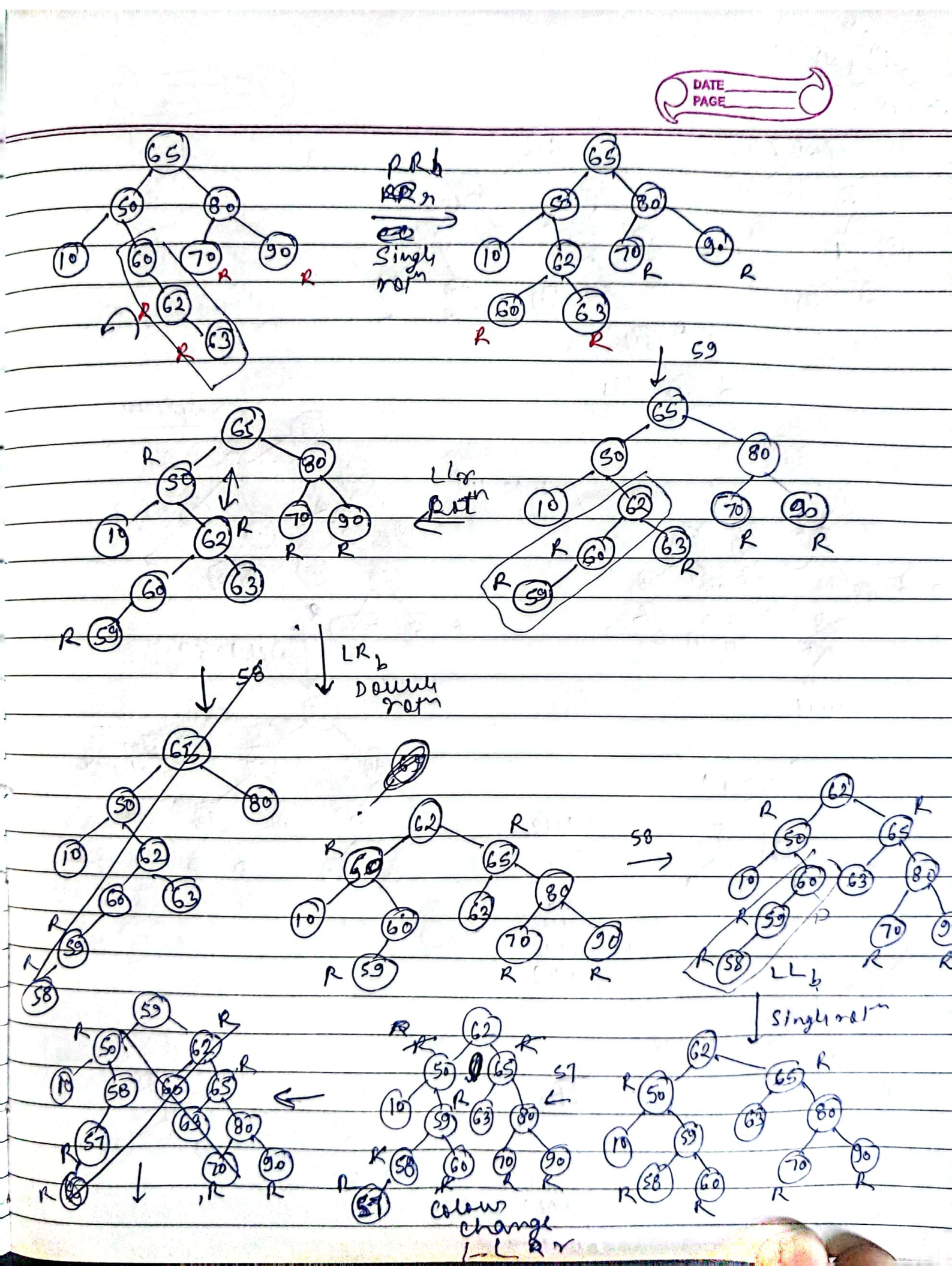
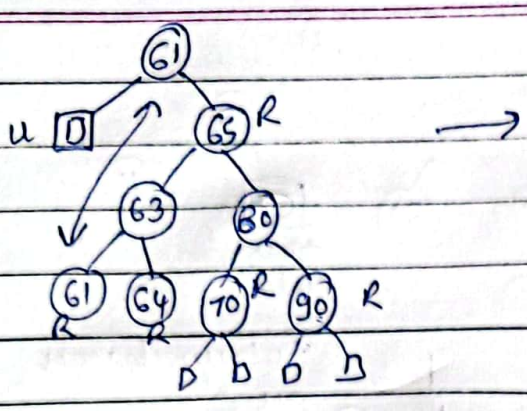
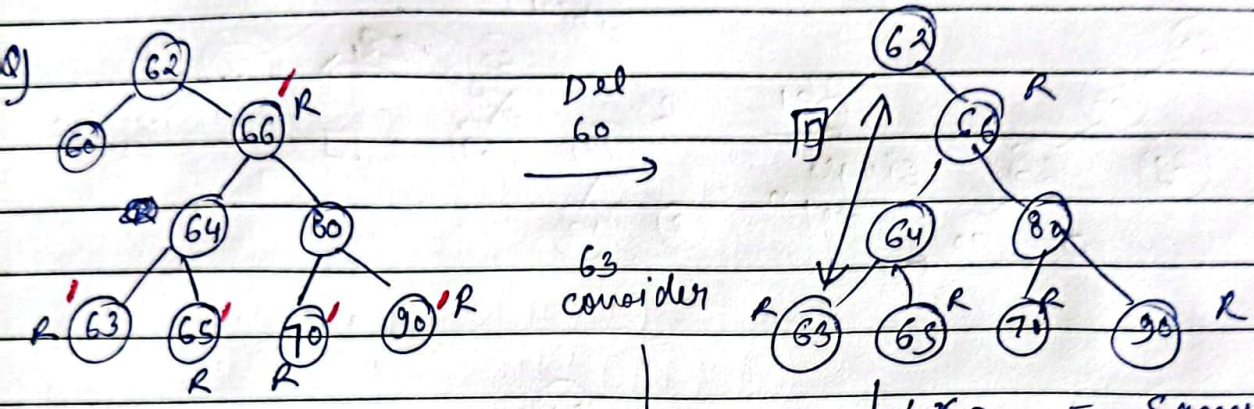
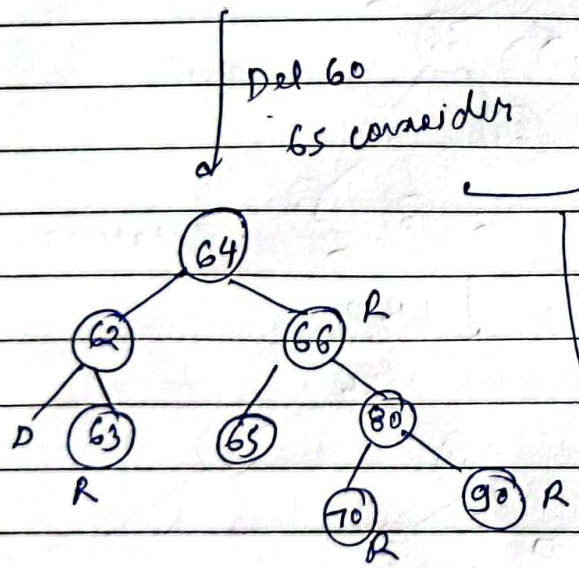
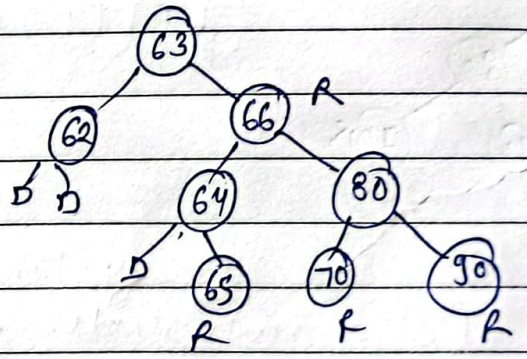

DATE
PAGE
RR
Single rotn
59
LL
Rotn
LR
Double rotn
58
57
Colour change
L-L R r





LR2 ≡ Same as LR1 (पहले दोनों Successor Double





जिसको consider किया उसे color change करो।

जो Ref में आता उसका color change

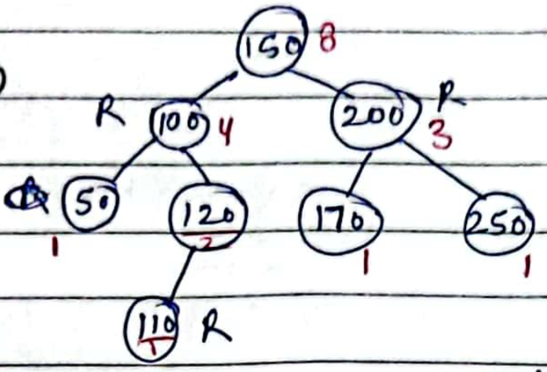
15.02.2024
Thursday.

DATE ____
PAGE ____

Augmenting Data Structure

↳ original Data structure maintain रहना चाहिए।
↳ Extra information store होनी चाहिए।

RB Trees
↳ O(log n)



Storing size of the subtrees.

(i) General order Statistical Operations
↳ Rank of any given value O(n)
(Inorder traversal से Order)
i^{th} smallest element → O(n)

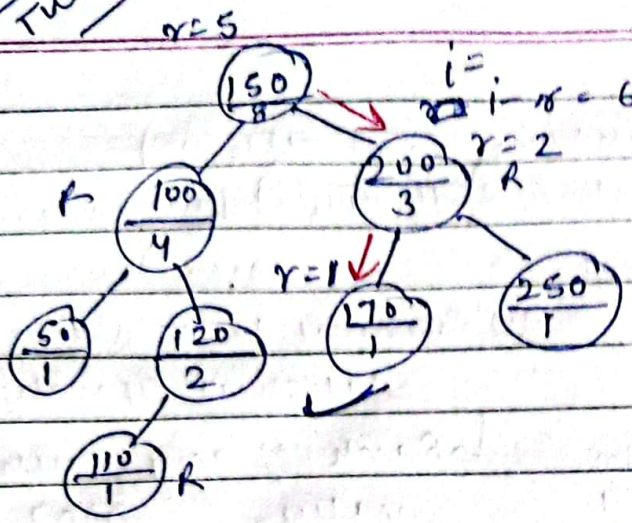
OS-Select (x, i) → Returns a pointer in the node containing the ith smallest element in the subtrees rooted at x.

OS-Rank (T, x) → Rank of x in the inorder traversal of Tree T.

Rank (lookout) = ~~sz[cur-node]~~ — sz[cur_node → left] + 1
left से जाओ तो no update in Rank (lookout)
Right से update Rank (lookout) = Rank(lookout) − sz[cur-node]

OS Select (x, i). (x : Root)
  r ← size [left [x]] + 1
  if i = r
    then return x
  else if i < r
    then return OS-Select (left [x], i)
  else return OS-Select (right [x], i − r)

20.02.2024
Tuesday



i = i - r = 6 - 5 = 1

OS-Select(150, 6)
6th rank rooted at 150
= 170

OS-Rank(T, 120)
r ← 2
y ← 120
y ≠ 150

OS_Rank(T, x):
r ← size(left[x]) + 1
y ← x
while y ≠ root[T] {
do if y = right[p[y]]
then
r ← r + size(left[p[y]]) + 1
y ← p[y] }
return r

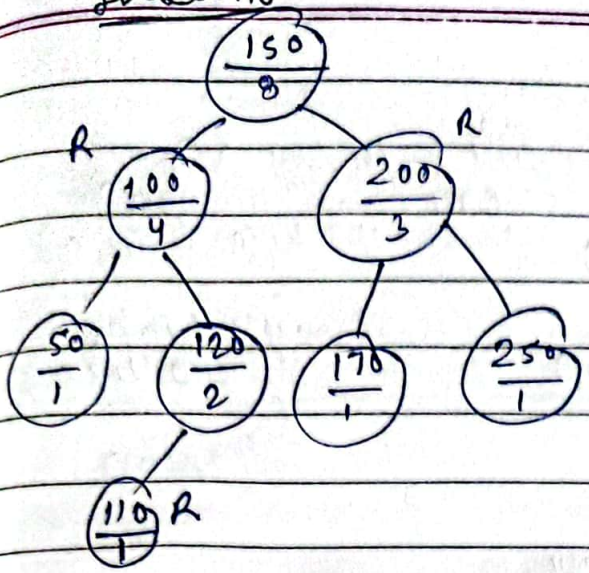
recursive
size store

Augmenting DS - Methodology

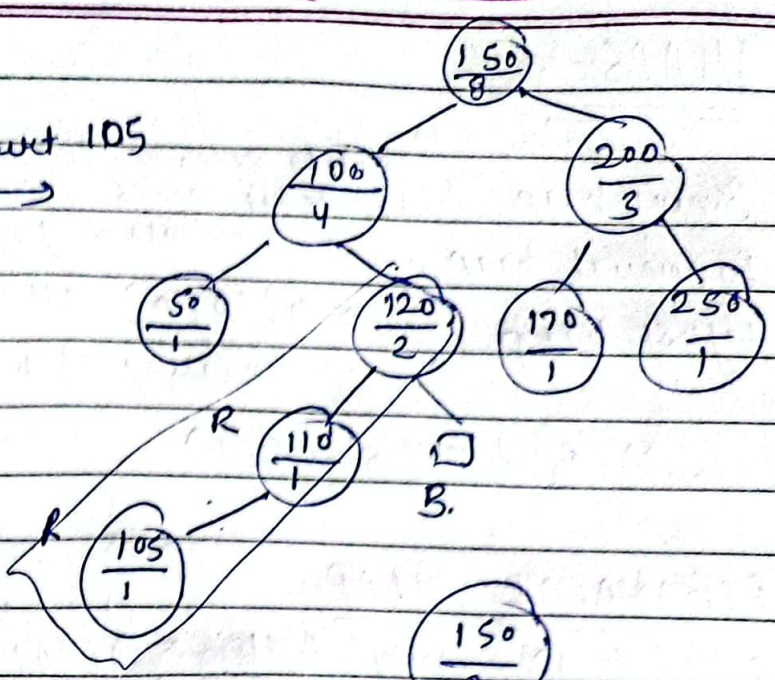
1. Identifying the underlying datastructure (DS) - RB Trees
2. Determine the additional info. required - size[x]
3. Verify that the additional info. can be maintained for the existing DS operations efficiently. → during inserting, deleting & searching
4. Develop the new operations.

extra info.

Insertion

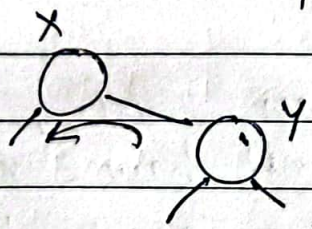


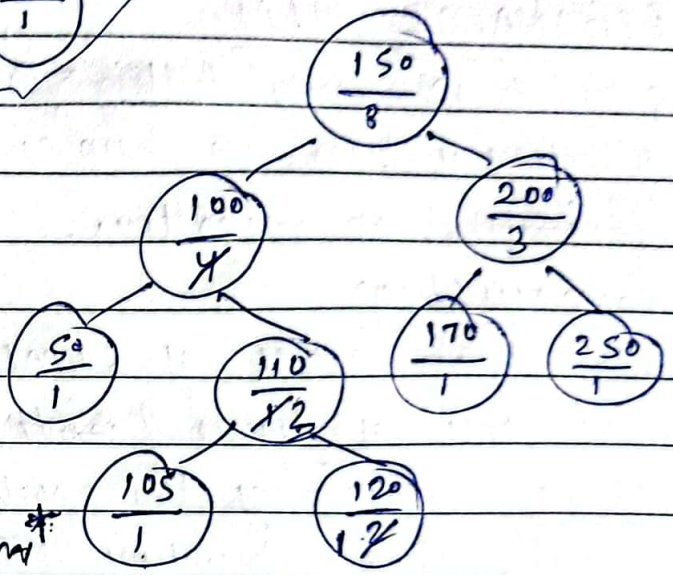
Insert 105 ⟶



Color change → No change in size

Rotation etc → constant time change





size[Y] ← size[x] // constant time

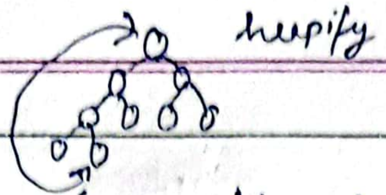
size[x] ← size[left[x]] + size[left[Y]] + 1

Deletion

(1) First pass it delete the node.
& update the size[x] for node ∀x ∈ path from root to pt. of delete.

(2) Color change ⟶ No change in size[x]
Rotation ⟶ constant time, change of size[x]

degenerated binomial heap.



HEAPS

Binary heap — Delete min = O(log n), Insert = O(log n)
Find ~~Extract~~ min = O(1), Extract min = O(log n)
creation of heap = O(n), union of 2 heaps = O(n)

Binomial heap → O(log n) all operation

Leftist heap — Creating of heap = Θ(n) → Θ(n)
union of 2 heaps = O(log n) → O(log n)
↳ Find min. = O(1)

FIBONACCI HEAP

other Θ(1) only deletion / Delete-key → O(log n) → Extract min

collection of trees in heap order.

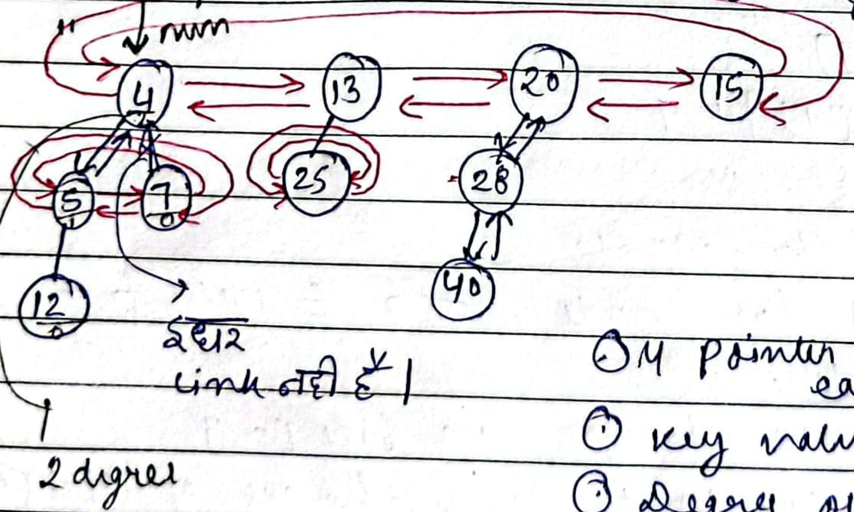
[Binomial heap restriction: Only 1 tree with given rank/degree

Fibonacci heap

1) More than 1 tree of same rank/degree can exist

3) A tree of degree / ~~node~~ rank i need not have 2^i nodes.

Idea: Postponing the work until the structure reaches the level where structure ऽत्राब हो ideal रहा हो।



↓ min. pointer
↳ pointing to overall minimum element.

ER child / parent pointer
TR parent
parent of 12 तो single child

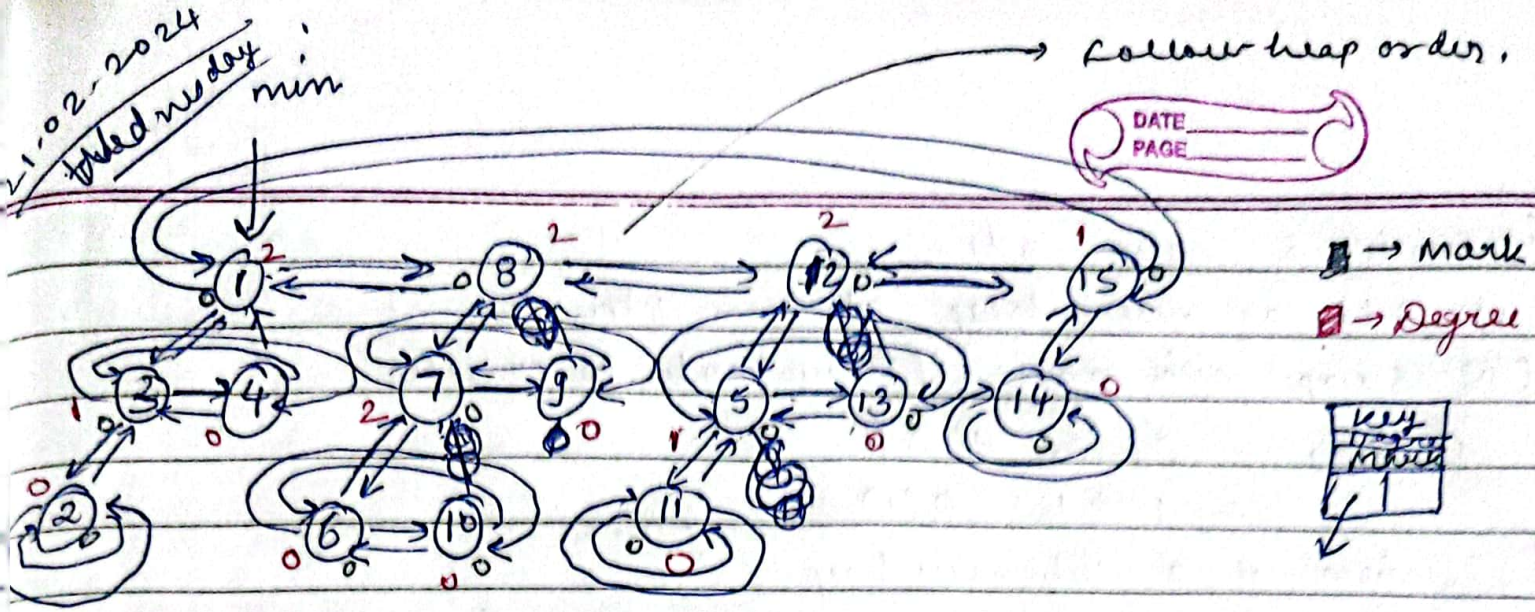
① 4 pointer at each
② key value
③ Degree of each node
④ mark (Boolean value on each node)

As soon as a node loses its child its mark = 1

⑤ total no. of nodes in heaps: n

21.02.2024

Fibonacci heap



Node Structure

Parent Pointer	
Child Pointer	
Left Sibling	Right Sibling
Data	
Degree	
Mark	

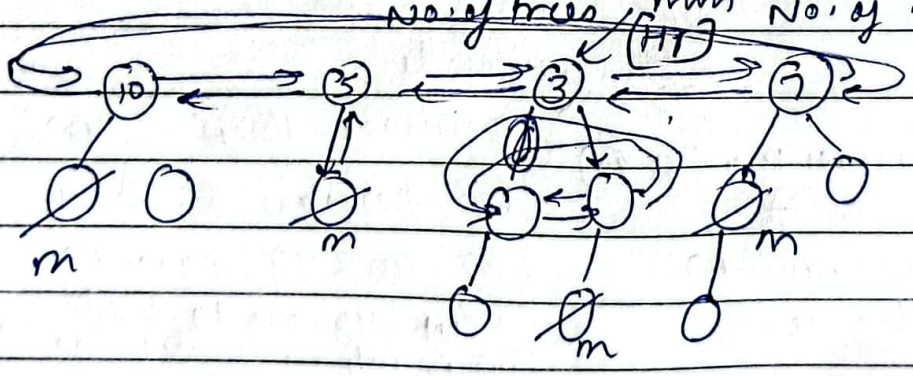
→ Parent से पहले child की pointer.

Doubly linked list:-

(i) Deletion O(1) (ii) Concatenation of 2 circular LL O(1)

Potential function - $\Phi = t(H) + 2m(H)$

No. of trees, min (H), No. of marked node (i)



$\Phi = 4 + 2 \times 4$

$= 12$

$D(n)$ = max. degree a node can have for a 'n' node fibonacci heap. $= O(\log(n))$ order.

Total pot^n of combined heaps $= \sum_{i=1} \Phi(\text{heap } i)$

(1) Insert a new node.

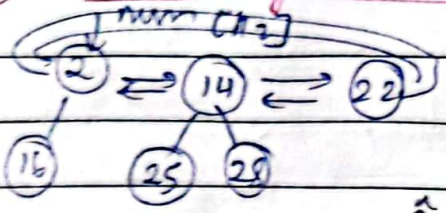
(i) Add as new heap at min. ptr.

(ii) Swap min. ptr. if new node is smaller.

$\hat{C}_{insert} = C_{insert} + \Delta\phi$

$= 1 + 1 = O(1)$

(2) Union of a Fibonacci heap.



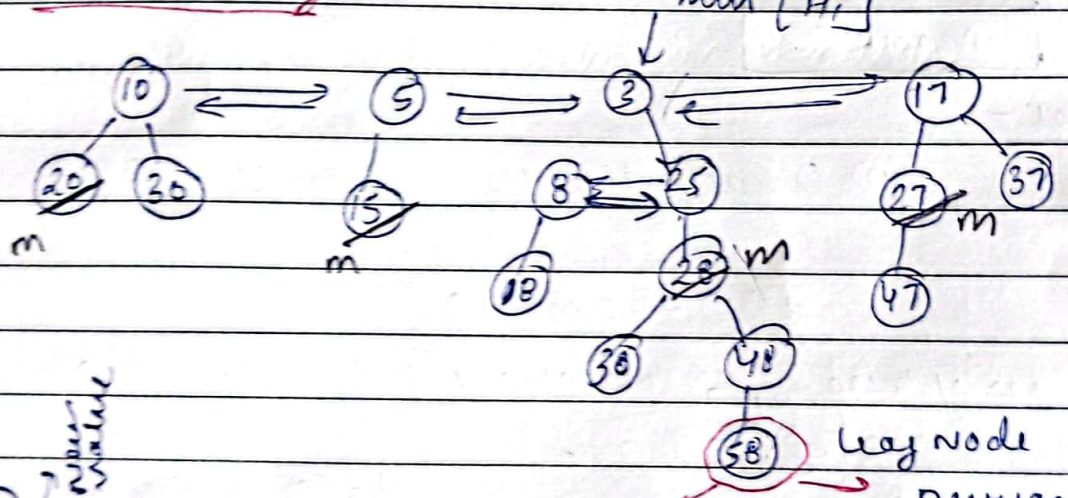
(i) Concatenate two list $O(1)$
(No limit on no. of trees & no. of degree)

(ii) Return on min. pointer.

$\hat{C}_{union} = C_{union} + \Delta\phi$

$= 1 + 0 = O(1)$

(3) Decrease key → We have pointer to that key



Key Node

Decrease-key (58, 50)

(i) Decrease ✗ ✓

(ii) 50 > 48 (parent)
No change

Decrease-key (58, 40)

(i) Decrease ✗ ✓

(ii) 40 < 48 (parent)
Cut 40 Node & add it in root list

(iii) Mark parent as marked? M = 1

If parent is marked = 1
↓
Cut it & propagate this operation (Cascading operation) until reach unmarked node or root node

Not marked
↓
marked = 1

Decrease key (H, x, k) → Pointer to key node

Check min. pointer

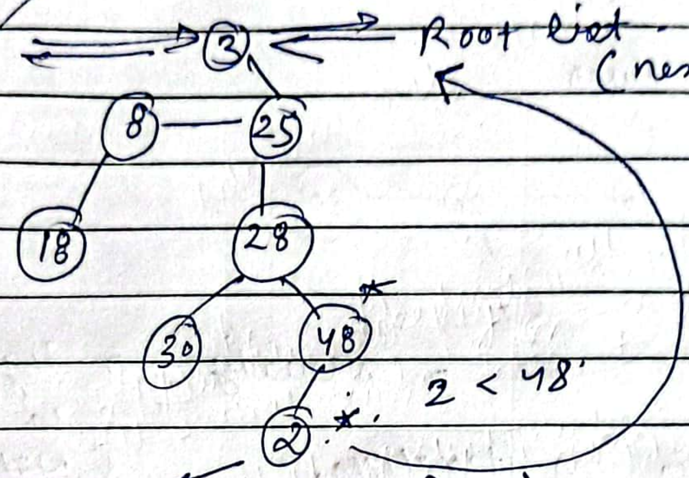
$O(1)$

22.02.2024
Thursday

Cascade cut length = x (say)

Initial trees: t, m

$$\hat{C}_{\text{decrease key}} = C_{\text{decrease key}} + \Delta\Phi$$
$$= x + [\{(t+x) + 2(m-x)\} - \{t + 2m\}]$$
$$= x + x - 2x = 0$$



$O(1)$ — marked become unmark

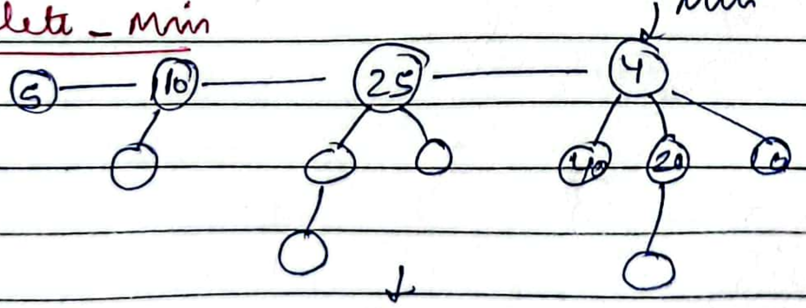
Repeat this process until we reach unmarked node

Ye marked था हो सकता and नहीं था।

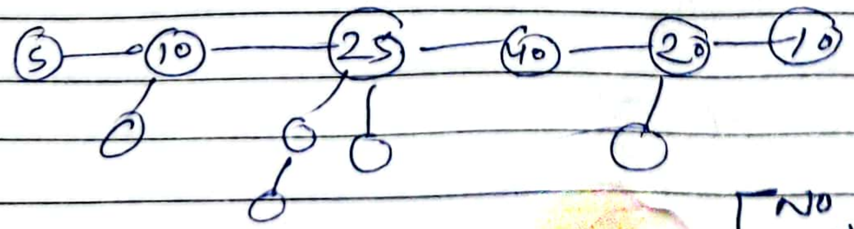
mark है तो	mark नहीं है तो
x	$x-1$
$\hat{C}_{dk} = 0$	$\hat{C}_{dk} = 2$

[dk = decrease key]

(4) Delete_Min



(i) Delete the min & add its child to the root list.



(ii) [Consolidating operation]
Traverse whole root list.
Merge two equal degree nodes.
[No more than one tree of same degree]

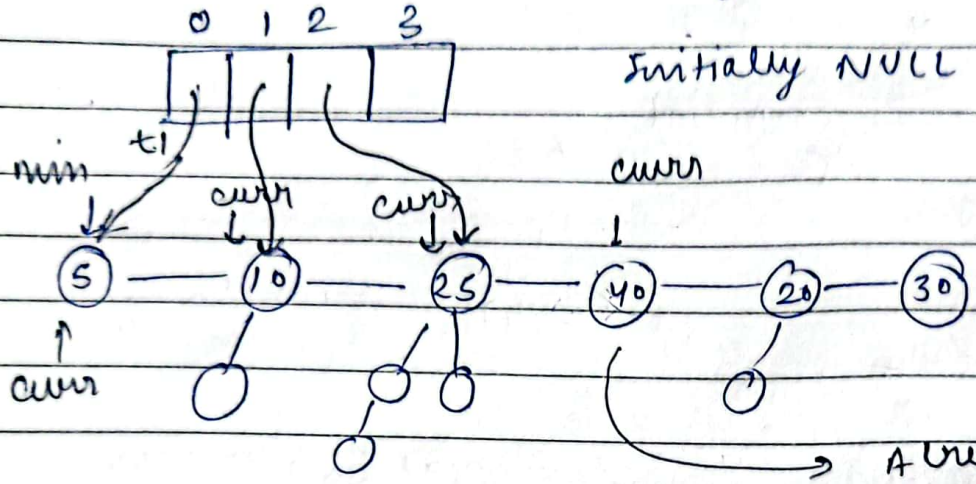
Colman.

How union?

⇒ use array to maintain the degree part of tree.

Size of array = $O(\log n) + 1$

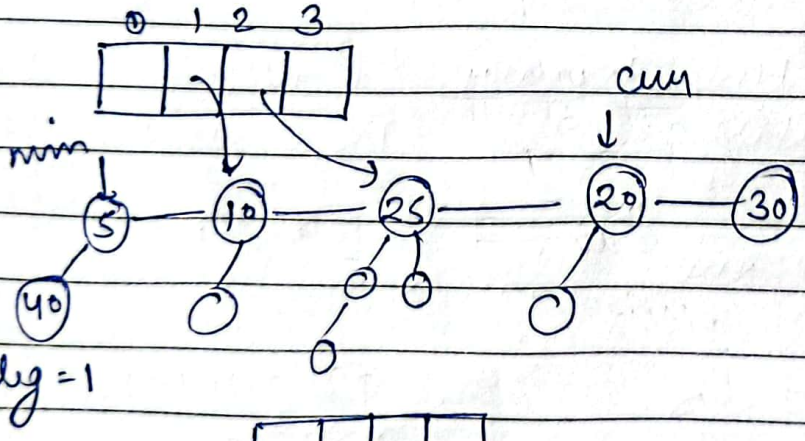
[Reason → $D(n) = (\log(n))$



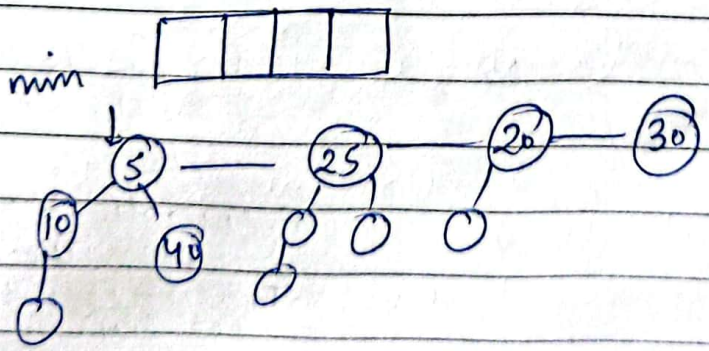
compare min. pointer in iteration update curr.

∴ Union of t1 & curr (tree)

[Union like Binomial heap]



deg = 1



2 Step manner

① ~~Extract~~ Delete min

② Consolidation process

t, m

initial

① → parent pointer change.

$\log(n)$

$t + \log(n)$ ②

final

$t + D(n) - 1$

max. degree of any node possible

m'

2 marked ✓ then unmarked

2 unmarked ✓ then marked

(i) Initial: $m + \log n$ — Final: $m - \log n$

(ii) Initial: m — Final: m

$\hat{C}_d = C_i + \Delta\phi$

$= \log n + (t - 1 + D(n)) + 2(m - \log n) - [t + 2m]$

$= \log n + t - 1 + D(n) + 2m - t - 2m - 2\log n$

$= -1 + D(n)$

$\hat{C}_d = C_i + \Delta\phi$

$= \log n + (t - 1 + \log n) + 2m - (t + 2m)$

$= 2\log n - 1$

$D(n)$

worst $= O(\log n)$

② Consolidation

$\Delta t = \log(n) - t$

$\Delta m = 0$

$\Delta\phi = \log n - t$

$\hat{C}_i = C_i + \Delta\phi = t + \log n - t$

$= \log n$

$\therefore$ ① + ② $= 2\log n + \log n = O(\log n)$

Consolidation process is called in Delete_min & Delete_key

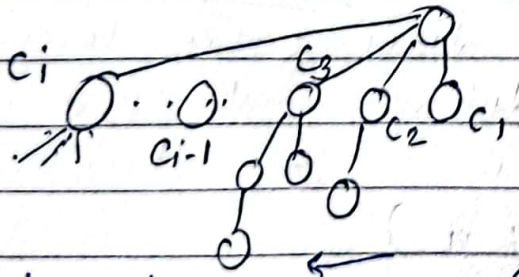
(3) Delete_key (H, x)
(i) Decrease (H, x, -∞)
(ii) Delete_min

Amortized complexity = O(log n)
of Delete_min & Delete_key
& for all others = O(1)

Each tree in a 'n' node FH has rank O(log n)

[Decrease key operation a tree is degenerated (i.e loses its child but its not too long)
As first cut of child mark the nodes
& second cut would lead to cut the node]

Let C_i be the i^{th} youngest child of any vertex x in FH where $i > 1$ (consolidation process)



Rank of C_i is atleast $(i-2)$

i-1 i-2 2 1 0 (degree)

just before joining. } any tree can lose only one of its direct child.
∴ least degree it can have is $(i-2)$

Any tree of rank 'k' in a 'n' node FH has atleast $F(k+2)$ descendants, (including vertex)

F(r+2) = (r+2)th Fibonacci number 'r' T(r)

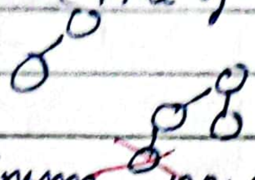
is the no. of nodes in a tree of rank 'r', ~~T(r)~~

. T(0) = 1 (Tree of degree 0 has 1 node) O

T(1) = 2

T(2) = 4 (node from 2 deg=1 node) ~~can have just 1 node~~

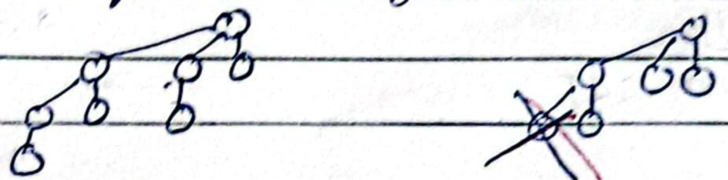
↳ maximum no. of node



What's minimum keeping its degree = 2

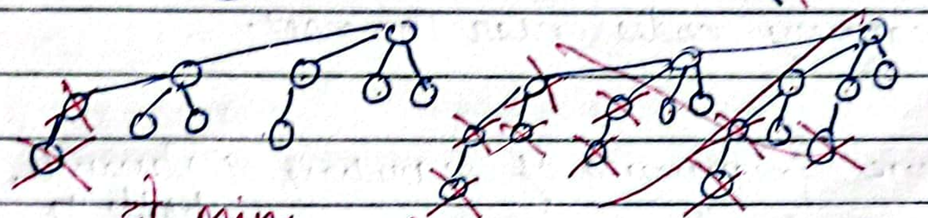
∴ T(2) = 3 (min. no. of nodes degree = 2 tree can have)

T(3) = 5



T(4) = 8

T(5) = 13



ये min. पहले वाली Tree.

Observe pattern

| T(n) = T(n−1) + T(n−2) |

F(r+2) ≤ n Fibonacci tree grows by exponential ~~logarithmic~~ order

r ↑ses ↓ exp(r) ↑ses ↓

Inference: o/p quantity increases by "exponential figures" then it is evident that r value increases by "logarithmic figures"

Accounting method Analysis.

Marked को करते ~~हटा, न node~~ time Node → 2 cost (1 credit है 1 उसकी credit का use root list में tree add करते time होता है।

Observation

29-08-2018 Wednesday

90-10 rule
90% times
10% code

DATE PAGE

Balanced BST

Data के case में हमें these 2 rules.

Two Cases :-

(I) Static ⇒ frequencies of elements along with probabilities of frequency of search operations.

⇓

Optimal BST

" More frequency nodes near the root. "

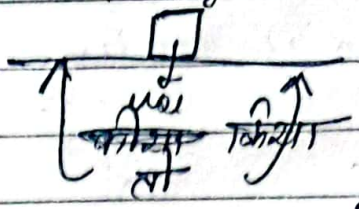
(II) Dynamic frequencies of searching of elements.
[Considering, " principle of locality holds "]

⇓

Splay tree (Self adjusting DS)

Normal binary search tree.
Single operation can take O(n) time
Amortized : O(log n) time for series of operation
(Even skew tree over series of operation becomes balanced trees)

① Principle of locality.



पास वाले की chances बढ़ जाती हैं / use होने की।

② 90-10 Rule
↳ major operation of code
↳ for robust coding part code + testing

SPLAY TREES :- It is a self-adjusted binary search tree (BST) in which every operation on element rearranges the tree so that the element is placed at the root position of the tree.
All the operations in a splay tree are involved with a common

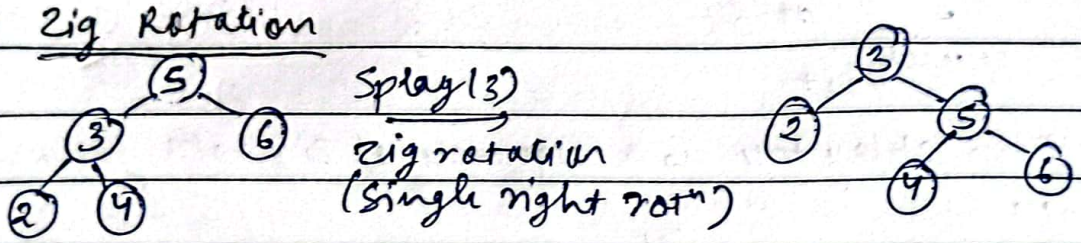
operation called "Splaying".

Splaying an element is the process of bringing it to the root ~~elem~~ position by performing suitable rotation operation

Rotations in Splay tree :-

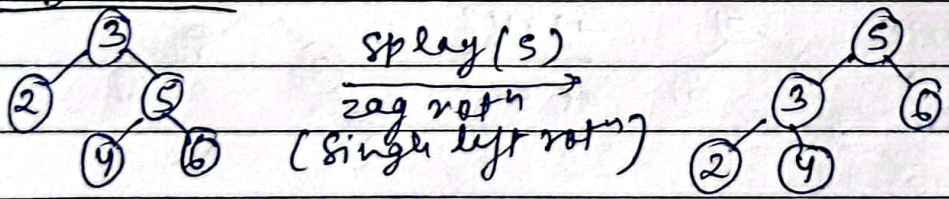
(i) Zig Rotation
(ii) Zag Rotation
(iii) Zig-Zig Rotation
(iv) Zag-Zag Rotation
(v) Zig-Zag Rotation
(vi) Zag-Zig Rotation

(i) Zig Rotation



(a) The zig rotn in splay tree is similar to the single right rotn of AVL tree.
(b) In zig rotn every node moves one position to the right from its current position.

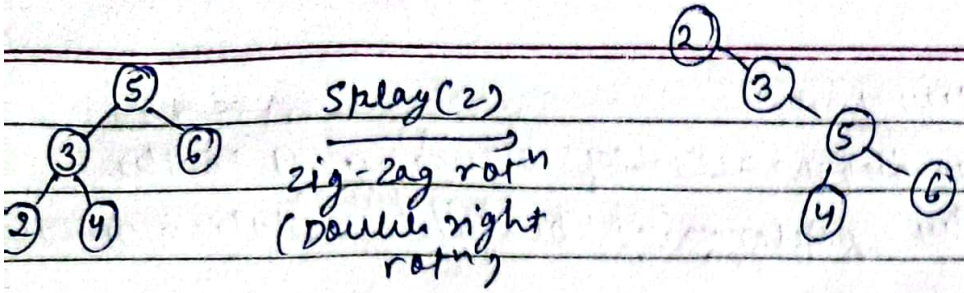
(ii) Zag Rotation



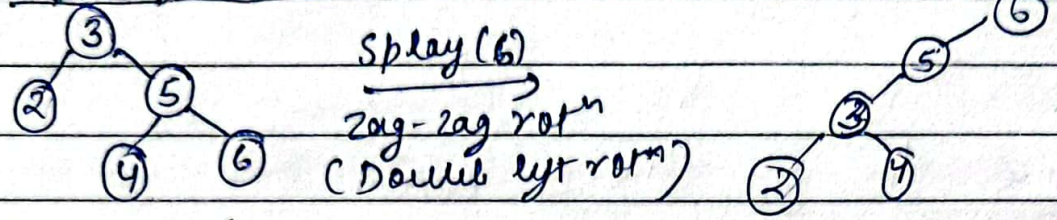
(a) In zag rotn in splay tree is similar to the single left rotn of in the AVL tree.
(b) In zag rotn every node moves one position to the left of from its current position.

(iii) Zig-Zig Rotation

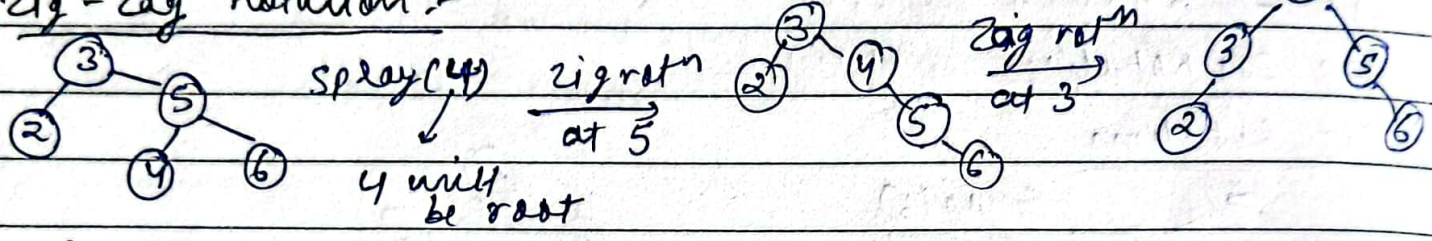
(a) The Zig-Zig rotation in splay tree is a double zig rotn.
(b) Here every node moves two position to the right from its current position.



[iv] Zag-Zag Rotation :-



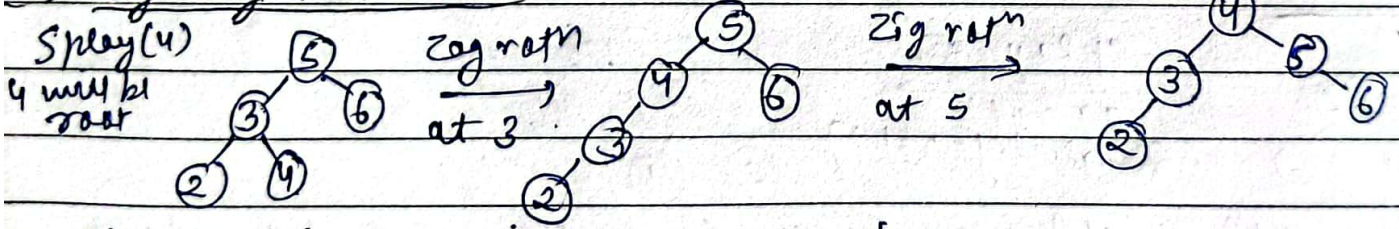
[v] Zig-Zag Rotation :-



a) The zig-zag rotn in splay tree is a sequence of zig rotn followed by the zag rotn

b) In zig-zag rotn every node will move one position to the right followed by one position to the left from its current position

vi) Zag-Zig Rotation :-



a) The zag-zig rotn in splay tree is a sequence of zag rotn followed by zig rotn.

b) In zag-zig rotn every node moves one position to the left followed by one position to the right from its current rotn.

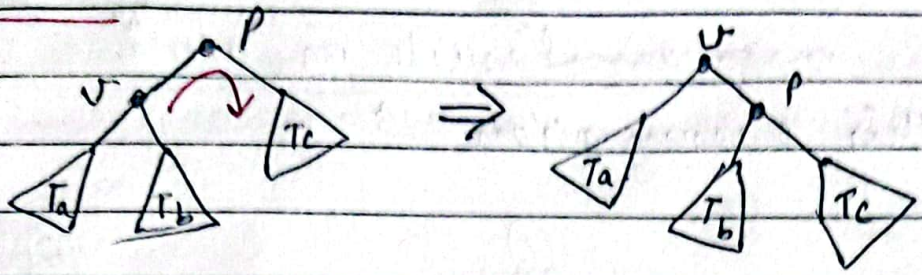
* Splay वाला root बनता है after rotn. Splay(x) perform the suitable rotn such that x root बन जाये।

29.02.2024
Wednesday

Splay (v) → vertex v to be made root

Splay Tree

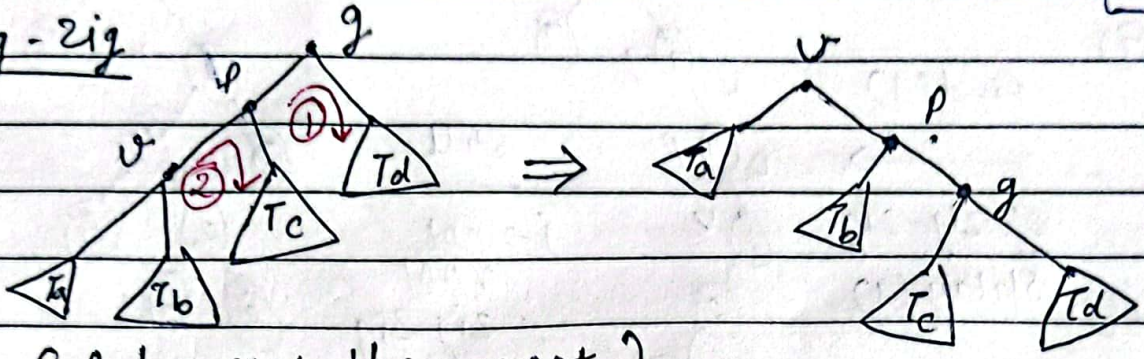
① zig



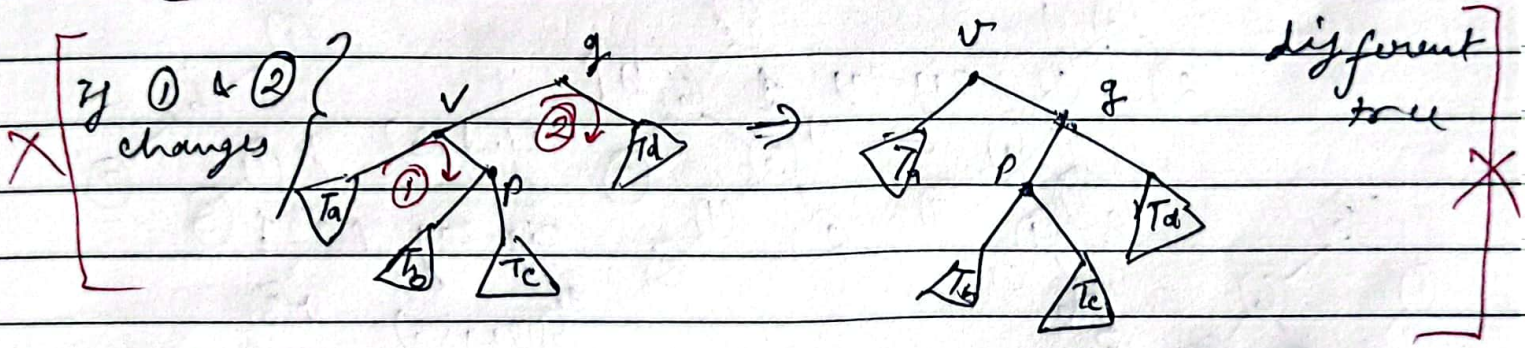
Single rotation

When you want to splay node just after root.

② zig - zig



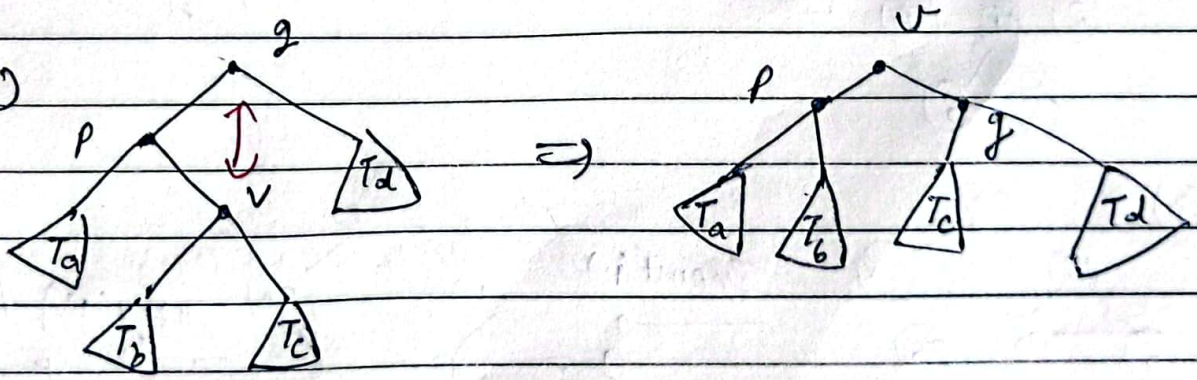
(g != root || g == root)
&& (direction(p) == direction(v))



if ① & ② changes

different tree

③ zig - zag
(Double rotn)
as of AVL



Types of Locality :-

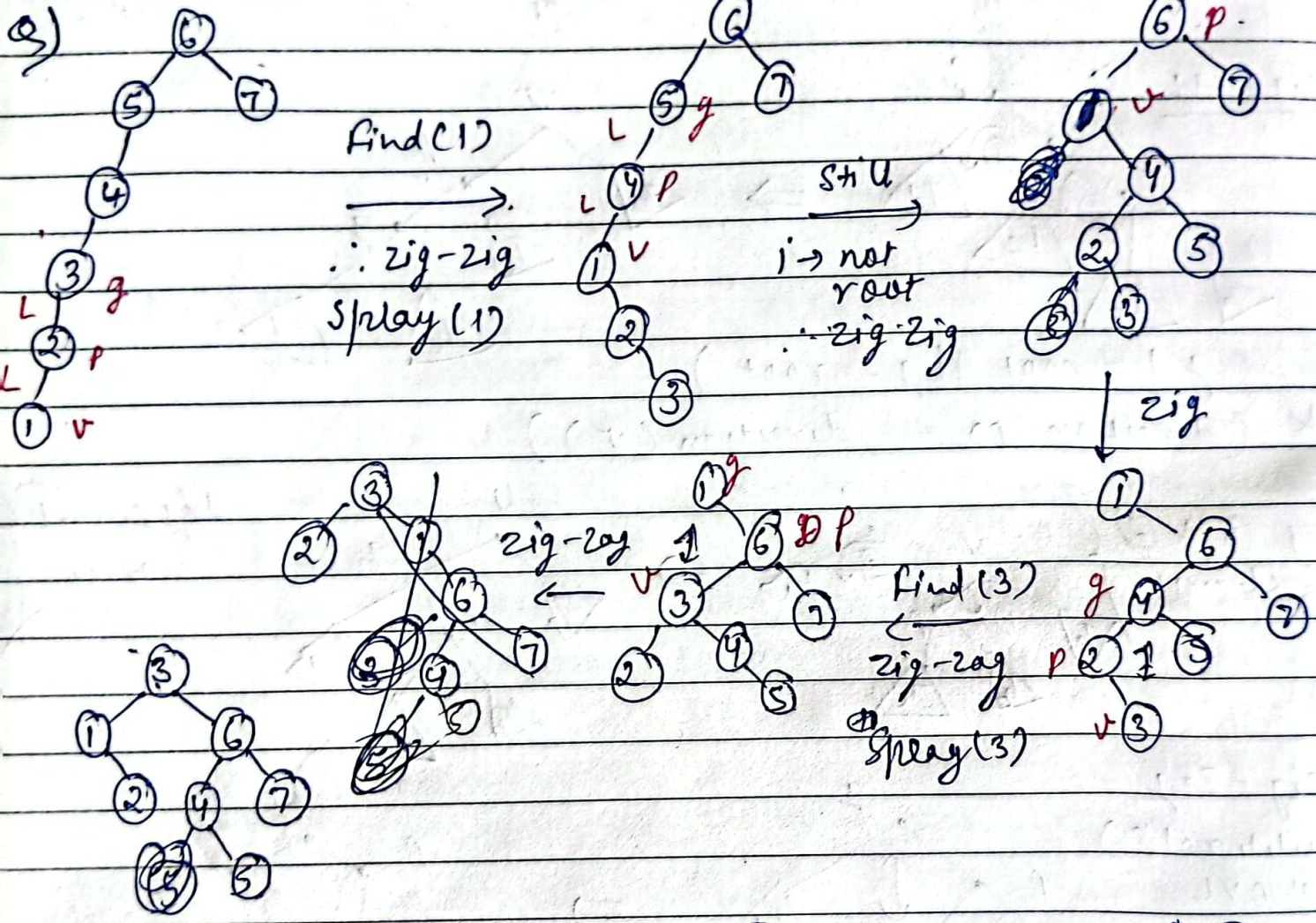
① Spatial locality → future access would would be closer to current access location
[sequential instruction, arrays]
आगे पीछे वाला Data

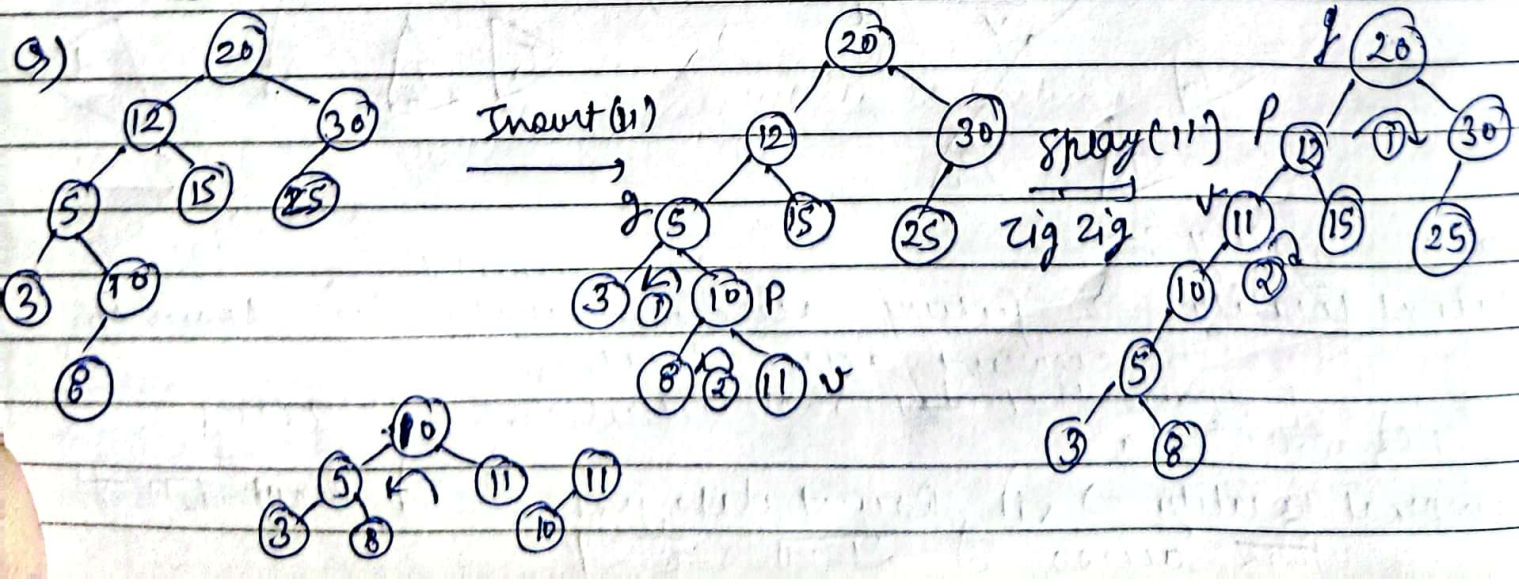
② Temporal locality → जो Recent data use हुआ है वो root के पास हो
पास access हो सकता है।

→ Assumption

Splay tree → (Spatial locality + Temporal locality) is ~~maintained~~ exists in computer world.

↳ exploit this assumption →

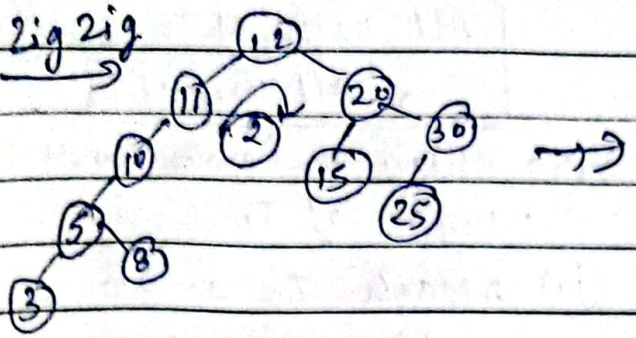


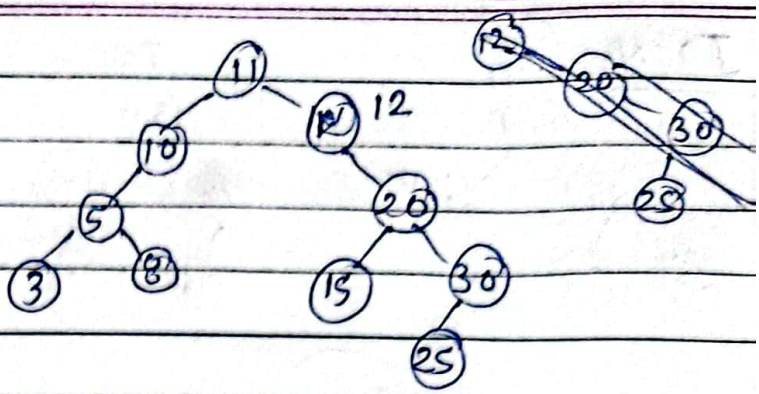


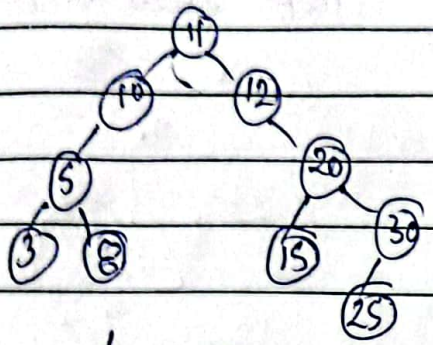
05.02.2024
Tuesday

DATE
PAGE

Zig Zig →







Find (14) Not found →

① search : Find (x)

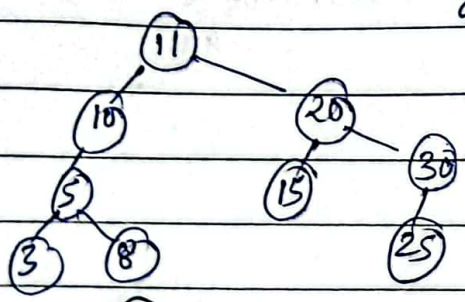
Success → Splay (x)

Not Success → splay (last node visited by the not success)

Delete 11

→ First do splay (11)

↳ Then BST root delete



② Insert: ~~is splay the node to be inserted~~

(i) Insert as per BST

(ii) Splay that node.

③ Delete:

(i) Splay the node to be deleted

(ii) Delete root as per BST rules

↳ Inorder successor

④ Delete max (i) splay max node

(ii) Find max (Right most node)

(iii) & delete as per BST rules

Right most node

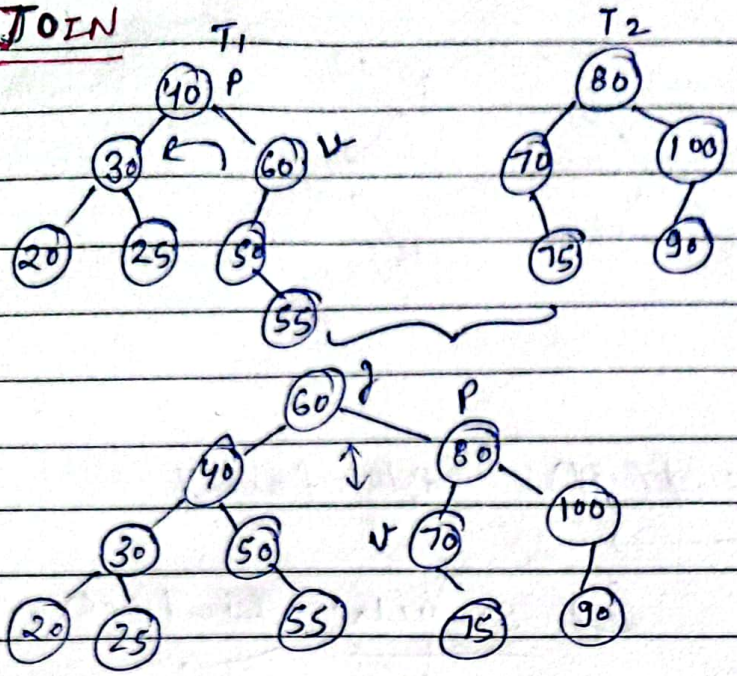
Right subtree will be NULL after opr.

⑤ Delete min → Left subtree NULL after opr.

(i) Splay min. node (leftmost node)

(ii) Delete root as per BST rules.

JOIN



All keys of T_1 < All key of T_2

(i) Splay the max node (key) of T_1

(ii) Attach T_2 as its right subtree

SPLIT

Split at key 70

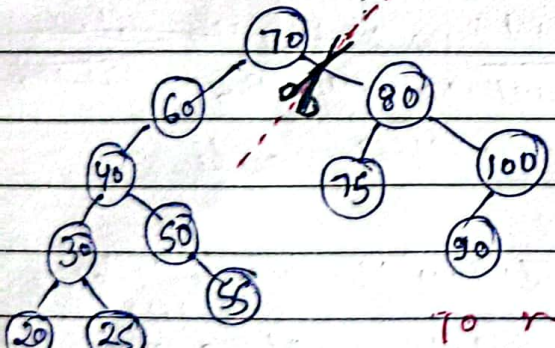
Two methods

~~(i) Splay 70 & detach the~~

method 1

(i) Splay 70

(ii) Detach the right link



method 2

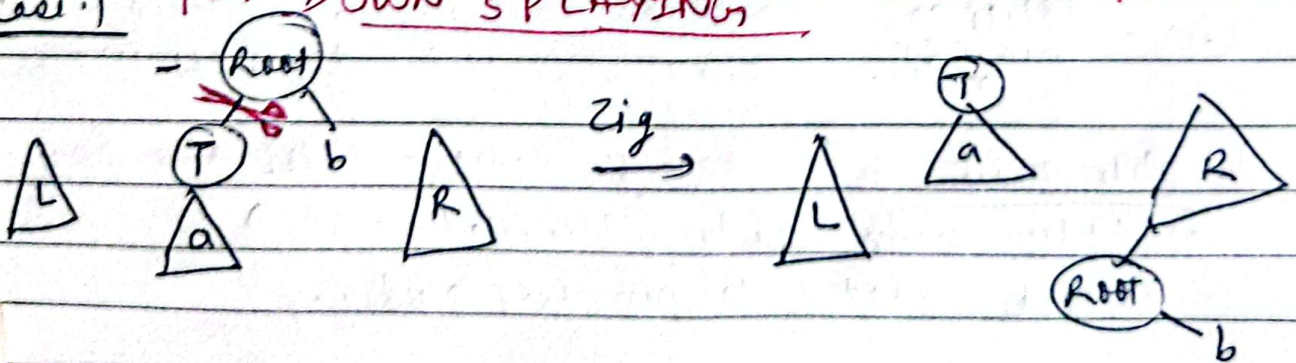
(i) Splay successor of 70

(ii) Detach at the left link

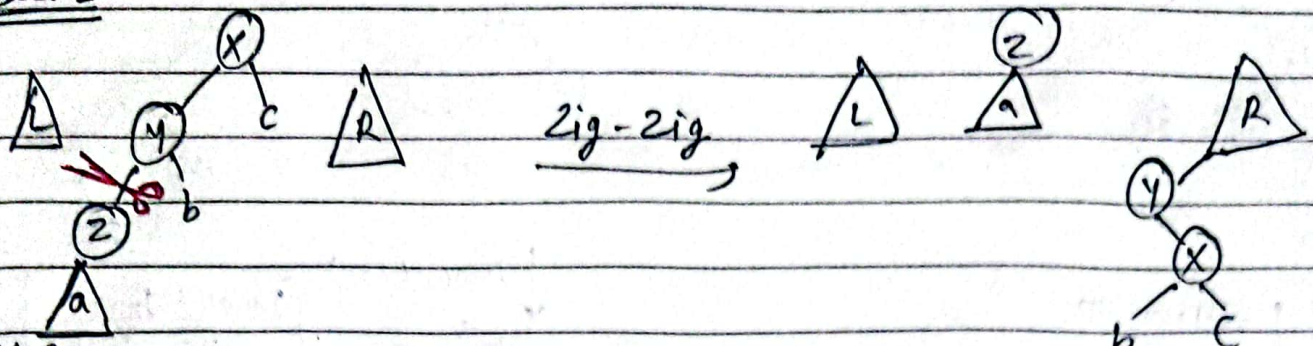
to reduce complexity we prune while splaying } Single traversal

Case 1

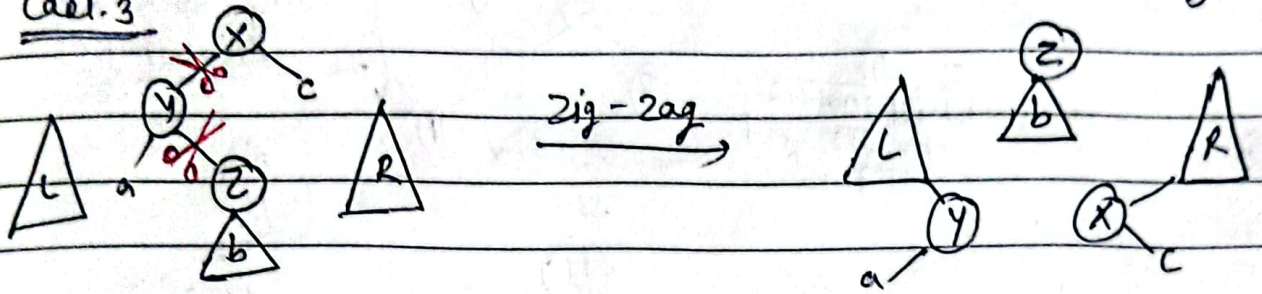
TOP DOWN SPLAYING



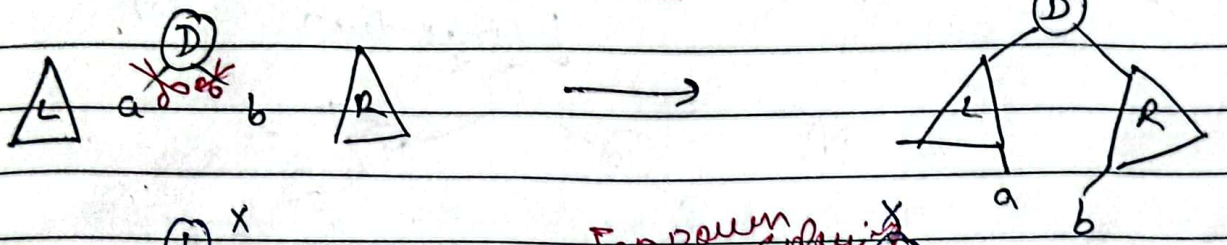
Case 2



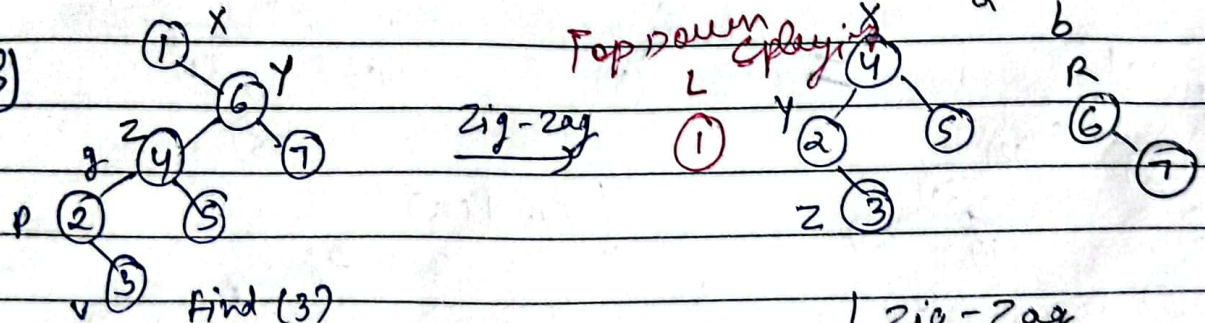
Case 3



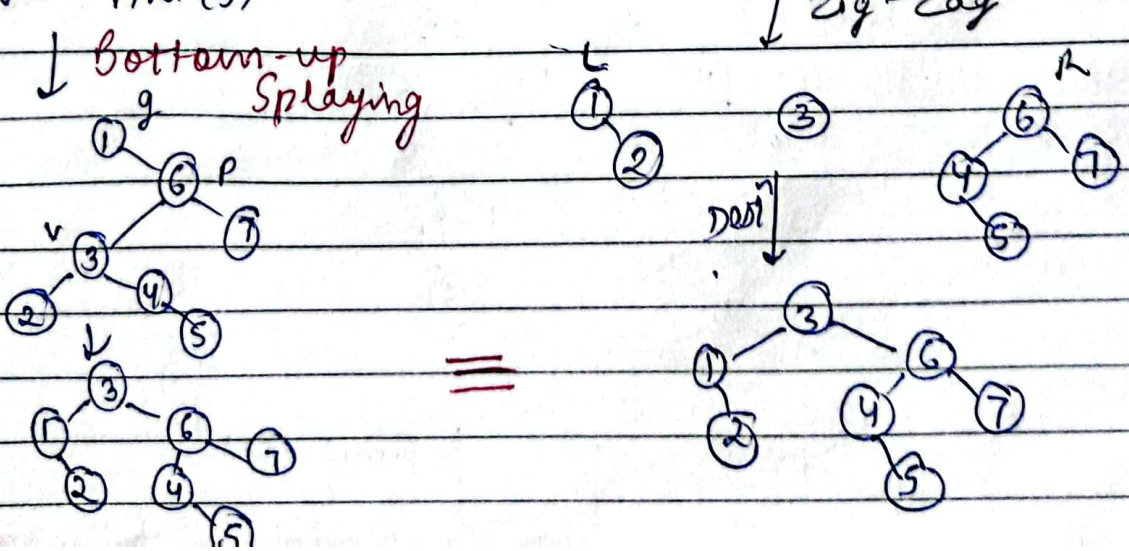
Final when we reach desp. node

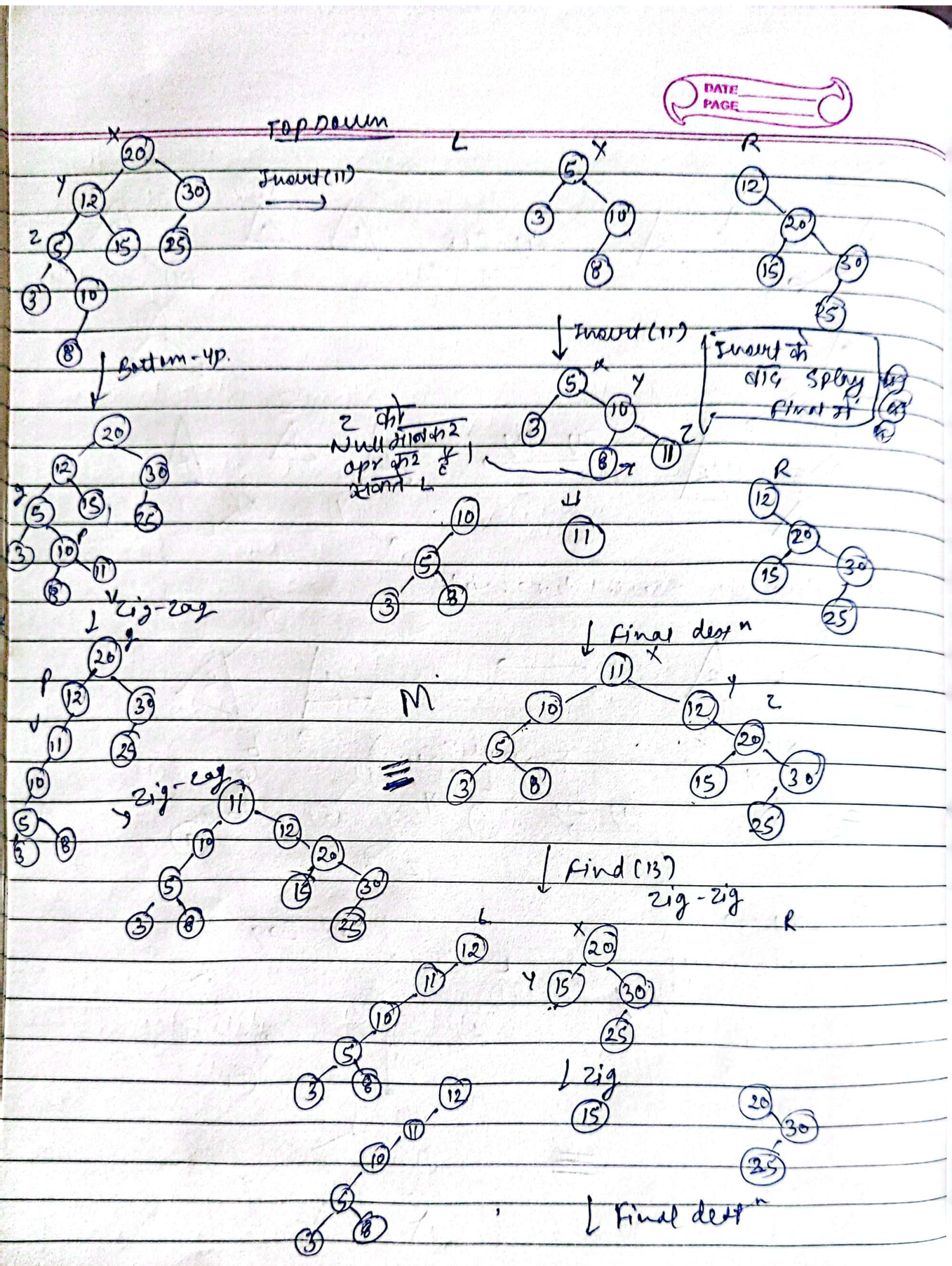


Q)

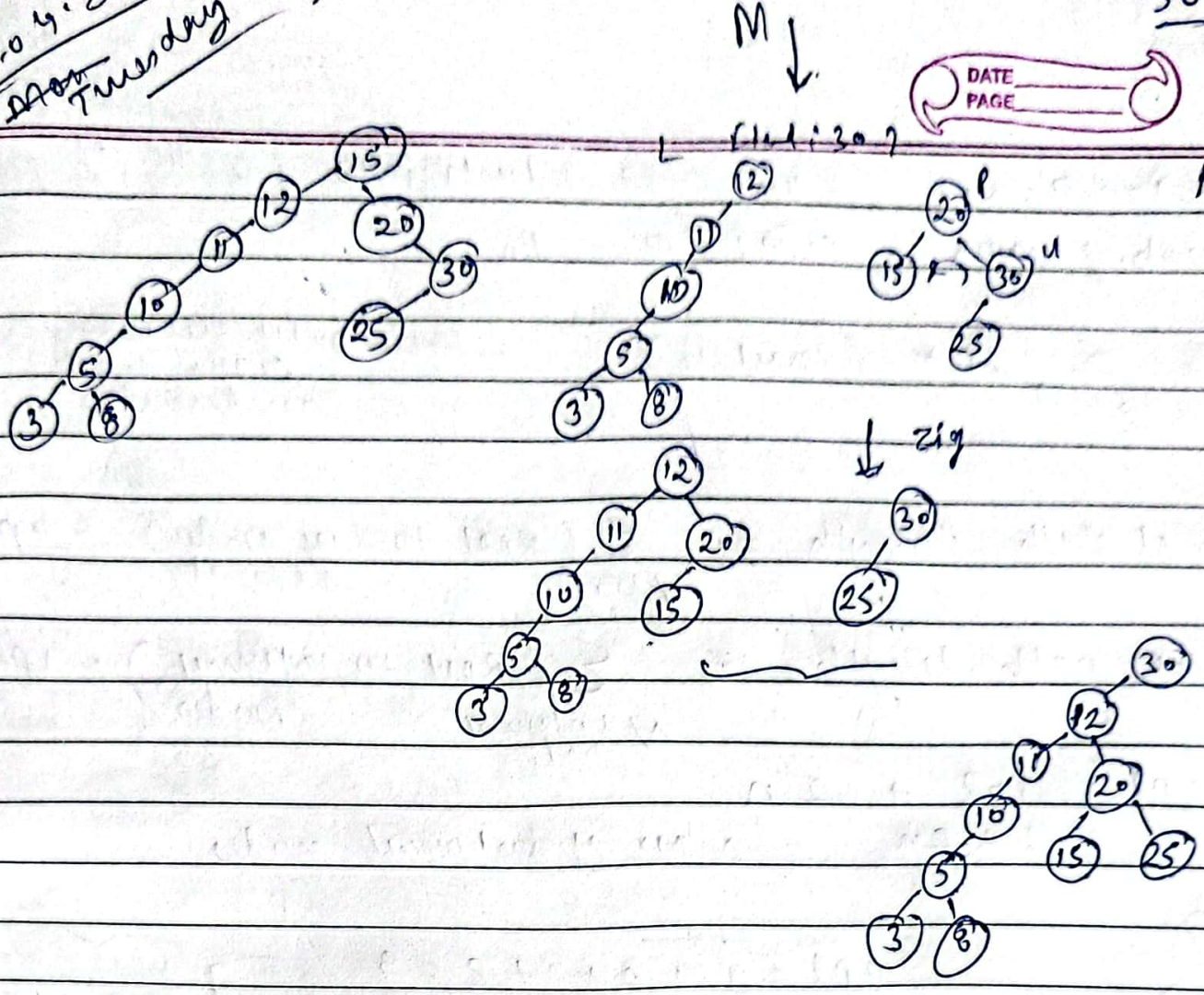


Find (3)



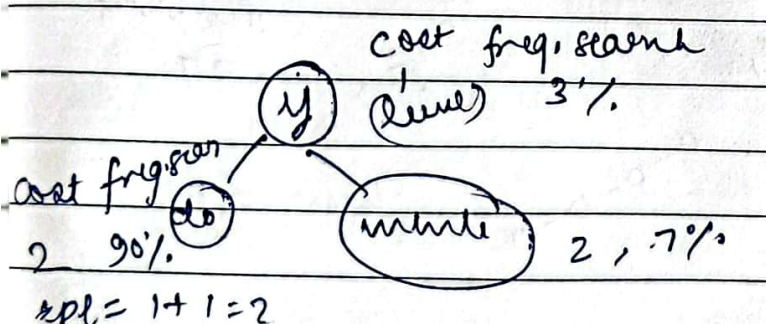

Top Down
Insert(11)
Bottom-up.
Zig-Zag
Zig-Zag
Insert के बाद Splay Final में
Null होना चाहिए
Final destn
Find(13)
Zig-Zig
Zig
Final destn

02.04.2024
Thursday

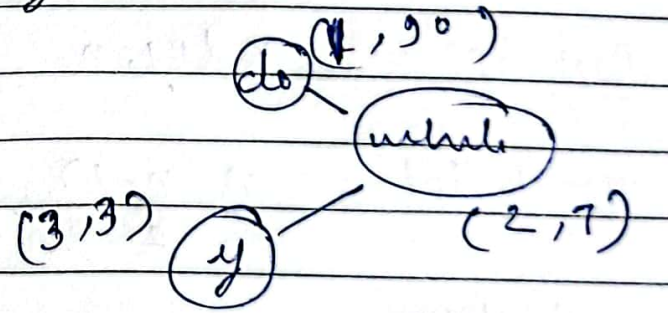


Optimal BST

Static set of identifiers : No insert, No Delete only search.



$wpl = 1 + 1 = 2$



Without freq. ~~set~~ prob.

Avg cost of search $= \frac{1+2+2}{3} = \frac{5}{3}$

With ~~to~~ freq. prob.

$ACS_2 = \frac{1 \times 90 + 2 \times 7 + 3 \times 3}{3}$

$= 30 + 3 + 4.67$

$= 37.67$

With freq. prob

$ACS_1 = \frac{1 \times 3 + 2 \times 90 + 2 \times 7}{3}$

$= 1 + 60 + 4.67 = 65.67$

for n internal nodes ⇒ n+1 external nodes in binary tree.

DATE
PAGE

increasing

$a_1 < a_2 < a_3 < \dots < a_n$ → Identifiers (order of prob.)

Its prob. of search $p_1 \; p_2 \; \dots \; p_n$

$$\text{Cost} = \sum_{1 \le i \le n} p_i * (\text{level } a_i)$$ → only successful searches are included.

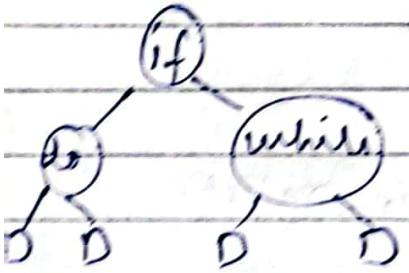
$$+ \; 3 + 3 + 2$$

External path length = $\sum_{\text{all leaf}}$ (Root to leaf nodes) = epl
external

Internal path length = $\sum_{\text{all internal nodes}}$ (Root to internal nodes) = ipl

$epl = ipl + 2n$

$E = I + 2n$ ↳ no. of internal nodes

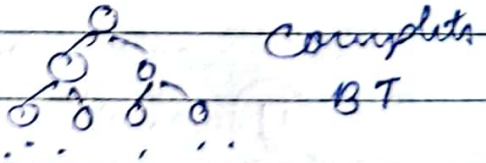


$epl = 2 + 2 + 2 + 2 = 8$

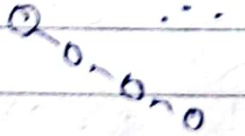
$ipl = 1 + 1 = 2$

$ipl = 2 \times 3 = 2 + 6 = 8$

Best ipl = $n \log_2 n$ order



worst ipl = $\frac{n(n-1)}{2}$



Application

① Symbol table (fixed data) only search, (Compiler Design)

→ $a_1 < a_2 < a_3 < \dots < a_n$ (n) (internal node)

→ $e_0 < e_1 < e_2 < e_3 < \dots < e_n$ (n+1) (external nodes)

Associated freq. prob.

$q_0 < q_1 < q_2 < q_3 < \dots < q_n$ (n+1)

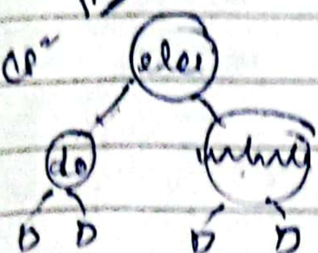
$$\left[\sum_{1\le i\le n} p_i + \sum_{i=0}^{n} q_i = 1\right]$$ → Summation of probability

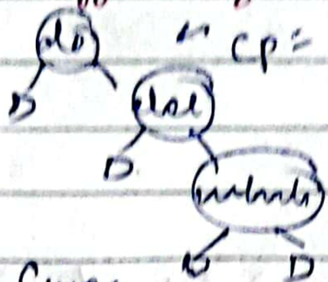
Total cost : $\sum_{1\le i\le n} p_i \times (\text{level}(a_i)) + \sum_{0\le j\le n} q_j \times (\text{level}(e_j))$

or $\sum_{0\le j\le n} q_j \times (\text{level}(e_j) - 1)$

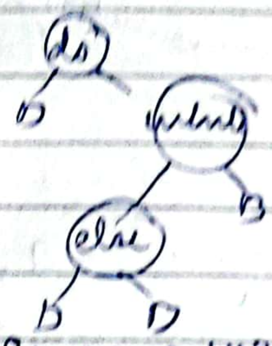
Optimal BST ← same cost will be different only.

1.9





Cp = 1.5



Cp = 1.6

Cwp = 2×4 + 1×2 = 10/7 (13)

lexicographically BST

Cwp = 1×2 + 2×2 + 3×2 = 2 + 4 + 6 = 12/7

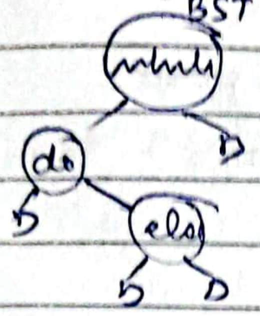
Cwp = 1×2 + 2×2 + 3×2 = 17/7

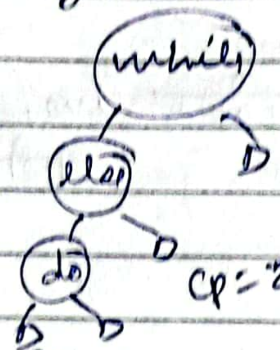
1×1 + 2×2 + 3×2 + 4×2 = 19/7

= 1 + 4 + 6 + 8

$^{6}C_3 / (3+1)$ = 5

$^{4}C_2 / (2+1)$ = 4×3 / (2×3) = 2





Cp = 2.65

No. of nodes = Prob. = 1/7

level → No. of comp.

Cwp = Cp = 2.05

Cwp = $\frac{1}{7} + \frac{1}{7} + \frac{2}{7} + \frac{2}{7} + \frac{3}{7} + \frac{3}{7} + \frac{3}{7}$

$= \frac{15}{7}$

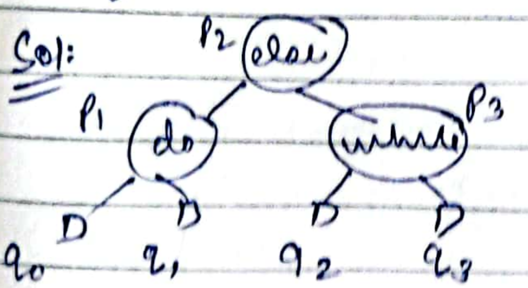
$^{10}C_5 / 6$ = 42

Q) $p_1 = 0.5$ $q_0 = 0.15$

$p_2 = 0.1$ $q_1 = 0.1$ $q_3 = 0.05$

$p_3 = 0.05$ $q_2 = 0.05$

Sol:



1×0.1 + 2×0.5 + 2×0.05

+ 2×0.15 + 2×0.1 + 2×0.05 + 2×0.05

(ej - 1)

= 0.1 + 1 + 0.10 +

0.3 + 0.2 + 0.1 + 0.1

= 1.9

03.04.2024
Thursday

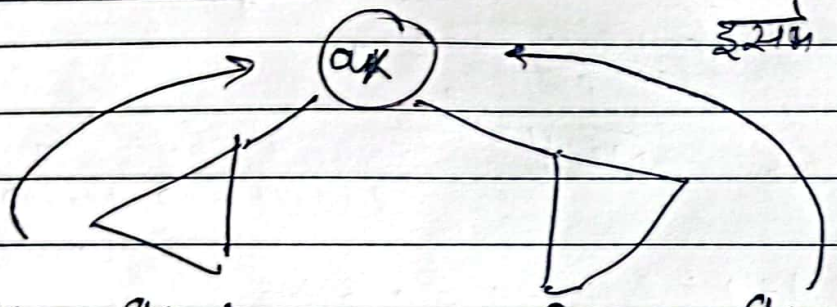
cost of finding cost of all possible BST trees = n × No. of trees in BST

No. of BST $= n \times \frac{(2n)!}{(n+1)!\,n!} = \frac{2n C_n}{n+1}$

n=1	1	exponentially
n=2	2	
n=3	5	
n=4	14	
n=5	42	

$a_1 < a_2 < a_3 < \cdots < a_n$ → 2 node tree optimal

Intuition for DP



इसमें जितना ans. होगा

ans पता है | ans. पता है

Base case.

① Single node tree का optimal answer होगा.

② a_1, a_2, a_3

$a_1 < a_2 < \cdots a_n$

Let Tij be an OBST for identifiers $a_{i+1}, a_{i+2}, \cdots a_j$

$[e_i, e_{i+1}, \cdots, e_j]$

Let Cij = cost of tree Tij

rij = root of cij

wij = weight of Tij $= q_i + \sum_{i+1 \le k \le j} (p_k + q_k)$ = sum of prob.

Tii = empty tree $(i+1 = k < i)$ → invalid

Cost of Cii = 0

Wii = qi (see formula 2)

Need to find —

Ron, Con, Rin, Won [1 to n nodes]

Let Tij be an OBST for identifiers $a_{i+1}, a_{i+2}, \ldots a_j$

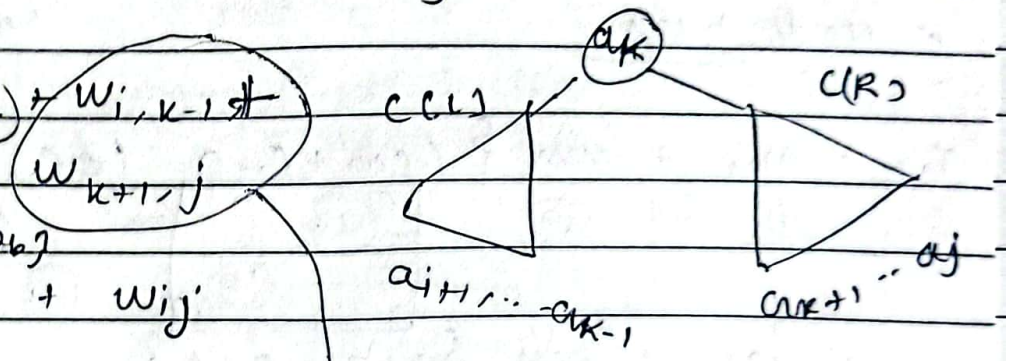
$[e_i, e_{i+1}, \ldots e_j]$

and if $r_{ij} = k$ $[i+1 \le k \le j]$

$C_{ij} = C(L) + C(R) + p_k + W_{i,k-1} + W_{k+1,j}$

a_k → cost (sum of prob)

$\Rightarrow C_{ij} = C_{i,k-1} + C_{kj} + W_{ij}$



$\Rightarrow C_{i,k-1} + C_{k,j} + W_{ij} =$

$W_{ij} + \min_{i<k\le j} [C_{(i,k-1)} + C_{(k,j)}]$

Left subtree ~~2 to or~~ 2 to j will ~~for~~ 2 to or same for right

↳ 1 node comb^n → optimal ✓

2 node comb^n

a_1, a_2 → a_2 root $a_2\ a_3$ same $a_3\ a_4$ same ....

1 root

find optimal ✓

3 node comb^n

a_1, a_2, a_3

a_1 root a_2 root a_3 root

$a_2\ a_3$ are in T_{ij} a_1 a_3 a_1, a_2 ✓

Q) $n = 4$

$a_1 < a_2 < a_3 < a_4$

do if return minds

p_1	p_2	p_3	p_4
3	3	1	1

q_0	q_1	q_2	q_3	q_4
2	3	1	1	1

Sol: 1 Node tree

$T_{01} = C_{01} = p_1 + q_0 + q_1 = 8$

$\hookrightarrow a_1$

$T_{12} = C_{12} = p_2 + q_1 + q_2 = 7$

$\hookrightarrow a_2$

$T_{23} = C_{23} = p_3 + q_2 + q_3 = 3$

$\hookrightarrow a_3$

$T_{34} = C_{34} = p_4 + q_3 + q_4 = 3$

$\hookrightarrow a_4$

2 Node trees

$T_{02} = w_{02} + \min \{ (c_{00} + C_{12}), (C_{01} + C_{22}) \}$

$\Rightarrow$

$= 12 + \min \{ 0+7, 8+0 \}$

$= 19$

$r_{02} = a_1$

$C_{02} = 19$

$T_{13} = w_{13} + \min \{ (C_{11} + C_{23}), (C_{12} + C_{33}) \}$

$= (+ \min \{ 0+3, 7+0 \}$

$p_2 + p_3 + q_1 + q_2 + q_3$

$= 3 + 1 + 3 + 1 + 1$

$= 9 + 3 =$

12

$C_{13} = 12$

$r_{13} = a_2$

3 root

4 root

$T_{24} = w_{24} + \min \{ (C_{22} + C_{34}), (C_{23}, C_{44}) \}$

$= p_3 + p_4 + q_2 + q_3 + q_4$

$= 1 + 1 + 1 + 1 + 1$

$= 5$

$\min \{ 0+3, (3+0) \}$

3

$+$

$= 8$

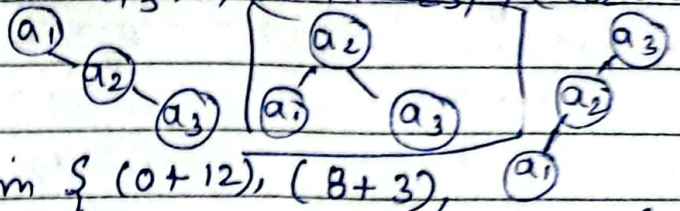
$r_{24} = a_3$

$C_{24} = 8$

3 node tree

T_{03} — a_1 as root, a_2 as root, a_3 as root

$$C_{03} = w_{03} + \min\{(c_{00}+c_{13}), (c_{01}+c_{23}), (c_{02}+c_{33})\}$$



$$= p_1+p_2+p_3 + q_0+q_1+q_2+q_3 + \min\{(0+12), (8+3), (19+0)\}$$

$$3+3+1+2+3+1+1 = 14+11 = 25$$

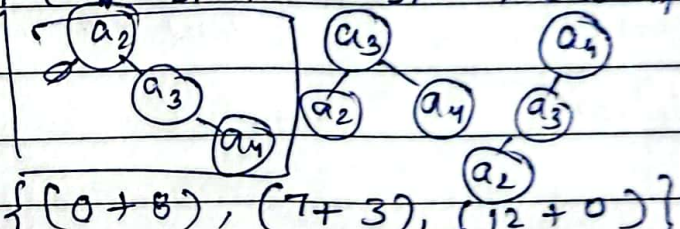
$C_{03} = 25$, $r_{03} = a_2$

T_{14} ○

$$C_{14} = w_{14}$$

root 2 3 4

$$= p_2+p_3+p_4 + q_1+q_2+q_3+q_4 + \min\{(c_{11}+c_{24}), (c_{12}+c_{34}), (c_{13}+c_{44})\}$$



$$= 3+1+1+3+1+1+1 = 11+8 = 19$$

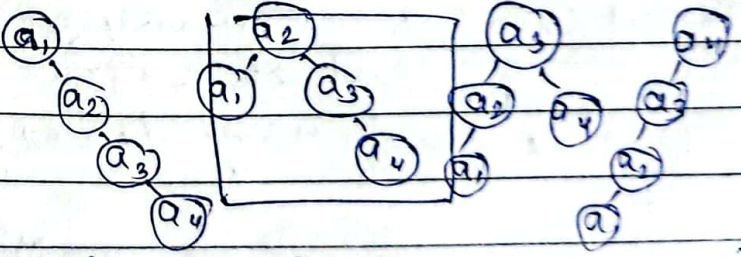
$$\min\{(0+8), (7+3), (12+0)\}$$

$C_{14} = 19$, $r_{14} = a_2$

4 node tree

T_{04} — a_1 a_2 a_3 a_4

$$C_{04} = w_{04} + \min\{(c_{00}+c_{14}), (c_{01}+c_{24}), (c_{02}+c_{34}), (c_{03}+c_{44})\}$$



$$= p_1+p_2+p_3+p_4 + q_0+q_1+q_2+q_3+q_4$$

$$= 3+3+1+1+2+3+1+1+1 = 16 + \min\{(0+19), (8+8), (19+3), (25+0)\}$$

19 16 22 25

$$= 16+16 = 32$$

$C_{04} = 32$, $r_{04} = a_2$

Time complexity:

m node trees → m comparisions

$O(mn^2) = O(n^3)$ comparisions

not amortized

04.04.2024
Thursday

Dictionary Operation

→ Implementation different

- Balanced BST – RT Tree & AVL – O(log n)
- Splay Trees – O(log n) → For a series of ~~long~~ operation
- Treaps → ensures not sure
- Sorted list → Array — O(log n) search, O(n) insert/delete
  - Linked list — O(n) search

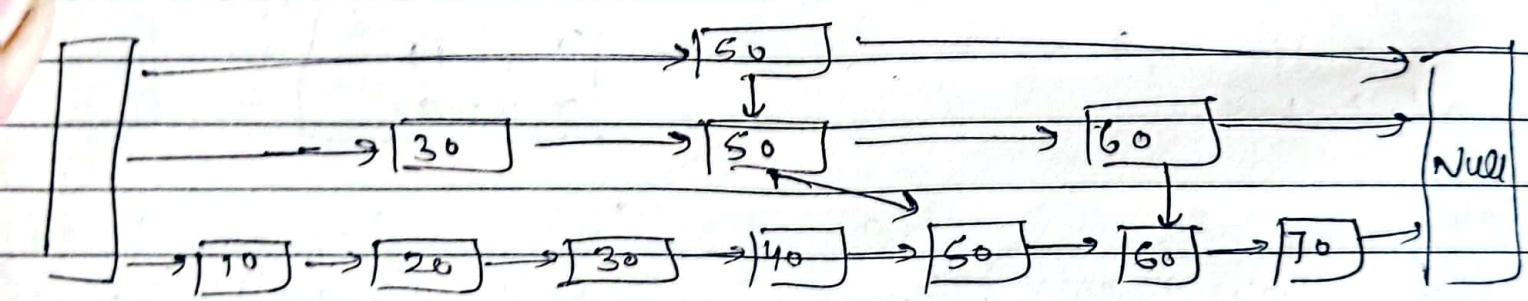
Based on this Skip list

Skip List

Linked list (sorted) → 10 → 20 → 30 → 40 → 50 → 60 →

But no use in improving complexity.

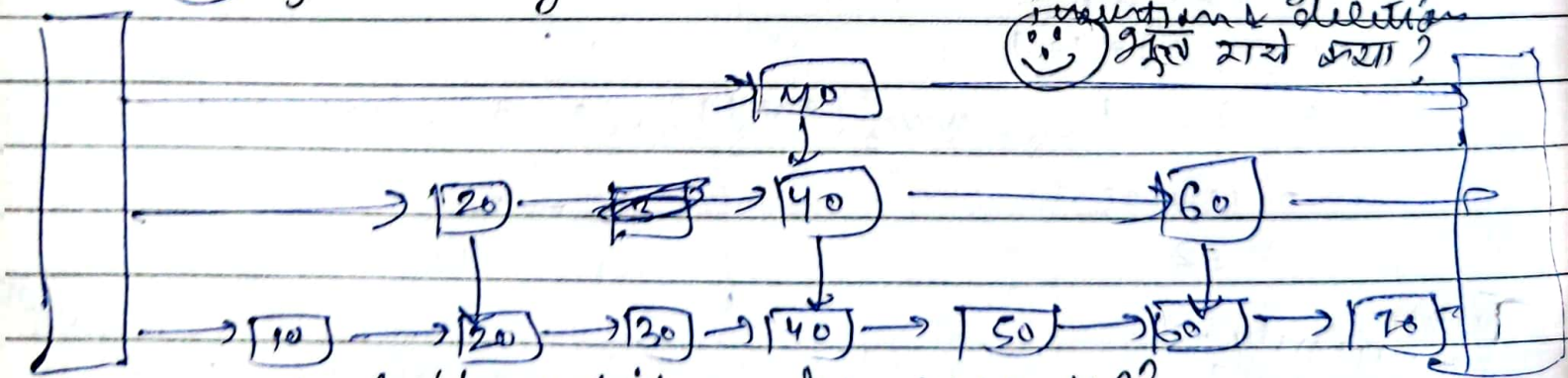
Skip list



Perfect Skip list (Binary search type skipping list)

mid of 3 or 4 ?

↳ such thing do not exist (Reason: dynamic distribution of nodes) insertion & deletion होने पर क्या होगा?



Q) How to decide which node goes up?

Sol: Toss a coin 50% prob. that node goes up the list & 50% prob. that does not goes up

2 list → $\sqrt{n}$
3 list → $\sqrt[3]{n}$

Randomised data ~~list~~ structure → depends on level.

Perfect no. of levels for 'n' nodes → $\log_2(n)$
(but can be choice of implementation)

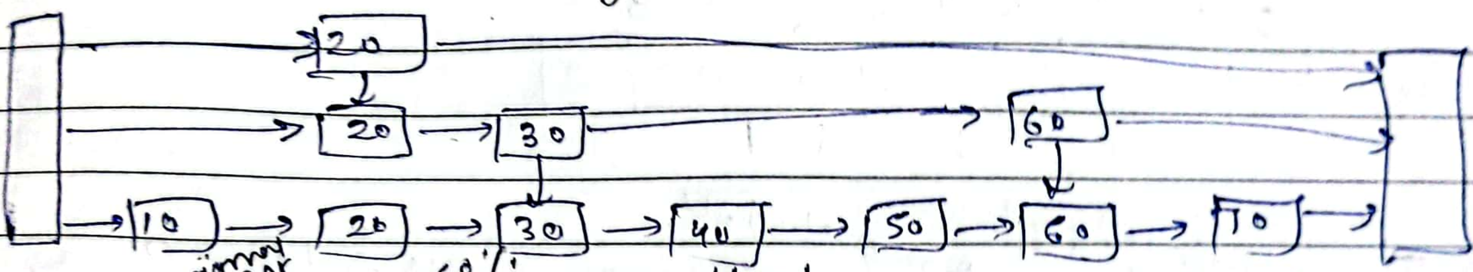
Time complexity of Search = $O(\log n)$

Time Complexity of Delete = $O(\log n)$ { $O(\log n)$ Search + $O(\log n)$ level del, ins.

Not Perfect Skip List ~~Randomised skip list~~

from any level i to level i+1 ⇒ 50% nodes are moving upwards.

No. of levels = $O(\log n)$



original cost, 50%, upward level, same level

$C(j) = 1 + \frac{1}{2}C(j-1) + \frac{1}{2}C(j)$ → 50%

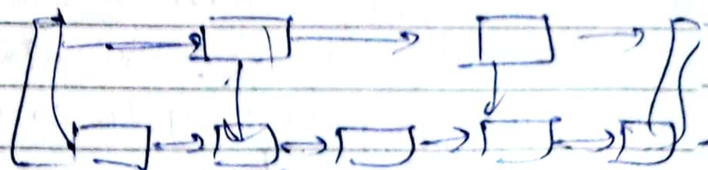
⇒ $C(j) = 2 + C(j-1)$
$= 2 + 2 + C(j-2)$
$= 2 + 2 + \cdots (\log n)$ times $+ C(0)$

⇒ $C(j) = 2(\log n)$ → Time complexity

$n + \frac{n}{2} + \frac{n}{2^2}$
$n(1 + \frac{1}{2} + \frac{1}{2^2}$
$n (\frac{1}{2})^{\log n}$
$2n$ $2^{\log_2 n}$

Space complexity → $O(2n)$

Insert / Delete
(i) Find position (search)
(ii) Insert / Delete
(iii) coin toss करके पता करो
ऊपर जा same रहेगा



$\frac{K \cdot n \cdot K}{K^2}$ $K\lceil\frac{n}{K}\rceil$ $(K+1)\frac{n}{K}$ $K = \sqrt{n}$

n n/2 K ← ~~O(log n)~~

09.04.2024
Tuesday

BLOOM FILTER → Probabilistic data structure

↳ Used to check whether an element is a part of set or not.
(login करते वक़्त username check)

This can give FP (False Positive) ✓ in some cases.
Identifies a membership in the set but can be FP results (False Positive)

But never gives FN (Username is not a part of set)

Hash fⁿ takes a i/p & gives a fixed size o/p

Bloom filter uses Hashing.

	0	1	2	3	4	5	6	7	8	9
BF	0	0	0	0	0	0	0	0	0	0
			1		1		1	1		

↗ multiple different hash fⁿs ↳ FP confirm prob. ↑ses

k hash functions value →	x_1	x_2	x_3 (Check for)
↓ k_1 (value)	2	4	1
k_2 (val)	10	6	0
k_3 (val)	7	7	6

7 ↓ mark 1 in BF

10 → means it outside range

→ If all fields = 0 Then confirm not part of set
else There is chance of its ~~existence~~ not existence
"Can't say"

① Cache में particular page exist करता है कि नहीं ~~है~~ अगर है।
② Membership operation check करना

Akami → 75% Downloads only once
They will only bring to cache if more than once a file is downloaded.

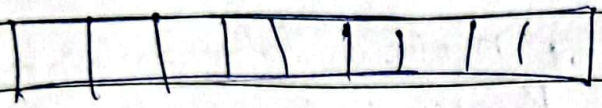
Hash function is independent of each other

DATE
PAGE

First time download होने पर BF में set करते
& next time confirm करते हैं।

100,000,000 files
↓
75,000,000 download single time
↓
other more than 1

K Hash fn (K=7) by Akash
80,000,000 × 10 Unique entries in BF



file1 → Check
exist → can / can't be
not exist "Confirm"
single time add first time
directly get from main server "Download"

Check in Cache
↓
Search on distributed server
✓ ✗

Distributed Caching
Certain web pages are cached in server else page is downloaded
When we check in several servers to find if its cached in some then we get directly there.
① "Searching time on every distributed server"
② "Increased network traffic"

If No. of elements ↑ but Size (BF) remains same then FP ↑ & FN ↓ → Confirmation decreases
↳ Not confirmation increased.

Not possible to delete particular element

Putting & searching always possible

Wikipedia
& element exists but "1"

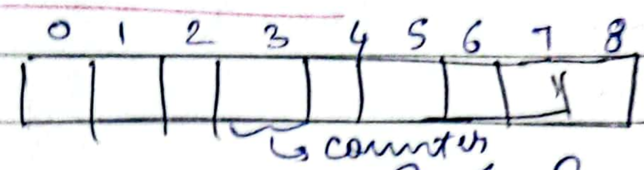
10.04.2024 Wednesday.

Application ✓

DATE
PAGE

Counting Bloom Filter

4 bit counter



more increment ~~कीया~~ किया तो overflow in single cell
more decrease (16 से ज्यादा) तो FN आ सकता है।

Q) m blocks लिए BF में तो कितने n elements' membership effectively can be done → value of K optimizations.

Sol: m: Size of array (Bloom Filter)
n: No. of members in the set

Optimal value for $k = \left(\frac{m}{n}\right) \ln(2)$ → Probabilistic

False +ve rate = $\left(1 - e^{-kn/m}\right)^k$

m ~~का~~ भी नहीं पता

Time ∝ No. of Hash function

Time ↑ ① Slower ~~Has~~ Bloom Filter
② Bloom Filter will be filled soon
↙
∴ k can't be very large
~~Coulbe~~: Counter intuitive for FN rate more

Array

Q a[0] ---- a[n-1] n integers
a[l...r] → highest value

computational geometry → Interval trees

Priority trees

Range trees

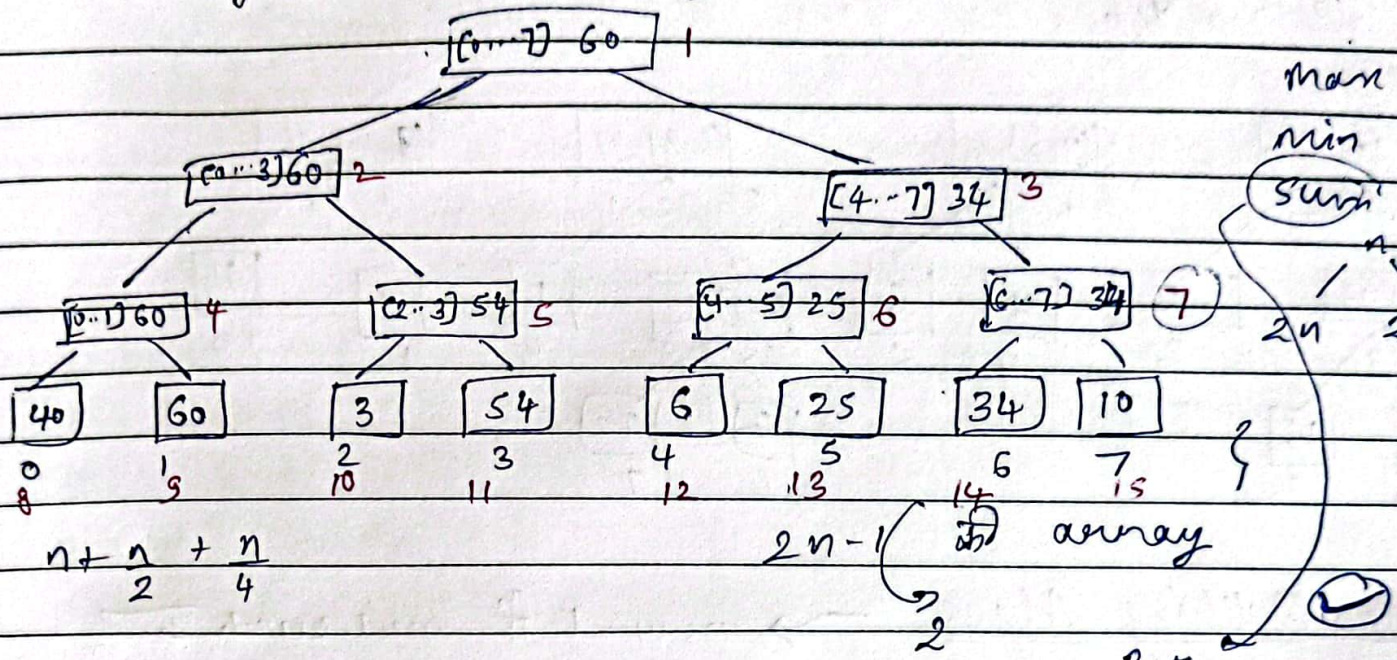
DATE
PAGE

Search range → O(n) For m queries : O(nm)

SEGMENT TREES → for m queries : O(m log n)

Static Set of Data

For range (l, r) → 2 log(n) comparision



main min sum

n

2n 2n+1

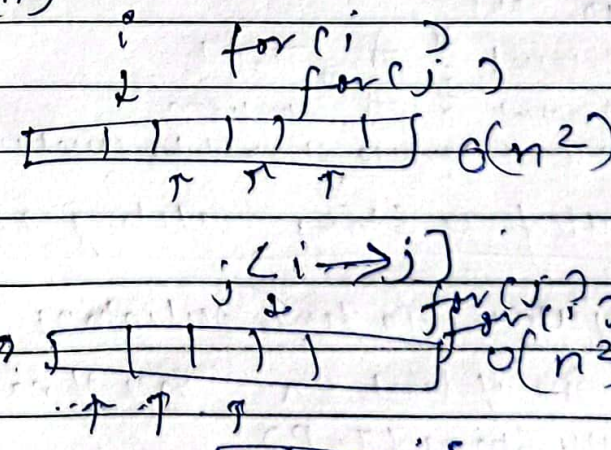
Better use Prefix sum (if no dynamic changes)

If updates are higher use Segment Tree log(n)

Constant time (Space)

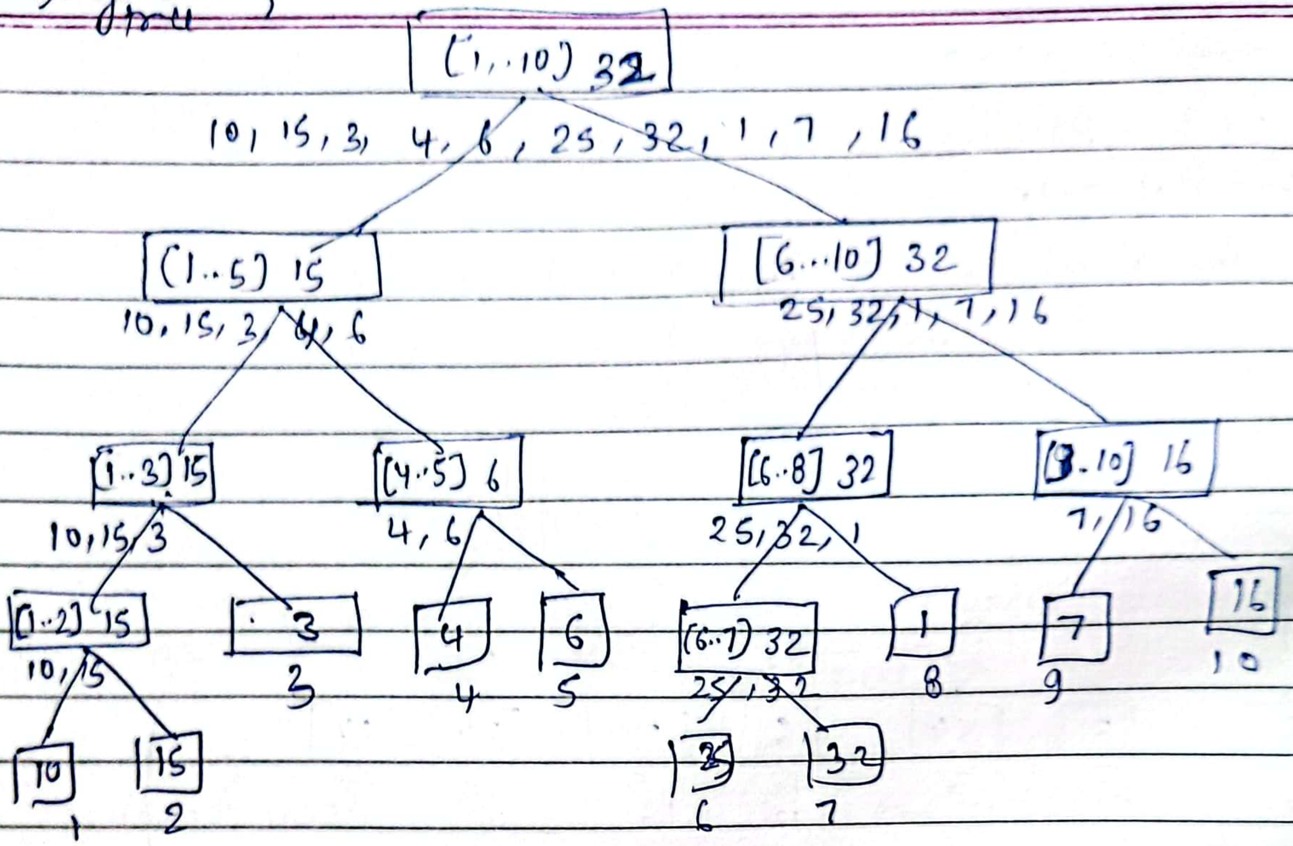
	0	1	2	3	4	5
0	40	60	60	60	60	60
1						
2						
3						
4						
5						

Prefix Sum



16.04.2024 Monday.

Segment tree } storing ranges also



String Matching Algos

→ ○ ~~ms word में word search करना~~
○ grep in linux.

Brute force

ababacabcaba → string |s| = n
abc → pattern |p| = m

} O(nm) ↑ Nested for loop approach.

Algo cond^n
- Present है तो बताओ।
- Absent है तो confirm
- ~~All instances should be counted~~

O((n-m) m) = O(nm)

① Brute force String matching algo. (Naive string matching Algo)

Objective: To find substring in the main string

Size of text = n   Size of pattern = m   where m <= n

Naive String (T, P)

1. n = T.length
2. m = P.length
3. for s = 0 to n-m; → O(n)   } O(nm)
4. if P[1..m] = T[s+1....s+m] → O(m)
5. print "Pattern Found";

(Either reduce 'n' comparisions
or reduce O(n) from loop
→ To reduce TC
&

DATE
PAGE

⓪ Rabin Karp String Matching Algorithm :- → Reducing m comparision

① It uses concept of Hashing.
② It works on numeric values.

Practically TC = O(n)
Mathematically TC = O(nm)
(Theoritically) ↳ if hash same then 'm' compare

(i) a b c → 1+2+3 = 6
b a c → 2+1+3 = 6
spurious hit
(ii) a b c → $1\times10^2 + 2\times10^1 + 3\times10^0$
b a c → $2\times10^2 + 1\times10^1 + 3\times10^0$ ≠
but integer overflow.
(iii) Hash algo on (ii) collision chance [Spurious hit]

hash value = (hash value of previous 'm' character − value of first character) × 10 + value of last character

Rabin Karp (T, P, d, q) → prime no. for mod; d → base ↳ ①

1. n = T.length
2. m = P.length
3. h = d^{m-1} mod q
4. p = 0 // start index location
5. t_0 = 0
6. for i = 1 to m // (initial value)
    p = (d*p + P[i]) mod q // Pattern's hash value
    t_0 = (d×t_0 + T[i]) mod q // first 'm' chars of text's hash
for s = 0 to n−m :
    if p = t*s
        if P[1...m] = T[s+1 ... s+m]
            print "Pattern matched"
    if s < n−m
        t_{s+1} = {d * (t_s − T[s+1] * h) + T[s+m+1]} mod q
        ① // From ①

Spurious Hit:

Hash value of pattern matches with hash value of text but character of pattern does not match with the character of text.

18.04.2024 Thursday

KNUTH MORRIS PRATT (KMP) ALGO (String matching algo.)

It works on the principle of suffix and prefix of a string (pattern)

We have to construct a prefix table for a given pattern before applying the KMP algorithm.

Prefix table size = m

	a	b	a	b	d
(PT) π	0	0	1	2	0

check string under consideration (length - 1) not whole

	a	b	a	b	a	c	a
(PT) π	0	0	1	2	3	0	#

Prefix Table Calculation — O(m)

Compute PT(P) → Pattern

1. m = pattern.length
2. let π[1:m] be an array
3. k = 0
4. π[1] = 0
5. for q = 2 to m
6. while k > 0 & P[k+1] ≠ P[q]
7. k = π[k]
8. if P[k+1] == P[q]
9. k = k+1
10. π[q] = k
11. return π

i = 1

	1	2	3	4	5	6	7	8	9	10	11	12
Text	a	b	a	a	b	a	b	c	a	b	a	a b
Pattern	a	b	a	b	d							
	1	2	3	4	5							

q = 0

i = 1
q = 0

(1) We are going t[i] = P[q+1]

(2)(i) if t[i] = p[q+1] :
Then increment both i & q.

(ii) if t[i] != p[q+1] :
bring q to π[q] (q = π[q])
if 'q' is already 0 then increment 'i'
if q comes to m then print "Pattern Found"

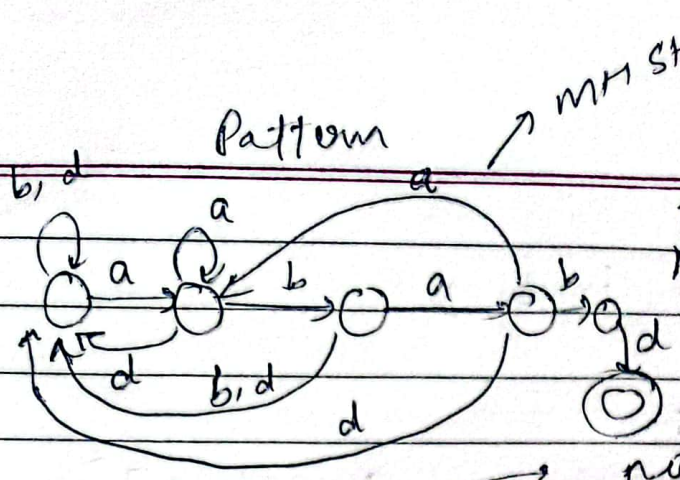
KMP(T, P) : # Text = T Pattern = P
1. n = T. length ; m = P. length
2. π = compute PT(P);
3. q = 0
4. ~~i=1~~ for i = 1 to n do :
while q > 0 & P[q+1] ≠ T[i] (match नहीं तो Back जाओ)
q = π[q]
if P[q+1] == T[i] :
q = q+1
if q == m :
print ("Pattern found")

Practically time complexity of this algo : O(n)

FINITE AUTOMATA BASED ALGO (string matching algos)

Based on the concept of prefix & suffix of a string
A Finite Automata M is constructed for the pattern. → 5 tuple machine Q Σ δ q_i F

TOC { states, i/p, transition, initial, Final }



Text: abaababcabcaab
pattern: ababd

O(n)

no. of input alphabets → $O(m^3)$ pattern finding pattern

Compute FA (P, Σ):

1. m = P.length
2. for q = 0 to m:
   for each a ∈ Σ
   k = min(m+1, q+2) # m से उपर नहीं
   repeat
   k = k-1
   until $P_k \sqsupset P_q a$ → any i/p symbol
   $\delta(q, a) = k$
   return δ

Runs until false comes

$a \sqsupset \in a$.
a is suffix of $\in a$
$\in$ epsilon

FA (T, n, δ, m): → O(n) scanning
q = 0
for i = 1 to n:
   q = δ(q, T[i])
   if q == m
      print ("Pattern found")