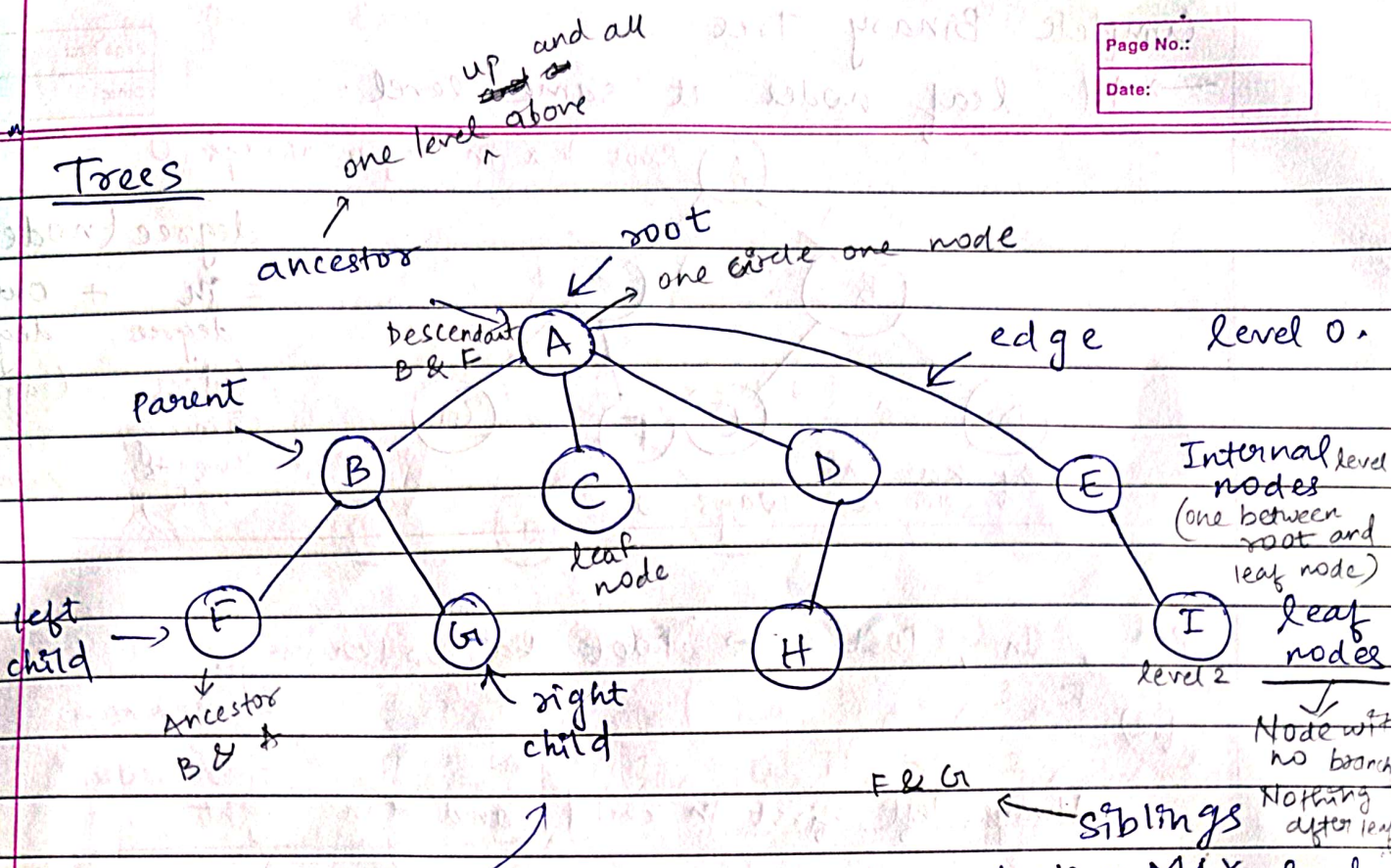
Page No.:
Date:

Trees

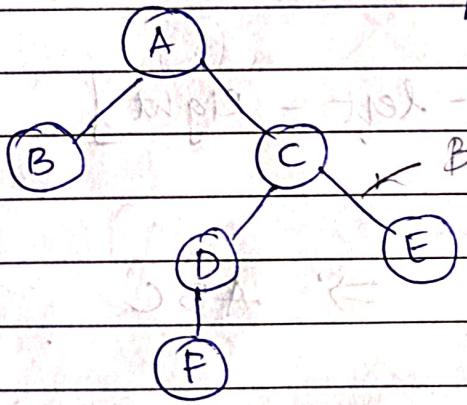


→ Non linear DS

Binary Tree (Node with ~~2 children / 2 branches~~ 0/1/2 branches then its binary tree)

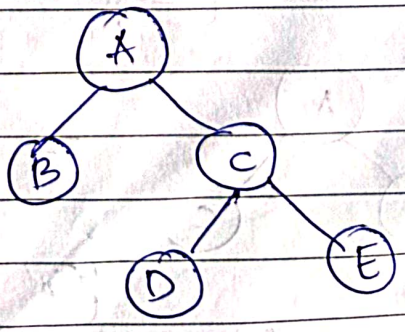
0 to 2

0/1/2 → either of 3



Strictly Binary Tree

0 or 2 → Branches.

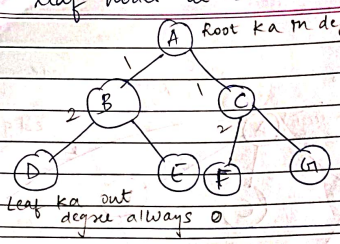


Complete Binary Tree

→ All leaf nodes at same level.

Complete Binary Tree

↳ All leaf nodes at same level.



Root ka in degree always 0

Leaf ka out degree always 0

degree (node) = in degree (edges coming towards node) + out degree (edges going away from node)

Pre, In, Post → Root ka order.

(Root before)

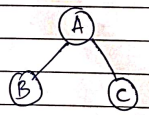
Always left first in child and then right

Binary Tree Traversal

1) Pre Order
2) In order
3) Post Order.

1) Pre order [Root - left - Right]

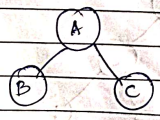
Root in start



⇒ ABC

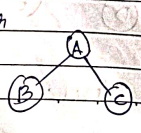
2) In Order [left - Root - Right]

Root in middle of left and right



⇒ BAC

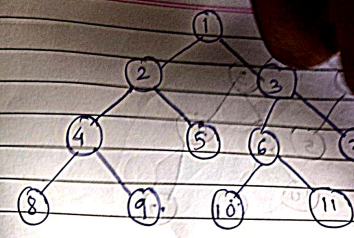
3) Post Order [left - Right - Root]



⇒ BCA

Root in last

5 M Q. in exa



Preorder - 1, 2, 4, 8, 9, 5, 3, 6, 10, 11, 7, 12.

Inorder - 8, 4, 9, 2, 5, 1, 10, 11, 6, 3, 7, 12

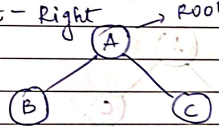
Postorder - 8, 9, 4, 5, 2, 10, 11, 6, 12, 7, 3, 1.

Tree Order Traversals. (Binary Tree Traversal)

0/1/2 branches

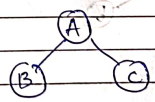
1) Inorder

left - Root - Right → Root



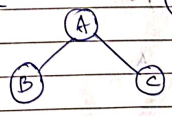
B - A - C

2) Preorder (Root - left - Right)



A - B - C

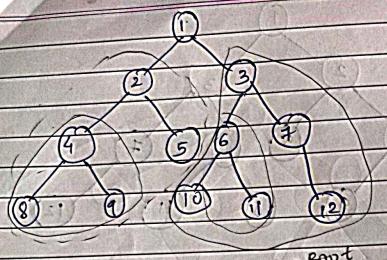
3) Post Order (left - Right - Root)



~~left~~

Mid 28 Sep
7th Oct – Assignment 2

Date:

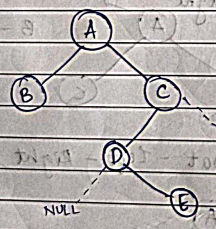


→ Root

Inorder – 8, 4, 9, 2, 5, 1, 10, 6, 11, 3, 7, 12
(All left subtree parts; All right subtree parts)

Preorder – 1, 2, 4, 8, 9, 5, 3, 6, 10, 11, 7, 12

Postorder – 8, 9, 4, 5, 2, 10, 11, 6, 12, 7, 3, 1

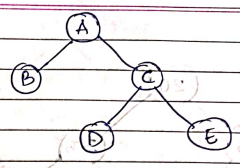


Inorder – B, A, D, E, C

Preorder – A, B, C, D, E

Postorder – B, E, D, C, A

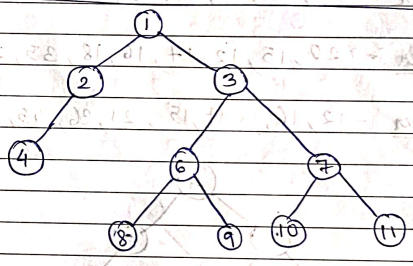
Page No.:
Date:



Inorder – B, A, D, C, E

Preorder – A, B, C, D, E

Postorder – B, D, E, C, A

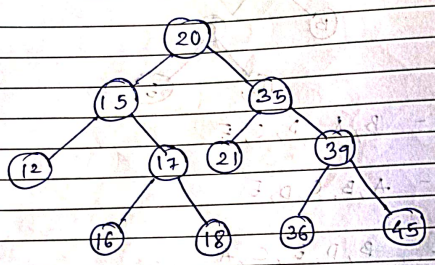


Inorder – ~~4, 2, 8, 6, 9~~ 4, 2, 1, 8, 6, 9, 3, 10, 7, 11

Preorder – 1, 2, 4, 3, 6, 8, 9, 7, 10, 11

Postorder – 4, 2, 8, 9, 6, 10, 11, 7, 3, 1

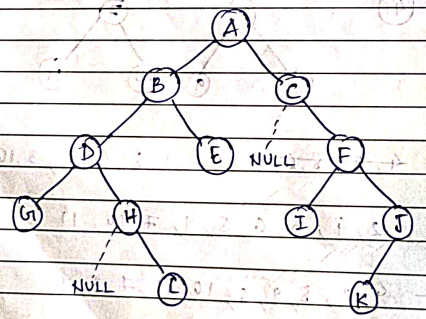
Q. Solve.



Inorder : 12, 15, 16, 17, 18, 20, 21, 35, 36, 39, 45

Preorder : 20, 15, 12, 17, 16, 18, 35, 21, 39, 36, 45

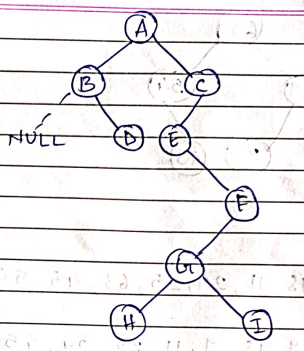
Postorder - 12, 16, 18, 17, 15, 21, 36, 45, 39, 35, 20



Inorder: G, D, H, L, B, E, A, C, I, F, K J

Preorder: A, B, D, G, H, L, E, C, F, I, J, K

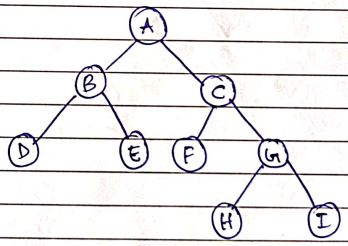
Postorder : G, L, H, D, E, B, I, K, J, F, C, A



Inorder : B, D, A, E, H, G, I, F, C

Preorder: A, B, D, C, E, F, G, H, I

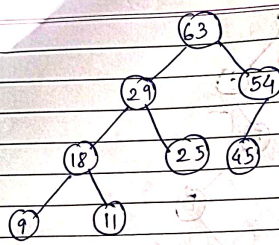
Postorder: D, B, H, I, G, F, E, C, A



Inorder : D, B, E, A, F, C, H, G, I.

Preorder : A, B, D, E, C, F, G, H, I

Postorder : D, E, B, F, H, I, G, C, A



Inorder: - 9, 18, 11, 29, 25, 63, 45, 54

Preorder: 63, 29, 18, 9, 11, 25, 54, 45

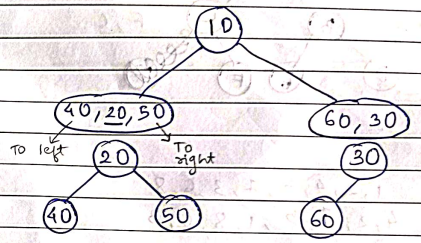
Postorder: 9, 11, 18, 25, 29, 45, 54, 63

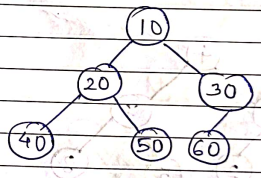
10 M Qs, in Exam.

Construct a Binary Tree.

1) Inorder - ~~40, 20, 50, 10, 60, 30~~

(Leftmost for root) Preorder - 10, 20, 40, 50, 30, 60 → see leftmost for root

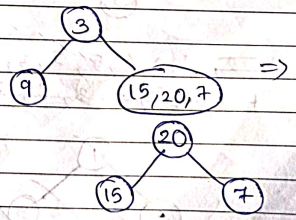




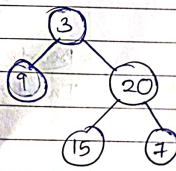
Inorder: 9, 3, 15, 20, 7 → See for left and right branches of root

Preorder: 3, 9, 20, 15, 7

Root node.



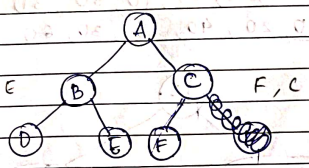
⇒



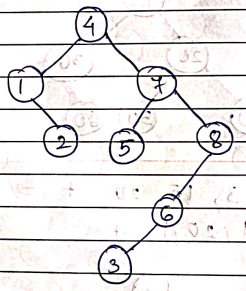
Always see preorder for ~~roots~~ ~~subtree~~ roots and for elements on right and left of root

Page No.:
Date:

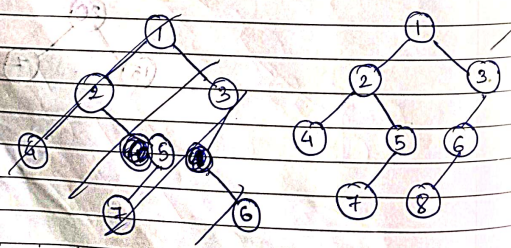
Inorder : D B E A F C
Preorder : A B D E C F



Inorder : 1, 2, 4, 5, 7, 3, 6, 8
Preorder : 4, 2, 7, 5, 1, 8, 6, 3



Preorder : ~~4, 0~~ 1, 2, 4, 5, 7, 3, 6, 8
Inorder : ~~1, 2, 4, 5, 7, 3, 6, 8~~ 4, 2, 7, 5, 1, 8, 6, 3

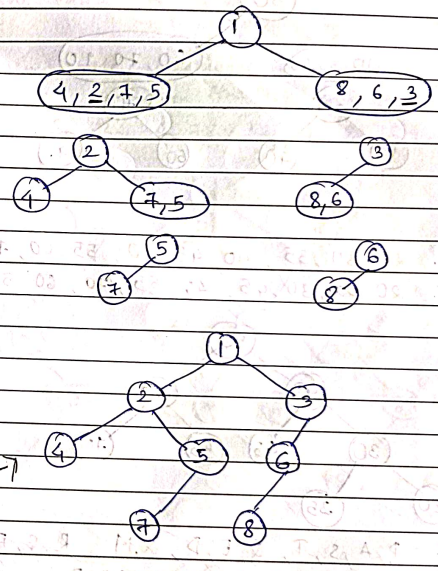


Always see preorder and postorder for position of roots.

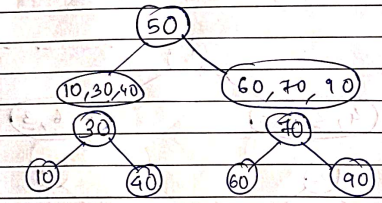
Page No.:
Date:

Inorder - 4, 2, 7, 5, 1, 8, 6, 3

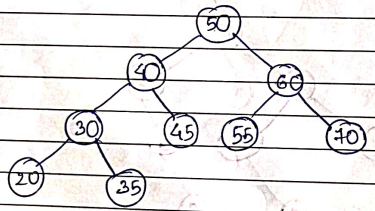
Postorder - 4, 7, 5, 2, 8, ~~6, 3, 1~~ 6, 3, 1
see rightmost for root



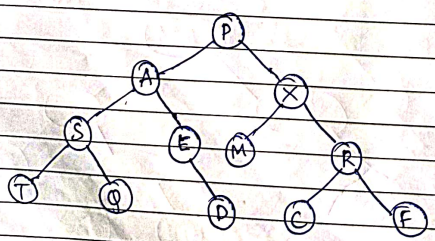
Inorder : 10, 30, 40, 50, 60, 70, 90
Postorder : 10, 40, 30, 60, 90, 70, 50 ← Root node



Inorder : 20, 30, 35, 40, 45, 50, 55, 60, 70
Postorder : 20, 35, 30, 45, 40, 55, 70, 60, 50



Preorder : P, A, S, T, Q, E, D, X, M, R, C, F
Inorder : T, S, Q, A, E, D, P, M, X, C, R, F



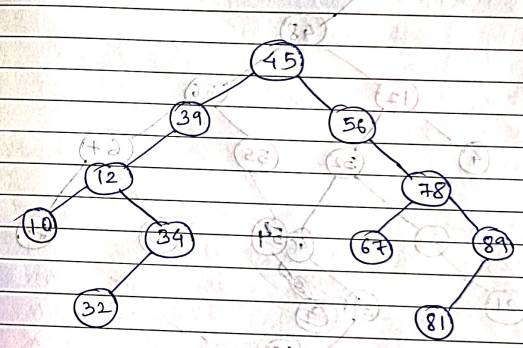
10M Q. in exam

For Binary Trees only → Binary Search Tree (BST) → First node at beginning is root node

→ If K is the data of root node then
→ all left child < K
→ all right child ≥ K
↳ If considered for equal then is ambiguity for equal to only root node repeated ↑
If any no. is repeated write Note: 23 is repeated

Every new node you insert will be a leaf node.
To cross check when you find its inorder you get proper ascending order.

Q. 45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67, 81

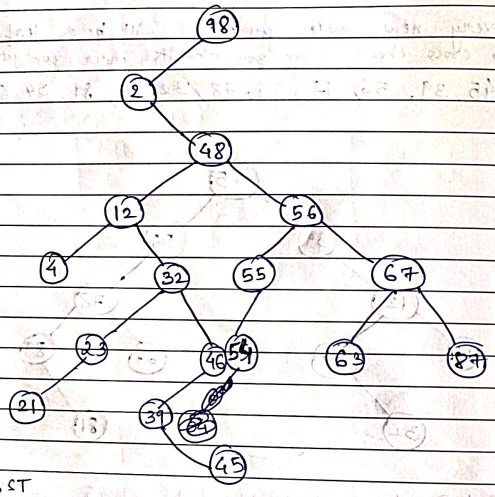


Find Inorder. — 10, 12, 32, 34, 39, 45, 56, 67, 78, 81, 89

If root node is repeated then go to right

98, 2, 48, 12, 56, 32, 4, 67, 23, 87, 23, 55, 46, 21, 39, 45, 54, 63

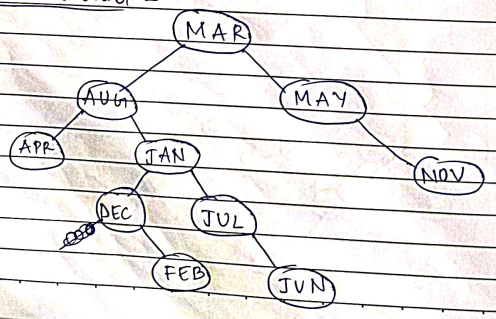
Note: 23 is repeated.



Create BST

MAR, MAY, NOV, AUG, APR, JAN, DEC, JUL, FEB, ~~JAN~~ JUN

Alphabetical Order –



Delete node in tree

↓

Not in lab but there soo final. It's difficult

Actually, Tree order Traversal expected → Imp

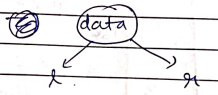
(99.9% BST Imp) → In lab, exam, etc

For this

You must know this.

BST Code

```
#include <stdio.h>
struct node
{
    int data;
    struct node * left, * right
};
struct node * root; = NULL
void create tree(struct node * root)
{
    root = NULL;
}
```



This not required if root already initialized to NULL.

```
struct node * insert_ele (struct node * root, int val)
{
    struct node * nn, * ptr, * temp;
    nn = (struct node *) malloc (size of (struct node))
    nn → data = val;
    nn → l = NULL;
    nn → r = NULL;

    // 1st node.
    if (root == NULL)
    {
        root = nn;
        root → l = NULL;
        root → r = NULL;
    }
    else
    { temp = NULL; → X (will do if not written)
      ptr = root;
```

x6 need to write 2 times but mam told to write (ask MAM)

what we can do ?
Not how we can do of ADT

```
while (ptr != NULL)
{
    temp = ptr;
    if (val < ptr -> data)
        ptr = ptr -> left;
    else
        ptr = ptr -> right;
}
```

→ finding proper parent.

```
if (val < temp -> data)
    temp -> left = nn;
else
    temp -> right = nn;
return root;
}
```

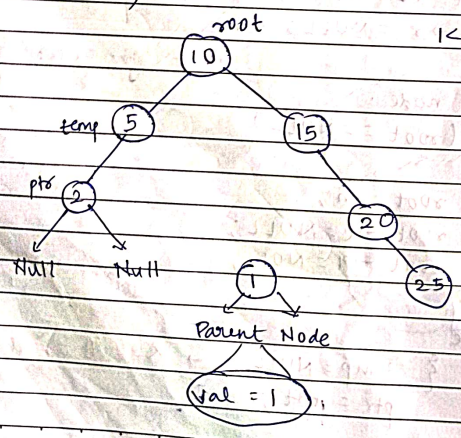
10, 5, 15, 20, 2, 25, 1

temp = ptr

ptr = ptr -> next;

~~root~~ temp = NULL

1 < 10



Tree Order Traversals (Using recursion)

```
void Inorder (struct node * root)
{                    Left -> Root - Right
    if (root != NULL)
    {
        inorder (root -> left);
        printf ("%d", root -> data);
        inorder (root -> right);
    }
}

void preorder (struct node * root)
{                    Root - Left - Right
    if (root != NULL)
    {
        ~~preorder~~ printf ("%d", root -> data);
        preorder (root -> left);
        preorder (root -> right);
    }
}

void postorder (struct node * root)
{                    Left - Right - Root
    if (root != NULL)
    {
        postorder (root -> left);
        postorder (root -> right);
        printf ~~postorder~~ ("%d", root -> data);
```

Inorder

In(20)
if(✓)
In(20→L)=10
print 10
In(20→R)

In(10)
if(✓)
In(10→L)=5
print 10
In(10→R)

In(30)

In(5)
if(✓)
In(5→L)=2
print 5
In(5→R)

In(6)
if(✓)
6→L → NULL
print 6
6→R → NULL

NULL

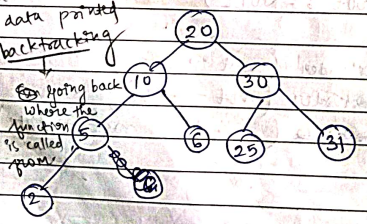
In(2)
if(✓)
In(2→L)
print 2
In(2→R)

NULL

NULL

2 5 10 6 20 25 30 31

First we come at 11 the leaf for finding the root until root != NULL and then data printed using backtracking



Left Heavy LH → Left Heavy (1)
Right heavy RH (-1)
(0)

Height of tree is no of levels

LL → Left Left

Rotate critical node (Node with BF not -1, 0, 1)

Rotations name based on type of insertion.

AVL Tree → sure short Q. (IMP)

Balanced Binary Tree (It's a BST)

⇒ AVL Tree

⇒ G.M. Adelson → Velskii → Landes (Founded / Invented)

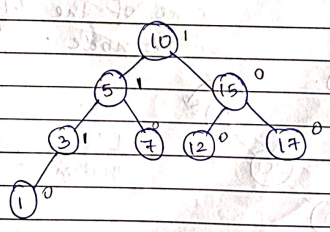
AVL

* Height of Tree = Maximum level of its leaves.

Balanced Binary Tree
Height of two subtree of every node never differ by more than 1.

∴ BF (Balanced Factor) = Height of Left subtree − Height of right subtree.

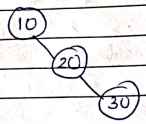
∴ B.F = −1 or 0 or 1



Balanced factor of leaf node is 0.

Page No.:
Date:

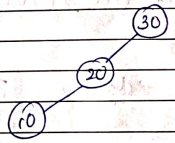
① Insert 10, 20, 30



RH – Right High

(Right Heavy Tree)

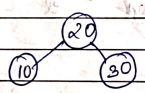
② Insert 30, 20, 10



LH – Left High

(Left Heavy Tree)

③ Insert 20, 10, 30



EH – Even High.

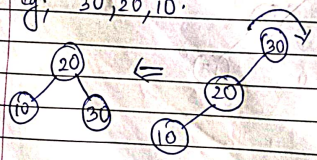
If Balanced factor other than -1, 1, or 0 then tree is unbalanced and the node with such B.F is the critical node

If AVL is not balanced we need to Balance using rotation

shows how the tree has the data stored

1) LL Rotation – (Left-Left) New node inserted into the left subtree of the left subtree of critical node.

eg; 30, 20, 10.



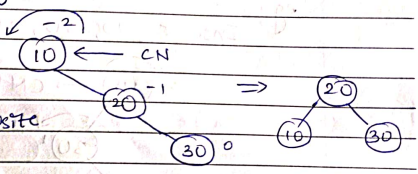
LL Rotation = Rotate in opposite direction (towards right)

Only rotate the critical node

Page No.:
Date:

2) RR Rotation (Right-Right) – New node inserted into the right subtree of the right subtree of the CN.

eg; Insert 10, 20, 30

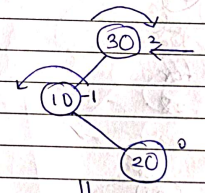


RR Rotation = Rotate in opposite direction. (towards left)

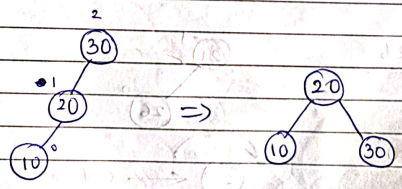
Only rotate the critical node.

3) LR Rotation (Left-Right) – New node inserted in the right subtree of the left subtree of the Critical node.

eg; Insert 30, 10, 20.

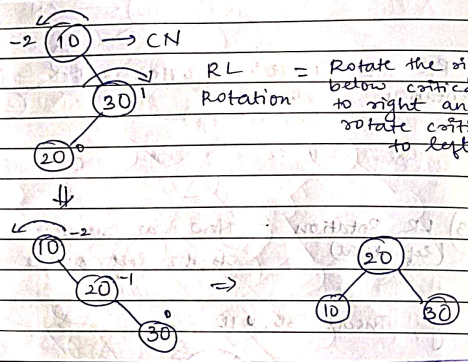


LRV = Rotate the left node below critical node to left and then rotate critical node to right.



4) RL Rotation - New node inserted in the left subtree of the right subtree of Critical Node.

eg; Insert 10, 30, 20

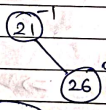


RL Rotation = Rotate the right node below critical node to right and then rotate critical node to left

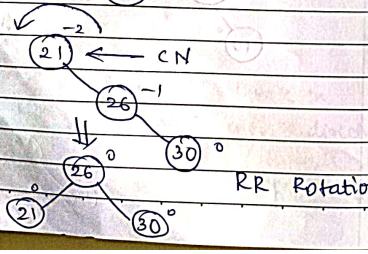
Construct AVL Tree for 21, 26, 30, 9, 4, 14, 28.

1) Insert 21

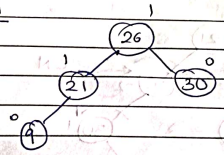
2) Insert 26



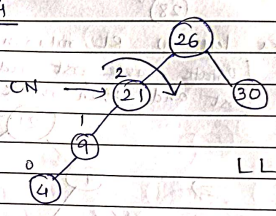
3) Insert 30



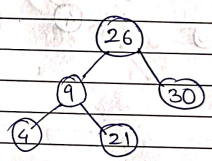
4) Insert 9



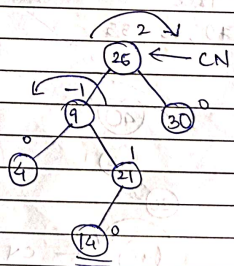
5) Insert 4



LL Rotation.



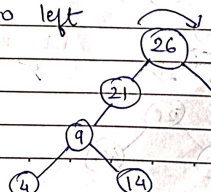
6) Insert 14



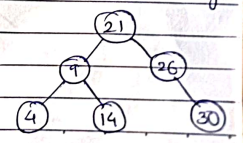
Insertion in left ka right so LR ~~rotation~~

LR Rotation.

=> Rotate 9 to left

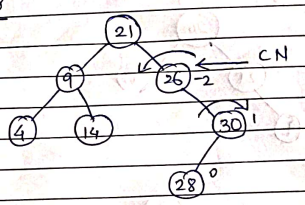


=> Rotate 26 to right



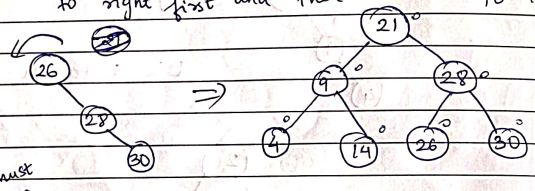
Page No.:
Date:

3) Insert 28



Insertion at right ka left so RL rotation

RL Rotation. (Rotate ~~right first~~ node at right of CN to right first and then rotate CN to left)



whenever there is ambiguity like here about 28 then insert it using logic of BST like here 28 must come in 26 ka right when 30 rotated.

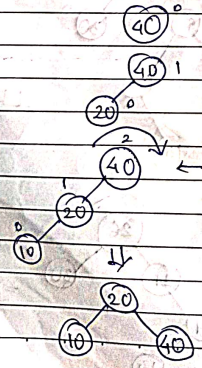
Q. Construct AVL Tree

40, 20, 10, 30, 70, 60, 55

Insert 40

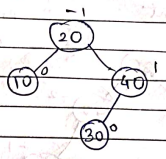
Insert 20

Insert 10

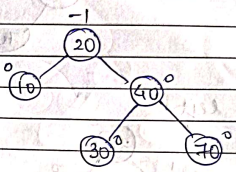


LL Rotation

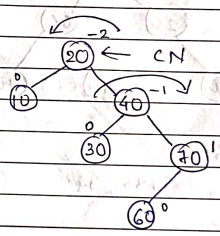
Insert 30



Insert 70

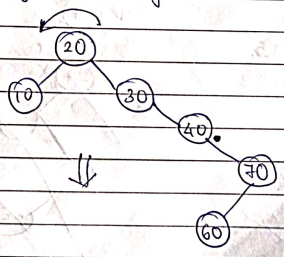


Insert 60

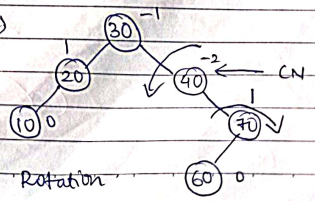


RL Rotation. side
(Insertion at right ka left me)

Rotating 40 to right



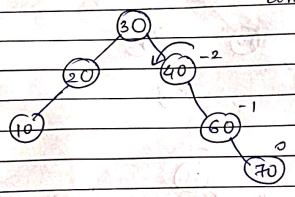
Rotating 20 to left



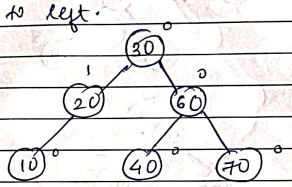
(Insertion at right ka left)

RL Rotation

Rotating 70 to right (If 70 goes right then according to principle of BST 60 must connect to 40 ka right)



Rotating 40 to left.



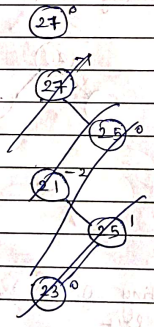
Q. Construct AVL Tree.

27, 25, 23, 29, 35, 33, 34

Insert 27

~~Insert 25~~

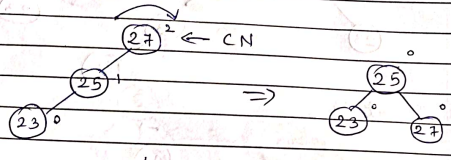
~~Insert 23~~



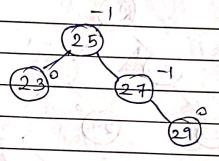
~~RL Rotation~~

Insert 25,

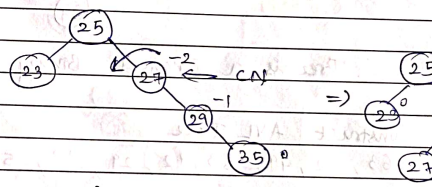
Insert 23,



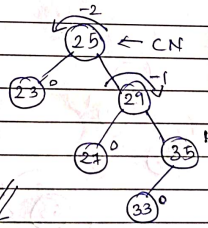
Insert 29,



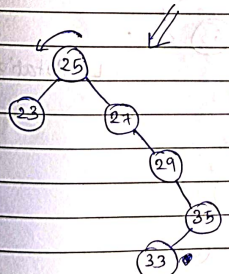
Insert 35,

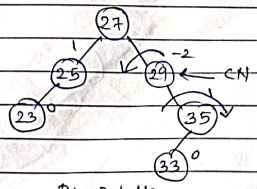


Insert 33,

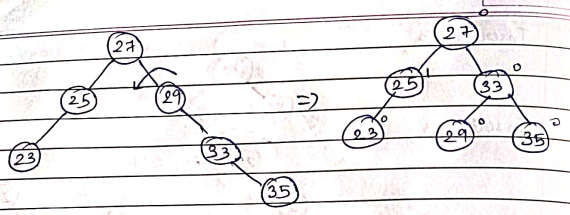


RL Rotation.

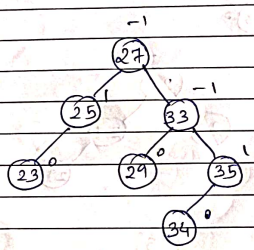




RL Rotation



Insert 34,

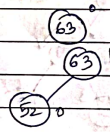


∴ Tree is a Balanced Binary Tree.

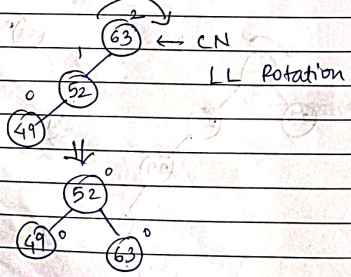
Q. Construct AVL Tree
63, 52, 49, 83, 92, 29, 23, 54, 13, 99.

Insert 63,

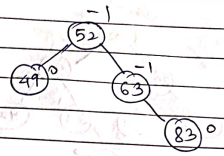
Insert 52,



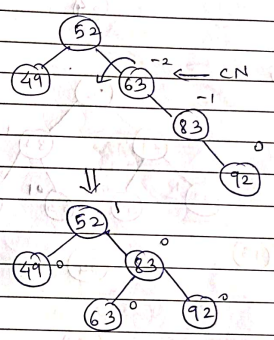
Insert 49,



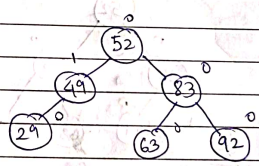
Insert 83,



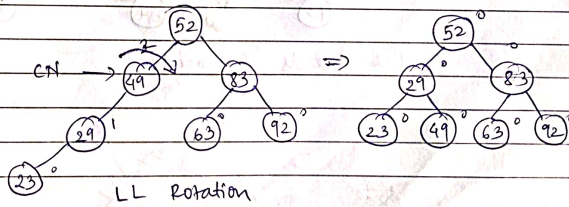
Insert 92,



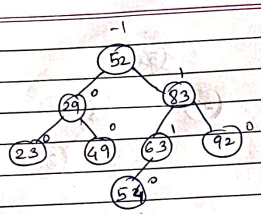
Insert 29,



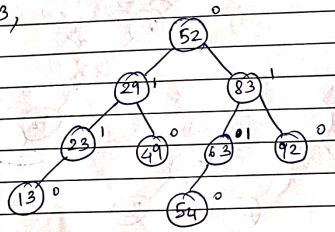
Insert 23,



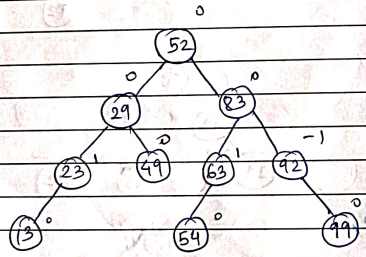
Insert 54,



Insert 13,



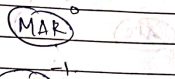
Insert 99,



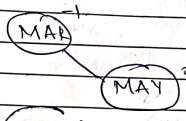
∴ The tree is balanced.

Q. Construct AVL using alphabetical order for MAR, MAY, NOV, AUG, APR, JAN, DEC, JUL, FEB, JUN

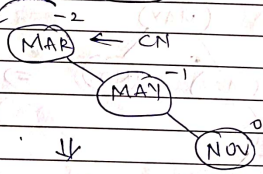
Insert MAR,



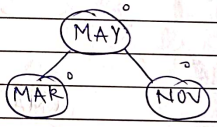
Insert MAY,



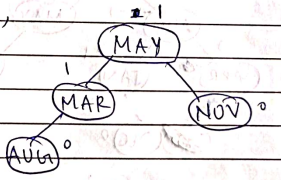
Insert NOV,



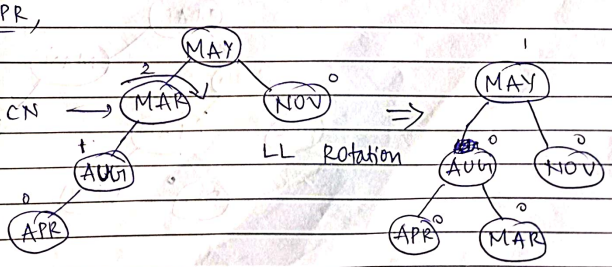
RR Rotation



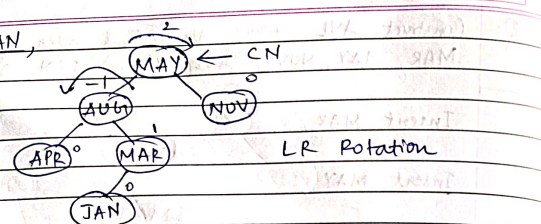
Insert AUG,



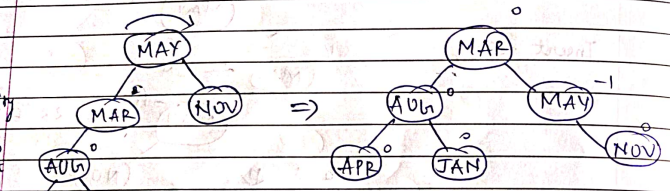
Insert APR,



Insert JAN,

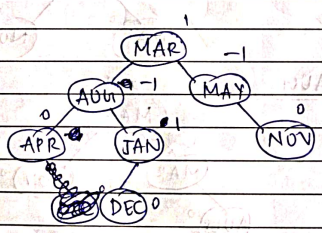


LR Rotation

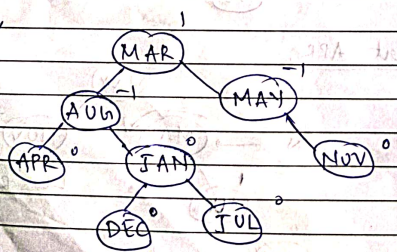


Ambiguity about JAN so insert it according to rule of BST

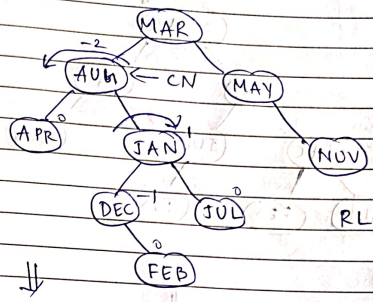
Insert DEC,



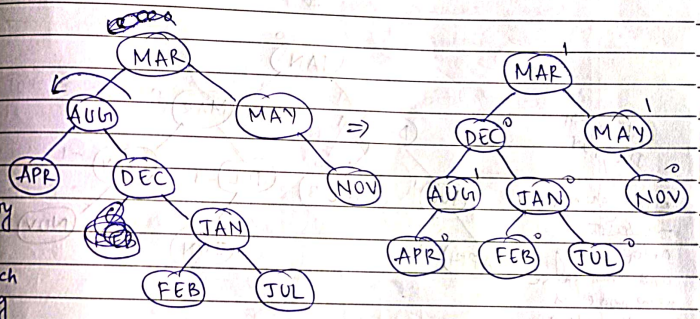
Insert JUL,



Insert FEB,

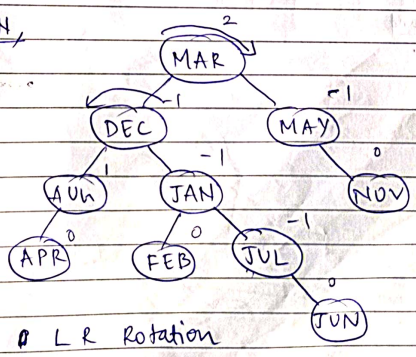


RL Rotation

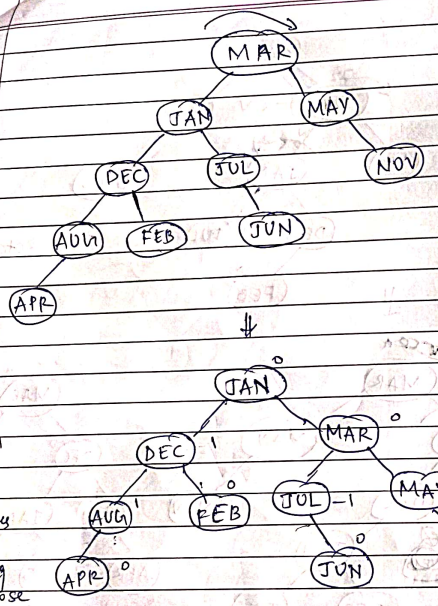


Ambiguity about FEB so attach it using BST rule.

Insert JUN,



LR Rotation



Ambiguity in JUL, just make changes ~~of left~~ after rotating and then chose new node as start node;
Here JAN for JUL and JUN.

Huffman Encoding Technique.

→ ASCII → fixed length coding → each letter 7 bit coding
7 bits
A → 7 bits → more
Z → 7 bits frequency.

→ Huffman : Variable length Encoding said to use
~~Encoding~~
✓ Efficient storage.
✓ Strategy
→ Shorter codes for more frequent words
→ longer codes for less frequent words

Total no. of bits = Σ (no. of bit * freq)

eg; MISSISSIPPI

Step 1 – Construct freq table. (This on last)

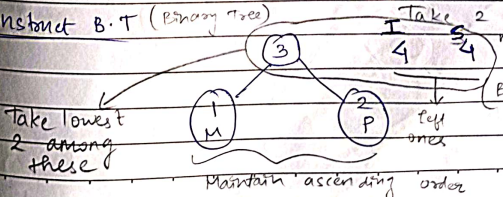
M – 1	– 100	1 X 3 = 3
I – 4	– 11	4 X 2 = 8
S – 4	– 0	4 X 1 = 4
P – 2	– 101	2 X 3 = 6

Total = 21 bits

Step 2 – Arrange in ~~A-0~~ A·O (~~Alphabetical~~ order) Ascending
Arrange frequencies in ascending

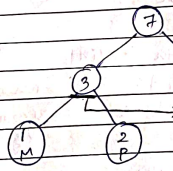
M	P	I	S
1	2	4	4

Construct B.T (Binary Tree)



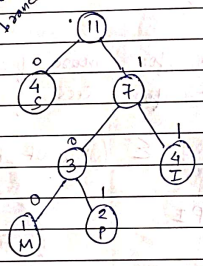
Page No.:
Date:

S
4



→ Maintain the order according to ascending order. (3 less than 4 hence 4 on right.

we are concerned about the bits not about the code code can be taken we branch anyhow. there we branch choose 0 for left branch and 1 for right branch



All left branches leveled as 0 and all right branches as 1

(Reach the letter from the root node)

2) MALYALAM

3) DATA STRUCTURE (If DATA STRUCTURE then consider space also as a character)

4)

Symbol	P	q	e	r	i
Frequency	20	17	33	25	40

Hacking, Alteration of data hence encoding required

AABC — Medium — XXXX

Transmitter — Receiver

Decode AABC

Page No.:
Date:

Letter	Frequency	Code	Bits.
M	1	100	3
I	4	11	8
S	4	0	4
P	2	101	6

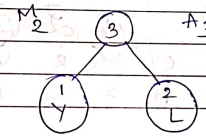
Total bits = 21

← for MISSISSIPPI final ans.

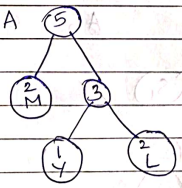
2) MALYALAM

M - 2
A - 3
L - 2
Y - 1

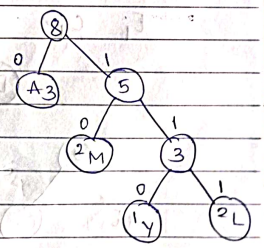
Y	L	M	A
1	2	2	3



If we take A_3 first or generated 3 first with 2 it will do. Though the code for every bit would be ~~2 bits~~ 2 digits it will do. Not concerned about code concerned about bits.



⇒



Letter	Frequency	Code	Bits.
Y	1	110	1×3 = 3
L	2	111	2×3 = 6
M	2	10	2×2 = 4
A	3	0	3×1 = 3

Total bits = 4+3+6+3 = 16 bits.

3) DATA STRUCTURE (If space present then consider space also as a digit)

D - 1
A - 2
S - 1
T - 3
R - 2
U - 2
C - 1
E - 1

2 2 2 2 4 3

1 1 1 1 2 2 2 3

1 1 1 2 2 2 2 3

2 4

2 2 2 2 2 3

D	S	E	C	A	R	U	T
1	1	1	1	2	2	2	3

E_1 C_1 (2) D_1 S_1 A_2 R_2 U_2 T_3

(2) (2) A_2 R_2 U_2 T_3

E_1 C_1 D_1 S_1

Take the pair of two which are the least in ascending order.

A_2 R_2 U_2 T_3

8

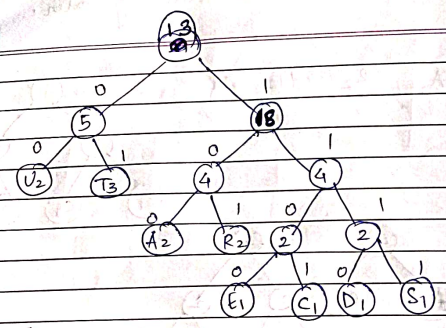
(4) (4)

A_2 R_2 (2) (2)

E_1 C_1 D_1 S_1

(5) (5)

U_2 T_3



Letter	Frequency	Code	~~No of bits~~ · Total no. of bits
D	1	1110	1×4 = 4
A	2	100	2×3 = 6
S	1	1111	1×4 = 4
T	3	01	3×2 = 6
R	2	101	2×3 = 6
U	2	00	2×2 = 4
C	1	11101	1×4 = 4
E	1	11100	1×4 = 4
			38 bits.

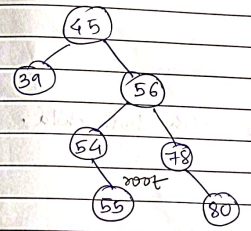
Successor / predecessor → Inorder traversal key

Left subtree ke liye MAX
Right subtree ke liye MIN

Deletion of a Node in BST

- ✓ properties remain intact
- ✓ Nodes are not lost

Case 1: Deleting a node with no child i.e Leaf.

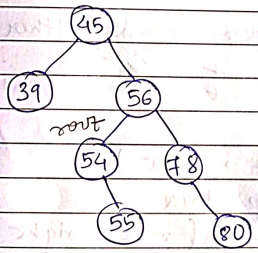


Delete 55

1) Find the node
2) Free the node

```
if (root → left == NULL
    && root → right == NULL)
{
    free(root);
    return NULL;
}
```

Case 2: Deleting a node with one child.



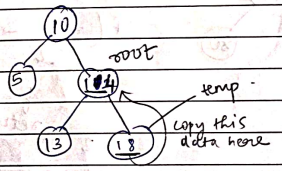
Delete 54

Strategy - i) Find the node
Replace the data of the node to be deleted with its child.

```
//right child.
if (root → left == NULL)
{
    temp = root;
    temp = temp → right;
    root → data = temp → data;
    free(temp);
}
```

// left child
else if (root → right == NULL)
{
temp = root;
temp = temp → left;
root → data = temp → data;
free (temp);
}

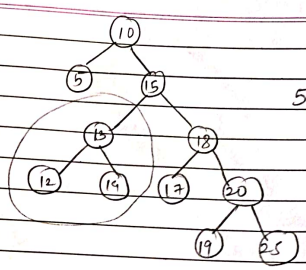
Case 3 : Delete the node with both child.



Inorder Traversal – 5, 10, 14, 12, 18

1) Find the node
2) Delete the node & replace it with either successor or predecessor.
↳ node before root. parent
node after root parent

temp = FindMin (root → right)
root → data = temp → data
root → right = delete (root → right, temp → data);



5 10 12 13 14 15 17 18
19 20 25

~~Huffman coding~~

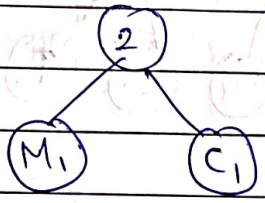
Data Structures.

Huffmann Coding

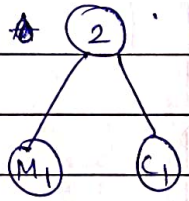
MACHHINDRANATH

M - 1
A - 3
C - 1
H - 3
I - 1
N - 2
D - 1
R - 1
T - 1

M	C	I	D	R	T	N	A	H
~~1~~	~~1~~	1	1	1	1	2	3	3



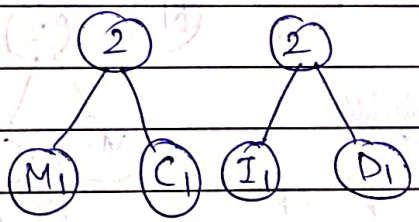
I D R T N ~~A~~ 2 A H
1 1 1 1 2 3 3



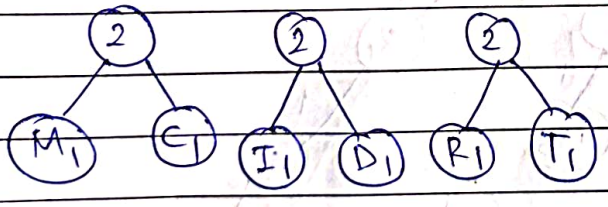
can arrange in any order for equal no.s

Only the two ~~nodes~~ taken must be the least from ascending order

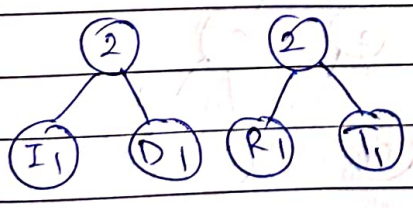
R T N 2 2 A H
1 1 2 3 3

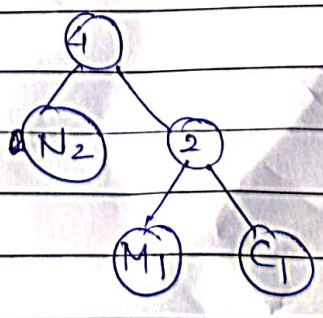


N A H
2 3 3



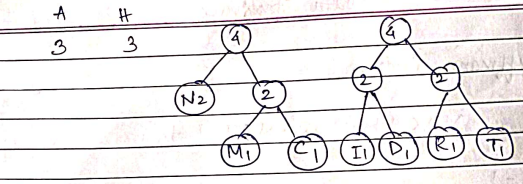
A H
3 3

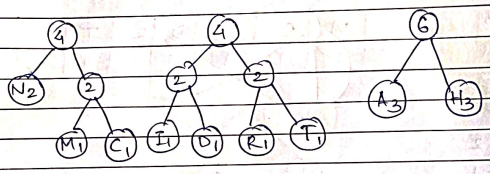


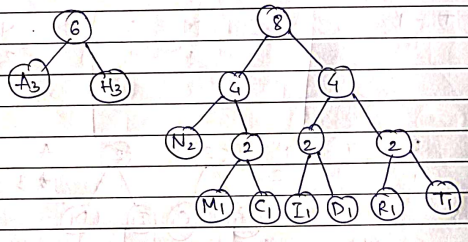


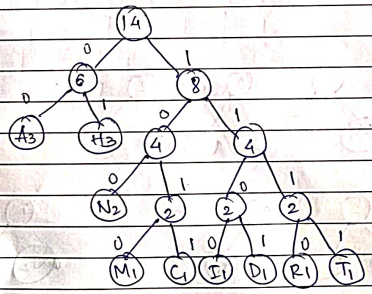
Always take pair of least two

A H
3 3









code can be different only no. of bits must match.

Letter	Frequency	Code	Total no. of bits.
M	1	1010	1×4 = 4
C	1	1011	1×4 = 4.
I	1	1100	1×4 = 4.
D	1	1101	1×4 = 4.
R	1	1110	1×4 = 4
T	1	1111	1×4 = 4
N	2	100	2×3 = 6
A	3	00	3×2 = 6
H	3	01	3×2 = 6
			42 bits.

04/10/24

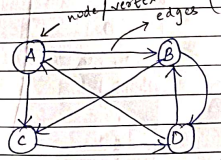
Graphs

Page No.:
Date:

Graphs

Ways to represent a graph in computer is Memory.

eg.



G = { Vertices, Edges }
collection of vertices and e...

Two types

Directed (with arrows) → Degree (In and out separate counted)

Undirected (without arrows) → Degree (In and out degree counted together)

1) Adjacency Matrix

A always square matrix

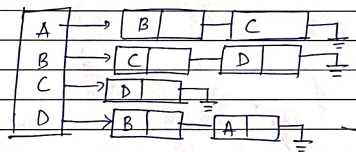
	A	B	C	D
A	0	1	1	0
B	0	0	1	1
C	0	0	0	1
D	1	1	0	0

A to adjacent is B and C

Presence of edge = 1
Absence " " = 0

↳ To show adjacency.

Adjacency List



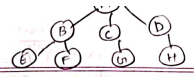
→ Easy to update (Insert and delete)

→ Most of memory go in address.

Which is better · Why ? → Depends on which application we want.

~~Not~~ Both are better

What are different



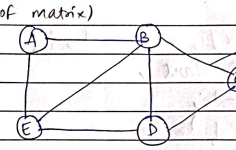
A-B-C-D-E-F-G-H

Page No.:
Date:

Graph Traversals
↳ is drawn breadthwise.

1) Breadth first search – DS → queue.
(on basis of matrix)

can be applied to any of non linear data structure applicable to trees)



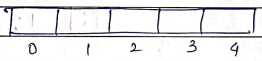
first node and then its direct connection and the visit individual connection and their further connections

~~Matrix~~ Matrix –

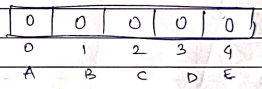
	A	B	C	D	E
A					
B					
C					
D					
E					

Undirected matrix both side connected

queue



Visited



Maps use graphs.

WAP to Implement BFS

```
void BFS ( int adj[ ][ ], int v, int start)
{
    int q[MAX], r=f=-1, i;
    q[++r] = start;
    v[start] = 1;
    while (r != f)
    {
        start = q[++f];
        printf("%c \t", start+65);
        for (i=0; i<MAX; i++)
        {
            if (adj[start][i] == 1 && v[i] == 0)
            {
                q[++r] = i;
                v[i] = 1;
            }
        }
    }
}
```

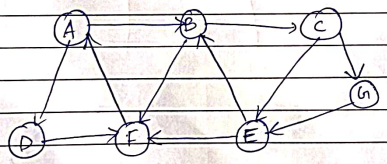
For passing:

```
bfs (adj, v, 0);
```

(0 → start index zero.)

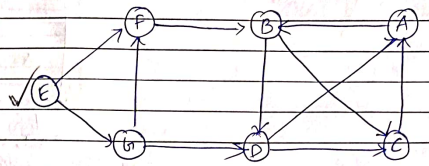
Any node can be struck off just after removing it from queue and store in queue and visited queue.

eg,



A B D C F E G

RAM → Not the name (It's the type of memory)



E – F – G – B – D – C – A

(E = Start)

Searching [10 M Sure Q.] V.V. V Imp Topic

Hashing – (Insertion as well as Searching)

→ Searching technique
→ Key mapped to some address in the hash table.
→ Using Hash function.

Hash Function

→ Function that transforms a key to table Index / add

an attribute ← Key → Hash function → address
that uniquely determine a tuple.

(Key)
138 → Key % 10 + 1 → 9 h (Key)

n/(n)

Table size (No. of rows) — any no.

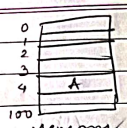
In exam function given

primary clustering – group on one side
secondary " – group in btwn

Page No.:
Date:

Rehashing → Two functions

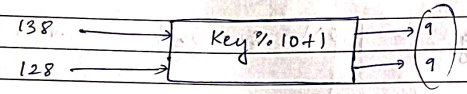
Not in order



Memory inside
↓
RAM, ROM, Cache
Secondary
(Hard Disk)

Synonyms

→ All those keys that hash/map to same table index.



Collision

→ An event for synonyms
→ Keys mapped to same index

ID	Name	Score
10	A	–
20	B	–
30	C	–
40	D	–
50	E	–
60	F	–

Memory Access Methods –

→ Direct Access / Random
→ Sequential Access.

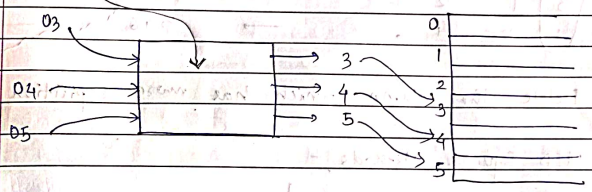
Table size 10 → 0–9 → for Key % Table size
10 → 1–10 → for Key % Table + 1

Page No.:
Date:

Types of Hash Function

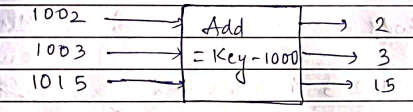
1) Direct Hashing

→ Key = Address
→ No manipulation.



→ Limited situations
→ No synonyms → No collision. (Advantage)

2) Subtraction Method



Advantage – No synonyms → No collision.

3) Modulo Division Method

→ Division Remainder Method
→ add = (key % Table size) + 1

Key = 105, Total size = 100.

add (105 % 100) + 1 = 6

mostly not used

4) Digit Extraction - X

	Key				Add
1	2	3	4	→	234
2	7	9	3	→	793
1	8	6	5	→	865
3	1	2	7	→	127
1	5	4	6	→	546

Delete the column which has more repetition.

5) MID Square Methods -

Key = 986
$(Key)^2$ = 97 (21) 96
add

Key = 123
$(Key)^2$ = 15129
∴ add = 51/12

6) Folding Methods -

Memory
RAM, ROM, Cache, Secon... (Has...

1) Key : 1 2 3 4 5 6 7 8 9

123
456
789
1368
↑ discard
add

Fold Shift

321
456
987
1764
discard
Folded like a box on w... here, 456 add

Fold Boundary.

Page No.:
Date:

Collision Resolution

- Open Addressing
- Chained / Linking

☆ Open Addressing

→ Hash Table search for an alternative open position where the new element can be placed

i) Linear probing -

1) Key = 50, 71, 30, 82, 61

Hash function = Modulo division
Table size = 10

* Count the number of collisions

h(50) = 50 % 10 = 0+1 = 1
h(71) = 71 % 10 = 1+1 = 2
h(30) = 30 % 10 = 0+1 = (1)

h(30) = (2) ←
h(30) = 3

Collision
h(82) = 2+1 = (3)
∴ h(82) = 4
h(61) = 1+1 = 2
= 3
= 4
= 5

0	50 / (30)
1	71 (30)
2	30 ✓
3	82
4	61
5	
6	
7	
8	
9	

Limitation - Primary Clustering

2) Quadratic Probing -

$= (h + i^2) + \text{table}$

table size = 10 [0-9]

Key = 99, 33, 23, 44, 56, 43, 19

$h(99) = 99 \% 10 = 9 \quad (i = 0)$

$h(33) = 33 \% 10 = 3 \quad (i = 0)$

$h(23) = 23 \% 10 = 3\ (C_1) \longrightarrow (i = 0)$

$h(23) = (23 + 1^2) \% 10 = 24 \% 10 = 4 \longrightarrow (i = 1)$

$h(44) = 44 \% 10 = 4\ (C_2) \longrightarrow (i = 0)$

$h(44) = (44 + 1^2) \% 10 = 45 \% 10 = 5 \longrightarrow (i = 1)$

$h(56) = 56 \% 10 = 6$

$h(43) = 43 \% 10 = 3\ (C_3) \longrightarrow (i = 0)$

$h(43) = (43 + 1^2) \% 10 = 4\ (C_4) \longrightarrow (i = 1)$

$= (43 + 2^2) \% 10 = 47 \% 10 = 7 \longrightarrow (i = 2)$

$h(19) = 19 \% 10 = 9\ (C_5) \longrightarrow (i = 0)$

$= (19 + 1)^2 \% 10 = 0$

Add	Key
0	19
1	
2	
3	33
4	23
5	44
6	56
7	43
8	

Limitation ↓ Secondary Clustering

3) Rehashing / Double Hashing

Mostly not asked

$h_1 \longrightarrow$ primary HF
$h_2 \longrightarrow$ secondary HF
} Two Hash Function

Key (138) Table size = 10

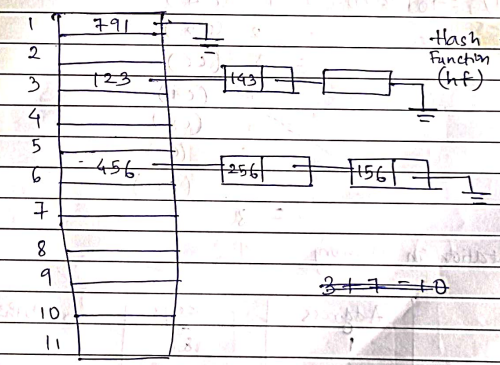
$h_1 = 138 \% 10 + 1 = 9$

$h_2 = (9) - 4 = 5 \longrightarrow$ Any function.

☆ Chaining / Linked List Resolution

Uses linked list. Hence more memory.

Keys : 123, 456, 791, 143, 256, 156



Hash = Key % 10
Function (HF)

~~3 + 7 = 10~~

Q. Consider a Hash Table of size 10. Use Linear Probing to insert the keys 72, 27, 36, 24, 63, 81, 92, 101. Use the function $h(k) = \text{mod } 10$.

→

$h(72) = 72 \% 10 = 2$
$h(27) = 27 \% 10 = 7$
$h(36) = 36 \% 10 = 6$
$h(24) = 24 \% 10 = 4$
$h(63) = 63 \% 10 = 3$
$h(81) = 81 \% 10 = 1$
$h(92) = 92 \% 10 = 2$ (C1)
$= 3$ (C2)
$= 4$ (C3)
$= 5$
$h(101) = 101 \% 10 = 1$ (C4)
$= 2$ (C5)
$= 3$ (C6)
$= 4$ (C7)
$= 5$ (C8)
$= 6$ (C9)
$= 7$ (C10)
$= 8$

Representation in Memory -

Address	keys
0	
1	81
2	72
3	63
4	24
5	92
6	36
7	27
8	101

→ Nothing at 0

OR

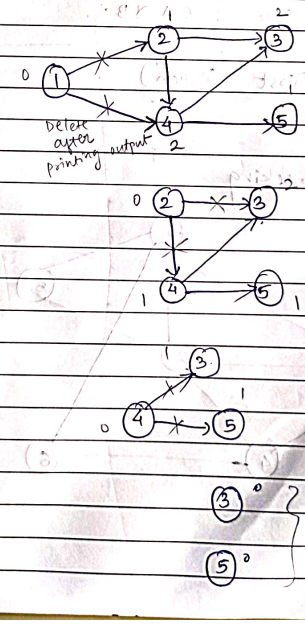
-	81	72	63	24	92	36	27	101	9	10
0	1	2	3	4	5	6	7	8		

Topological Sorting (order in which we visit nodes)

→ DAG Directed Acyclic graph.
→ $u \to v$ so in order (Not be any cycle existing)
u should come b4 v. (u can't do topo sorting if there is cycle)
→ atleast 1 order
→ Use in graphs, networks, routing, etc.
→ Start with vertex having indegree zero. (Nothing coming towards node.)

There can be more than one order

In exam if not given show egs for both cyclic and acyclic sorting



D/p

~~1 2~~ 1
1 2
1 2 4
Final o/p { 1 2 4 3 5 ✓ / 1 2 4 5 3 ✓

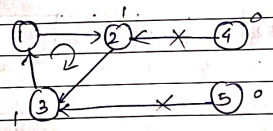
ACAB
CADB

Keep on pushing and popping
↓
Stack backtracking

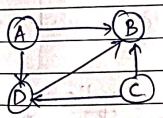
Page No.:
Date:

Q Solve.

2)



3)

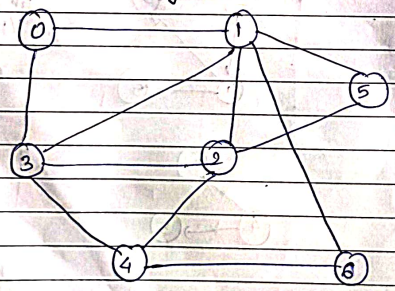


O/p 2) → case 1 : 0 4 5 - - . (Not possible)
case 2 : 0 5 4 . . - .

3) → case 1 : ACDB
case 2 : CADB.

DFS (Depth First Search)

✓ Uses stack
✓ Involves backtracking



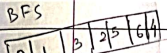
DFS → 40 10 30 60 70 20 50
BFS - 40 10 20 30 50 60 70

Page No.:
Date:

Adj. Matrix.

	0	1	2	3	4	5	6
0	0	1	0	1	0	0	0
1	1	0	1	1	0	1	1
2	0	1	0	1	1	1	0
3	1	1	1	0	1	0	0
4	0	0	1	1	0	0	1
5	0	1	1	0	0	0	0
6	0	1	0	0	1	0	0

BFS

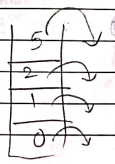


0 1 3 2 5 6 4

After completing one level then next level.

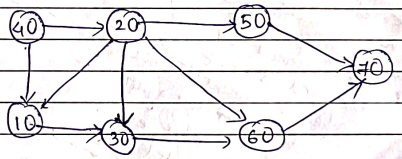
O/p → 0 1 2 3 4 6 5

In last



Keep on changing the row with column visited.

eg;



Start vertex
↓
40
given in q.

Draw matrix in ascending order only

Print before pushing

Adj Matrix -

	40	10	20	30	60	50	70
40	0	1	1	0	0	0	0
10	0	0	0	1	0	0	0
20	0	1	0	1	1	1	0
30	0	0	0	0	1	0	0
60	0	0	0	0	0	0	1
50							
70							

	10	20	30	40	50	60	70
10	0	0	1	0	0	0	0
20	1	0	1	0	1	1	0
30	0	0	0	0	0	1	0
40	1	1	0	0	0	0	0
50	0	0	0	0	0	0	1
60	0	0	0	0	0	0	1
70	0	0	0	0	0	0	0

First visit start vertex and print its adjacency and then keep on changing the start to next visited/printed element after start.

BFS → 40, 10, 20, 30, 50, 60, 70

Visit the start vertex then visit the first adjacent element and then move to that elements row. and do the same. If the row is empty or had no adjacent element then pop that element and move to previous element and then find its next adjacent element. If not then pop it and again move to prev element of visited list.

DFS → 40, 10, 30, 60, 70, 20, 50.

Stores more than one data in a specific node. (Height limited searching easy)

B+ Tree

No node at middle nodes. They keep data on leaf nodes

m-way Search tree

1) m → Order of the tree

m-1 → Keys per node
m → No. of subtrees.

eg; 1) m = 2 m-1 = 1 data / Keys.

Binary tree.

2) m = 3 , m-1 = 2 keys