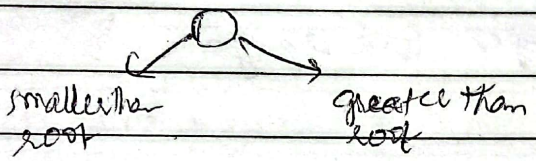
Date 29 NOV 21

Binary Tree

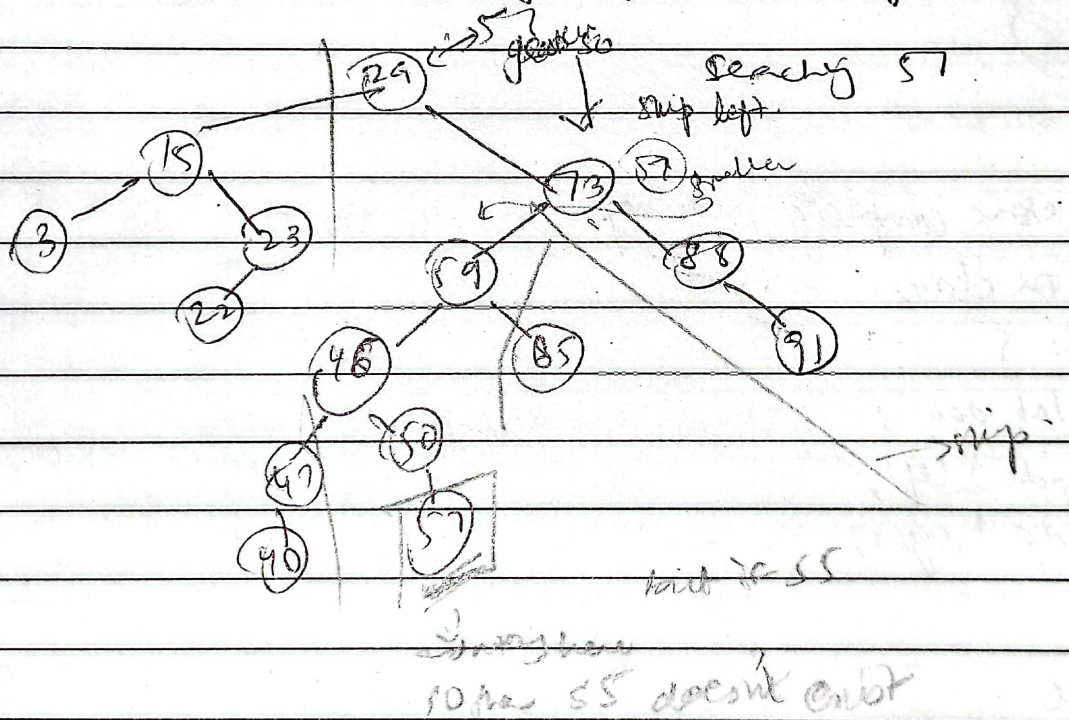
each node can have almost 2 children.

Binary search tree :-

→ follows an order



→ complexity is reduced by half at each point



→ structure depends on the order of keys.

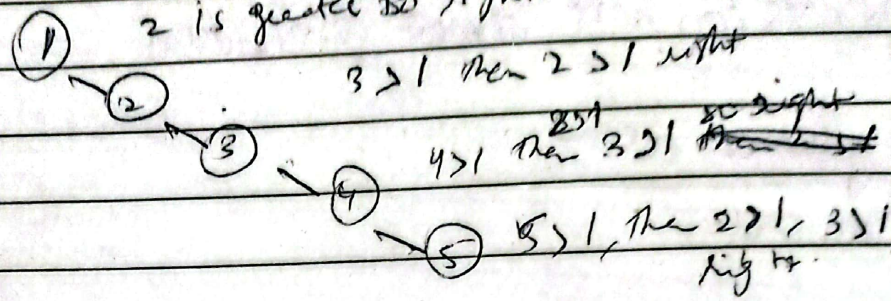
heap - has insertion, deletion & pass restructure.

Date

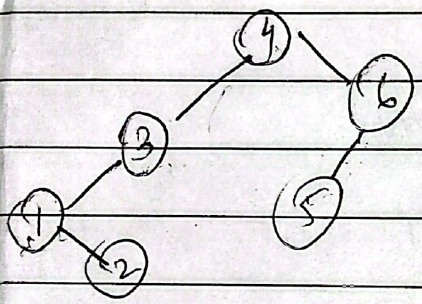
Branches:

1 2 3 4 5 → keys.

Insertion :-



4 3 1 6 5 2 order change, tree change.



→ can only store complete in array so will store this in structure, or class-

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

leaf :- data.
both parts → Null

Insert, update

↓

search value then change its value.

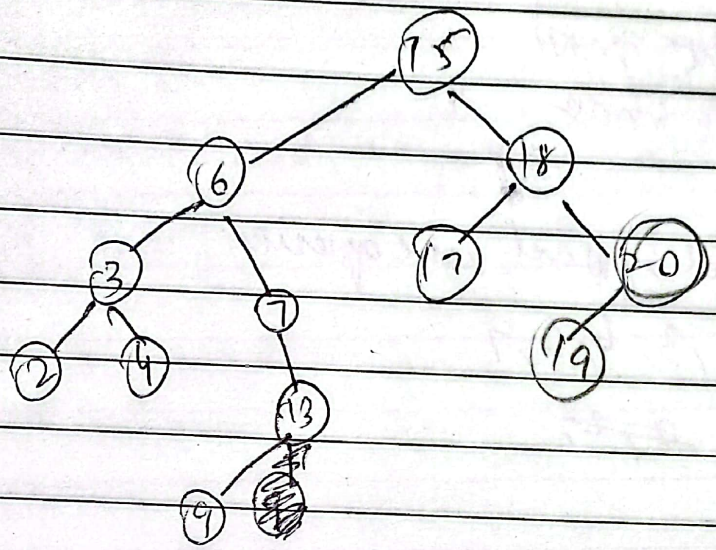
Traversal → breadfirst traversal BFT, DFS

→ Nodes are keys that have values stored on it.

Date

in students id can't be repeated
gpa can be repeated

Insert 19



→ what if says insert 18.
→ can't write same values.
→ we always talk about keys in this, id, can't be repeated

Path → 15 → 18 → 20 → 19

search 21 path → 15 → 18 → 20. not found. / NULL.

Update: change (20) to (21)

15 → 18 → 20.
21

Deletion

3 cases:- search.

① → if is leaf node, doesn't matter.
② → it has one child.
③ → it has 2 children.

① If leaf. no issue, just delete it after searching.
use parent node & next = NULL

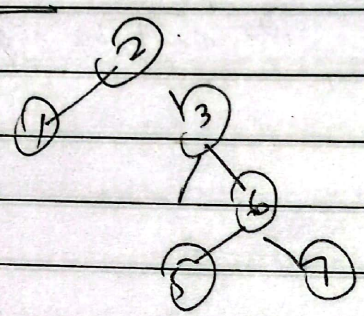
② has one child:, place the child to that node's place

③ if has 2 children, right tree, left most tree will be the smallest so replace root with it if deleting root.
or left tree, right most value

Date

max → right tree's right most
min → left tree is left most.

BFT traversal. Breadth first level by level, use queues.



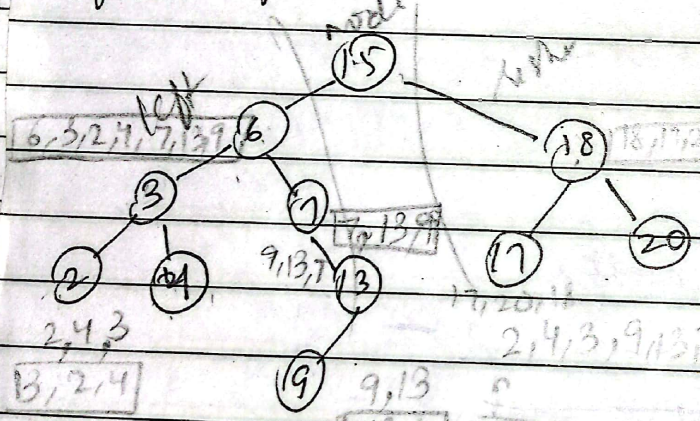
2, 1, 3, 6, 5, 7

~~2, 1,~~

DFT traversal. Depth first search traversal

Inorder, Preorder, Postorder. [uses stack]

left node right — sorted order — NLR — LRN



Inorder: LNR 6, 15, 18 Ascending

~~2, 3, 4, 6, 7, 13, 9, 15, 17, 18, 20~~

2 3 4 6 7 9 13 15 17 18 20

→ is sorted order.

Preorder: NLR

Hometasks → postorder, preorder.

15, 6, 3, 2, 4, 7, 13, 9, 18, 17, 20.

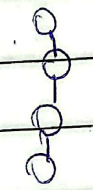
Date _____

POST = Push
Push unvisited child
pop (display)

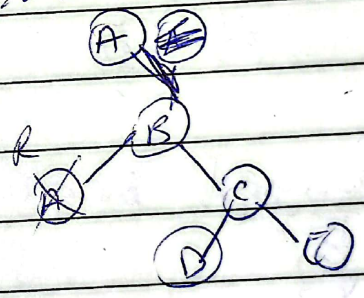
Pre = display when push
push unvisited
pop

Inorder
push if left subs. leave unvisited
display else
right,
backtrack main display.

DSW algorithm
→ after inserting, balances.
→ has 2 phases
1st phase → forms a backbone like
2nd phase
→ that balances, in which it rotates every second node (alternate ko dekha)



linear banadeta hai.



→ rotate A, will become parent of its parent

```
CreateBackbone (root, n){
    temp = root;
    while (temp != NULL) { → till leaf node
        if (temp->left != NULL)
```

if no left child, becomes right child.

one child

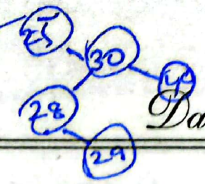
left

right

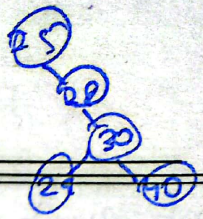
will become left subtree's rightmost child.

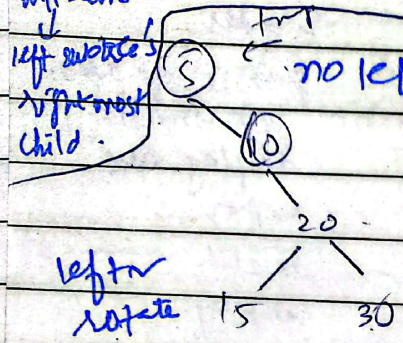
right subtree ka left most child.

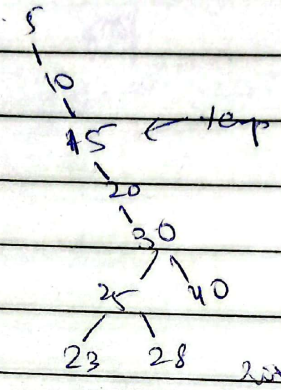
eg

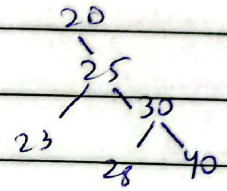


Date







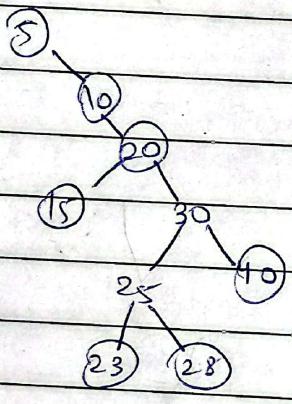


rotate 25 / big 23 ~ it.

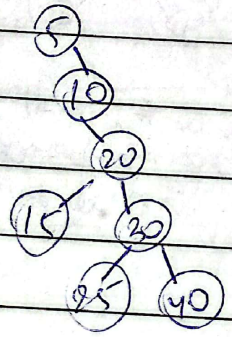
~~make 25 of third gets parent~~

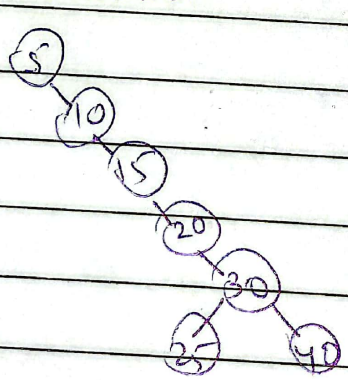
it was right child

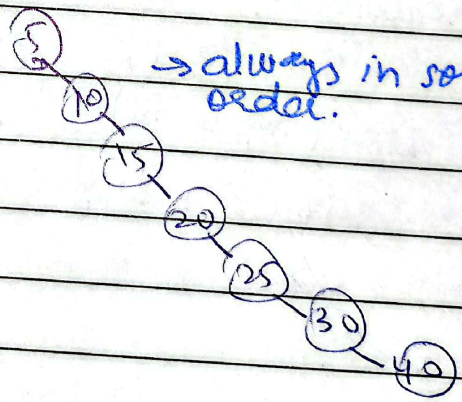
or right subtree. ka left most child banega.



6. exercise 1 Create backbone.



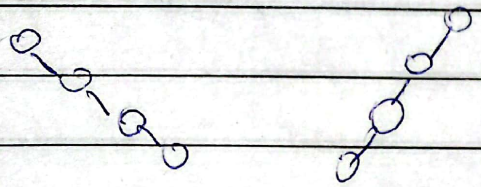




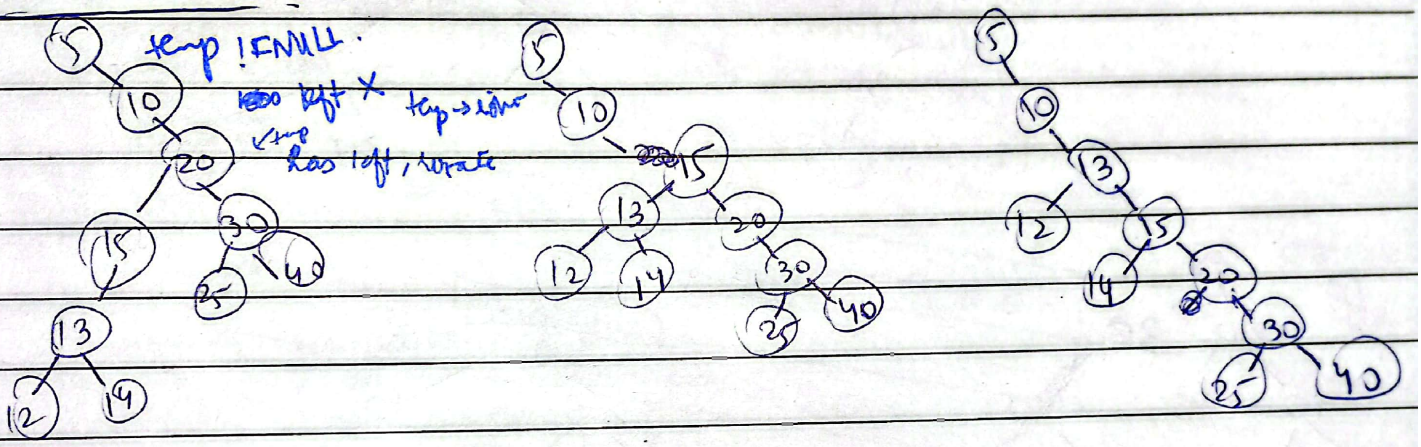
→ always in sorted order.

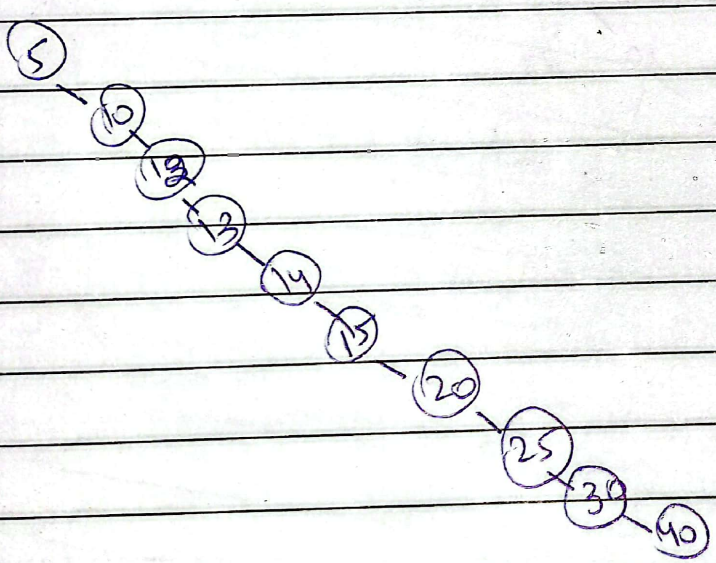
Date ~~goves~~ for rotation

1. temp.
2. has 1 child.
3. has 2 children.



Create backbone:-





Second phase

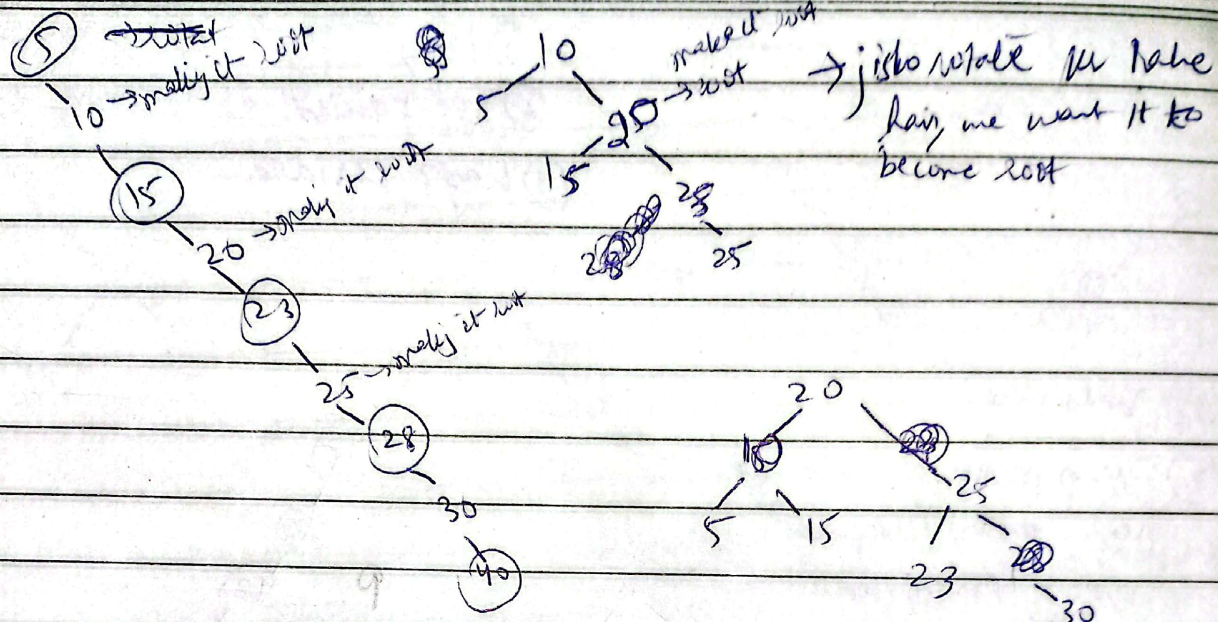
→ Rotate every second node.

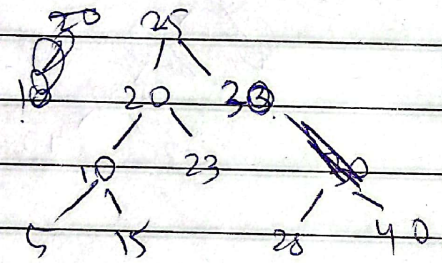
Previously ↑ rotation. Now ↓ rotate.

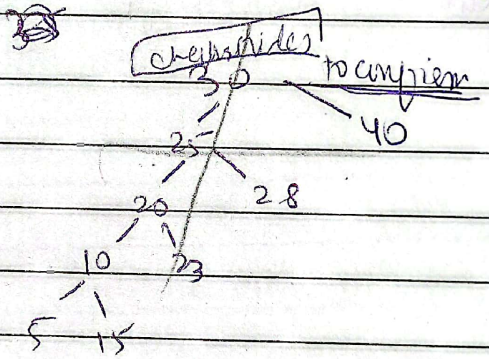
parent becomes child.

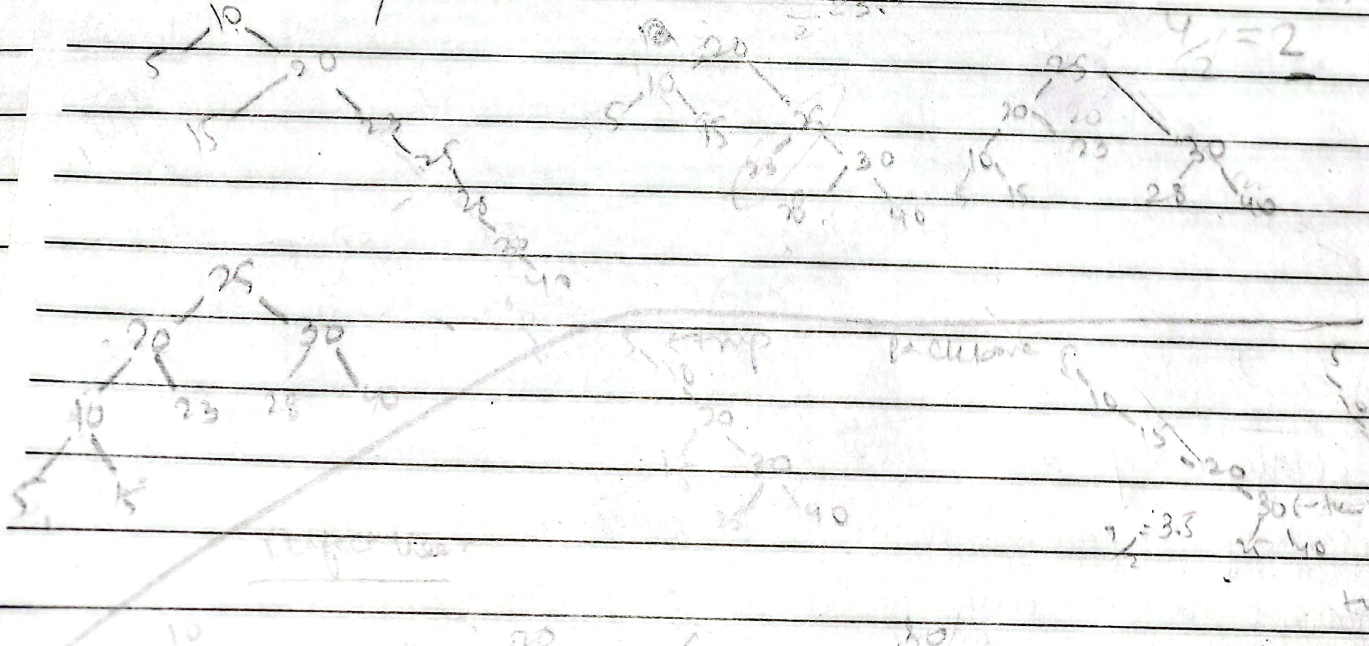
Balanced Binary tree = $2^n - 1$

Date



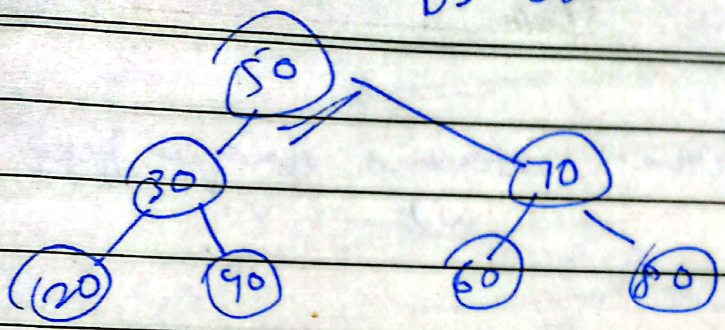






DS Lab

Date



successor: use greater.

predecessor: use petit / chote.

inorder always gives sequential data.

20, 30, 40, 50, 60, 70, 80

30's predecessor = 20.

30's successor = 40

Deletion:

Delete root, leaf or successive child (i.e parent and a child so

If has one child, swap child w its data, then delete it.

has 2 children, will swap greater one i.e right child's value.

right is most left for the min value.

Date

Huffman:

compresses string format, attaches unique freq wit.

aaabbcedeefghhbcd.

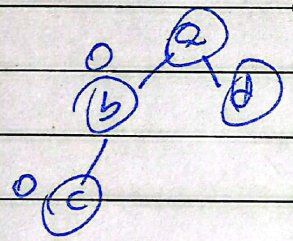
freq
a = 3
b = 3
c = 2
d = 2
e = 3
f = 1
g = 1
h = 2

compressed = a3b3c2d2e3fgh2

construct BST for it.

code length of the most repeated letter is the least.

code length = height



leaf node height = 0

give 0 height to left, 1 to right

code length:-

c = 0 0

b = 0

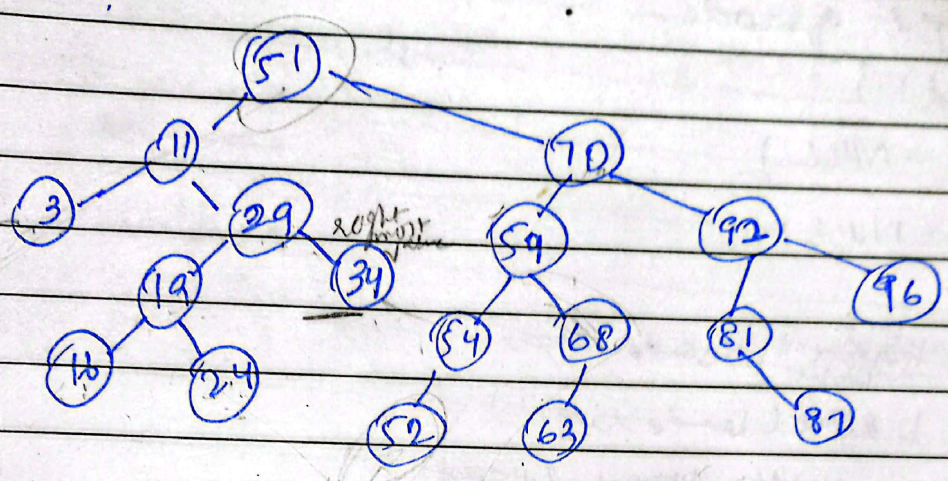
d = 1

→ 1st identify 2 min values.

f = 0 0 1 1

DS Theory.

Date 30 NOV '21



Inorder predecessor : LST → Right most value.

Inorder secessor : : Right ST → left most value

PNS

51 Predecessor of 51 : 34 (nearest 51 and biggest in left subtree)

BFS Breadth first Search → by Queue

DFS Depth First " → by stack

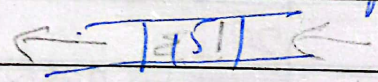
BFS → level k hisaabse.

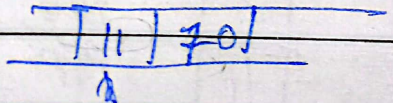
51, 11, 70, 3, 29, 59, 92, 19, 34, 54, 68, 81, 96, ...

To access it:

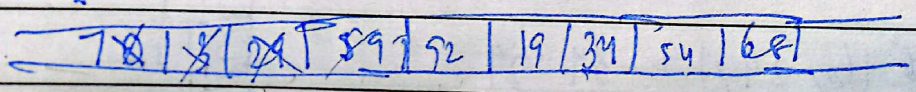
will use queue for it.

→ Initially empty, insert root, then extract display then insert its offsprings.





11 delete, then insert its offsprings.

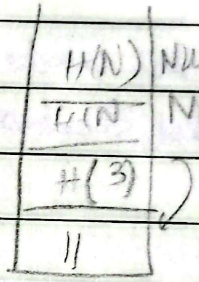


.... unless the queue is empty.

Date

Calculate height of node.

```
Height ( head ) {
    if( head == NULL ).
        return NULL;
    else {
    LHeight = Height (head, left)
    RHeight = Height ( head, right)
        if ( L Height > R Height)
            return (L.Height +1)
        else
            return ( R.Height +1)
    }
```



DFS will start from a root, then traversal its tree till depth.

→ start w root
→ temp variable "visited = 0" change to 1 if has visited
→ left subtree, then left side if exists
If no child, start popping all the stack

→ basically postorder

Display = 3, 16, 24, 19, 34, 29
11

DS Lab Date 7 Dec '21

Hashing function

To search normal data takes time. If desire take long in searching, insertion and deletion takes time.

Hash fnc helps in this.

Key is transformed by hash function into reduced form for data to be stored in hash table.

Hash table size ÷ max element se zyada rakhna hota hai.

{11, 22, 33, 44, 56, 72}

hash fnc ⇒.

index = 11 % 6.
(key)

11 % 6 = 5

22 % 6 ⇒ 4

33 % 6 = 3

44 % 6 = 2

56 % 6 = 2

72 % 6 = 0

0	72
1	
2	44
3	33
4	22
5	11

Folding

123 456

123

456

579

0 — 9

Date

To ~~deal~~ deal with collision, probing

① Linear probing
$h(k) + i$ → probing no.

② Separate chaining
→ is implemented by linked list and data pointed by pointers.
→ using area outside hash table that is why called open hashing

Task 1 → {10, 11, 12, 13, 14, 15, 16, 17, 18, 19}
k = 10

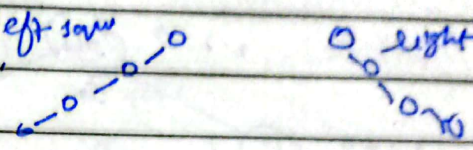
#include <list> → Stl based list library.
now we don't need to implement linked list

list<int>* hashtable → hashtable acc to list size
int hashfun(int x) {
return (x % hash_bucket); } → size of table.

DS.

Date 7 Dec'21

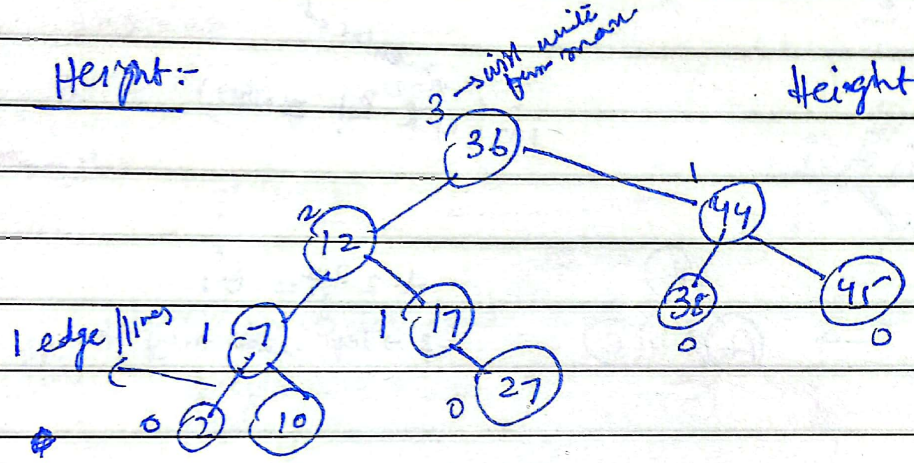
AVL tree. imp



→ Balances jis point pe kch modify ho.

① Height
② Balance factor

Height:-



Height(n) = max (H(L), H(R)) + 1

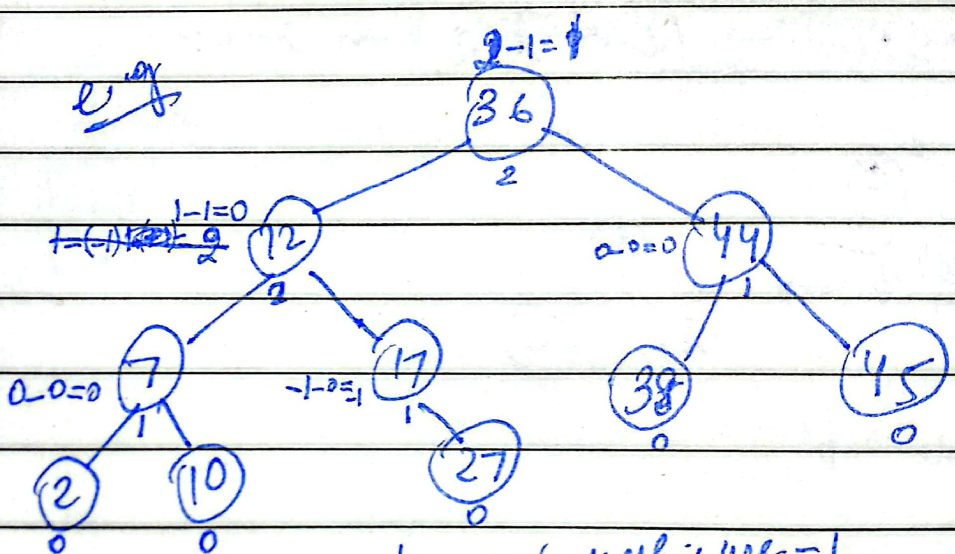
Balance factor

Difference of height bw right and left subtree

Balance factor = H(L) − H(R)

↳ can be 1, 0, −1

Other than that, node will be unbalanced.



Balance factors for this tree are either 1, 0 or −1 so is balanced.

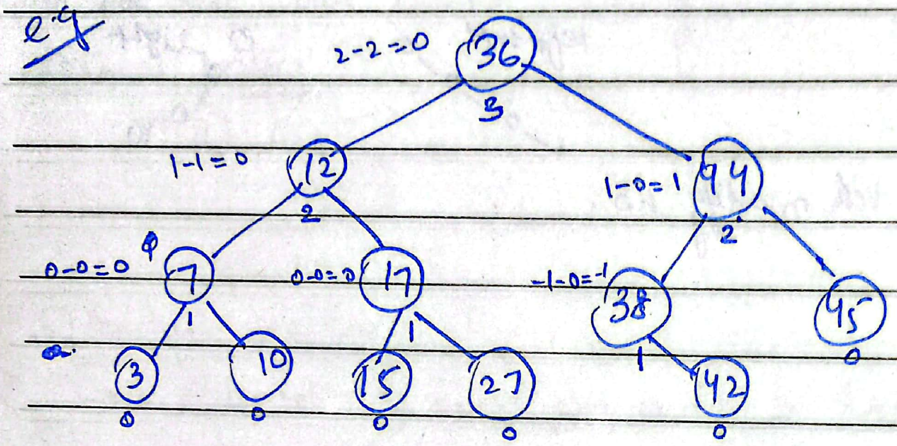
if child isn't there, use −1

Key:-
Height → below the node.
Balance factor → left side of node.

For leaf, height, balance factor = 0

Date

e.g.



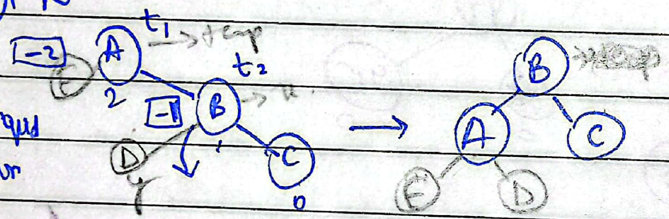
Balanced.

Rotation

single rotation:-

if (n.R.R! = NULL)
imbalanced

① Left R

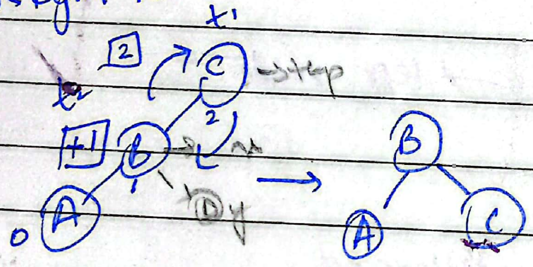


is right squed so left rotation

$t_2 \cdot left = t_1$

-2 → left rotation.

② Right R



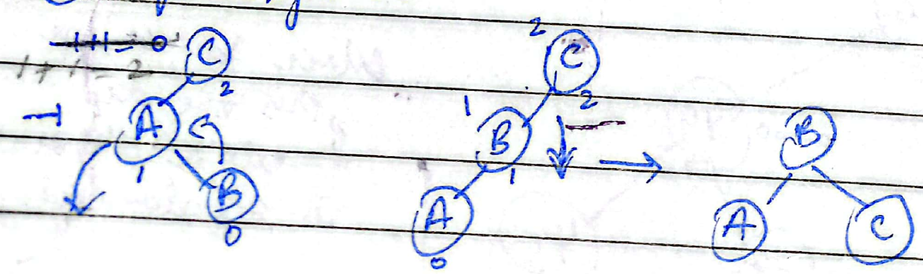
2 → right R.

left side skewed so right R.

$t_2 \rightarrow right = t_1$

Double rotation → 2 baar rotations hona hi hoti hain one:-

③ left Right R



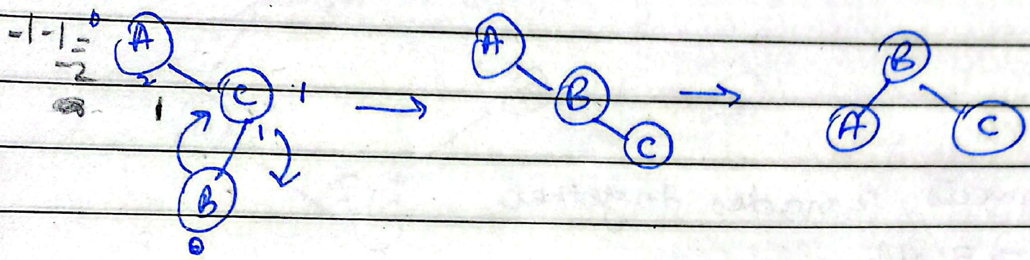
imbalance ke child pe

imbalance pe:

R
L
RL
LR

Date ____________

④ Right Left R



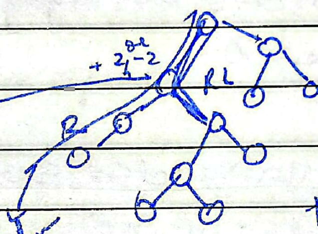
Insertion

As it self-balanced → Binary → checks at every insertion if its balanced

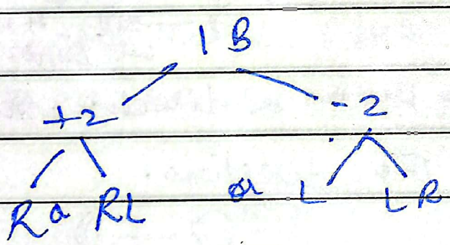
R →
L

on imbalance's child ← RL → on that node.

LR



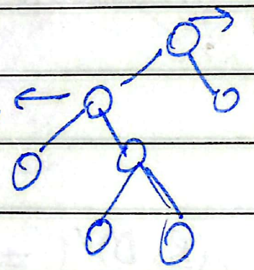
If imbalance from right tha → Jiski height zyada hogi uthar se imbalance hoga. BST / stacks / queues / hashing



not R, RL just an e.g. ←

→ choose 1 formula. +2, -2 ya dosra tareeka.

→ it imbalance, ← from right (LR)



Date 13 Dec '21

Graph

Node = vertices

edges = connects 2 nodes together

set of nodes → graph.

Properties of graph:

V(G) no. of vertices can't be empty. i.e nodes are imp.

E(G) edges can be 0.

Directed vs UnDirected graph

Directed :- edges have a direction, can only move towards that direction, arrows connect

A is linked to D but D is not linked to A

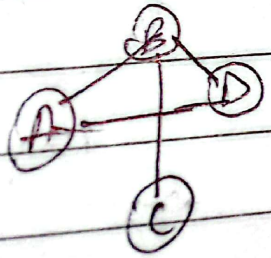
Undirected :- edges don't have a direction. We can move on both the directions.

A linked to D
D linked to A

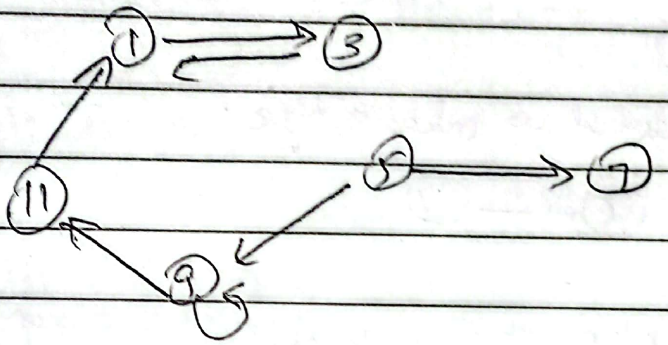
Cardinality :- No. of elements

$V(G) = \{A, B, C, D\} = 4$

$E(G) = \{(A,B), (A,D), (B,C), (B,D)\} = 4$



Date



$V(G) = \{1, 3, 5, 7, 9, 11\}$

$E(G) = \{(1,3), (3,1), (5,9), (9,11), (5,7), (9,9), (11,1)\}$

2 sides, will write seperately.

loop

Trees Vs Graphs.

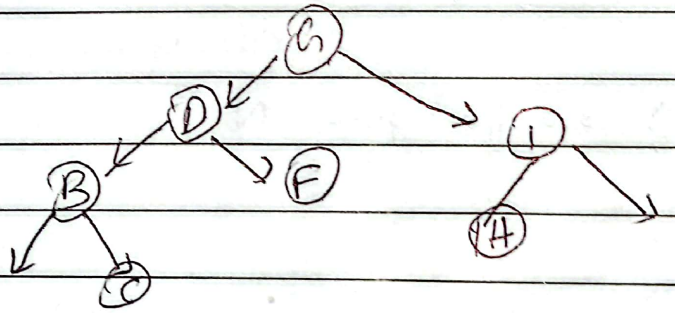
Trees are specialized version of graph.

↳ siblings nodes aren't connected w each other.

↳ so can also deal it as graphs.

↳ directed graph. (bcz can't go down to upwards if not recursive

BSF, DSF are also used for traversal of graph.



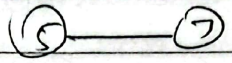
parent Points to child.

$G(V) = \{A, B, C, \ldots$

Date

Terminologies:

Adjacent nodes: linked w one edge.

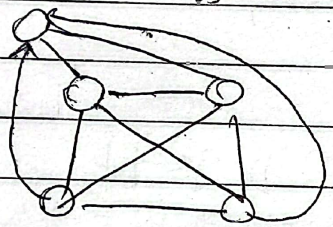


Path: sequence of vertices that connects 2 nodes.

Complete graph:-

(like complete tree:- all parents should have all children

Every vertex is linked w every other vertex.



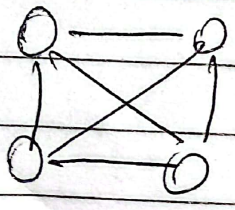
→ can or can't have loops.

No. of edges in complete undirected graph w N vertices:-

vertices = 5 = N

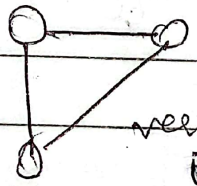
edges = 10

if complete graph → no. of edges = N



N = 4

E = 6



vertices = 3

E = 3

$$E = N * \frac{(N-1)}{2}$$

N = 5 E = 10

N = 4 E = 6

N = 3 E = 3

Date

For directed graph:- no of edges = $N * (N-1)$

$N^2 - N$

N^2 ka data

so complexity = N^2

Complexity = $O(N^2)$

Weighted graphs.

weight is associated with each edge.

value.

e.g. Journey from Khi to Es?

if concerned about cost, weighted.

else not

Implementation Representation :- → 2 methods.

Adjacency list

Adjacency matrices

Array-based

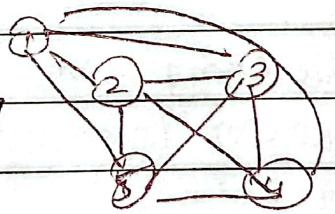
vertices in 1D array

edges in 2D array

Adjacency Matrix.

2D array.

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



Complete graph.

diagonals = 0

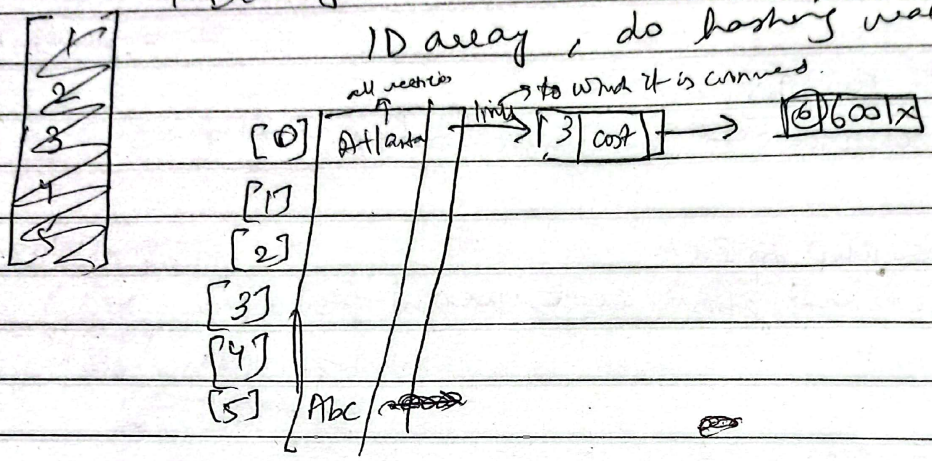
rest = 1

if weighted, values / costs will come in place of 1.

Date

Adjancey list → linked list

~~1-D array~~



Advantages of Adjancency list, matrices.

Matrix 0's utilize of space.

① Dense graph. → more edges (less 0's) → less space utilized in array → 2D array better.

sparse → less Edges. so not in this.

② Connectivity can be easily test like a[1][3]=1 ✓

chain more fine, matrix less.

list

sparse → less edge. → in matrix more 0, not good

linked list, dense, but save links; wastage of space

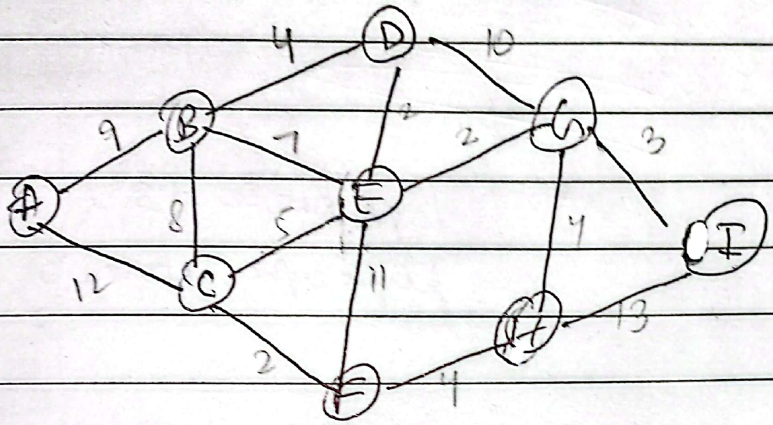
But for sparse not too extensive in linkedlist (chain)

chain also decreases searching time. O(n)

Date

Problem

Adjacency list, matrix?



Matrix:-

	A	B	C	D	E	F	G	H	I
A	0	9	12	0	0	0	0	0	0
B	9	0	8	4	7	0	0	0	0
C	12	8	0	0	5	2	0	0	0
D	0	4	0	0	2	0	10	0	0
E	0	7	5	2	0	11	2	0	0
F	0	0	2	0	11	0	0	4	0
G	0	0	0	10	2	0	0	4	3
H	0	0	0	0	0	4	4	0	13
I	0	0	0	0	0	0	3	13	0

list

A → B|9| → C|12|X

B → A|9| → C|8| → D|4| → E|7|X

C → A|12| → B|8| → E|5| → F|2|X

D → B|4| → E|2| → G|10|X

E → B|7| → C|5| → D|2| → F|11| → G|2|X

F →

G

H

I

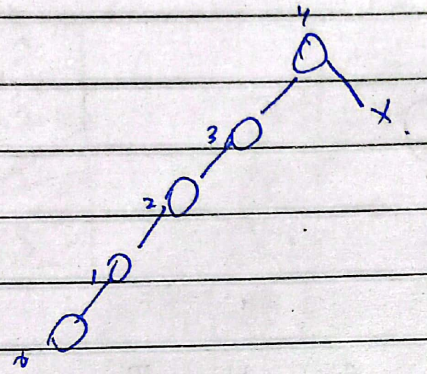
DS Lab

Date 14 Dec' 21

AVL Tree

BST = 0 to 2 children

height → reverse of level.
starts from leaf to root



height left subtree = 3

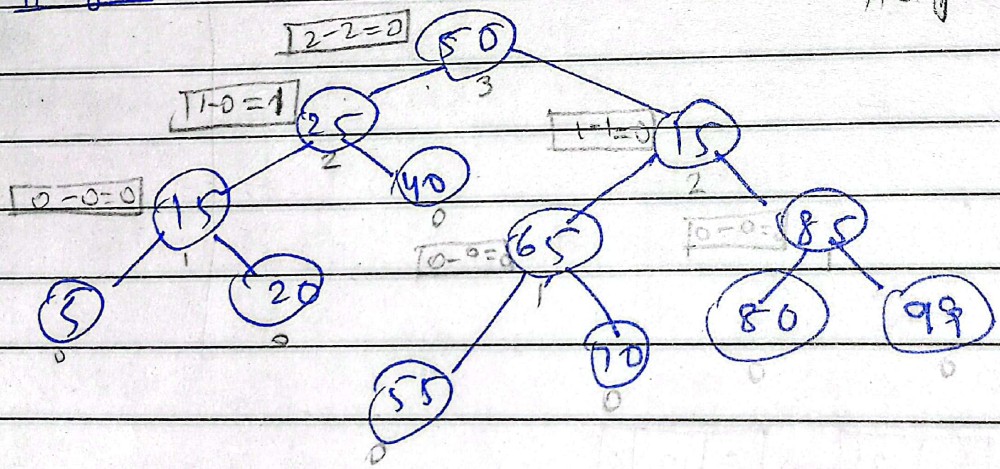
height right subtree = 0

we'll rotate to balance height

For that, we'll rotate it ~~+1>0<+1~~ -1, 0, 1

Heights - beneath a node

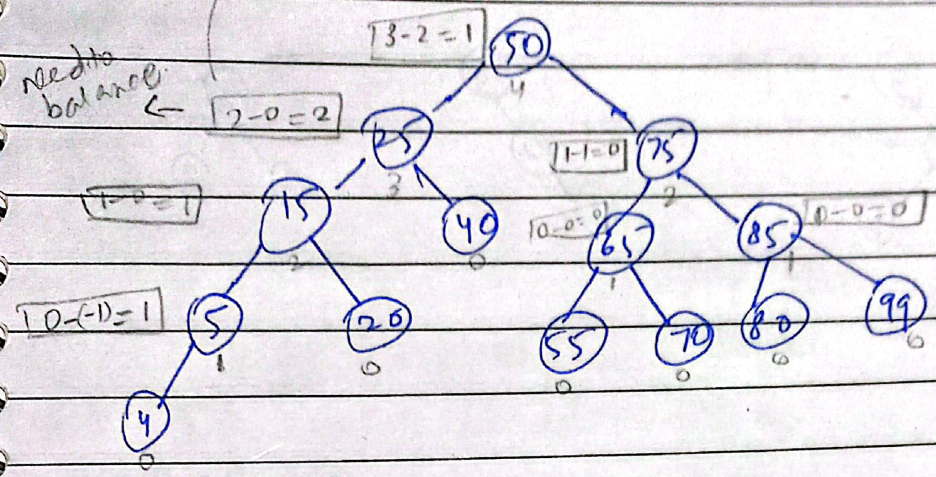
~~Height w.r.t. level~~



Balance factor:- on left of node

→ If BF of every node is either 1, 0, -1 then it is called AVL tree.

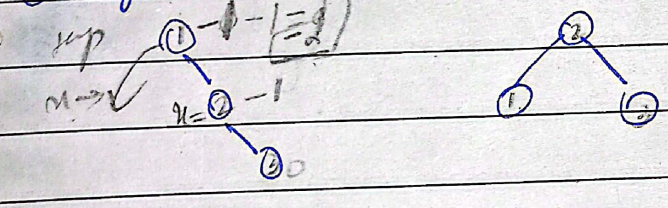
Date



→ want take root in balance factor b/c we've calculated every successive node.

Rotations

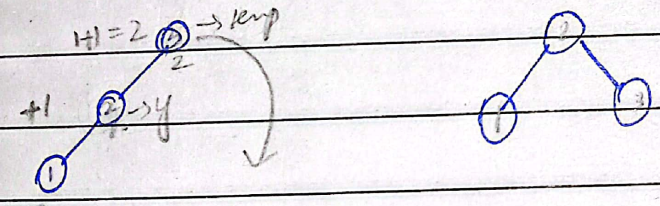
① Left R



If root node's BF = -2, Left R
and Rotation on BF = -1
same sign = 1 rotation

```
temp = root
n = temp->right
n->left = temp
temp = NULL
```

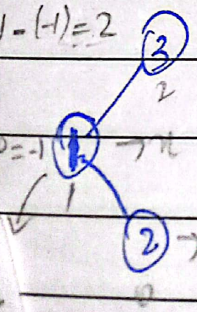
② Right R.



If root node's BF = 2, Right R
and rotation on BF = 1
parent - think of its child
same sign = 1 rotation

```
temp = root
y = temp->left
y->right = temp
temp = NULL
```

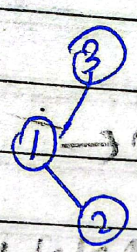
③ Left Right R



Root's BF = 2
immediate's BF = -1, then left right
diff signs = 2 rotation

1's call left rotation
① exchange data of 1,2

Date



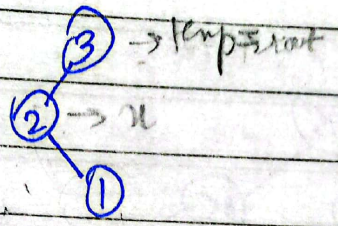
① swap data of 1, 2

int temp;

temp = x->data

x->data = x->right

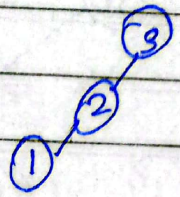
x->right = temp

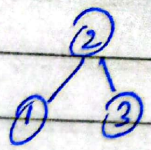


② left rotation

x->left = x->right

x->right = NULL





③ Right rotation

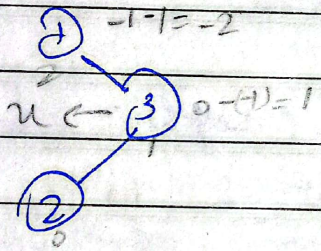
temp = root

x = temp->left

x->right = temp

temp = NULL

Right left R.



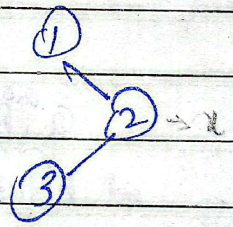
① swap ③, ②

int temp

temp = x->data

x->data = x->left

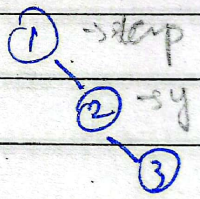
x->left = temp

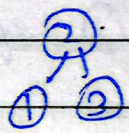


② Right rotation

x->right = x->left

x->left = NULL





③ Left rotation

temp = left

y = temp->right

y->left = temp

temp = NULL

DS

Date 16 Dec 21

Graph search

will check if some path exists b/w 2 nodes. (starting to ending)

DFS, BFS → for traversal.

↓ stack ↓ queue

DFS.

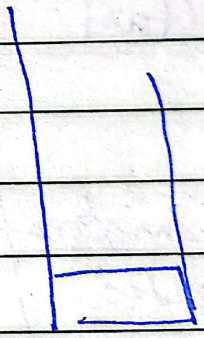
we check if it has adjacent nodes, push.

else pop.

Choose a path. travel downwards. till end,

till deadend.

then backtrack then again haan.

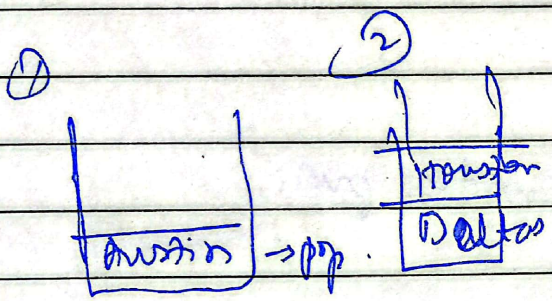


→ see slides for graph.

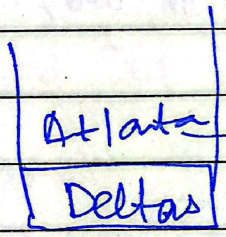
Date

austin → starting washington → ending.

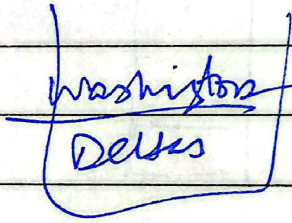
starting → push
then pop to check if it is ending node.
if node (will visit its adjacent nodes)



not found
so pop houston
(check if it is ending)
if not so, insert its adjacent nodes.



→ pop. (not last)
so insert houston, washington
already visited so wont insert this again.
Will only insert washington.



→ pop.
found.

→ Checking if it is ending when popping

→ mark visited when inserting

Date

see notes for graph.

BFS found = false

q: enqueue (Start)

do

q: dequeue.

mark visited when pop

| Austin |

pop.

| Dallas | Houston |

pop/dequeue.

| Houston | Chicago | Denver |

| Chicago | Denver | Atlanta |

| Denver | Atlanta | Denver |

will now mark denver = visited when pop.

pop (Chicago, Atlanta)

marked not marked.

| Atlanta | Denver | Atlanta |

pop (end).

| Denver | Atlanta | Washington |

Date

Transfer | Washington

Shortest path

→ If weights are associated w edges in graph, we'll want shortest path possible. So we get minimum weight for a path.

Starting node ka or all nodes ka w each other. shortest distance.

Single source shortest path problem.

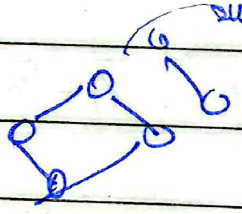
Sometimes 2 starting nodes.

Dijkstra → is only for single node.

Dijkstra's Algorithm

→ for both directed, undirected
→ none of the weights should be neg.
→ Should be a connected graph.

NOT connected =



unconnected, can't travel between.

Pseudocode

Date _____

Maintain 2 sets

→ 1 for all nodes/vertex.

→ 2 for shortest path jisko milta jayega, add kiste gaye hain in it.

If 6 nodes

Array[6]

distance[6]

0	∞	∞	∞	∞	∞

↓ Source (Generally on 0th index)

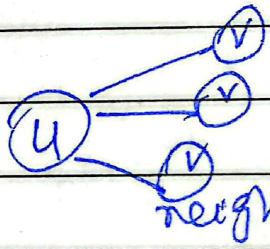
Initialize

source w = 0

rest w = ∞

S → for

Q → contains all nodes



on u, adjacent nodes of u = V.

~~10~~ p-1. $\langle u, v \rangle = u_1 v_1 + u_2 v_2 + u_3 v_3$

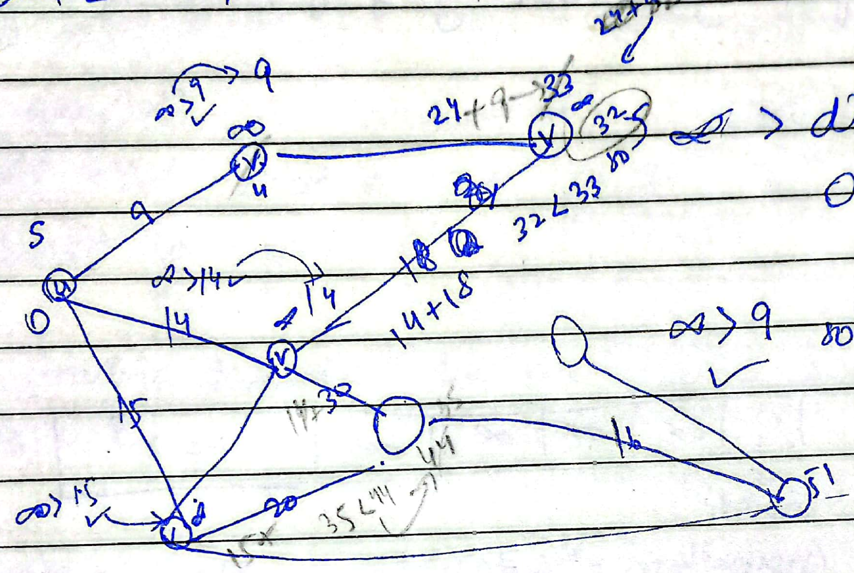
$u = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ $v = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$ $\langle u, v \rangle = a \cdot e + b \cdot f + c g + d h$

normal $\|u\| = \sqrt{u_1^2 + u_2^2}$

e.g. shortest path from s to t. $\|u\| = \sqrt{a^2 + b^2 + c^2 + d^2}$ for matrix

S = { 2, 3, 4, ... }

dist = 0,



shortest distance

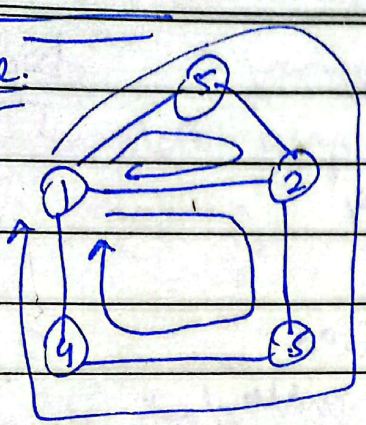
choose 1
Now, adj = u.
e.g. 9 wala will become u → has 1 adj, update its value
same here for this

Queue → order of placement fells

start ka jo order → wohi shortest distance

DS. Date 20 Dec '21

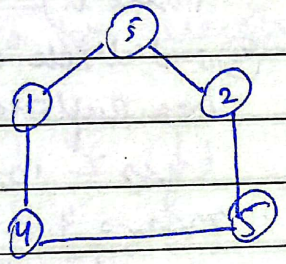
Cycle.



cycle: start and end on the same node

3 cycles.

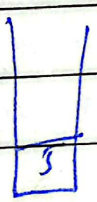
→ An algorithm to check if any cycle exists in a graph.



DFS, BFS → will change these there se for this traversal. of cycle.

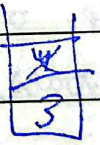
→ used to mark visited in DFS BFS.
after dequeue → after inserting

→ If any edge is taking us to any visited, there exists a cycle.



push(5)

pop.
5 → visited



5 4

Then 2 insert, pop, marked visited, Then

Then 3 4 " " " "

Then 4 " 4 " "

5 4

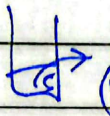
When done in stack, visited

After popping and visiting all its adjacent nods ⇒ explore.
↳ After popping 4, explore 2, 3. → unexplored

If explored node is found in any adj node, Cycle exists there.

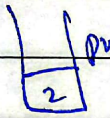
Date

marked visited

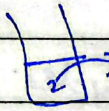


explore (pop)
2 adj.

Output
5

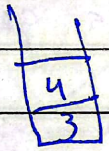


visited

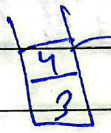


2 explore.
visit its children/adj

2.



visited.



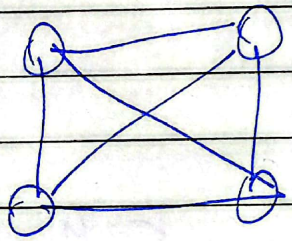
pop. explore ((1) Check if has unvisited node, insert it)

(2) ~~and check if~~ has explored node.
(yes 2 is explored) means 4 to 2 has ~~an~~ edge that forms a cycle.)

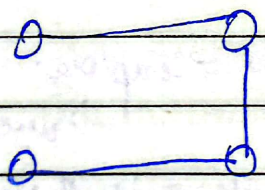
4.

Spanning tree.

→ subgraph of graph (G) that has the ability to only hold the nodes.
→ has the only edges that are required to hold all the nodes, doesn't have extra s.


→ is complete graph.

Spanning tree properties graph:-


edges are reduced.

edges = n − 1

n = nodes = 4

edges = 3

Date

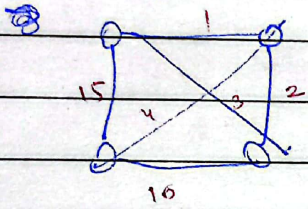
→ n^{n-2} spanning trees are possible. for any complete graph.

$4^{4-2} = 4^2 = 16$ spanning trees.

→ possible for any types graphs (not only complete graph)

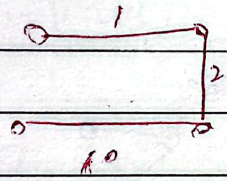
MST (Minimum spanning tree) → for connected graph only

→ is for weighted graph..



4 nodes
Total spanning tree = $n^{n-2} = 16$
↳ all will have diff sum of costs (weights)

→ ~~minimum sum~~ → MST

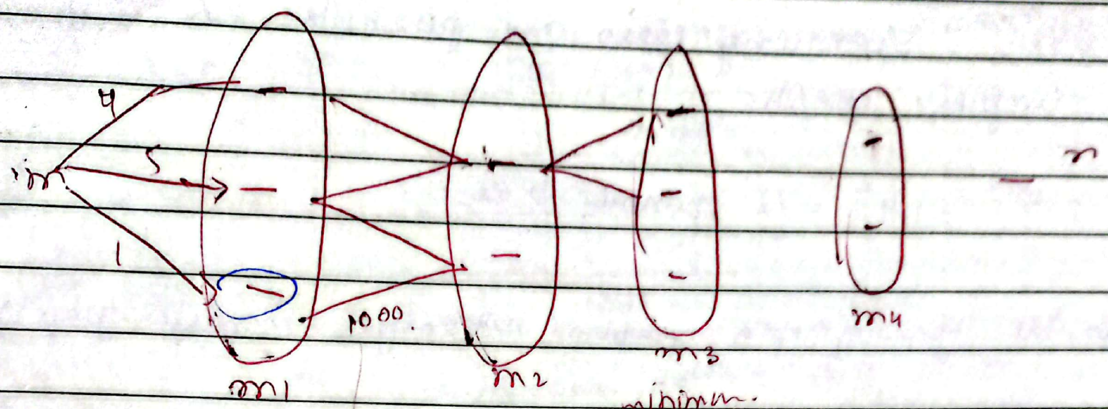


MST :- ~~Brute~~ force approach
calculate all possible spanning tree, then select the one with the minimum cost. but it's highly expensive computation.

→ not applicable in real life
so we use ① Prim's ② Kruskal → greedy algorithms.

Greedy algorithms :- → should be a connected graph.
↳ used to solve problems.
→ see options, use the most optimal (better)

Date

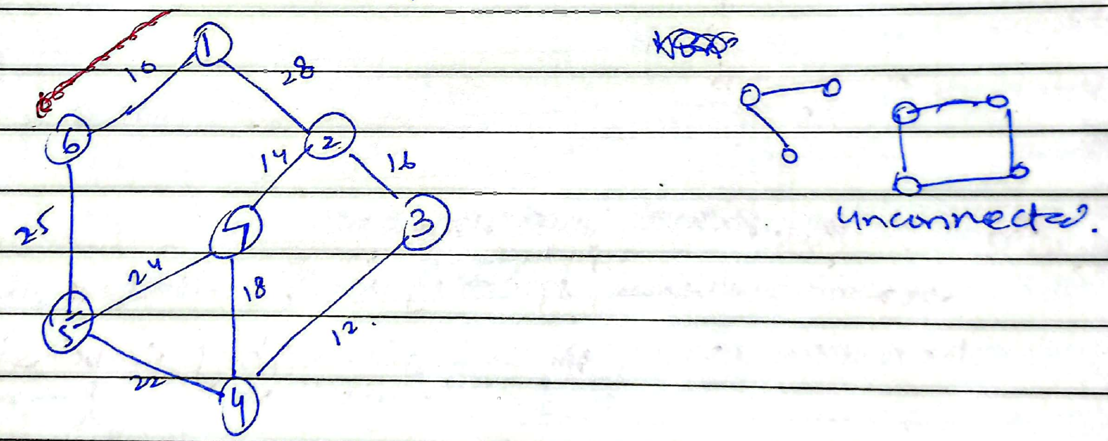


reach m to n in the cheapest cost (minimum)

wont check aage ka for m wala step.
Har steppe sbse kum.

→ doesn't always promise the best result bcz kan be costly aage steps main.

① **Prims** → should be connected graph.



① Choose the smallest weighted edge of the whole graph
① to ⑥

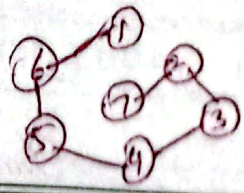
② Check options from 1 to 2 or 6 to 5 choose smallest

MST = { (1,6) 10, (6,5) 25, (5,4) 22, (4,3) 12, (3,2) 16, (2,7) 14 }

6 edges = n-1

③ Then (5,7) 24, (5,4) 22 ✓, (1,2) 28, ~~(4,7) 18~~, (4,3) 12 then (3,2) 16, (1,2) 28

cost = 10+25+22+12+16+14 = 99



Date

Check when taking any edge if there is any cycle present

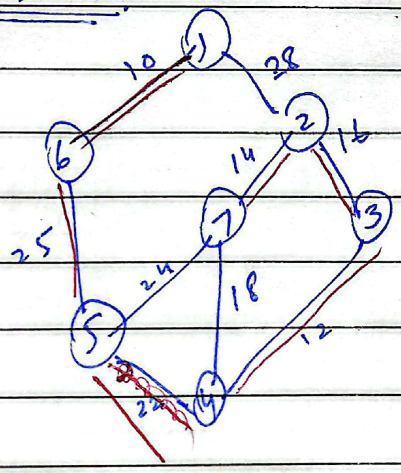
(4) (5,7) , (4,7) , (4,3) , (1,2)
24 18 12 ✓ 28

(5) (5,7) , (4,7) , ~~(4,3)~~ , (1,2) , (3,2)
24 18 ~~12~~ 28 16 ✓

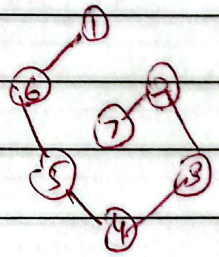
(6) (5,7) , (4,7) , (1,2) , (2,7)
24 18 28 14 ✓

Note:-
→ If any cycle is being generated at some point, we don't choose that edge.

(2) Kruskaus.



{(1,6) , (3,4) , (7,2) ,
10 12 14
(2,3) , ~~(4,7)~~ , (5,4) , ~~(5,7)~~
16 ↓ 18 cycle 22 cycle
(5,6) ~~(1,2)~~ }
25 28 cycle



(1) Choose edge w smallest ~~edge~~ weight := 10

(2) next smallest = 12 check if it's not forming a cycle, if not, add.

(3) ~~and~~ so on.

Spanning tree = {(1,6) , (3,4) , (7,2) , (2,3) , (5,4) , (5,6)}
10 12 14 16 22 25

cost = 10+12+14+16+22+25
= 99. same as prims.

Date

→ cost always same by both algorithm.
→ edges not always same.
→ even if same, order will be different.

Kruskal: choosing small by N nodes.
(N-1) times.

cost = n^2 → similar to heap. remove = O(1)
↓
heapify = O(log n)

O(n-1) → choosing smallest edges almost n.
cost = n^2

~~O(log n)~~ O(log n) → of Kruskal.

① Ford algorithm. ② Topology sort → read / home task.

bismah k B tree notes

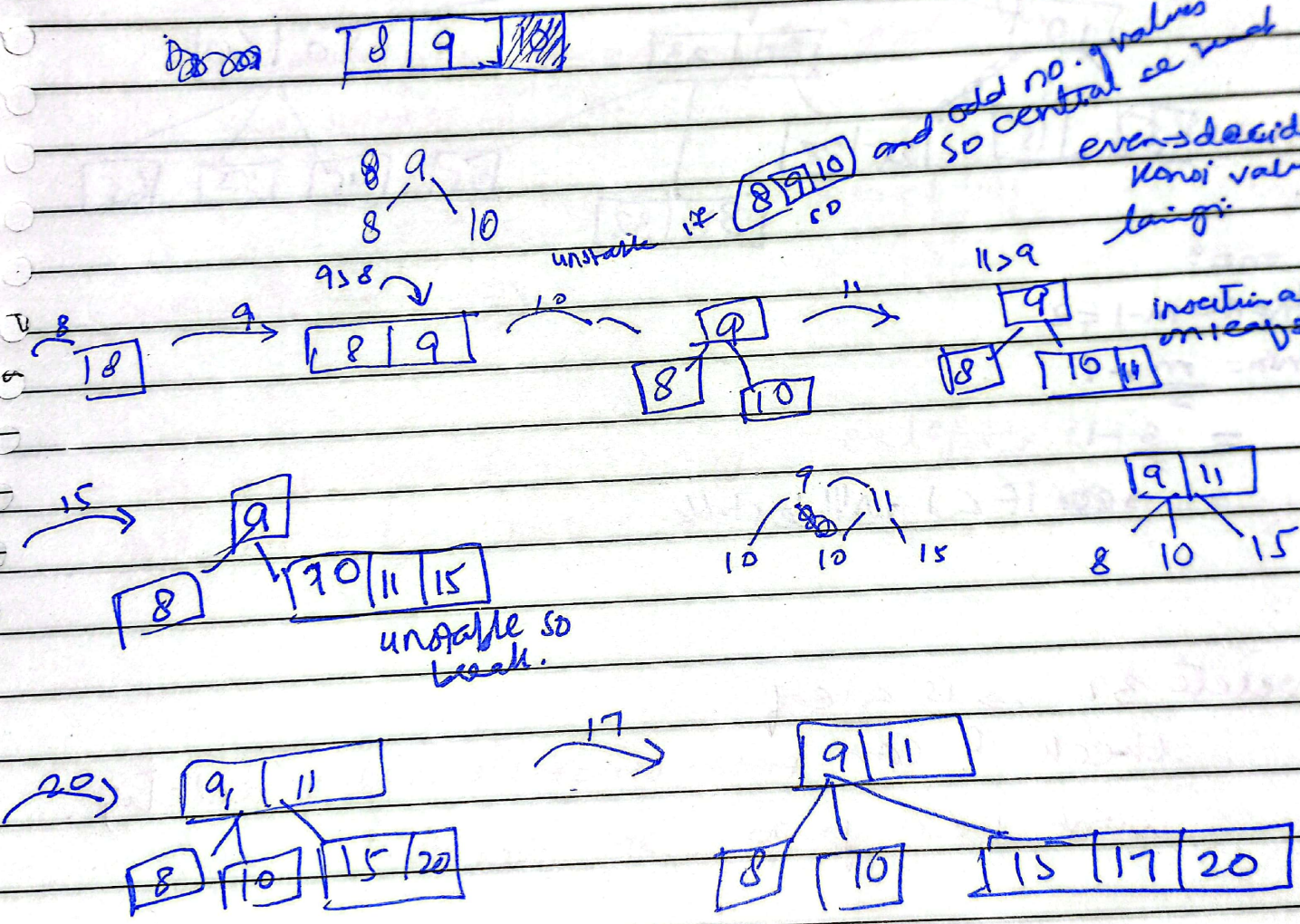
Date

B-Tree nocode, insertion, deletion.

8, 9, 10, 11, 15, 20, 17

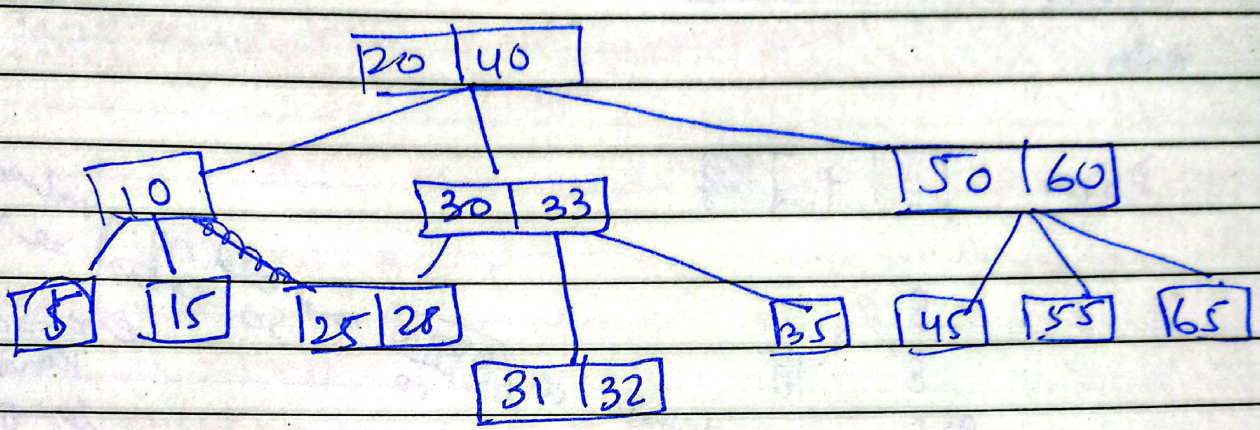
m = 3 so max child = 2 min = 0.

so max nodes = m - 1 = 2



Date

Deletion



$m = 3$

$max = m - 1 = 2$

$min = \frac{m}{2} - 1$

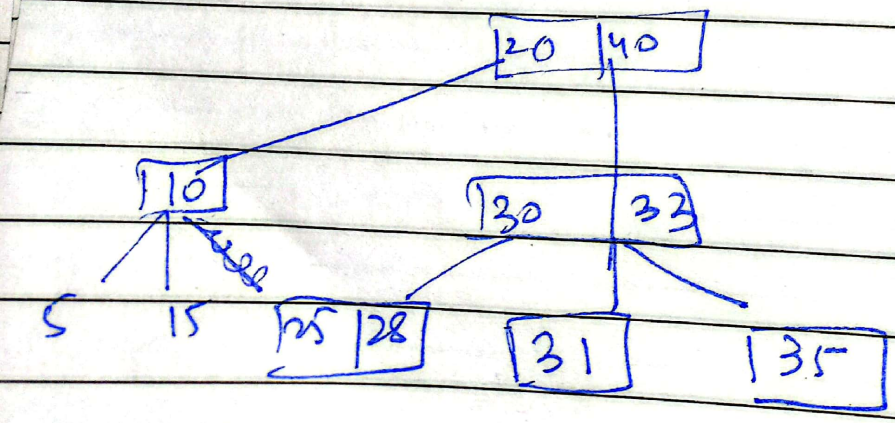
$= \frac{3}{2} - 1$

$= 1$ → if < 1 will be stable.

① delete 32 → is a leaf.

check if can then unstable after deleting.

won't be a prob so delete.



Date

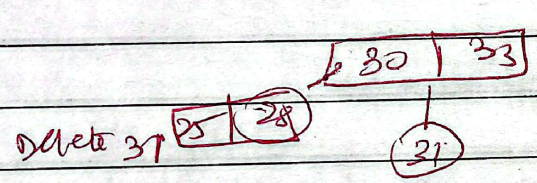
② Delete 31

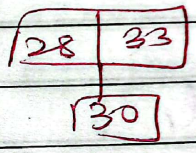
→ This will cause instability bcz 0 negative so will take value from siblings.

→ right se leta do wo unstable hogaya.

→ so will take from right

wo value phir parent ka bani hai.
or parent ki neeche.

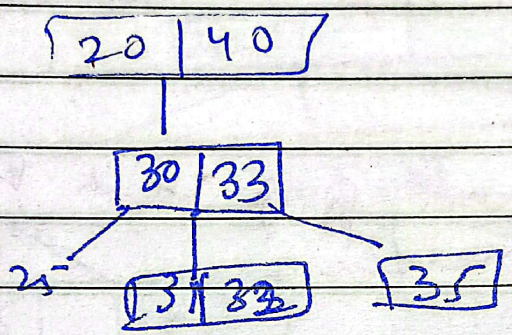




→ If dont have extra in right or left, then will take from parent after merging 2 left right.

Date

→ If deleting internal node.
→ will check if it has 2 children, replace it w its inorder successor (like in avl / bst)



codes → avl,