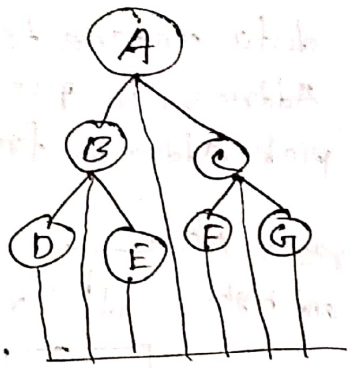
Level: A, B, C, D, E, F, G

BT Traversal Easy method 1

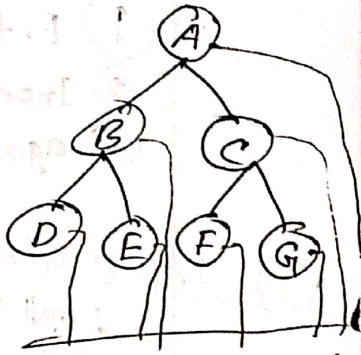
Lecture: 15



A, B, D, E, C, F, G



In: D, B, E, A, F, C, G

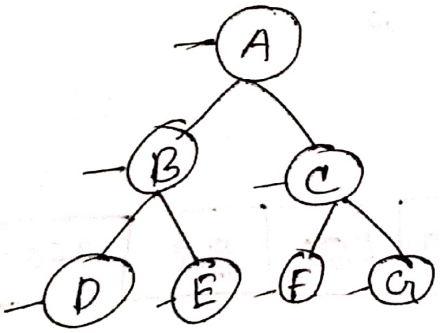


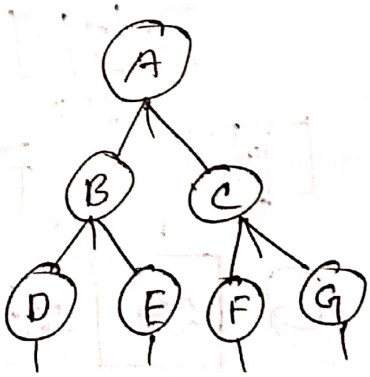
D, E, B, F, G, C, A

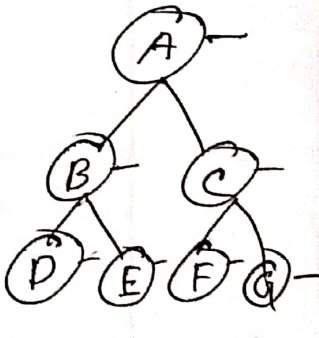
Post:

pre:

Lecture: 16







Post: D, E, B, F, G, C, A

In: D, B, E, A, F, C, G

pre: A, B, D, E, C, F, G

Lecture: 17

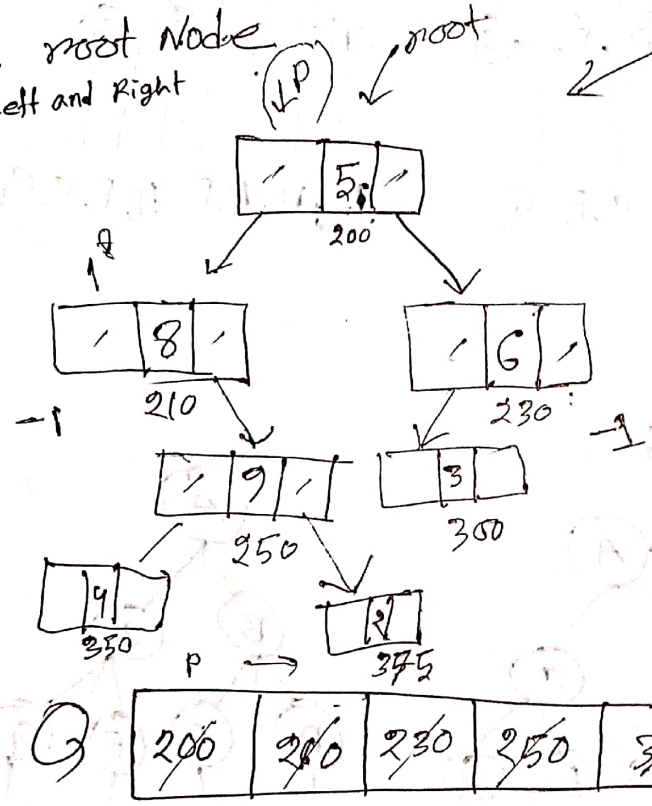
finger method

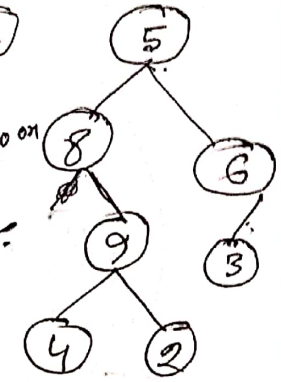
Create BT

Lecture 18

1. Enter data and create node if not -1
2. Insert Address in queue
3. again pick address from queue and go on

first step create root Node and null Left and Right child





Lchild or Rchild
Enter data if it is -1 then not create any node

(19)

create tree

Lecture 19

```
void create( )
{
    Node *p, *t;
    int x;
    Queue q;
    printf("Enter root value");
    cin >> x;
    root = malloc(sizeof(node));
    root->data = x;
    root->lchild = root->Rchild = 0;
    equeue(root);
    while( !isEmpty(q) )        // whenever queue is
                                // Not empty
    {
        p = dequeue(q);
        cout << "Enter left child "
        cin >> x;
        if(x != -1)
        {
            t = malloc: (sizeof (structure node));
            t->data = x;
            t->lchild = t->rchild = 0;
            endueue(t);  ⟹ insert in queu
        }
        ═══ Rchild same.
    }
}
```

20 / 21 > code → pdf

Tree Traversals

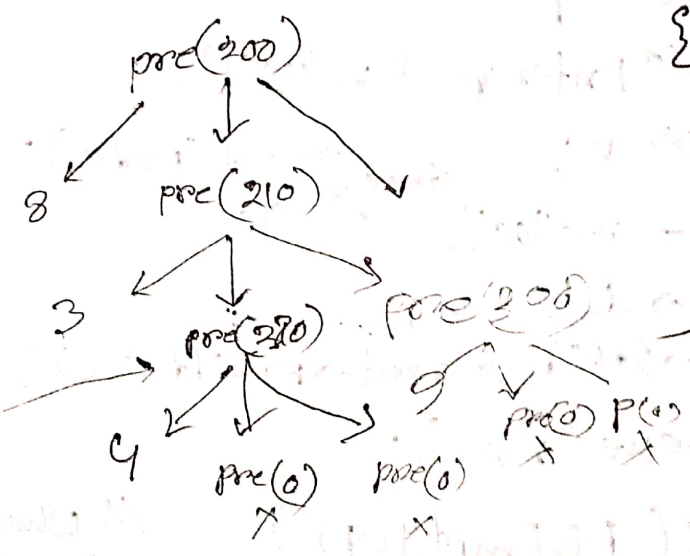
Lecture 22

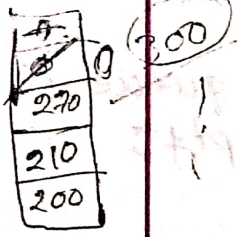
1. preorder
2. Inorder
3. postorder

(cout)

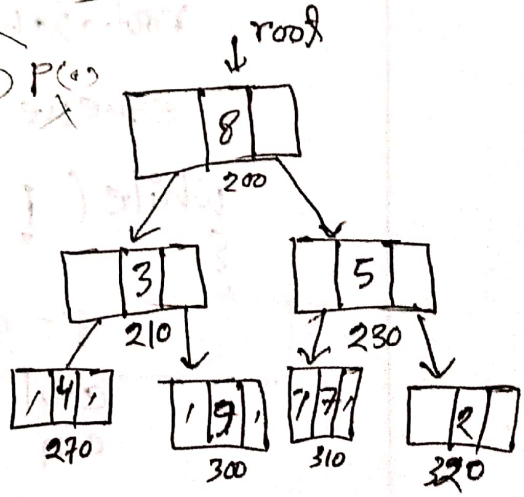
1 preorder

```
void preorder(Node *t)
{ if(t!=NULL)
  {cout << t->data;
   preorder(t->lchild);
  }preorder(t->rchild);
}
```





Stack



output: 8, 3, 4

total number of (n + n+1 = 2n+1)

- n → number of Node
- n+1 → number of pointers

Lecture 23 ⇒ Inorder

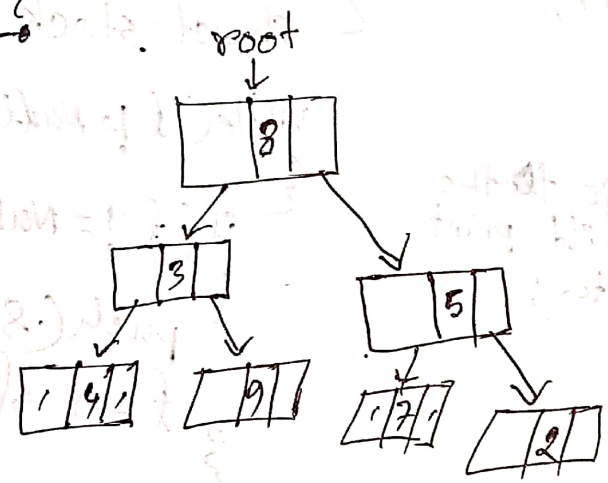
Lecture 24 : ⇒ ~~Inorder~~ & ~~Lecture : 25~~

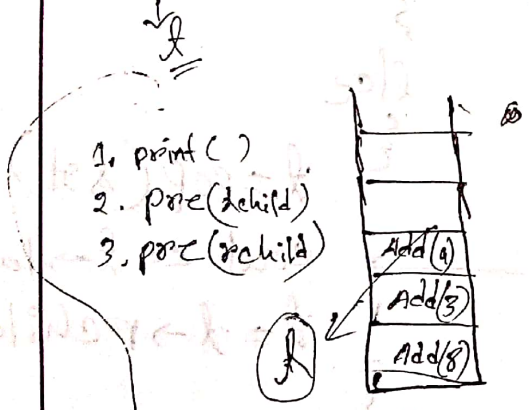
Iterative Tree Traversal

we have to use (stack) to convert Iterative tree traversals.

How?

1. preorder
2. postorder
3. Inorder






```
void preorder(Node *t)
{   struct stack st;
while( t != NULL // !ISEMPTY(st))
{
  if (t != NULL)
  { cout << t->data;
    push (&st, t);
    t = t->data
  }
  else
  { t = pop(&st);
    t = t->rchild
  }
}
```

Stack O/P = 8, 3, 4

Address is missing that's (t is null)

Why we have to use stack for remaing Address.

Lecture-25 Iterative traversals

Inorder

1. In(lchild);
   cout << t->data;
   In(rchild);

Before go to the rightchild print the data;

code

```
void Inorder (Node *t)
{ struct stack st;
  while( t != Null || !isEmpty)
  {
    if( t != Null)
    {
      push(&st, t);
      t = t->lchild;
    }
    else
    {
      t = pop(&st)
      cout << t->data;
      t = t->rchild;
    }
  }
}
```

Refer ID for Bonus and placemat under Java

Iterative tree traversals

100 * 1000 = 1,00,000
× 300
= 3,00,00,000

postorder

1. post (lchild)
2. post (rchild)
3. print( );

Fast of all both the side visite then print the data

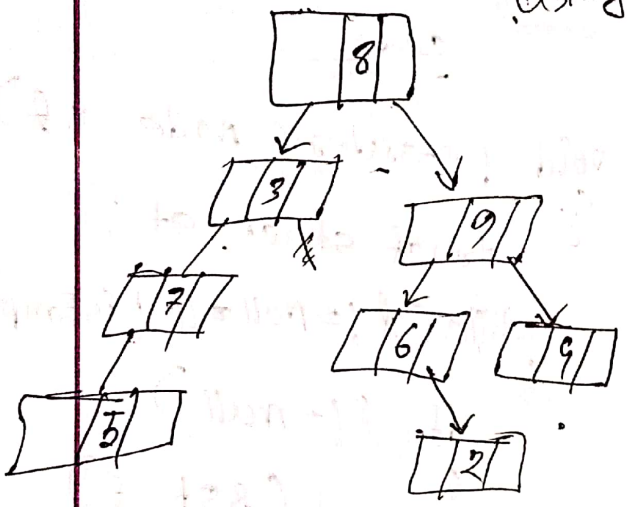
code

```
void postorder( Node *t)
{
  struct stack st;
  while (t != NULL || !isEmpty())
  {
    if (t != null)
    {
      push(&st, t)
      t = t->lchild;
    }
    else
    {
      int temp = pop(&st);
      if (temp > 0)
      {
        push(&st, -temp);
        t=(Node*)temp->rchild;
      }
      else
        { cout <<(Node*)temp->data
          t=null;
}}}
```

Lecture 27

Level order traversal.

using Queue

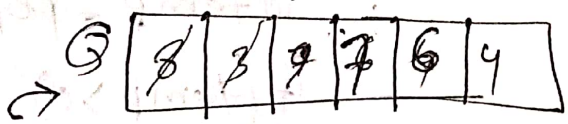


o/p = 8, 3, 9, 7, 6, 9, 5, 2

```
void levelorder(Node *p)
{
  Queue q;
  cout << p->data; //root
  equeue(&q, p);
  while(!isEmpty(&q))
  {
    p = dequeue(&q);
    if(p->lchild)
    {
      cout << p->data;
      enqueue(&q, p->lchild);
    }
    if(p->rchild)
    {
      cout << p->data;
      enqueue(&q, p->rchild);
    }
  }
}
```

Level order traversal using Queue

node type queue



insert address in queue.

pic address from queue and visite lchild and Rchild

lecture 28 → code.

Can we generate tree from traversals

lecture 29

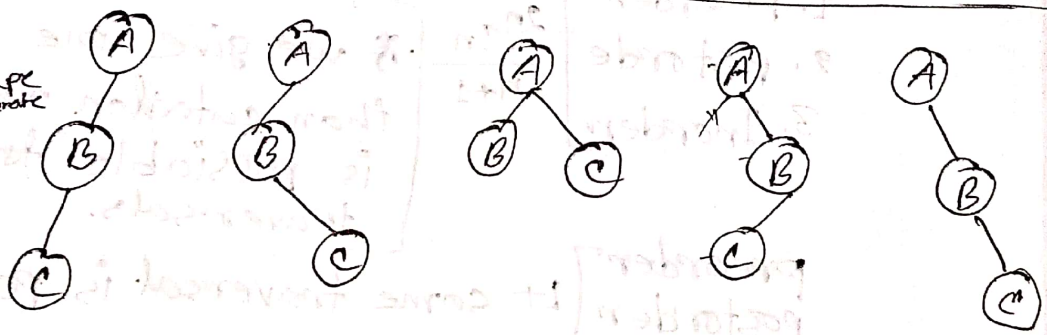
$n = 3$

A B C

preorder: A, B, C

from the preorder can we generate tree or not

$\frac{2nCn}{n+1}$ shape generate



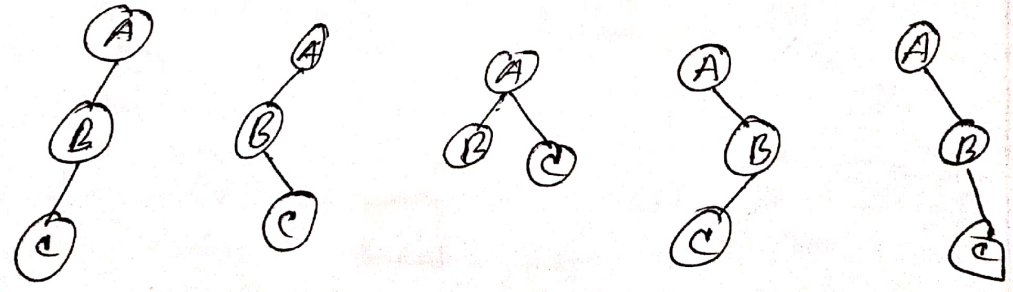
5 tree has same preorder ☹

conclusion is: if only preorder is given we can not generate tree

Now two traversals is given

preorder: A, B, C

postorder: C, B, A



pre: A, B, C pre: A, B, C

post: C, B, A post: C, B, A

more than two tree has same preorder and postorder

X

we want a single unique tree from ~~for~~ traversals.

Since:

1. preorder
2. postorde
3. Inorder

$\frac{2nCn}{n+1}$ · if we give one of them than catalon number tree is possiable for same traversals.

preorder
postorder
1+ same traversal is possiable.

** Which traversal is given me a unique tree

1. preorder + Inorder
2. postorder + Inorder

> tree make

Why inorder give me the (root)

(27) generating tree from traversals

Lecture: 30/1

Searching

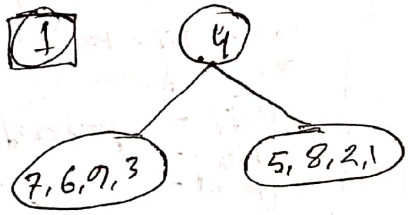
n → preorder: 4, 7, 9, 6, 3, 2, 5, 8, 1

n → Inorder: 7, 6, 9, 3, 4, 5, 8, 2, 1

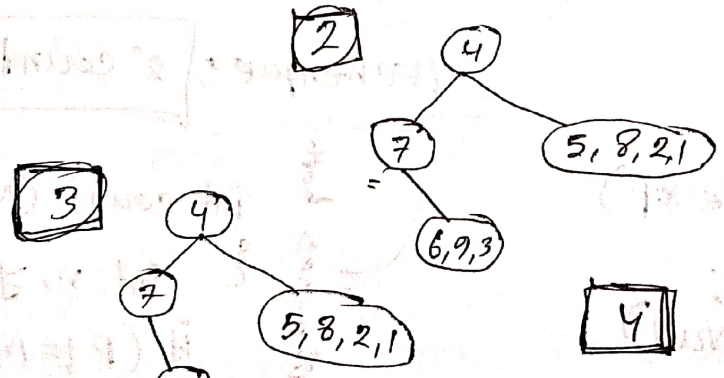
n*n = n^2

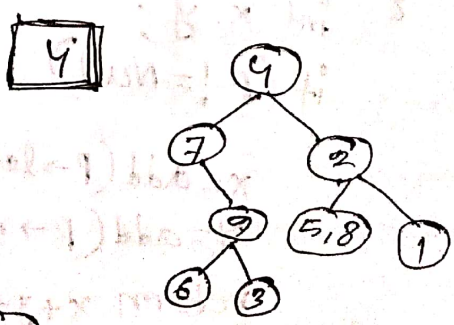
$O(n^2)$

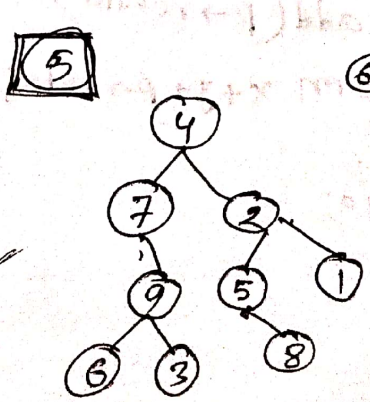
1. Take a node and take all the element of Inorder
   7, 6, 9, 3, 4, 5, 8, 2, 1
   then repeating step
   and traverse preorder (2)
   search in Inorder that take in one node.



searching
spliting

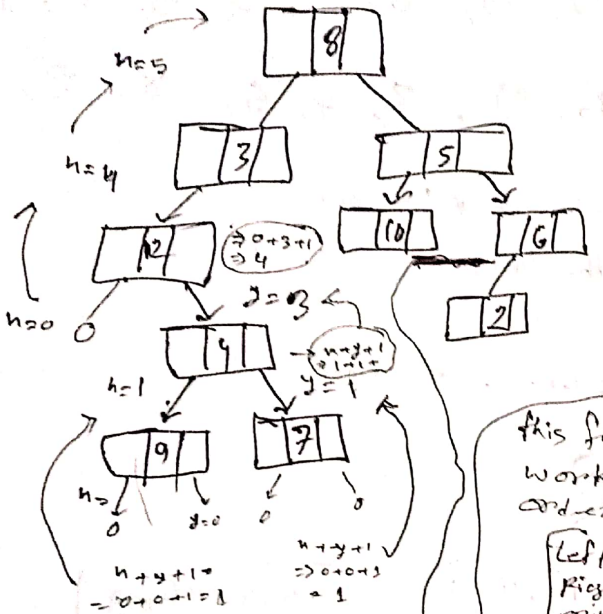






counting nodes and heights

Lecture 31




```
int count(Node *p)
{
  int x, y;
  if (p != Null)
  {
    x = count(p->lchild)
    y = count(p->rchild)
    return x+y+1;
  }
  return(0);
}
```

this function work like post order-ish form

Left – Right – Point – Postorder

Techenque: 2° count

add all element

```
int add(Node *p)
{
  int x, y;
  if (p != Null)
  {
    x = add(p->lchild)
    y = add(p->rchild)
    return x+y+p->data;
  }
  return 0;
}
```

this child: function program count who has both result = 3

```
int count (Node *p)
{
  int x, y;
  if (p != Null)
  {
    x = count(p->lchild)
    y = count(p->rchild)
    if(p->lchild && p->rchild)
      return x+y+1;
    else
    { return x+y;
    } }
  return 0;
}
```

Lecture 32 Code:

```
int fun(node *p)
{ int x, y;
  if (p != null)
  {
    x = fun(p->lchild);
    y = fun(p->rchild);
    if(x>y)
      return x+1;
    else
      return y+1;
  }
  return 0;
}
```

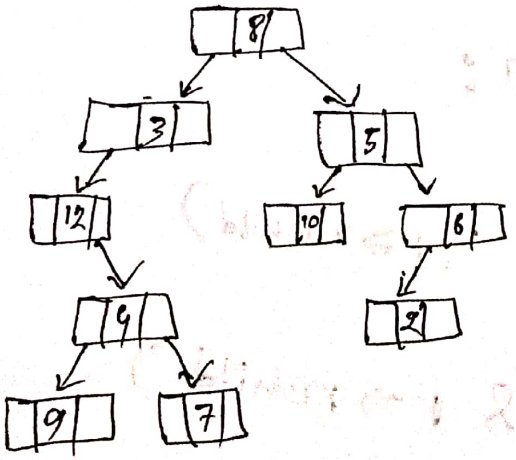
height

count Node

```
ind Nodes(Node *P)
{
  if (P->!= Null)
  return Nodes(P->lchild) + Nodes(P->rch
                +1;
  Return 0;
```

count leaf nodes

Lecture 33



counting Nodes

```
int Nodes(Node *P)
{ int x, y;
  if( p-> != Null)
  {
    x = Nodes(p->lchild)
    y = Nodes(R->rchild)
    return x+y+1;
  }
  Return 0;
}
```

when Leaf Node is found: both side is Null.

```
int Leaf(Node *p)
{
  int x, y;
  if (p != Null)
  {
     x= Leaf(p->lchild);
     y= Leaf(p->rchild);
     if (p->lchild == Null && p->rchild == Null)
          return x+y+1;
     else
          return x+y;
  }
  return 0;
}
```

2° or 1° || or use

all condition:

1. Leaf 0°
   if (!R->lchild && !p->rchild)
2. Node 2°
   if (p->lchild && p->rchild)
3. Node 1° or 2°
   if (p->lchild || p->rchild)
4. Node 1°
   if (p->lchild != Null && p->rchild == Null) || or
   opposite
   same another 😐

L ⊕ R = LR' + L'R (1)
if (p->lchild != Null ^ p->rchild != Null)

LR' or RL'