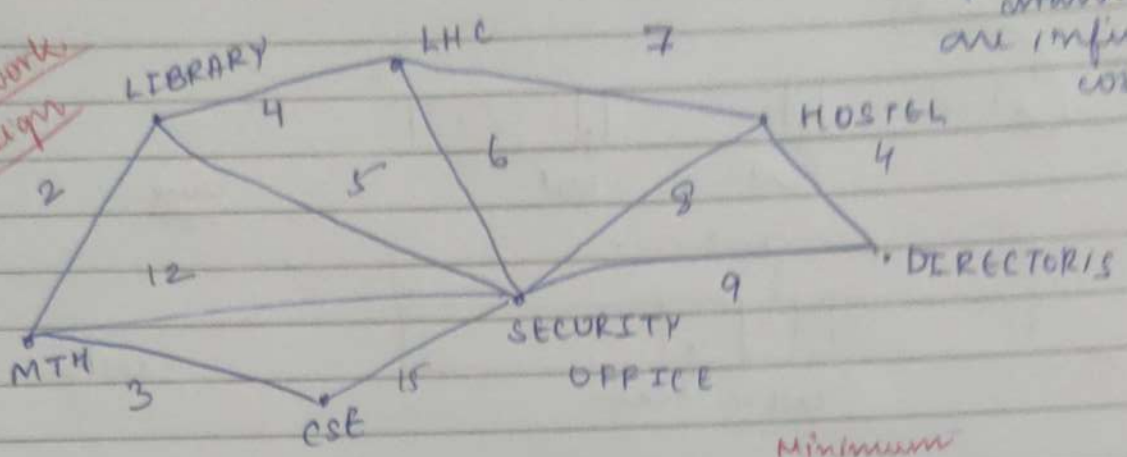
Module: Greedy & Dynamic Programming

Greedy algo design

Network design



Edges not drawn are infinite cost.

Minimum spanning Tree

(1) entire network is connected

(2) cost is minimized
distance btw^n & loc^n.

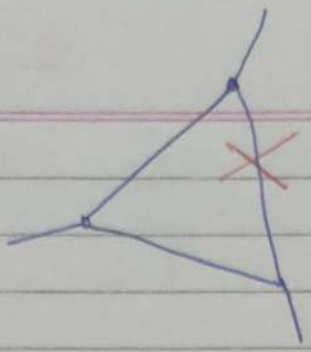
Only works for undirected graphs

LHC
HOSTEL
LIBRARY — LHC – 4 · — SEC – 5 — MTH – 2

~~Obs: cycle~~

~~Obs: cycle is~~

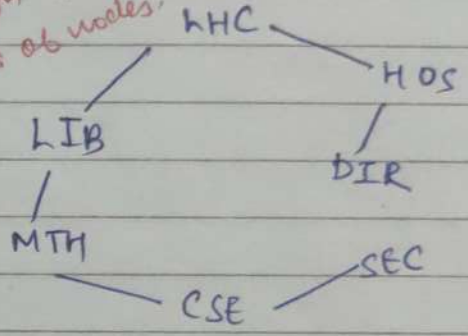
Cycle in output graph is never possible.



removing an edge from cycle → still connected & strictly less cost (if non-zero wt)

Defn [(Undirected) Tree]

An undirected graph
- connected ← ∃ a path b/w any 2 pairs of nodes
- acyclic (~~no~~ ~~cycle~~)



Claim:- Let T be an undirected tree on n nodes. T must always have n-1 edges.

Pf: Simulate process of building tree
Initially all disjoint
. . . . . .

Hint: Each edge (if put smartly) reduces the number of components by 1.

$n \xrightarrow{n-1} 1$

Ans **Induction**

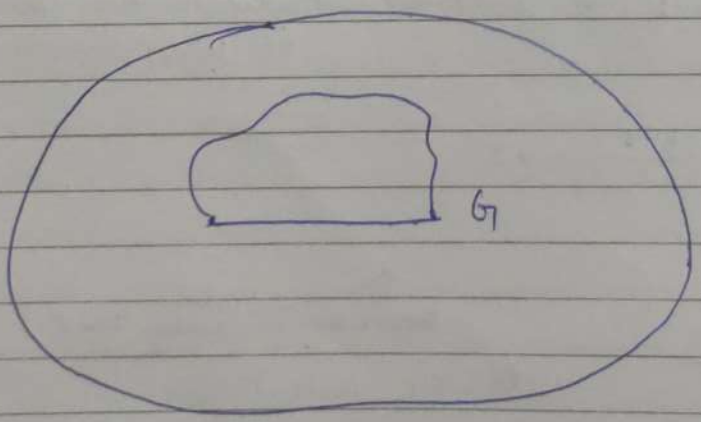
~~Division is only tree~~ also

Divide into trees.

Claim: If T a graph on n nodes
- $(n-1)$ edges
- connected

such a graph must be acyclic & hence a tree.

Pf:



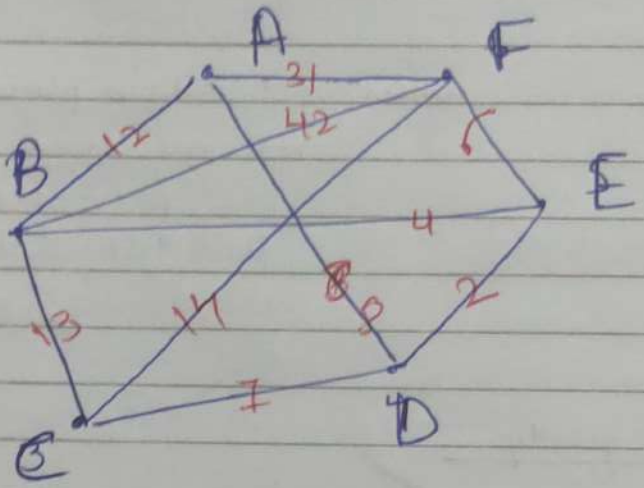
suppose there is a cycle
↓
Can remove one edge
↓
$(n-2)$ edges but still connected.

At some point all cycles finish but we have < n-1 edges

⇒⇐ to tree having = n-1 edges

∴ there is no such cycle.

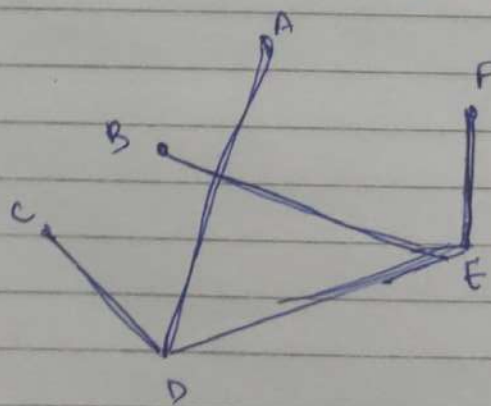
Characterization: Tree ⟺ n-1 edges
of nodes of graph. graph.



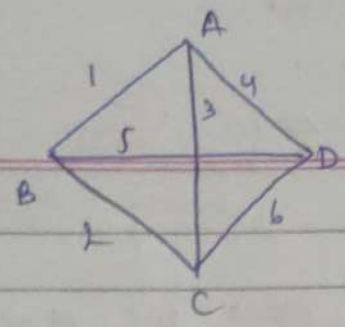
least wt.
spanning tree
Minimum
Spanning
Tree
connected

Choose smallest edges first (skip if cycle is created)

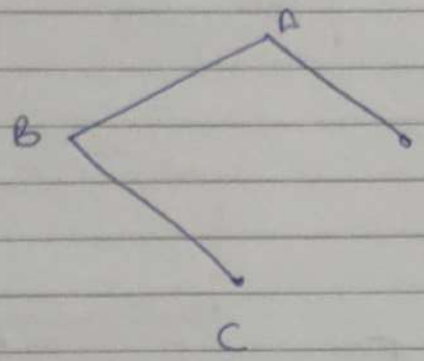
DE, BE, FE, CD, AD, BA, BC, CF, AF, BF



min spanning Tree



AB, BC, CA, AD, BD, CD



Min^M Spanning Tree

when n-1 edges placed we are done due to prev. claim such that acyclic then ~~connected~~ we are done.

General property of greedy

- algorithm is simple (take locally best step) ↳ that leads to globally best
(locally best ⇒ globally best)
- analysis is involved.

At each edge that does not lead to cycle introduced at least one new node into the system.

Procedure MST (G)

// G = (V, E)

// G is given as a weighted adj. list.

F ← set of edges E sorted lower to higher

ctr ← 0 , T ← ∅ , i ← 1

while (ctr ≠ n-1) :

e ← F[i]

if (T ∪ {e} does not have cycle)

∴ T ← T ∪ {e} ; T ← merge (u, v) ← Disjoint Union.

ctr++

endif

i++

endwhile

Find (u) ≡ Find (v)

Claim: The graph T will always be a tree.

∵ of prev claim

- n-1 edges
- acyclic

) connected?

Q2. Time

Usually Dichotomy: Easy algo hard to analyse

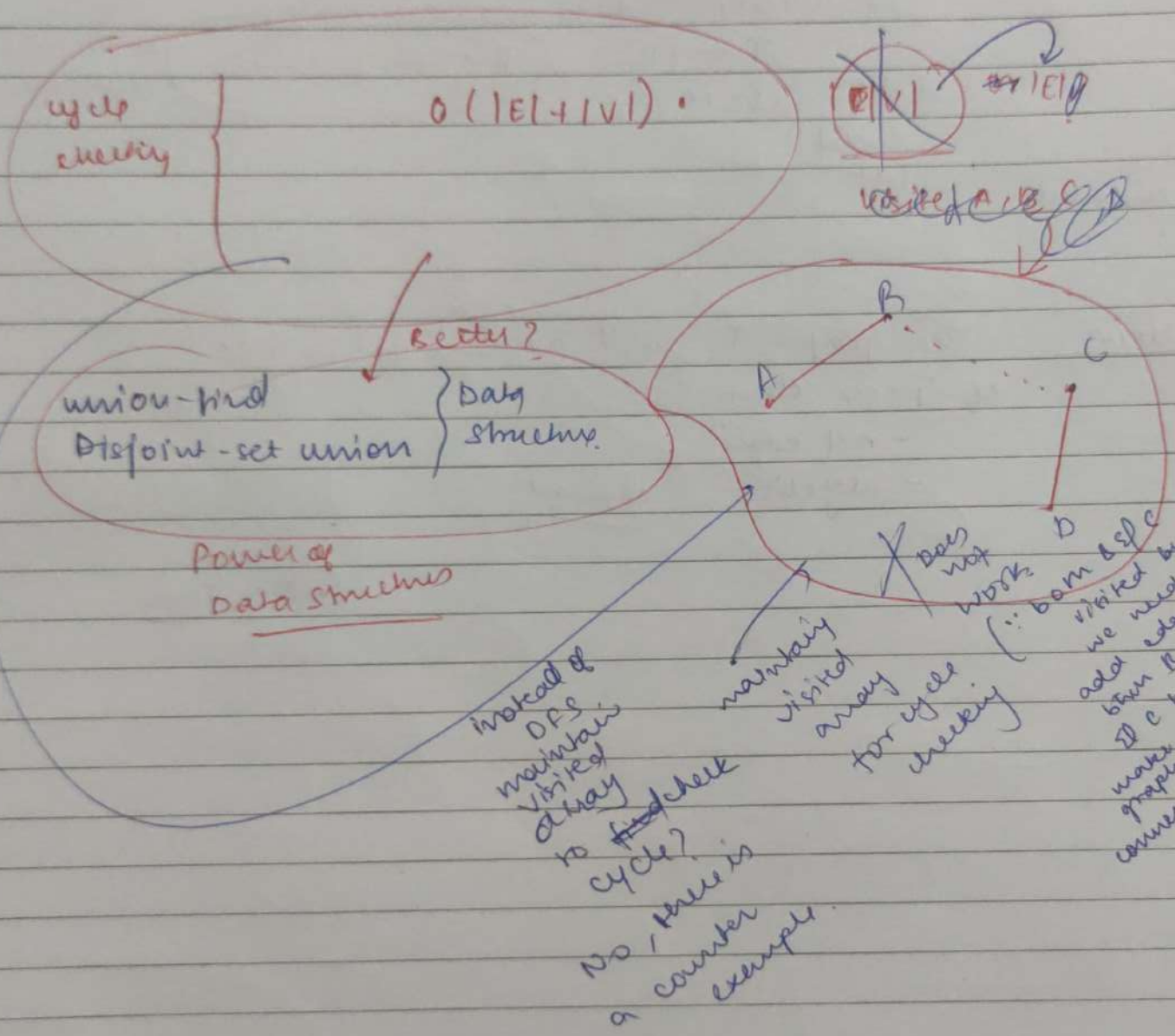
Q2. Time

Date: Page:

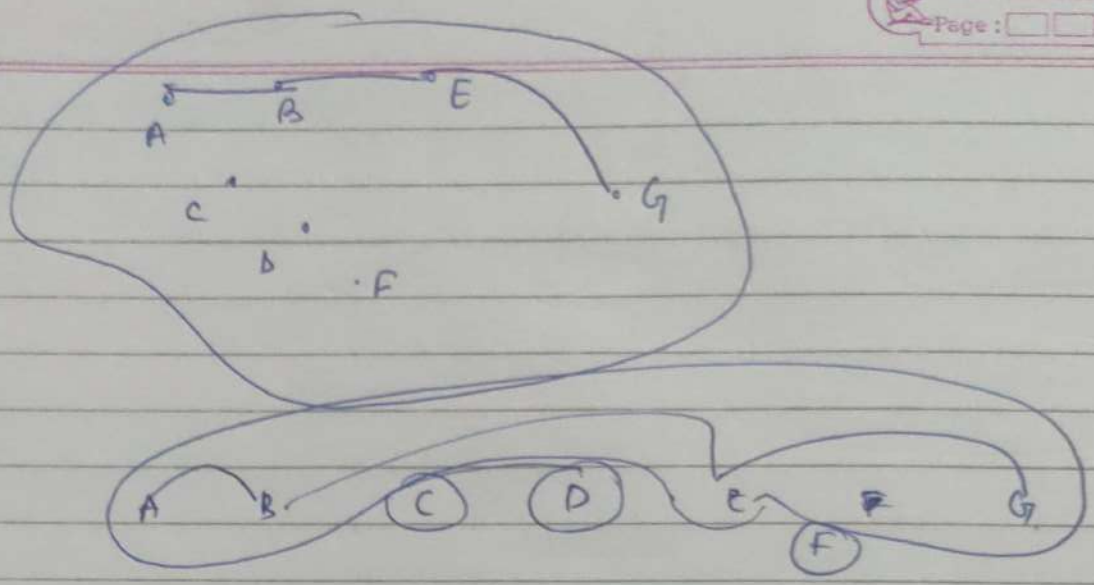
$O(|E| \log |E|)$ $\quad |E| \leq \left(\frac{|V|}{2}\right)$

Sorting

$\equiv O(|E| \log |V|)$



Disjoint-set Union



We are maintaining Disjoint sets/partitions
Partition of Vertexed.

$S_1, \ldots, S_k$ partition.

- $S_i \cap S_j = \emptyset$
- $\bigcup_i S_i = V$
- $u \in V$ & $v \in V$

Always think of operations required from a data structure.

Q: ~~Does~~ Do (u, v) belong to same partition.
Quick query

A: Yes; no (quickly)

Q: Merge partitions corresponding to u & v
Quick merge.

$\exists i$ $u \in S_i$, $\exists v \in S_i$

Merge (u, v) $S_i \leftarrow S_i \cup S_j$

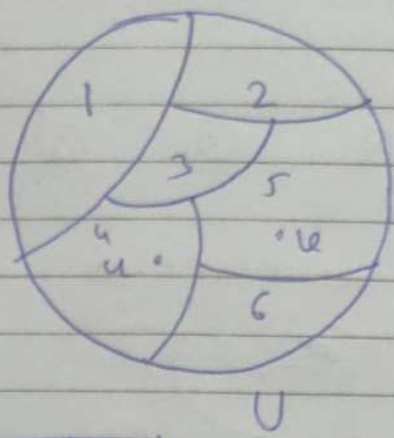
all other partitions are same

- Ask - h-15 start of end & dbts
- But we already ~~head~~ Q is full check before incrementing.
- ~~Tree~~ complete pf of Tree has n-1 edges.
- ~~Ask~~ Srutui or siv to check acyclic + n-1 edges ⇓ connected pf. 7

Lec-16

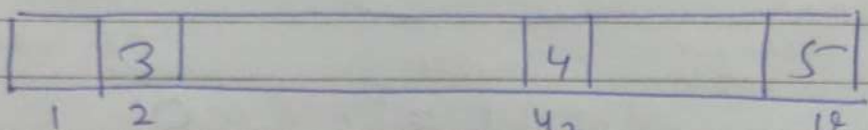
If prev implementation was

- ~~Find~~ (Equality) (u, v) : yes iff u, v ∈ same subset



- Merge (u, v) : union of two subsets u & v

Disjoint Set / Union - Find



→ Find is fast

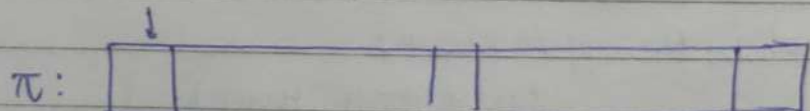
Naive merge: $S_i \leftarrow S_i \cup S_j$ may take entire array size → O(N) expensive

We will do find → O(log n)

merge → O(log n)

Directed Trees

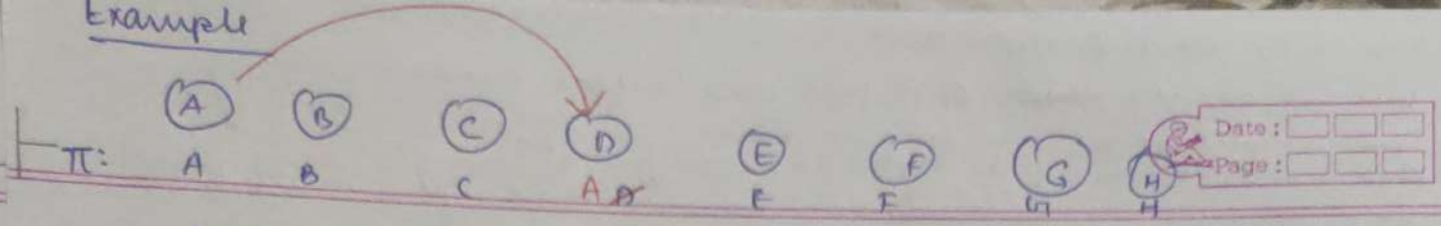
→ to store: maintain its parents



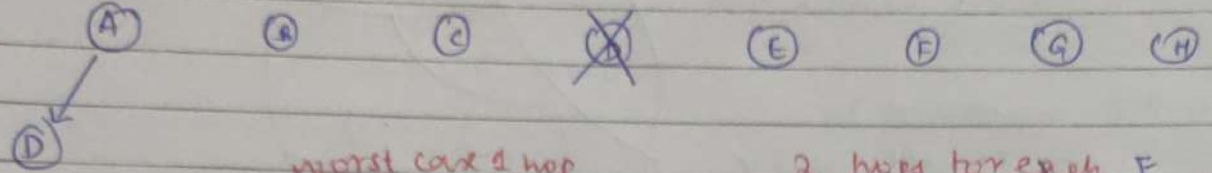
#parents ≤ 1

- We'll maintain each partition as a directed tree.
- Partition is identified with the root (source) of the tree.

Example



Merge (A, D)



worst case 1 hop for each node

2 hops for each E node

2 options A → E ↓ D OR E → A → D

Better

shallow trees are better.

Worse

root of tree

π is Helb

ht of A: 0, E: 1, D: 1

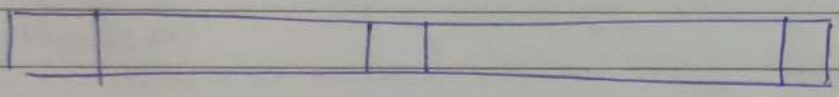
ht of E: 2, A: 1, D: 0

∴ shallow Trees are better

How do we know that the tree is shallow?

↳ maintain height array

ht:



Initially every ht is 0.

Procedure Find(u)

//returns root of tree containing u.

temp ← u

while(π(temp) ≠ temp)

temp ← π(temp)

endwhile

click

∴ ht is c log n

T.C ⇒ O(log n)

Procedure Merge (u, v)

// merge 2 trees containing u & v

if (Find(u) = Find(v))

return

endif

$\theta(\log n)$

$h_u \leftarrow ht(Find(u))$

$h_v \leftarrow ht(Find(v))$

if ($h_u > h_v$)

$\pi[Find(v)] = Find(u)$

$\pi[Find(v)] \leftarrow Find(u)$

else if ($h_v > h_u$)

$\pi[Find(u)] \leftarrow Find(v)$

else

$\pi[Find(v)] \leftarrow Find(u)$

$ht[Find(u)] \leftarrow ht[Find(u)] + 1$

Claim: If there are n nodes, the ht of tree is $\Theta(\log n)$ at most.

Pf: Try doing without it

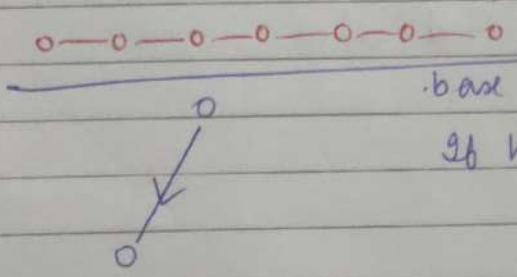
but we do Induction.

base: $n = 2$, height $= \log_2 n = 1$

I.H. $K \le n-1 \Rightarrow$ ht $\le C \cdot \log_2(K)$

Idea: If a tree of ht K then $\Rightarrow$ got $\ge 2^K \cdot C$ nodes

not shallow (dense tree)



base case: $K=1$

If ht $= 1 \Rightarrow$ tree has ≥ 2 nodes

I.S: : Note the first time tree's ht become $(K+1)$.

two trees of ht K merged.

left tree had $\ge 2^K$ nodes

right tree $\ge C \cdot 2^K$ nodes

the union tree $\ge C \cdot 2^{K+1}$ nodes

$n = C \cdot 2^K \Longrightarrow K = O(\log_2 n)$

□

Minimum Spanning Tree

Kruskal's Algo

```
Procedure MST(G)
    // G = (V, E)
    // G is given as weighted adjacency matrix
F ← sort list of edges from low wt to high wt
j ← 1
ctr ← 0
while (ctr ≠ n-1)
        (u,v) ← F[j]
        j++
        if (Find(u) = Find(v))
                continue
        else
                ctr++
                Merge(u, v)
        endif
endwhile
```

T.C.

sorting → E log E

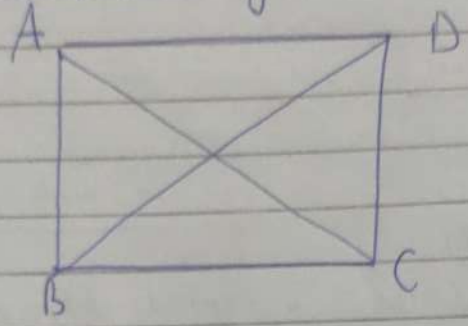
while loop → |E| log |V|

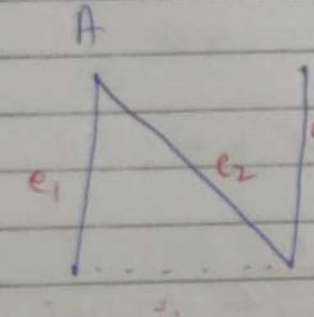
each iteration

Greedy → special easy rule that works

Q1: Is it correct?

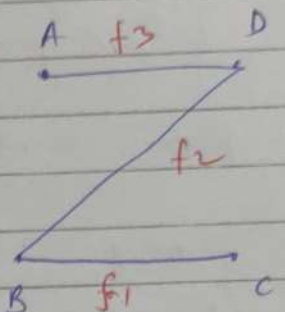
Pairing Property of Spanning Tree





a spanning tree

may not share an edge!



T_1 : T_2

cardinality same as ~~originally~~ both had originally

$T_1 \setminus T_2$	$T_2 \setminus T_1$
e_1	f_1
e_2	f_2
e_3	f_3

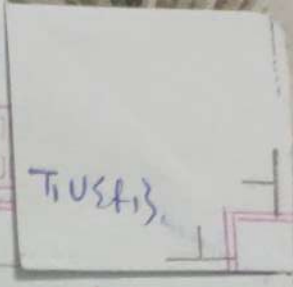
Pairing property

$T_1 \cup \{f_i\} \setminus \{e_i\}$ is still a spanning tree

6! pairing without property

Can pair AB to BC but AB not to AD

Lemma: For any spanning trees T_1 & T_2 s.t. $T_1 \neq T_2$, $\exists$ exists a valid pairing.



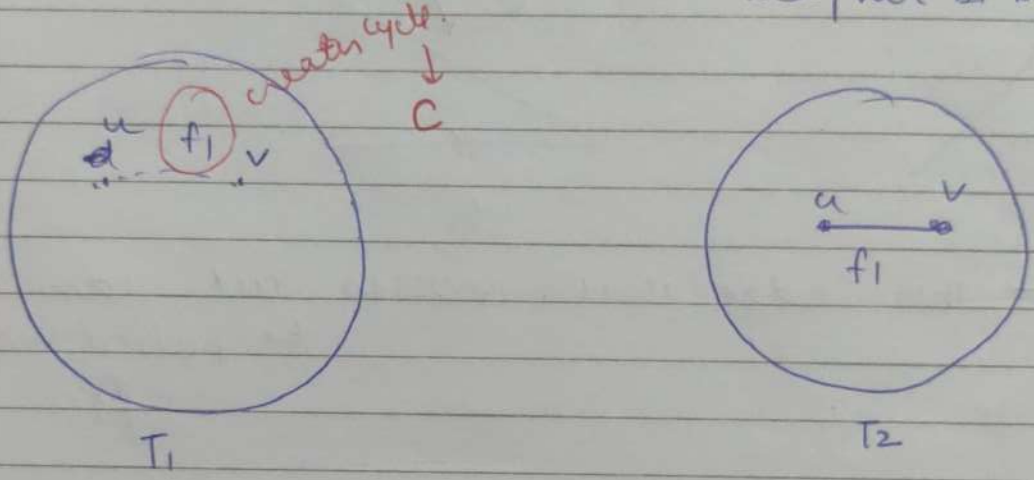
Asymmetry in def^n as T_1 becomes MST after every edge of not T_2

$T_1 \setminus T_2 = \{e_1 \ldots e_2\}$

$T_2 \setminus T_1 = \{f_1 \ldots f_\ell\}$

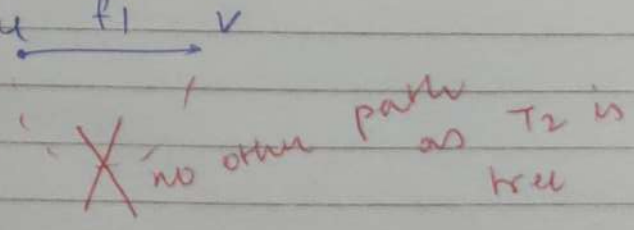
if $T_1 \cup \{f_i\} \setminus \{e_i\}$ ~~may not make~~ IsMST then

$T_2 \cup \{e_i\} \setminus \{f_i\}$ may not be MST (check)

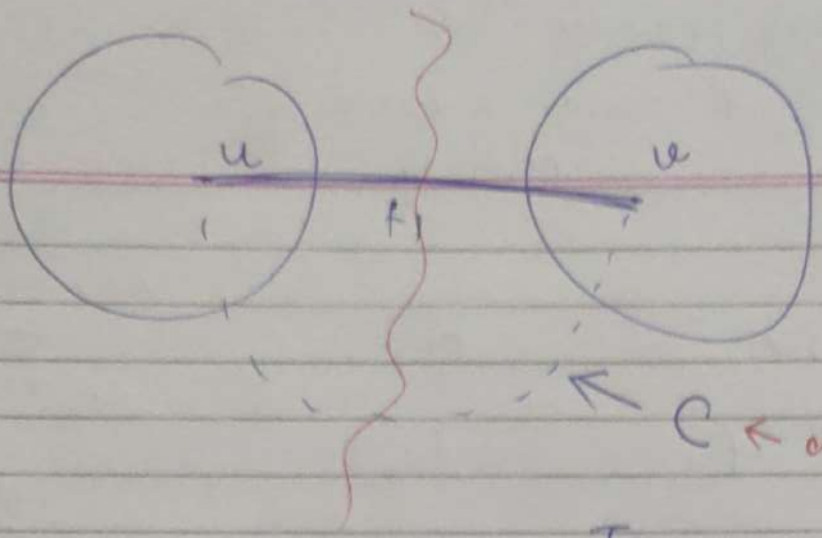


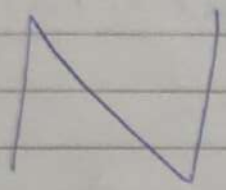
Pick some $f_1 \in T_2 \setminus T_1$

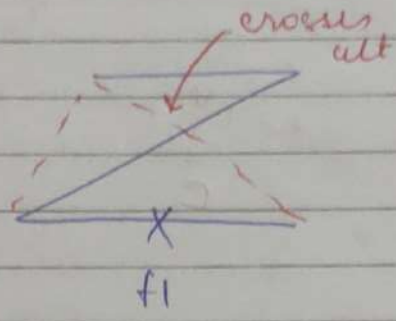
visualise C in T_2



if remove f_1 we get two subtrees







Claim ⇒ This edge that crosses cut can be paired with f_1.

- check find (u) code from notes
- ~~[illegible]~~ ht: n ⇒ log n nodes
~~log n nodes~~

- if $T_1 \cup \{t_i\} \setminus \{e_i\}$ in MST $\overset{?}{\Rightarrow}$ $T_2 \cup \{e_i\} \setminus \{t_i\}$ in MST

L-17

Pairing Property:

spanning Tree

$T_1 \neq T_2$

$\{e_1, \ldots, e_\ell\} \in T_1 \setminus T_2$ $\exists$ a pairing s.t. $\forall i$

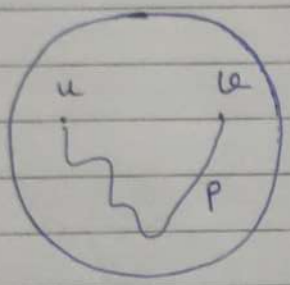
$\{f_1, \ldots, f_\ell\} \in T_2 \setminus T_1$

$\{\langle e_1, f_1\rangle, \ldots, \langle e_\ell, f_\ell\rangle\}$

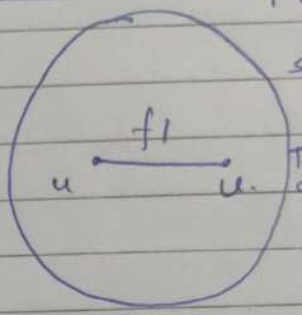
$T_1 \setminus \{e_i\} \cup \{f_i\}$ must be a spanning tree

Note: only $T_1 \setminus \{e_i\} \cup \{f_i\}$ is a spanning tree $\forall i$

$T_2 \setminus \{e_i\} \cup \{f_i\}$ is not necessarily $\forall i$.

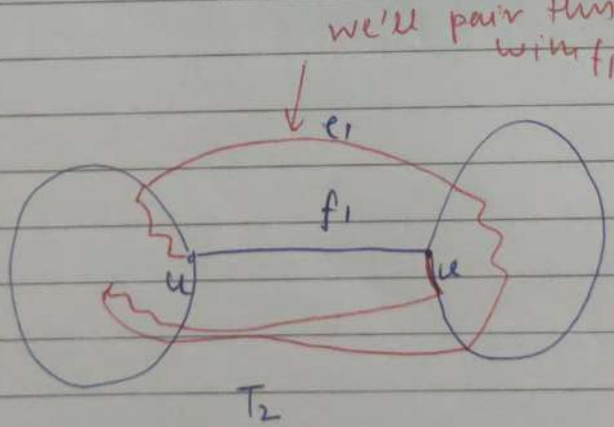


T_1



T_2.

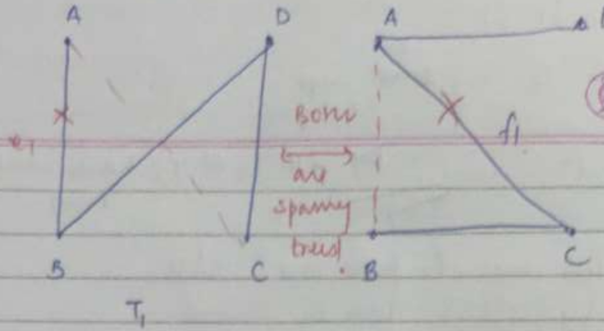
Tree → any edge is unique path.



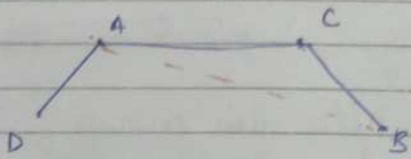
T_2

$e_1 \in T_1 \setminus T_2$ clearly ∵ o/w we'll have cycle in T_2. as the 2 subtrees are already connected by f_1

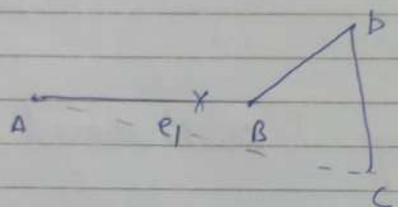
Induction
what trying to prove.



ABDC



Ex.
Try
more
pairing



$T_1 \cup \{f_1\} \setminus \{e_1\}$ is a spanning tree

$T_1 \cup \{f_1\}$ is a cycle
remove e_1
"n-1" edges & still connected as e_1 part of cycle
⇓
acyclic too
⇓
MST.

But why is $T_2 \cup \{e_1\} \setminus \{f_1\}$ a spanning tree?

$T_2 \cup \{e_1\} \rightarrow$ cycle
remove f_1
n-1 edges & still connected as f_1 part of cycle
⇓
acyclic ⇒ MST.

$T_3 = T_2 \cup \{e_1\} \setminus \{f\}$

↑ differs from T_2 by ~~one~~ $\ell-1$ edges

Then By Ind^n done ← By IH T_3 & T_2 have valid pairing

Base-case.

T_2 differ by T_2 from one edge.

claim: $\{T_3 \setminus T_2, T_2 \setminus T_3\}$ can be paired.

Clearly, ~~T_3 gets disconnected & Cycle~~ T_3 becomes T_2 & T_2 become T_3 after this exchange & both are ~~MST~~ spanning trees

hence valid pairing.

⇒ By sequence of pairings T_1 can be changed to T_2

#: 2nd one not MST

$\ell-1$ edges

Kruskal pf. Change the pt & get checked by sir

Correctness of Kruskal Algo

claim: let T^* be the spanning tree returned from Kruskal's algo

let T be any other spanning tree

$$wt(T^*) \leq wt(T)$$

Proof: Let $T^* \neq T$

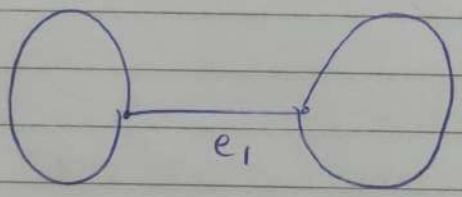
$\{e_1, \ldots, e_l\} \in T^* \setminus T$

$\{f_1, \ldots, f_l\} \in T \setminus T^*$

$\forall i$ $\boxed{wt(e_i) \leq wt(f_i)}$ WTS

suppose not.

$wt(f_i) < wt(e_i)$



we know $T^* \cup \{f_i\} \mid \{e_i\}$ $\forall i$ is a spanning tree

f_i ~~is put~~ was not put here.

only becoz it must have created a cycle ~~created cycle~~

~~had been put.~~

$\Downarrow$

$wt(f_i) \geq wt(e_i)$

This we can show for all i

$\Rightarrow wt(T) < wt(T^*)$

$\Rightarrow\Leftarrow$ there is a spanning tree of less wt than that created by Kruskal

Now put f_i in algo it wouldn't have put ~~it~~ because it would have taken ~~it~~ some other smaller edge

put all edges upto f_i

~~edges with~~ ~~edges before~~ ~~wt less than~~ ~~since put~~

Dynamic Programming

Divide conquer	D.P.
★ Subproblems are of similar size	★ Subproblems are very close to original problem.
★ Recursive (top-down)	★ bottom-up

Edit Distance

★ Auto-correct

★ What is the "distance" btw two words?

↓

Edit Distance

↙

Lecture → lecture

- # operations needed to convert one word into another word.

- delete
- insert
- replace

"This distance is Symmetric"

replace ⟹ replace
delete ⟹ insert
insert ⟹ delete

w_2: FLAVORY w_1: FAVORITE

(space up) insert replace delete (space below)

Another perspective (Alignment)

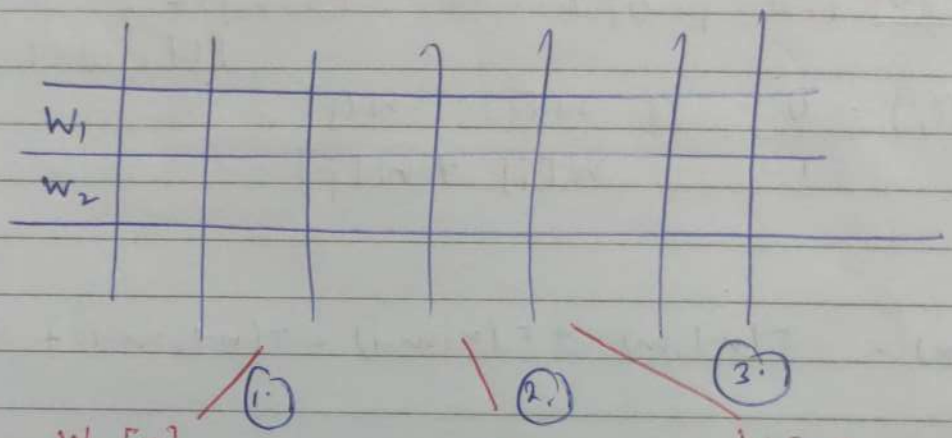
F	_	A	V	O	R	I	T	E
F	L	A	V	O	R	Y	-	-
	①					②	③	

Ex:

Prove : Alignment ⟺ Edit Distance

n chars , m chars.

what can be the last column?



if first option taken

$ED(n,m) = 1 + ED(n-1, m)$

if second option taken.

$ED(n,m) = 1 + ED(n, m-1)$

if third option taken

~~$ED(n,m) =$~~ $w_1[n] = w_2[m]$

$ED(n,m) = ED(n-1, m-1)$

$w_1[n] \neq w_2[m]$

$ED(n,m) = ED(n-1, m-1) + 1$

$W_1[1 \ldots i]$ $W_2[1 \ldots j]$

$ED(i,j)$ $i \geq 0, j \geq 0$

Edit Distance between $W_1[1 \ldots i]$ & $W_2[1 \ldots j]$

$i=0$ or $j=0$ $\Rightarrow$ ϕ string.

$$ED(n,m) = \min \begin{cases} ED(n-1,m)+1 \\ ED(n,m-1)+1 \\ ED(n-1,m-1)+\delta(n,m) \end{cases}$$

Base Case: $i=0, j=0, 1 \ldots m$, $ED(0,j) = j$

$ED(i,0) = i$

$\delta(i,j) = 0$ if $W[i] = W[j]$

$= 1$ $W[i] \neq W[j]$

$$T(n,m) = T(n-1,m) + T(n,m-1) + T(n-1,m-1) + \underbrace{\Theta(1)}_{c}$$

$$\geq 3T(n-1,m-1) + c$$

$$\geq 3^{\min\{n,m\}} + c$$

Huge!

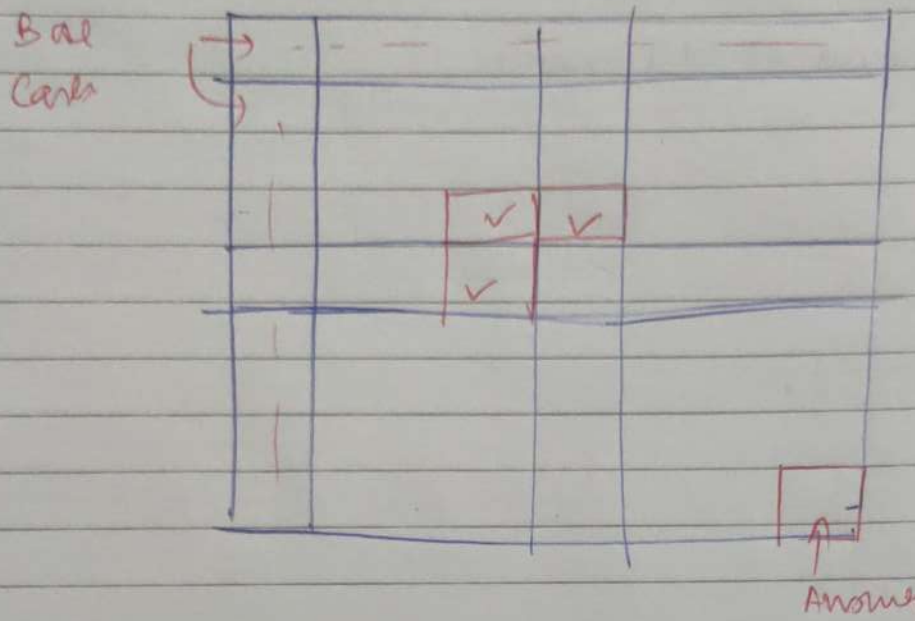
$\downarrow$

so we want to do bottom-up.

Dot

- what are we trying to prove using induction?
- why are we also showing that T_g used is also a ST.
- more thought needed on Kruskal correctness pf too.
- Prove Alignment $\Leftrightarrow$ edit Distance.

Create a matrix for ED



	∅	F	A	V	O	R	I	T	E
∅	0	1	2	3	4	5	6	7	8
F	1	0	1	2	3	4	5	6	7
L	2	1	1	2	3	4	5	6	7
A	3	2	1	2	3	4	5	6	7
V	4	3	2	1	2	3	4	5	6
O	5	4	3	2	1	2	3	4	5
R	6	5	4	3	2	1	2	3	4
Y	7	6	5	4	3	2	1	2	3

Alignment of F with FA.

path mat[i][2] corresponds to

0 → 0 → 1

F A
F —

Final path is here from (0,0)

Que: Recover the best alignment?

F — A V O R I T E
F L A V O R Y — —

Pseudo Code

Procedure EDIT(A, B, n, m)

// n = length(A), m = length(B)
ED ← empty table of size (n+1) × (m+1)

for j = 0 to m+1
    ED(0, j) ← j
for i = 0 to n+1
    ED(i, 0) ← i

for i = 1 to (n+1)
    for j = 1 to m+1
    ED(i, j) = min { ED[i-1][j] + 1,
                     ED[i][j-1] + 1,
                     ED[i-1][j-1] + 1   if A[i] ≠ B[j]
                     ED[i-1][j-1]   o/w.

incorrect indentation.

Ex: Find the alignment.
Hint align_A ↩ n+m empty array.
align_B ↩

if diagonal move write in both o/w right blank in one of them

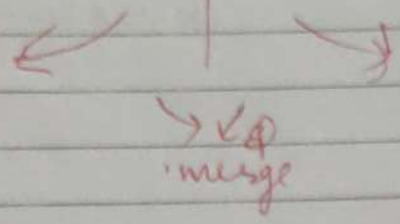
T.C. ⇒ θ(n·m)

vs Divide

Vs Divide & conquer.

FAVORITE ✗

FLAVORY--



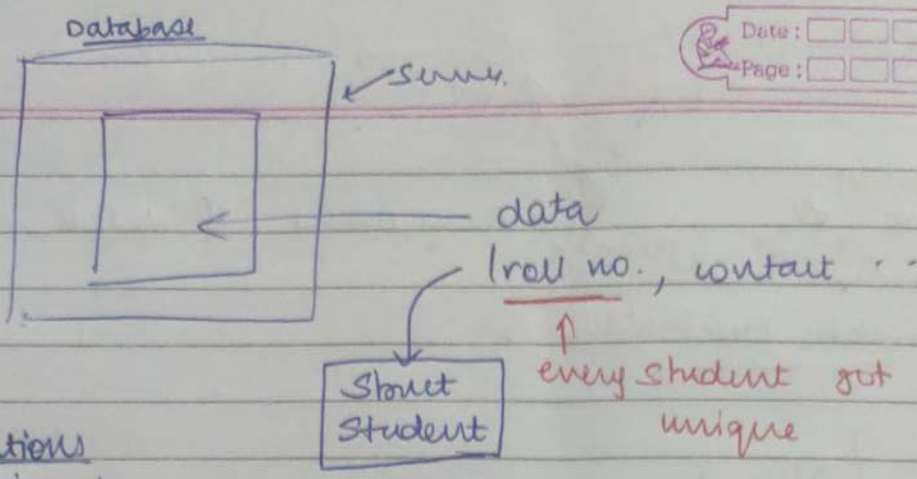
↘ ↙ &
merge

DP ⟺ DP Table.

L[i,j] ← parameterization in the a clever way is the key.

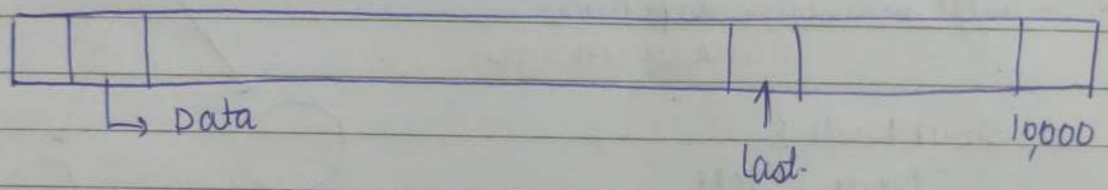
- Defining sub problems is extremely imp. (Ex: take last i & j letters in min edit distance problem)
- Table should not blow up

Binary Search Tree



Operations
- insert
- delete
- search
  (takes roll no. & returns ~~data~~ data)

? Array ?



- insert Arr[last] ← roll no.
  ~~last~~
  last ++.

- search ⇒ O(N) time

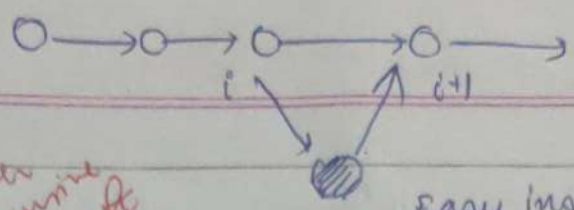
- delete ⇒ may have to shift etc. O(N)

Another idea : ~~Binary~~ Sorted Array !

? Sorted Array ?

Search : O(log n)
Insert : O(n)] shifting.
Delete : O(n)

? Linked list ?



↓ No search in expensive you can't access the middle in LL

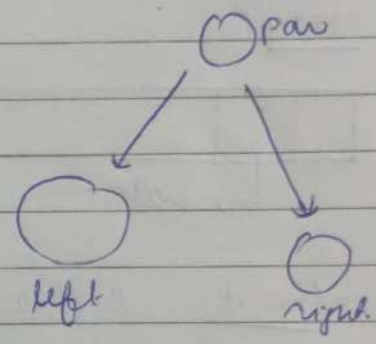
Easy insert & delete.

Binary search Tree (Balancing)

Insert
Delete
search
} O(log n)

19, 431, 572, 72, 1042, 2, 3

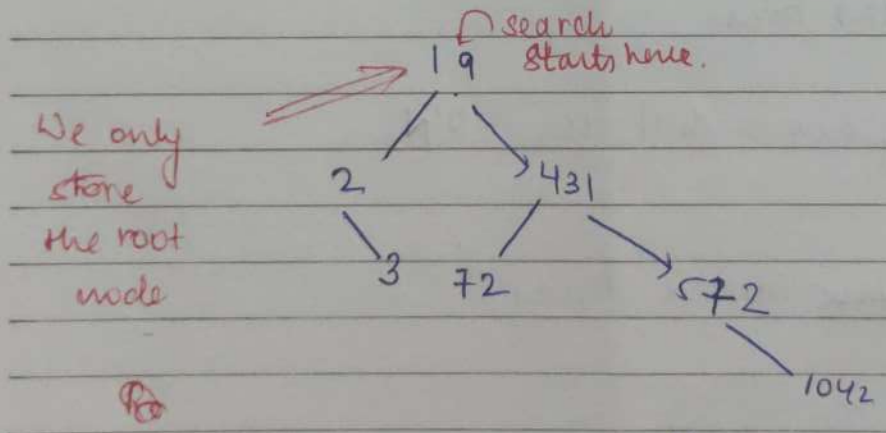
Binary Tree: [Directed]
- will maintain as pointers (Linked list ideal)




```
- struct Node {
      Node *left
      Node *right
      Node *parent
      Int  key
  - }
```

Pseudo code

Node .left
.right
.par
.key



We only store the root node

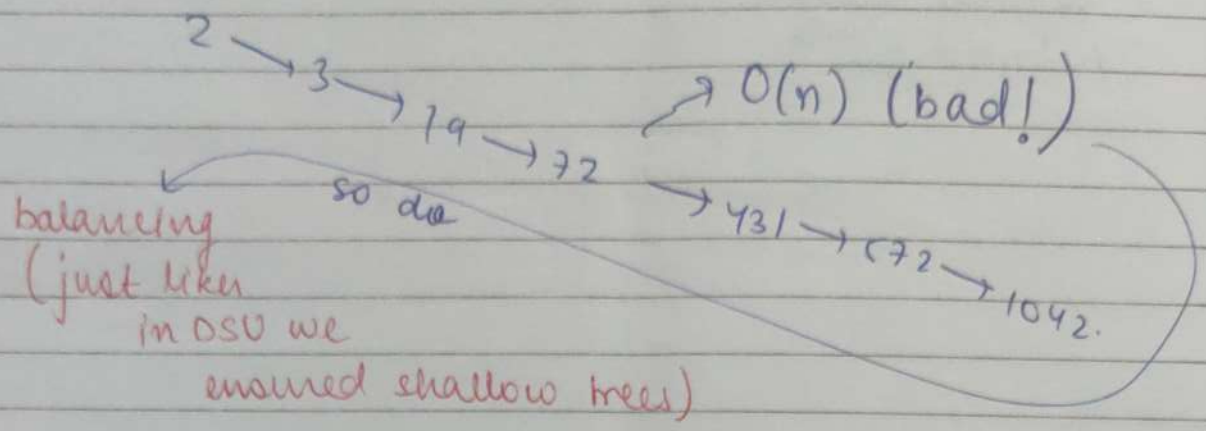
Root Node ≡ Tree

Assumption :- Keys are all diff^w (natural assumption as in databases there are distinct Primary Key)
- total ordering btwn key

Time

Suppose sorted 2, 3, 19, 72, 431, 572, 1042.

(goes to LL.



Procedure Insert

Lec-19

Binary Search Tree — Binary Tree
- Everything in left subtree < root < Everything in right subtree.

Procedure Insert (Node n, Node T)
Root

(Insert)

// n is the new node to be inserted
// into the tree T
if (T = NULL) return n endif.
Temp ← T
while (Temp ≠ NULL) ← always false except at bottom of tree.
    oldTemp ← Temp
    if (Temp.key = n.key) ← node already in tree. n.left = n.right = null return error endif

    else if (Temp.key > n.key)
        Temp ← Temp.~~right~~ left

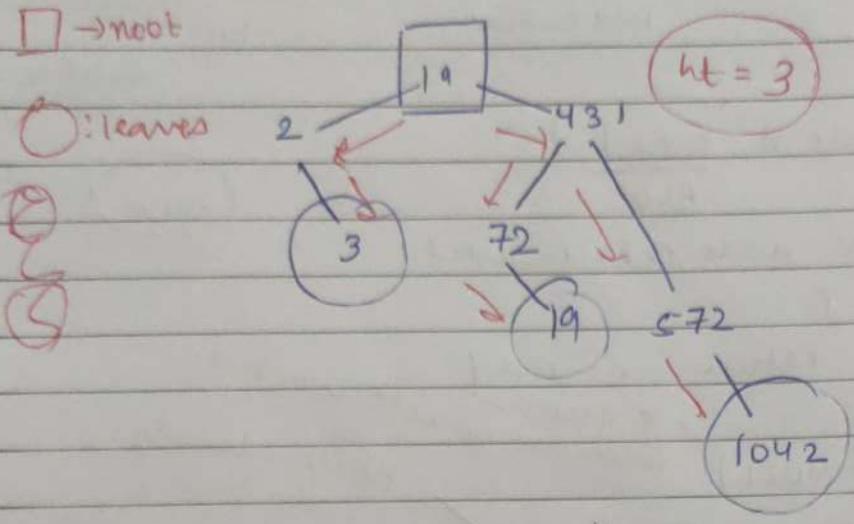
    else
        Temp ← Temp.right

if (oldTemp.key < n.key)
    oldTemp.right ← n
    n.parent ← oldTemp
    n.left ← NULL
    n.right ← NULL.

else if (oldTemp.key > n.key)
    oldTemp.left ← n
    n.parent ← oldTemp.
    n.left ← NULL
    n.right ← NULL

T.C.

ht of tree : maximum # hops of needed to go from root to any leaf of tree.



special of root
↓
parent
=
NULL

speciality of leaf
↓
both children
NULL

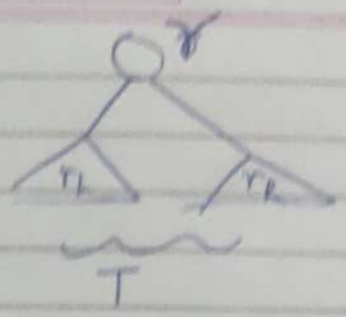
* Procedure search (~~node~~ Key, Node T)
// ~~Find value~~ returns either node
// correspondly to key or null
if (T = NULL) return null endif;

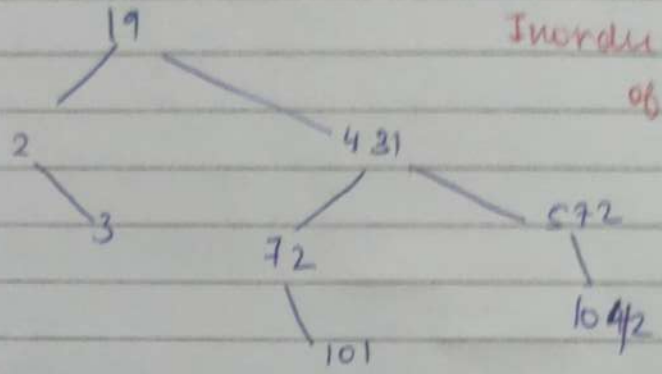
Temp ← T
while (T~~emp~~ ≠ NULL)
if (T~~emp~~.Key = Key)

Tree Traversal

similar to graphs
just that we donot have adj list.

- Inorder - Left Root Right (r_L r r_R)
- Preorder - Root Left Right
- Postorder - Left Right Root





Inorder Traversal of Binary Search Tree is sorted!

↓

Gives us another sorting algo

↓

construct Tree out of data & then inorder

2, 3, 19, 72, 101, 431, 572, 1042

19, 2, 3, 431, 72, 101, 572, 1042

3, 2, 101, 72, 1042, 572, 431, 19 ⇐ Post (check)

Procedure Inorder(Tree T)
if (T = NULL) return endif
Inorder (T. left)
// visit root
Print (T. Key)
Inorder (T. right)

Correctness ?

↓

Induction

↓

by IH the r_L & r_R will be inorder

~~If ⇒~~

Time ⇒ O(n)

with Pre & Post

smallest & largest can be called at any subtree node also.

Procedure smallest (Node T)
// returns smallest node (left most)
if (T = NULL) return NULL ~~else~~ endif

~~O(ht)~~

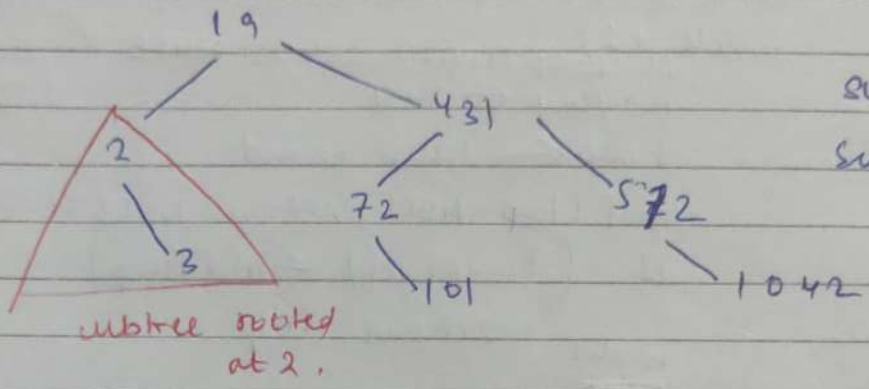
→ O(H)
H : ht.

procedure largest (Node T)
// returns largest node (which is right most)

Successor & Predecessor (key)

Guaranteed to exist.

Successor (key): returns the node whose key is immediately next to "key" in the inorder traversal/ sorted order.



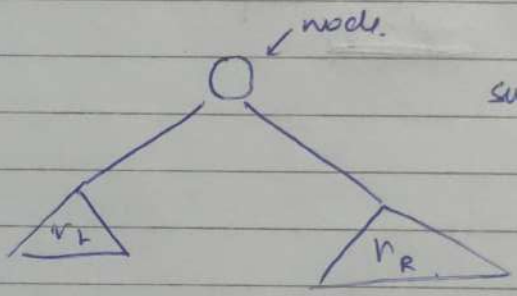
subtree rooted at 2.

Succ(19) = 72

Succ(3) = 19

How do we find?

Simpler case



succ(root) = smallest(r_R)

How to find where rooted in where 2 is largest

move parent → parent → parent (till right child)

↓

first time left child is parent

Procedure Successor (Node n)

// returns successor node of n.

if (n=NULL) return error

elseif (n.right ≠ NULL)
return Smallest(n.right)

else

temp ← n

while (1)

old Temp ← temp.

temp ← temp.parent.

if (temp = NULL) return NULL endif.

if (temp.left = oldTemp)
return tmp.

endif

endwhile.

endif

If n is largest node (i.e., no successor)

Time
↳ O(H)
(∵ Inorder is worse t.c.)

Procedure Predecessor (Node n)

// returns node immediately before n

// in the inorder traversal

After deletion BST property must be valid.
we can either do with succ. or predecessor
(in class)

Date:
Page:

return value

Procedure Delete (Node n)

Rune

// replace with successor

Idea/picture & break into subideas/cases

if (n = NULL) return error endif
if (n.right = NULL & n.left = null)
~~n ← NULL~~
~~return~~
if (n.parent = NULL)
return NULL
else
temp ← n.parent
if (n = temp.left)
temp.left ← NULL
if (n = temp.right)
temp.right ← NULL

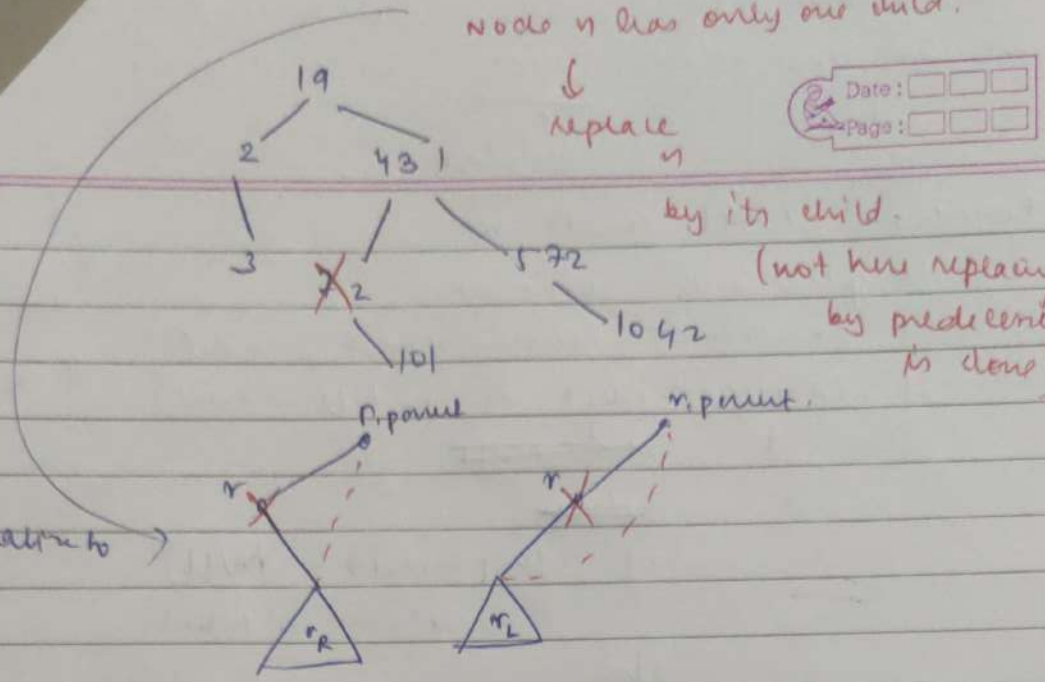
what to return?

else if (n.left = NULL)
// replace n with n.left
n_parent ← n.parent
n_right ← n.right
n.right.parent ← n.parent.
if (n = n_parent.left)
n_parent.left ← n_right
else
~~n_parent.right ← n_right parent~~
else n_parent.right ← n_right endif.

"duck."

else if (n.right = NULL)
n_parent ← n.parent
n_left ← n.left.
n_left.parent ← n_parent
if (n_parent.left = n)
n_parent.left ← n_left.
else
n_parent.right ← n_right

Node n has only one child.



↓ replace n by its child.

(not here replacing by predecessor as done)

Generalize to

else

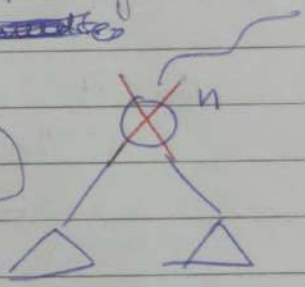
// n has got both left & right child.
// successor exists in right subtree
// return that

m ← successor(n)

// write if to check no case.

m.left ← n.left
n.left.parent ← m (m.parent ← NULL)
if (m.parent = NULL) return
check successor
if (n = n.parent.left)
n.parent.left ← m
else n.parent.right ← m
m.parent ← n.parent

if (n.right.left = NULL)



Replace n with successor

↓

2 cases

↓

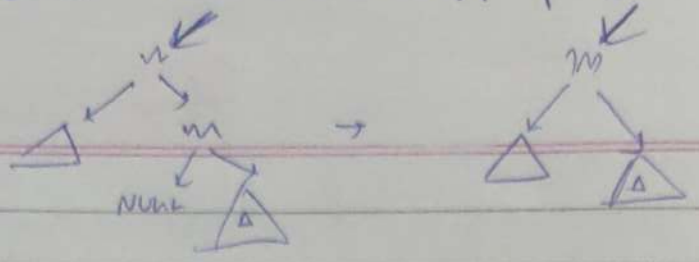
successor is left or right child only

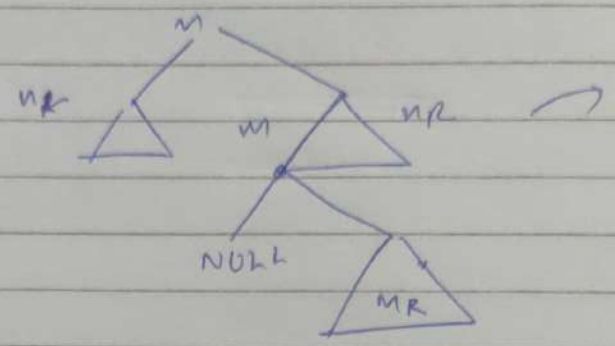
if successor is right child of n itself

m.left ← n.left n.left.parent ← m
m.right.parent ← m.parent
m.parent.(left) ← m.right

m.right ← n.right
n.right.parent ← m (m.left ← n.left)?
if (n.parent = NULL) return
m.parent ← n.parent
m.parent.(left/right) ← m

Dbt :- Frame Kruskal pf sf set checked by Sir's.



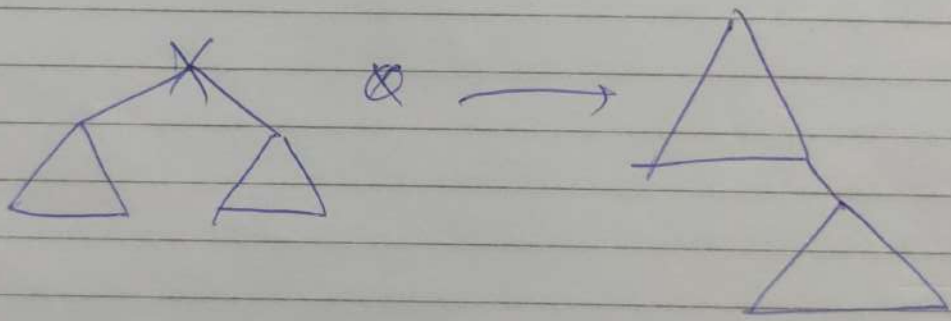


$n_L < n < n_R, m; m_R$

$m < n_R \mid m$

$m_R > m$

Ex: Check BST ordering preserved.



skewed ht
↳ Not good for M

Ex: Do with predecessor

Ex: Delete will reduce the ht by at most one.

All these 3 operations ⟶ $\log H$.

Balancing

lec-20

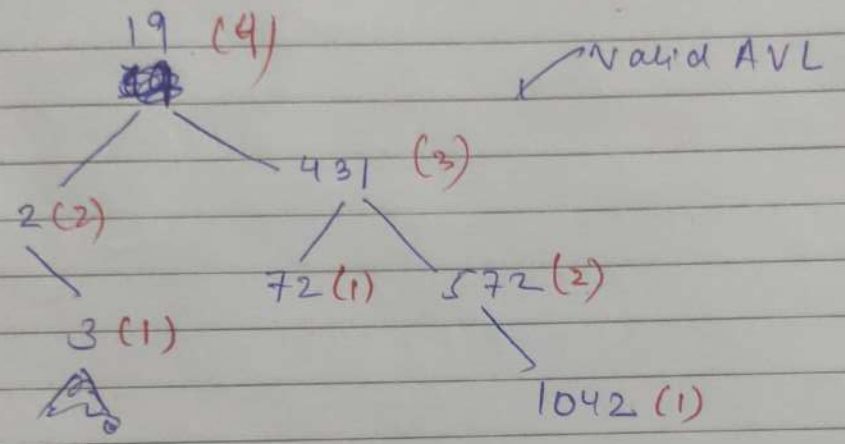
Tree Rotations & AVL Trees

- Improvement to BST
- ht is small.

Defⁿ (Ht of BST)
↳ define recursively.

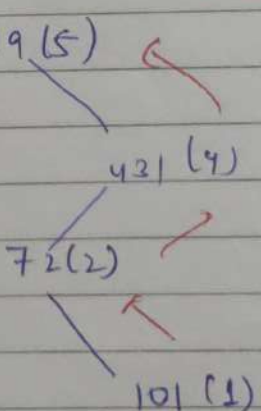
ht (NULL) = 0

ht (parent) = 1 + max { ht (left-child) + ht (right-child) }



* within node we'll maintain a ht field.

Insert & Updation



Since only one parent of we only need to update ht in His parent parent path.

```
start Node {
        key
              left
              right
              parent
        }      ht
```

ht code

Check

```
temp ← node n
while (temp ≠ NULL){
        temp.ht = 1+max{ ht(left-child), ht(right-child)}
        temp ← temp.parent.
}
```

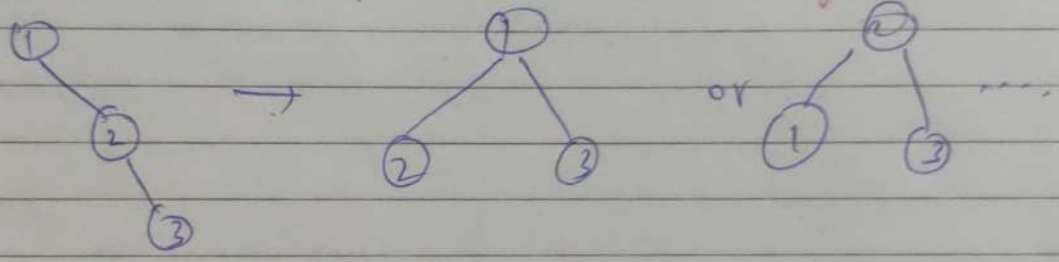
Defⁿ [AVL Tree] - A BST with the following additional property.

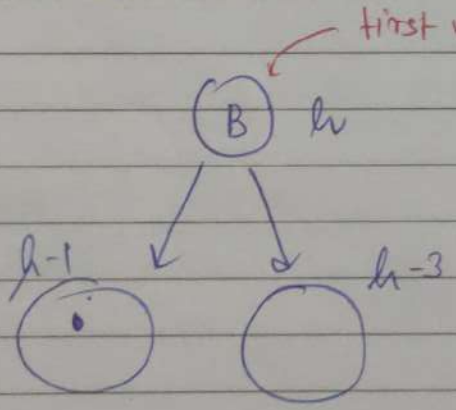
$$|height(node.left) - height(node.right)| \leq 1$$

$\forall$ node $\in$ Tree.

We can detect of AVL property violation while updating height. (check)

Restore AVL Property (Rotation / Balancing)

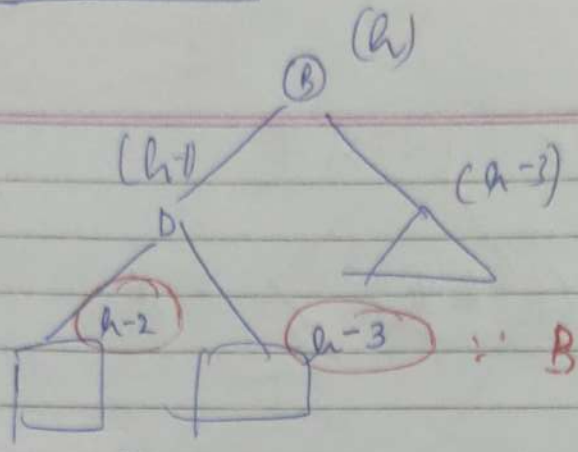




hts differ by = 2

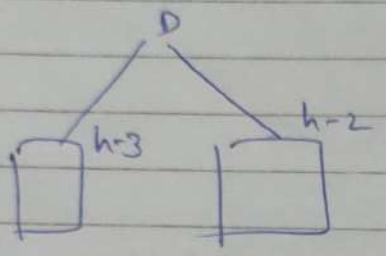
(∵ inserted only one node)

Open up :



∴ B is first violating node.

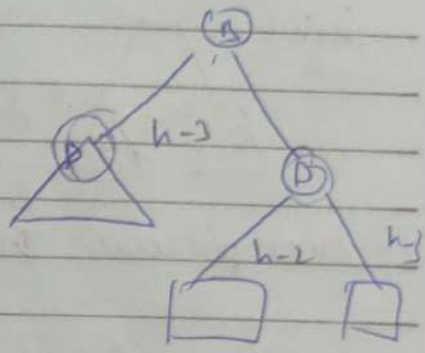
or



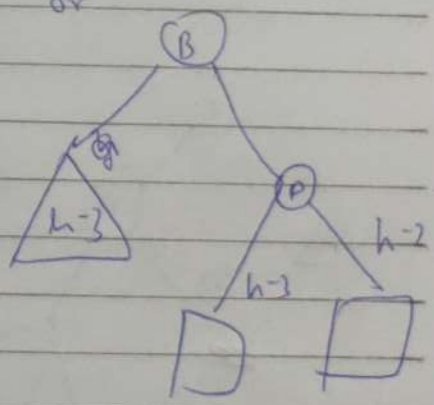
or

MS

Mirror Image (MI)



or

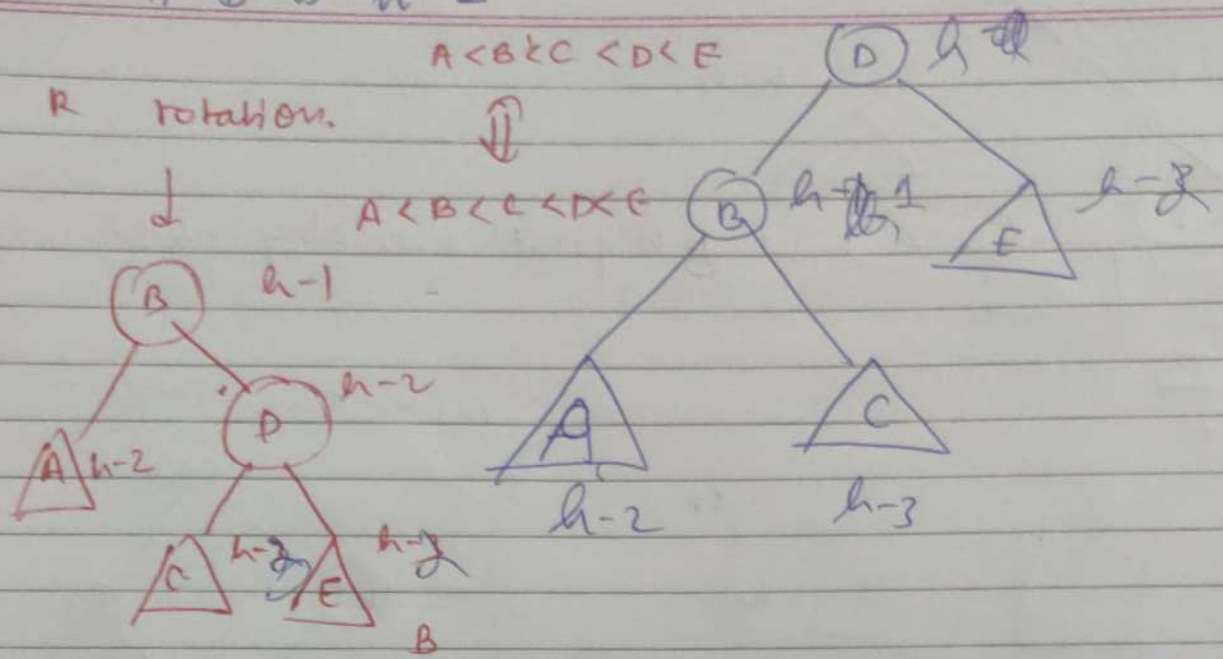


⇒ 2 cases of this is M.I.

Case - I

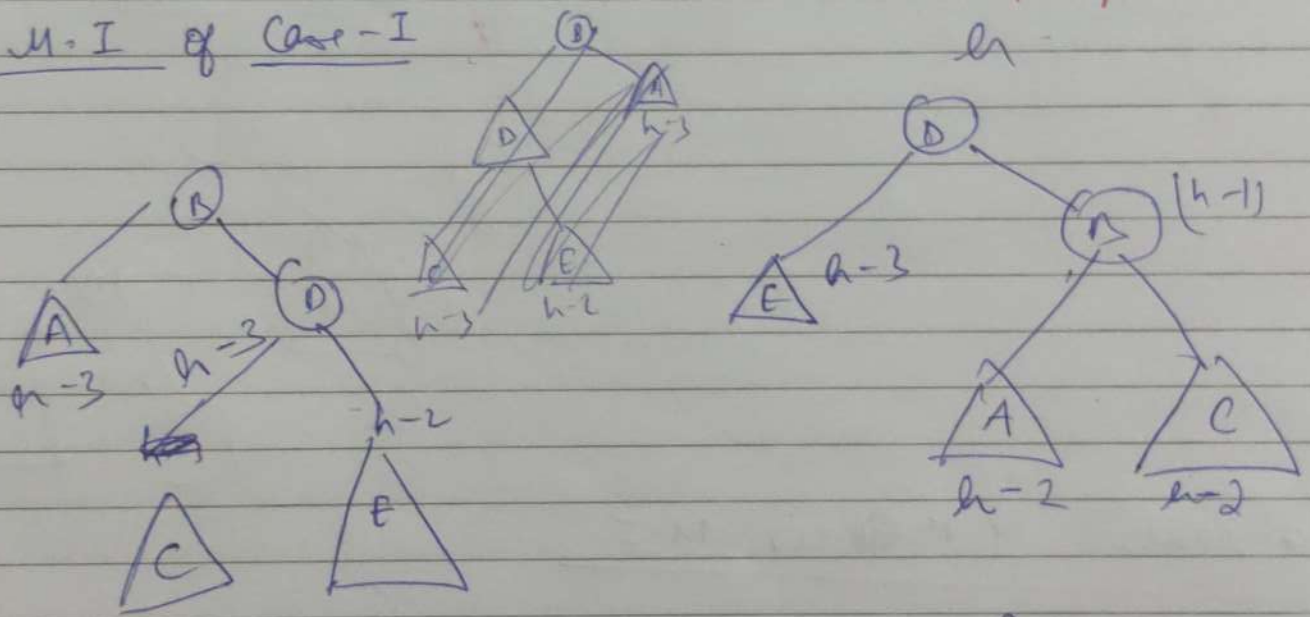
// left of D is h-1

// D is h-1



- shift preserves BST prop. & restores AVL prop.

M.I of Case - I



Ex: check preserves BST & AVL

Ex: Pseudo code.

~~Rotation decreases ht by~~

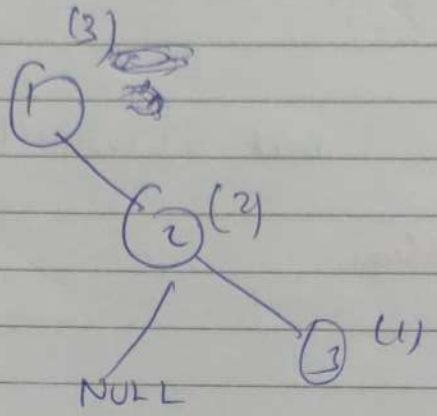
Remark: Insert followed by insertion will never ~~require~~ increase ht.

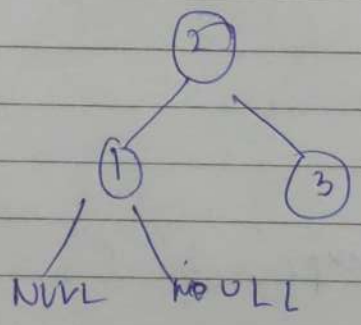
Date :
Page :

ht.

will it never inc? No inc. when perform rotation only.

$h = 3$

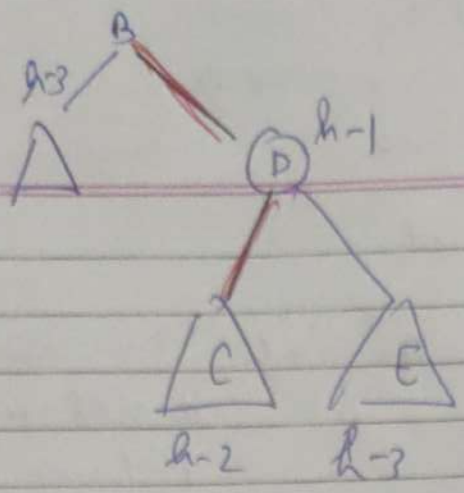




start with D & move to next ht child.
greater h greater ht of B one aligned
If child & grandchild, when align
↓
Case-1

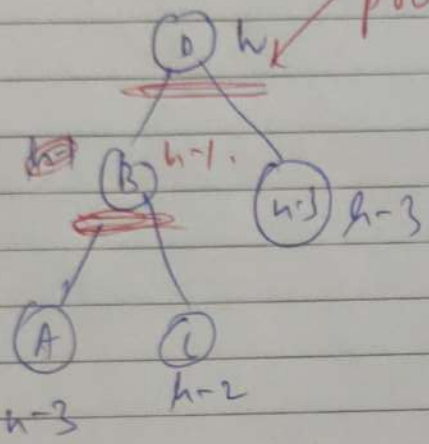
Check

Case-3

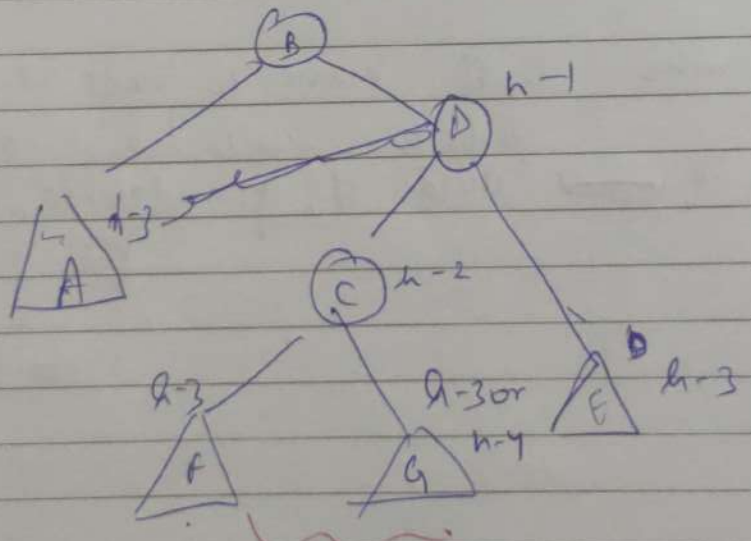


B D E
don't align

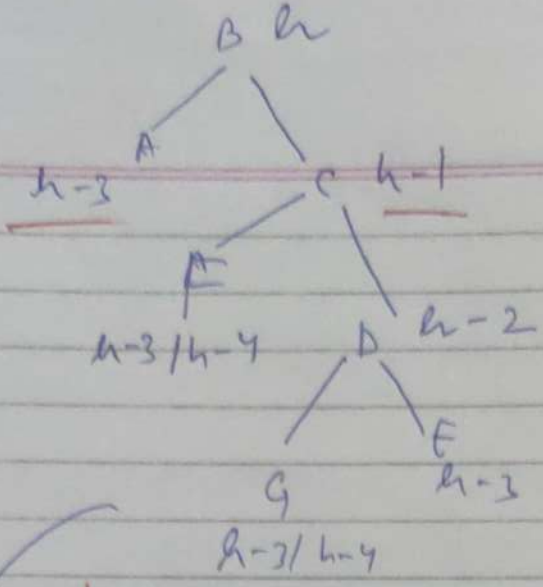
We can try rotation but it won't work.



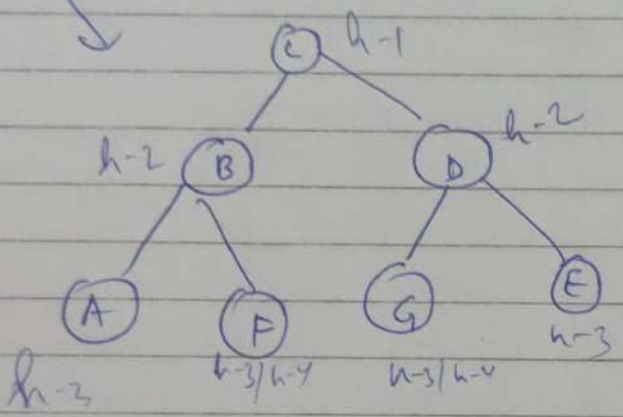
Do double rotation of expand C.



at least one is
h-3



one more rotation ~~rotate~~ rotate C & D



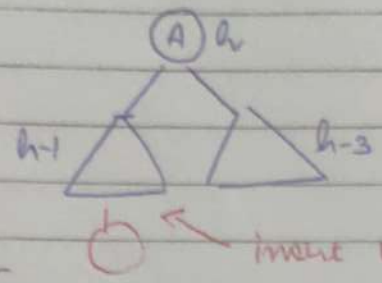
check that it solves the problem.

Plot - Balancing

(Intuition, 2 rotations?)

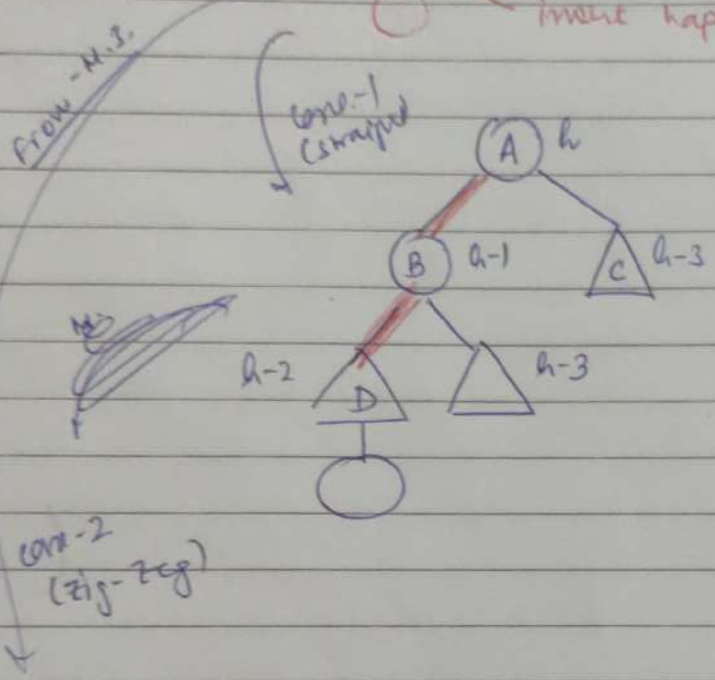
h-2| ~~Balancing~~

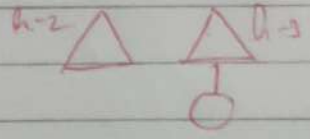
Balancing (Recap)



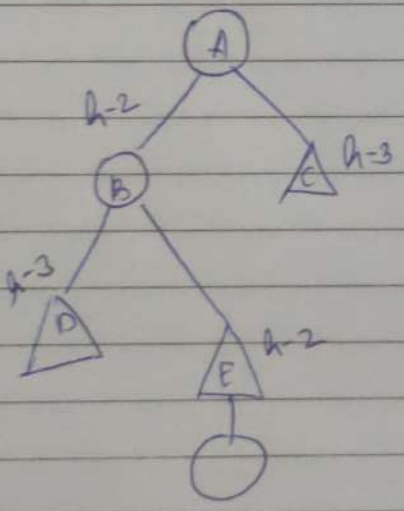
(The M. image case is similar)

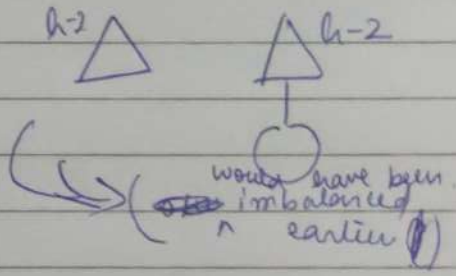
insert happened here.



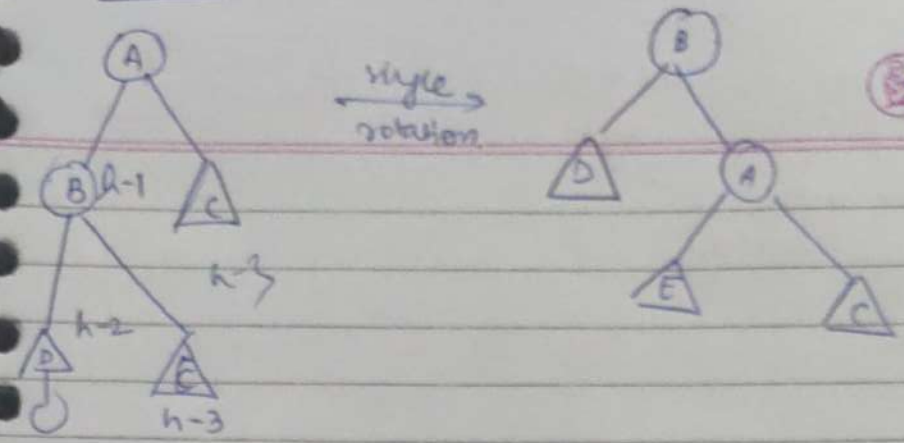


not possible w/o it would have been imbalanced.

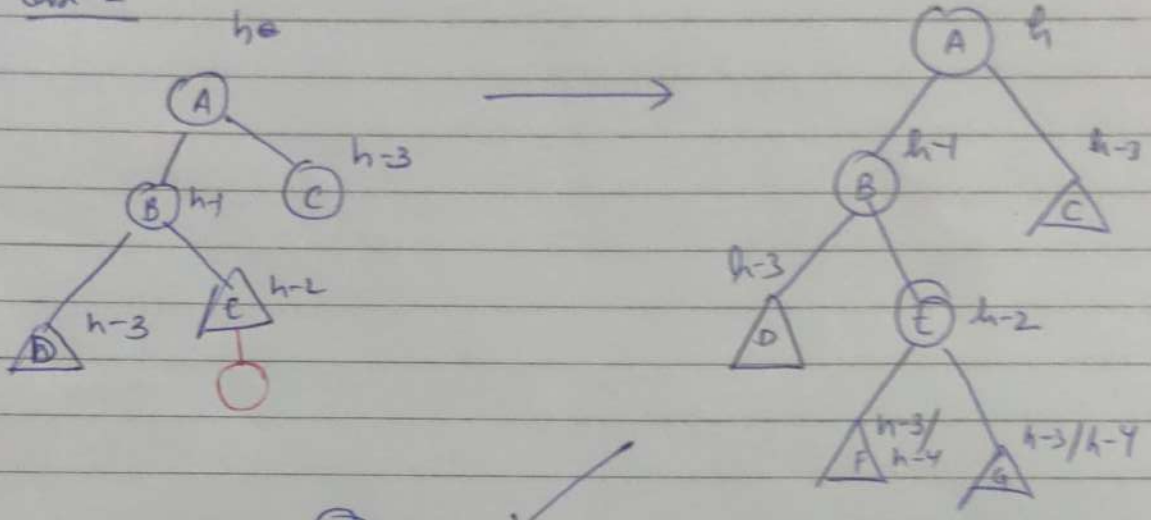




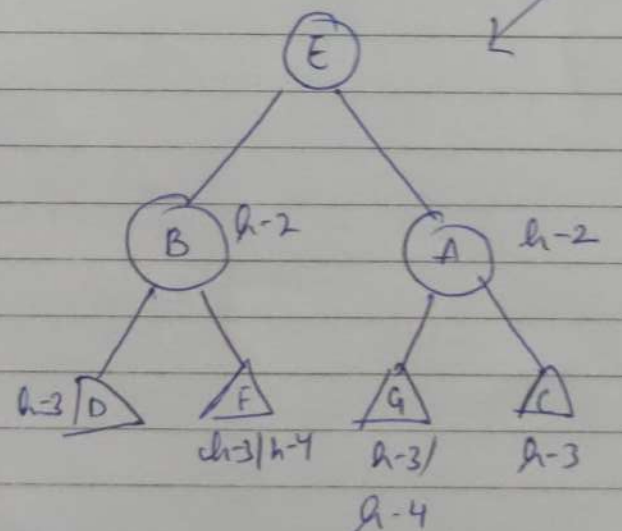
Case-1



Case-2



at least one is (h-3)



routines h remain AVL tree

Ex: Do for mirror images
- Write Pseudo Code

?

Time ⇒ O(1) + O(h)
= O(h)

For AVL tree h is not too lay.

AVL – Balanced tree

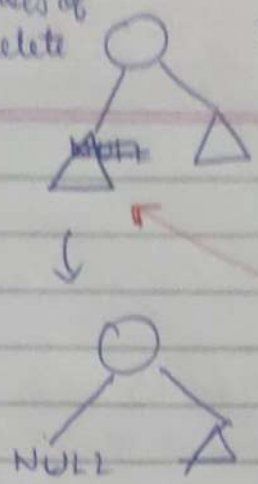
Sorting Algorithm (AVL sort)

$\underbrace{n \cdot O(h)}_{\text{insertion}} + \underbrace{O(n)}_{\text{in order traversal}} = O(h \log n)$

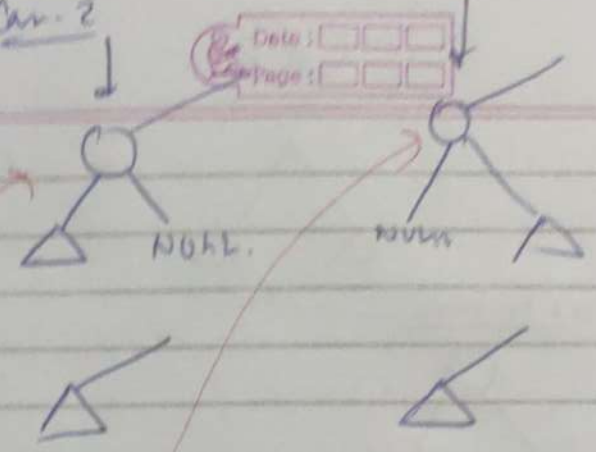
Restoring balance after delete

Cases of delete

Case -1



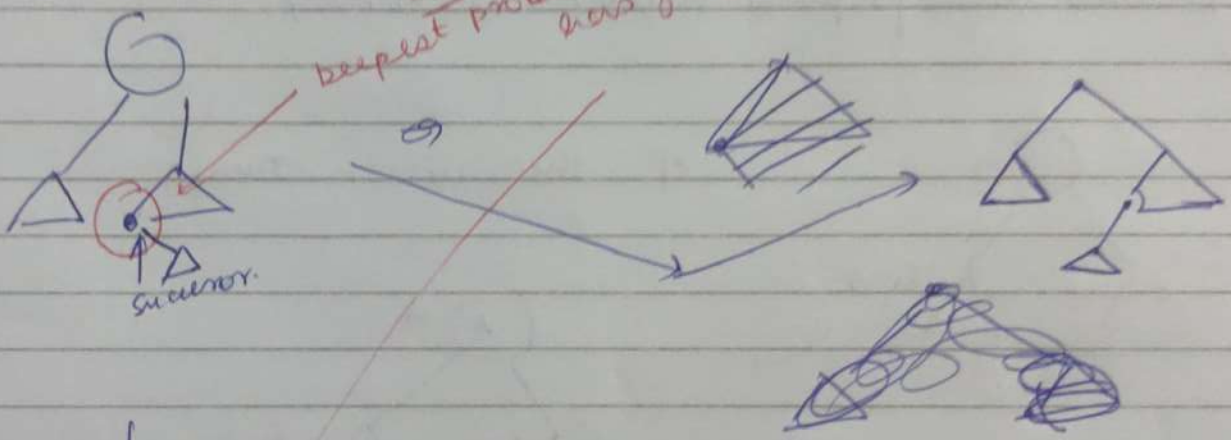
Case -2

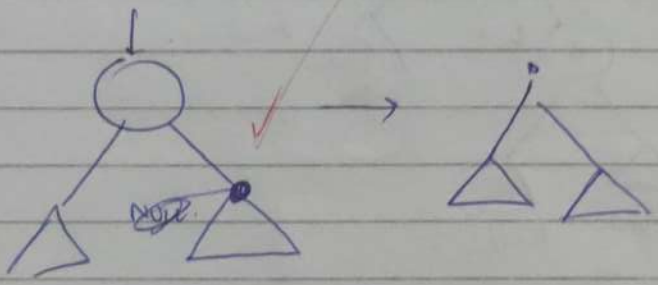


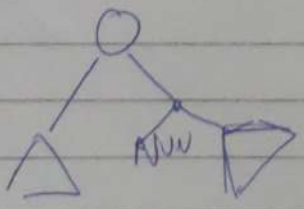
Case -3

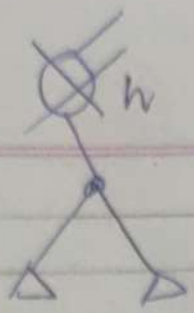
Case -4

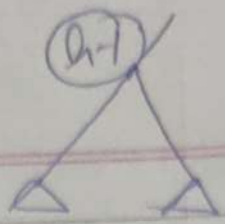
observe: deepest problem site has got only one child or no child





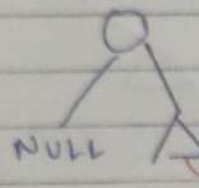






Date :
Page :

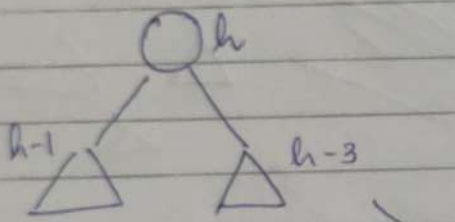
AVL Property



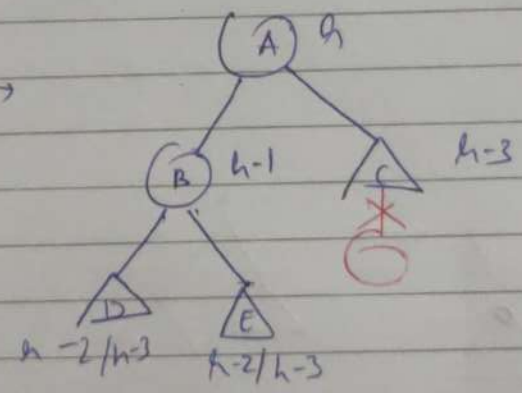
∴ Observe: location of first problem is either leaf or parent of a leaf.

We'll do like insert, go parent → parent → parent (up)

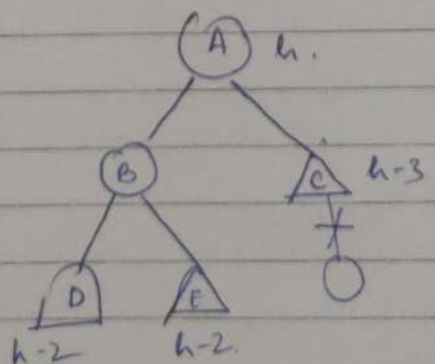
If where AVL property violated : DO something.



If the Mirror Image

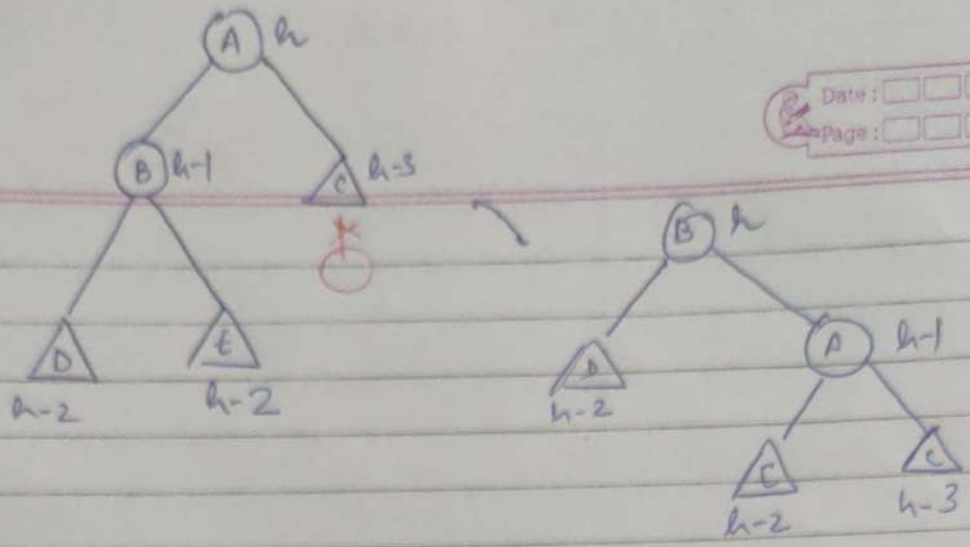


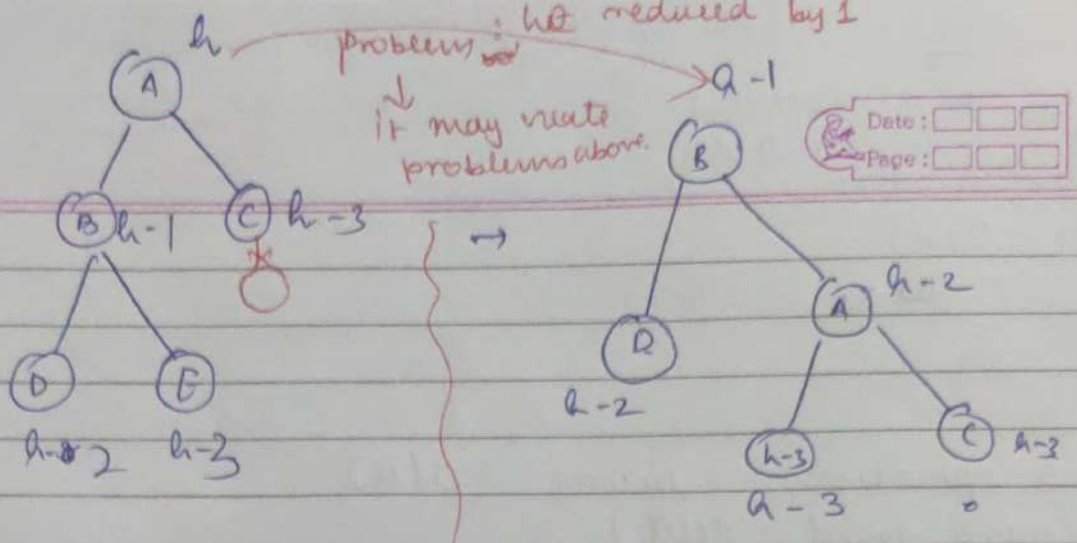
For delete we have another case.



Case-1 & Case-2 on line insert

Case-3





Remedy : go up (to parent) & check. if everything is alright if not fix it.

Is one level check enough ? (yeah ig)

O(H)

while ( )

time complexity:

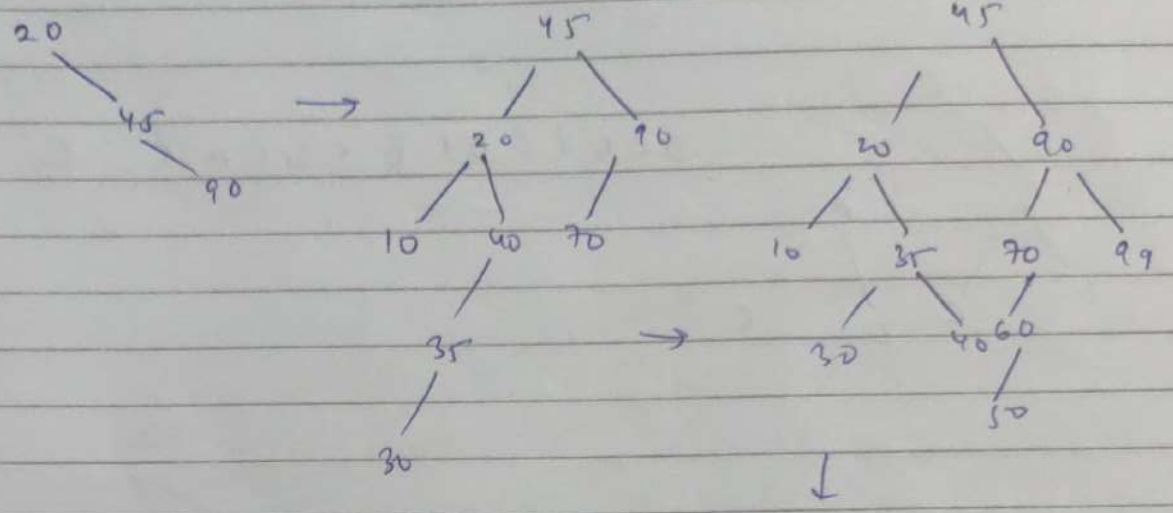
delete + rotations

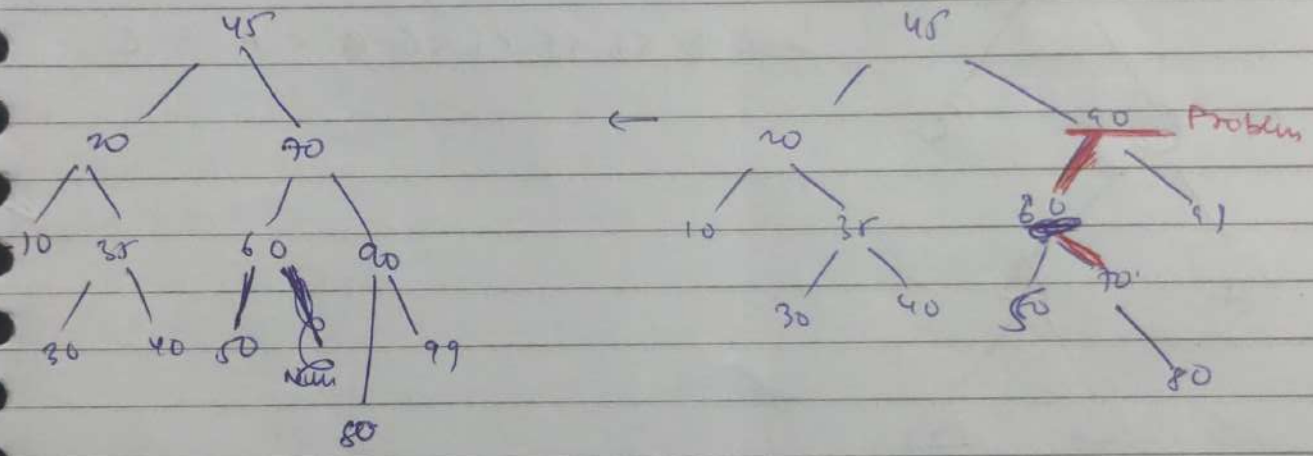
O(1) O(1) O(h)

= O(h)

∴ All 3 operations become O(h)
(search, insert, delete)

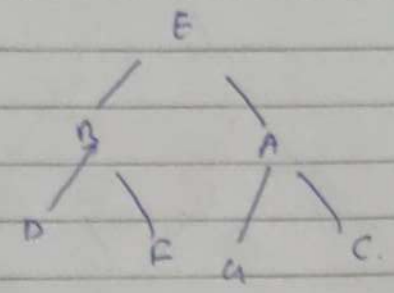
Example: 20, 45, 90, 70, 10, 40, 35, 30, 99, 60, 50, 80



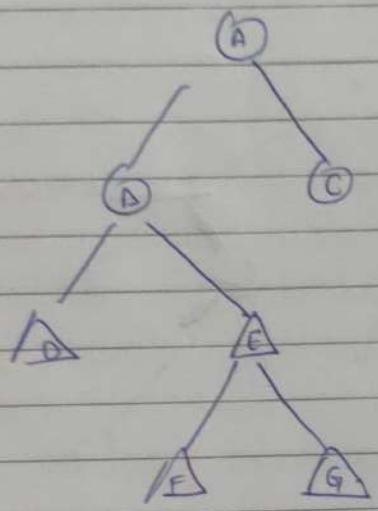


~~How can~~

How can we be sure that Balancing doesn't change the Min ~~Binary~~ Heap ~~Free~~ pty

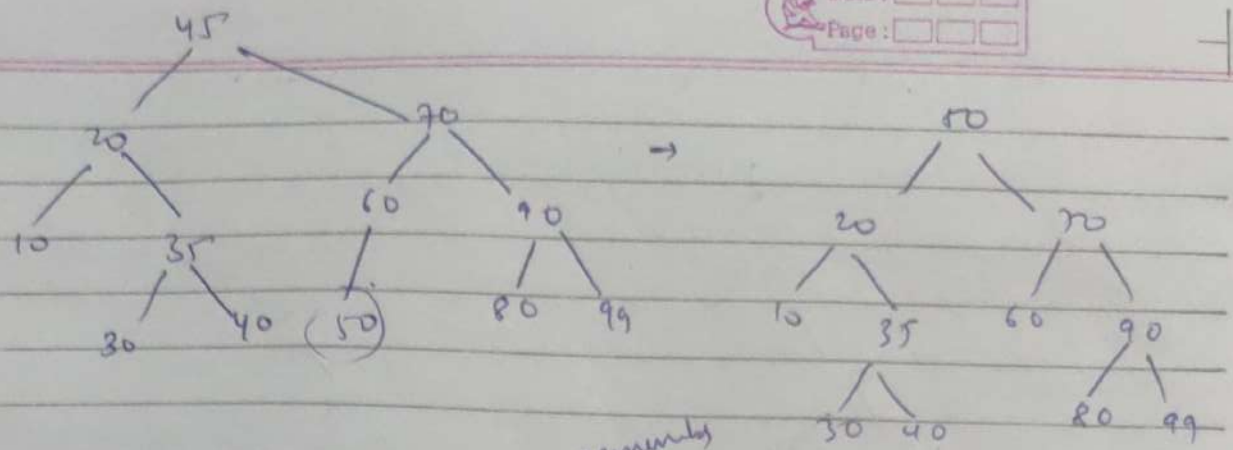


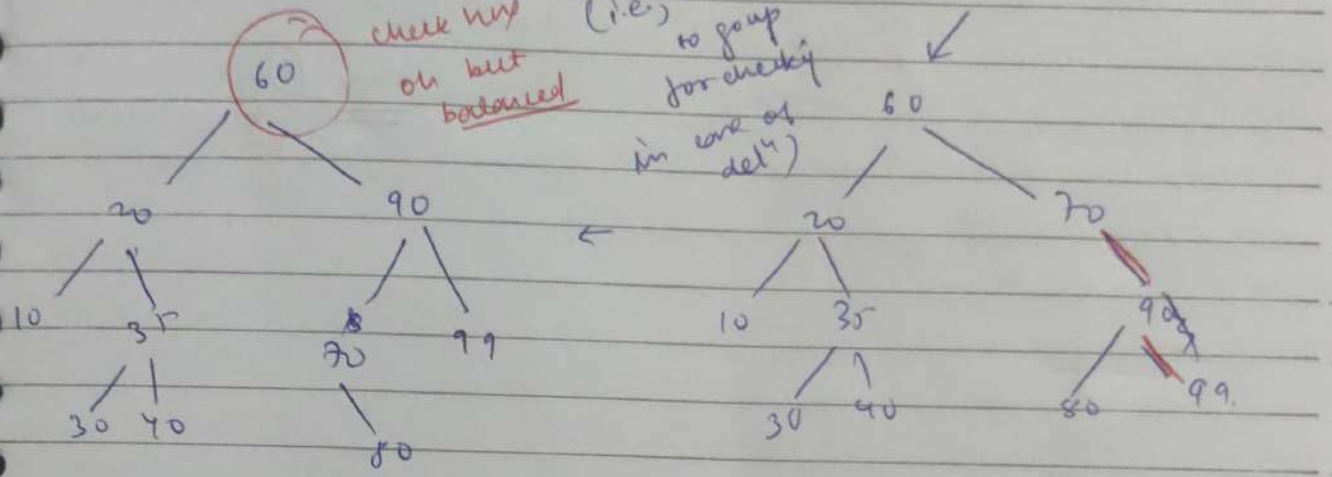
D < B < F < E < G < A < C

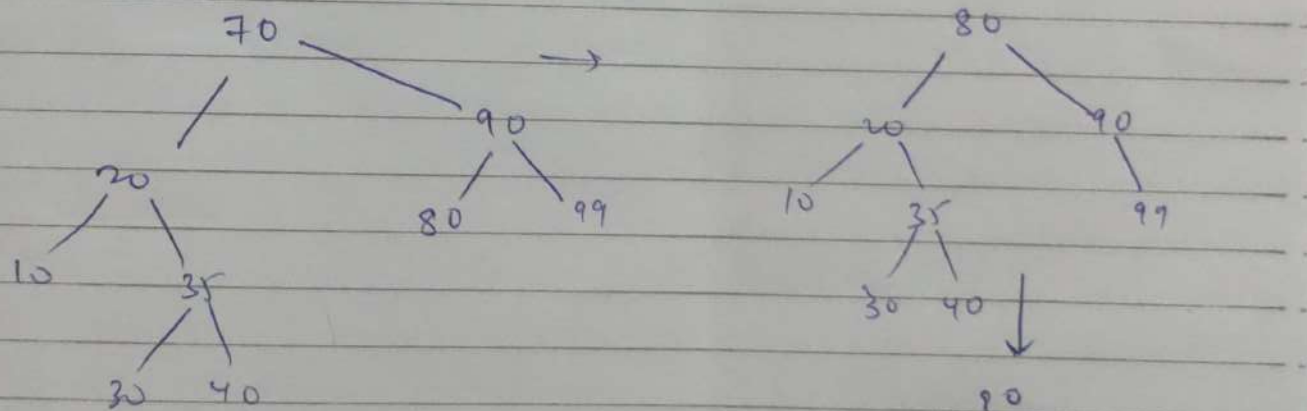


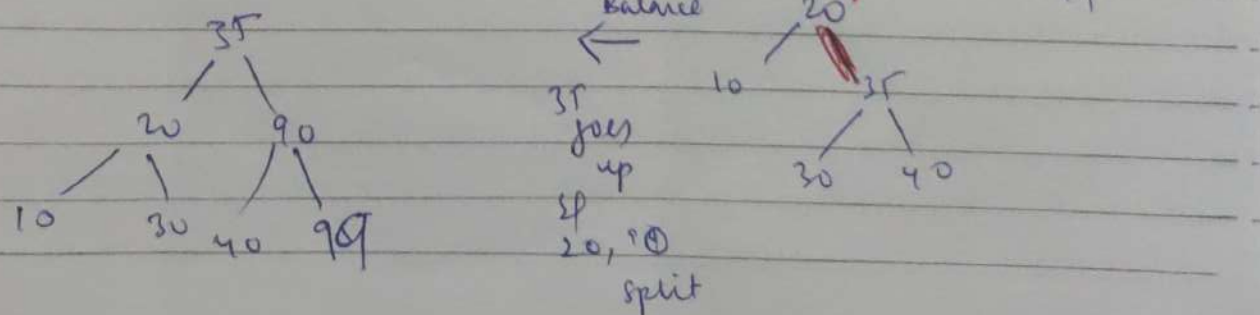
~~D~~ D < B < F < E < G < A < C

Keep deleting root

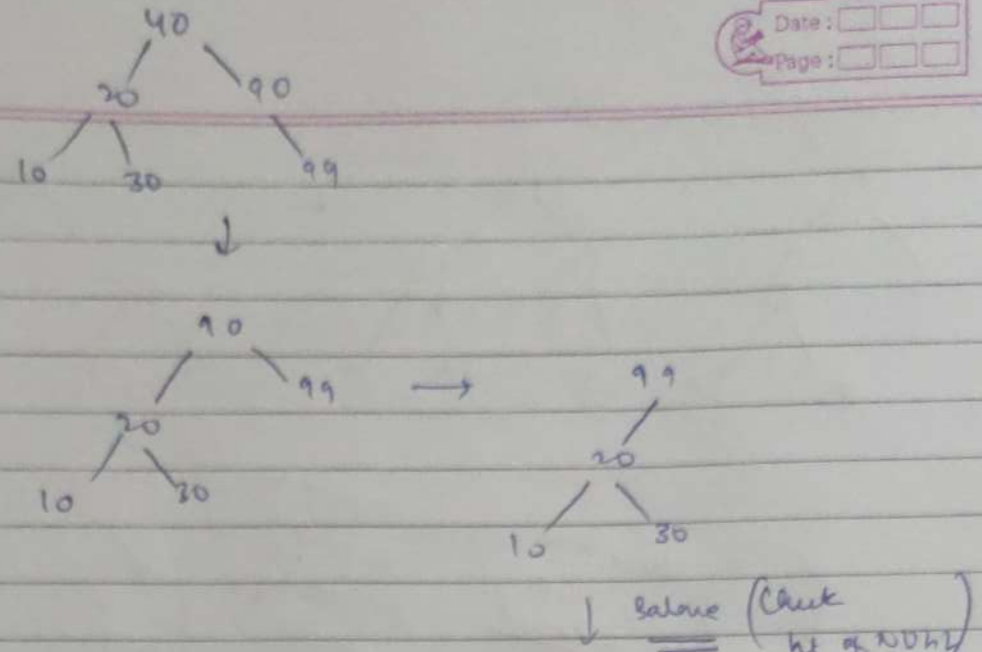








Post: exercise



↓ Balance (Check ht of NULL)

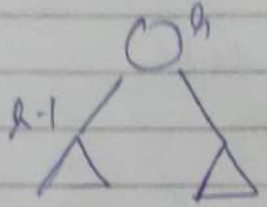
Claim: If T has n nodes, $h = O(\log n)$

Pf: (equivalent to showing) If AVL has ht h

$\Rightarrow \geq 2^{\Theta(h)}$ nodes.

$n(h)$ = min. # of nodes in ht h AVL tree

(We will use contradiction,



(worst case & (min. no. of nodes))

IH: $n(1) \ldots n(h-1)$

$n(h) \geq n(h-1) + n(h-2) + 1$

$n(h) > n(h-1) + n(h-2)$

} equal.

$n(0) = 1$

$n(1) = 1$

$n(h) = n(h-1) + n(h-2)$

$\Theta(h)$

$n(h) \geq 2$

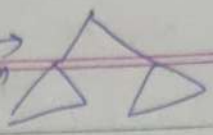
$\therefore$ Has $\log n$ ~~ht~~ contradiction.

$h = O(\log n)$

Doubts

-> How problem at leaf/parent of leaf? in case where

-> on deletion



-> Think what Case-3 delete

-> If at next step no new entries entered (??)

-> ~~from AVL~~

-> Delete If a parent ~~from AVL~~ satisfies AVL do we need to check grandpa.

but insert also h-1

-> worst case is Insert example last part the example of

-> Insert and delete pseudo code.

Date :
Page :

lec - 22

Matrix multiplication

A × B × C

10 × 5

check

$A_1 \times A_2 \ldots\ldots A_n$

$m_0 \times m_1 \quad m_1 \times m_2 \quad m_{n-1} \times m_n$

Minimise the total number of multiplic".

what paradigm to follow?

- Always first try greedy
  i) perform the cheapest consecutive pair first.
  (first example)

- divide & conquer
  (not clear where to divide to ~~~~ get half sizes)

- DP(Hammer) ! DP Table?
  what is the good way to define subproblems?

First see how many options.?

T_n : # ways we can form?

$(n-1)T(n-1)$ $T_n =$ ~~$(n-1) \times T_{n-1}$~~

Problem is equivalent to dividing into 2 parts.

$(A_1 \times A_2 \times \ldots A_i) \times (A_{i+1} \ldots \times A_n)$

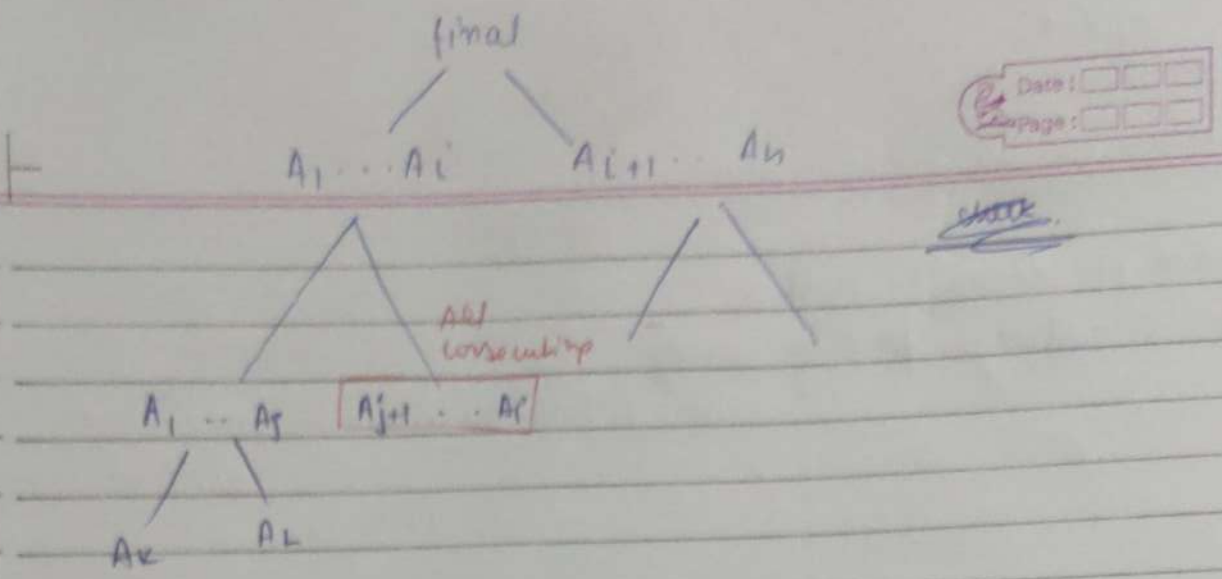
$$T_n = \sum_{i=1}^{n-1} T_i \cdot T_{n-i}$$ ← catalan number

$$= 2^{\Theta(n)}$$

Since can't try all ways (simple recursion doesn't work)

↓

DP.



$M[i,j] \leftarrow$ min # of operations needed to compute $A_i \times \cdots \times A_j$ $\quad j \geq i$

$M[i,i] \leftarrow 0 \quad \forall i$

$M[1,n] \leftarrow$ final answer.

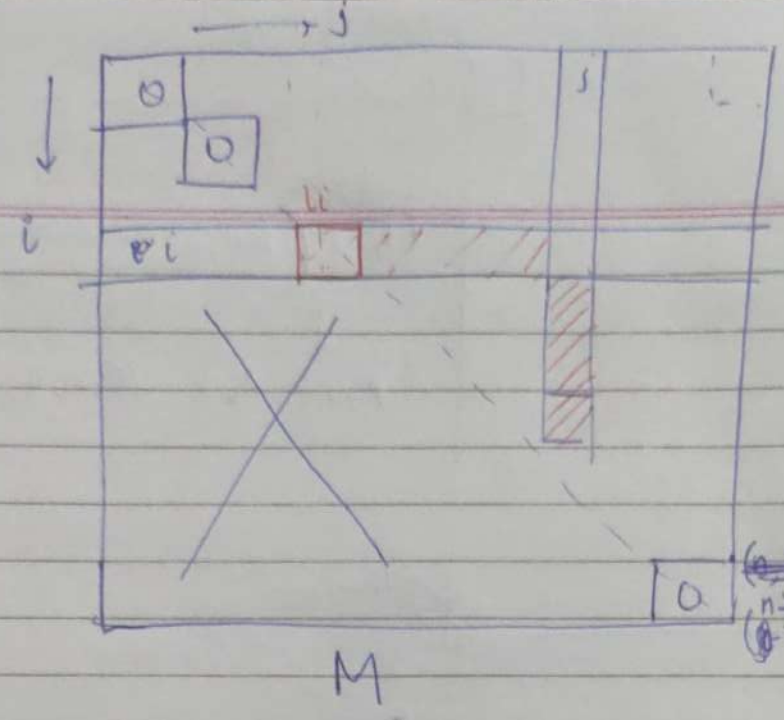
$M[i,j] \leftarrow$ $\quad i<j$

$\min_{i \leq k < j}$ $\{ M[i,k] + M[k+1,j] + m_{i-1} \times m_k \times m_j \}$

$A_i \cdots A_j$

$A_i \cdots A_k \quad A_{k+1} \cdots A_j$

A_i is of dim $m_{i-1} \times m_i$



i<j only makes sense.

so go diagonal by diagonal.

jth diagonal

(1, j)
(2, j+1)
⋮
(n-j+1, n)

Procedure MinMult ($m_0, \ldots m_n$)

for j = 1 to n

for i = 1 to n-j+1

if (j=1) M[i,i] ← 0

else

$M[i, i+j-1] = \min_{i \le k < i+j-1}$ { M[i,k] + M[k+1, i+j-1] + $m_{i-1} m_k m_{i+j-1}$ }

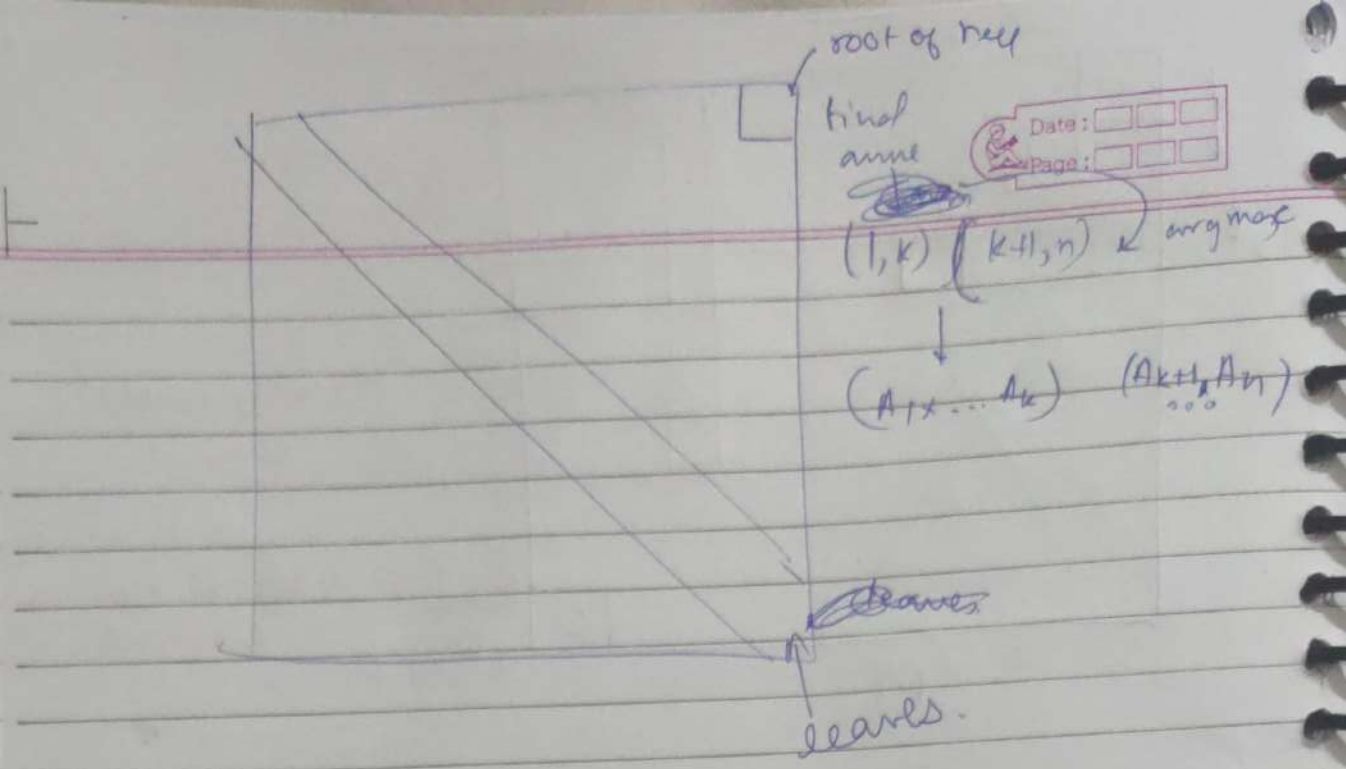
th entry of jth diagonal.

A_{i+j-1}

A_i ... A_{k+1}

Both terms are already computed.

How to recover the path ? $k^* = \operatorname{argmin}[i, i+j-1]$



DP problem : Parametrization.

All Pairs Shortest Path

} was not in Dijkstra

(1.) Neg. edge wts ✗ allowed
(2.) Source vertex is all possible
(3.) No neg. edge cycle.

fixed in Dijkstra

Suppose (1.) is not enforced, do Dijkstra for all vertices.

↓

$n \times O((m+n)\log n)$

↓

$O((m+n) \cdot \log n \cdot n)$

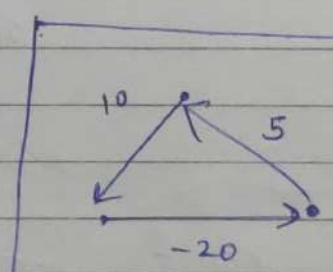
Algo we will see → $O(n^3)$

+ ~~-ve~~ -ve edge wts allowed.

(check Pf of Dijkstra needs +ve edge wts)

(✗ any negative wt cycle)

but



problem is ill-defined
$-\infty$ cost.

$m \approx n^2$ (Dense graph)

$\log n$ is small no.

Algorithm will also tell if there is a neg. edge wt cycle
reason out why then.

Again see Paradigms
We'll do DP.

we need to parametrize

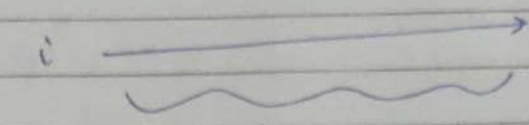
vertices: $\{1, 2 \ldots n\}$

dist (i, j)

↓

shortest path distance for vertex i to vertex j

(we'll take ~~undirected~~, will work for ~~direct~~ undirected too)



restrict. intermediate vertices.

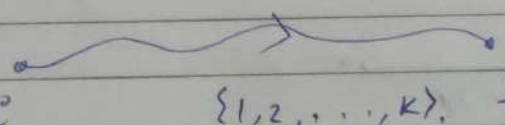
$\{0, 1, 2 \ldots k\}$ ← shortest path with 1 .... k intermediate vertices.

~~no intermediate vertices in shortest~~

shortest path with 0 intermediate vertices.

$dist(i, j, 0) \leftarrow wt(i, j)$ if $(i, j) \in E$ (i is parent, j is child)

$\leftarrow \infty$ o/w

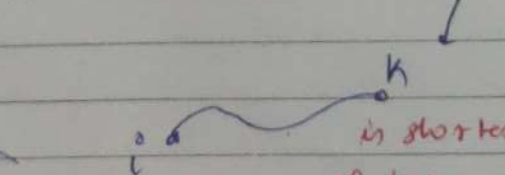


$dist(i, j, k) \leftarrow f(dist(i, j, k-1))$

either k is being used

k is not being used.

$dist(i, k, k-1) + dist(k, j, k-1)$

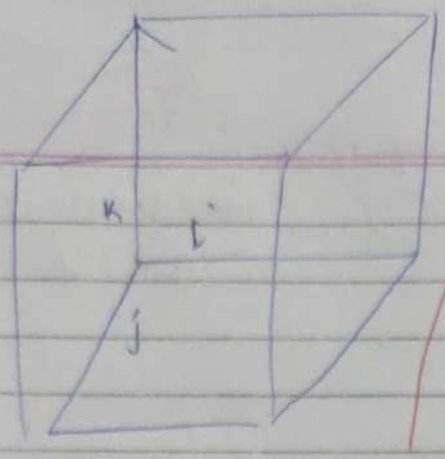


is shortest if intermediate vertices can only be 1 ... k-1

~~dist~~

$dist(i, j, k-1)$

(No k o/w cycle)



~~for~~

~~for i = 1 to n~~

Date :
Page :

~~i with outer for~~ ~~k = 0~~

for k = 1 to n

for i = 1 to n

for j = 1 to n { wt(i,j) ~~if~~

~~dist(i,j,0)~~ ~~∞~~ ~~c/w~~

dist(i,j,k) = min { dist(i,j,k-1) , dist(i,k,k-1) + dist(k,j,k-1) }

initialise for k = 0

for i = 1 to n

for j = 1 to n

dist(i,j,0) = ~~min~~ { wt(i,j) if (i,j) ∈ E ; 0 ; o/w }

recursive pattern.

largest (i,j) = ~~largest (i,j)~~ the last k you added.

$\binom{1}{1} \cdots \binom{n}{T_{n-i}}$

T_i

i k'' k' k''' j

Hwk

- Catalan Numbers

-kruskal pt

☆ largest (i,j) ← largest intermediate node in the shortest path from i to j.

say k = largest (i,kj)
then find.
largest (i,k) & largest (k,j) and so on...

☆ DP Table helps us generate the optimal soln also!

More Greedy
- rule based.
- non-Trivial
- Proof

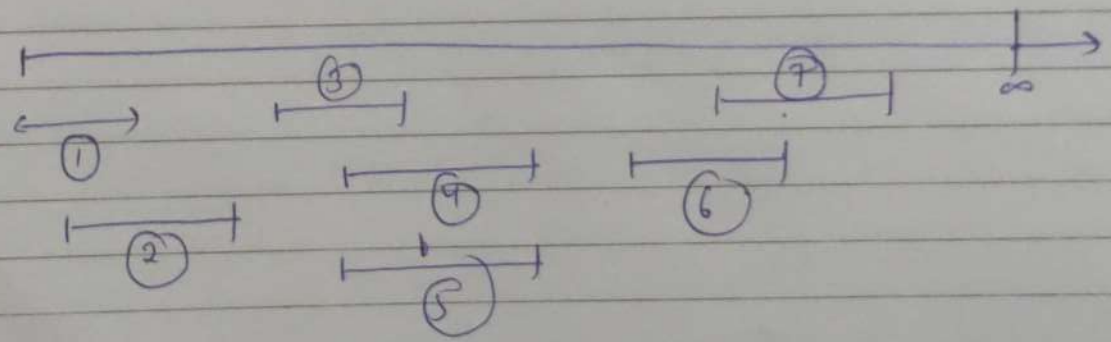
Interval scheduling

say lots of jobs to be scheduled. (ex: - scheduling in computer
- cloud service request)

↓
[start, finish]

Job	start	finish
i	s_i	f_i

Need ⇒ maximum subset that don't overlap.



1 3 6
to max^m 3 jobs can be done.

what rules can we try?

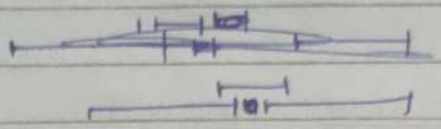
Rule 1: Select min - duration job first.

Date :
Page :

(i) select a job
(ii) remove all intersecting job

common among all rules

For all rules find pf or counter example

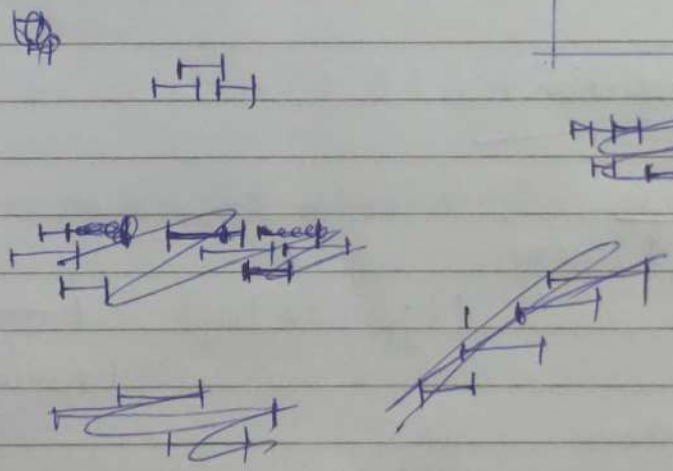


counter example ⇒

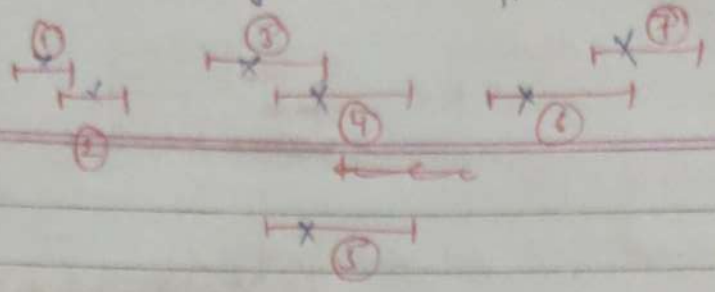
Rule 2: select one that blocks least no. of jobs.

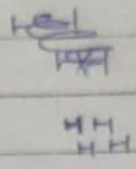
Exercise: Find a counter - example.

In Greedy there is a much more complex structure !



Rule 3: select job whose finish time is least.





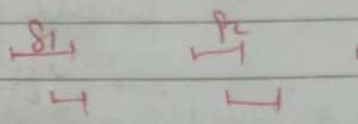
(1) (3) (6)

We'll prove sorted w.r.t. f.

If $O_1, O_2, \ldots O_k$ be "any" set of good jobs (do not overlap)

sorted w.r.t. f.

Let $g_1, g_2 \ldots g_m$ be the Rule 3 soln.

Show: $m \geq k$ (∵ )

> All g correct
but Rule 3 gives g_1, g_2, g_3)

(Pf by contradiction)

Assume $m < k$

Claim: $\forall\ 1 \leq j \leq m$

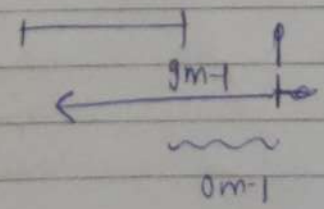
$$f(g_j) \leq f(o_j)$$ (remember o_j are sorted)

Pf by Induction

$j = 1$ by def^n (since g_1 finishes first)

I.H. Suppose $f(g_{j-1}) \leq f(o_{j-1})$ $\forall\ 1 \leq j \leq m-1$

I.S.



Wlog $f(g_m) \leq f(o_m)$

$f(o_m) \geq s(o_m) > f(o_{m-1})$ $\Rightarrow$ $f(o_m) \geq s(o_m) > f(g_{m-1})$

$\Rightarrow$ o_m is valid candidate to be put after g_{m-1}

&

we are selecting least finish time first after g_{m-1}

$\therefore f(g_m) \leq f(o_m)$

Why contradiction?

Due to Rule-3 $\Rightarrow$ No job starts after g_m finishes.

If we know

$$f(o_m) \geq f(g_m)$$

$\Rightarrow$ no jobs starting after o_m.

$\Rightarrow$ contradicts $(k > m)$

So finally, :- (i) sort all jobs by finish time

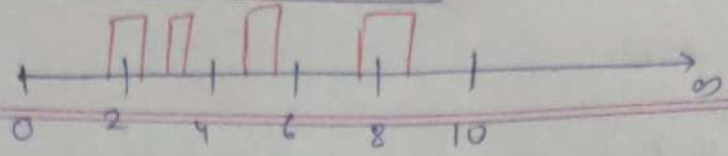
(ii) keep on picking jobs by Rule 3's strategy

T.C. = $O(n \log n) + O(n)$

(sorting) (scan)

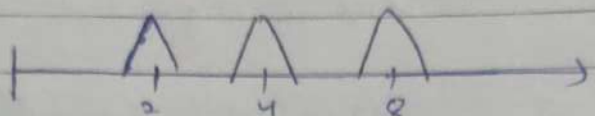
$= O(n \log n)$

Mobile Tower Placement



Given houses along Towers placed

Each tower covers $d = 2$ (inclusive of end pts)



[0,4] [2,6] [6,10]

Is such a covering always possible? → yes, one at each house

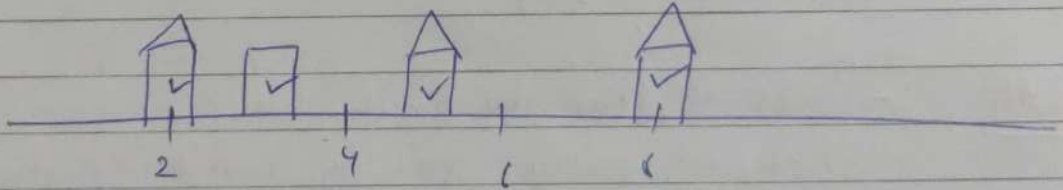
Given 4 towers we can cover all houses.

Problem: Minimize the number of towers.

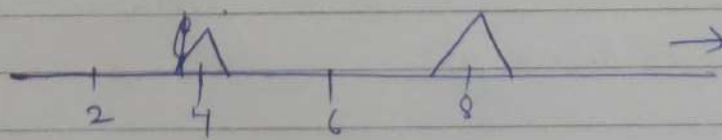
Input: Location of houses. $\in [0, \infty)$
range d

output: # of min. towers.
location of towers.

Rule 1: → Place at first uncovered house location & cover as many houses covered.

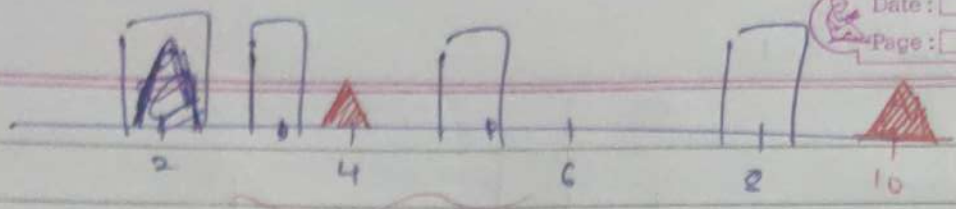


But we can do better



→ covers all houses

Rule 2: Place Tower at d distance away from last covered house.



Proof: Exchange argument (Even the last problem (scheduling) was exchange argument)

↑ change the soln to greedy soln

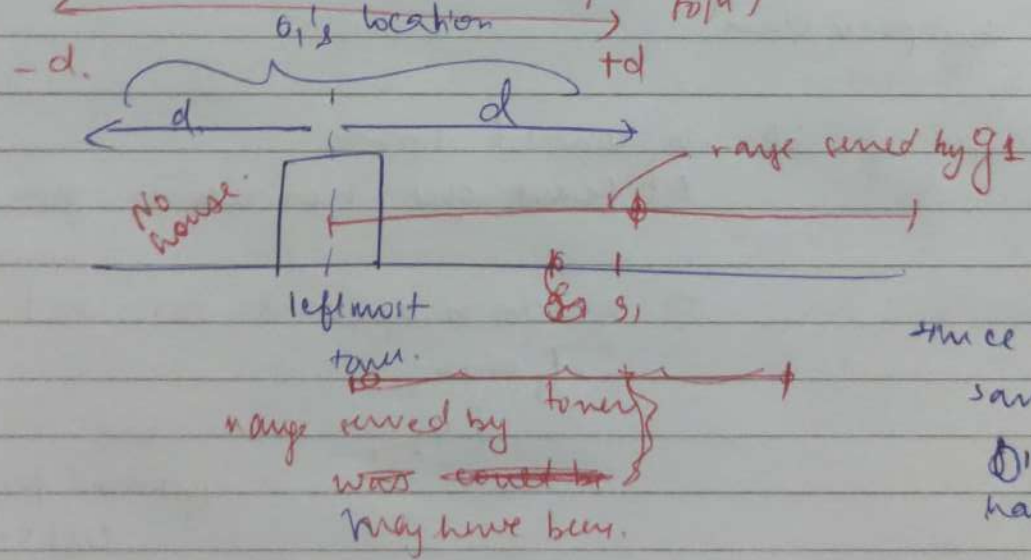
$O_1 \ldots O_k$ [sorted WLOG] be an optimal set of towers.

$g_1, O_2 \ldots O_k$ will also be optimal.

Claim: ↑ is also optimal

It suffices to show that g_1 cover whatever O_1 was exclusively covering.

(if O_1 not covering exclusively then O_1 can't be part of optimal soln)

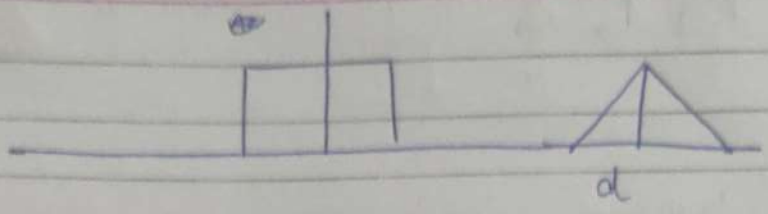


since range same as O_1 covered house covered

$\Rightarrow g_1$ can replace O_1

Let's proceed by induction on number of houses

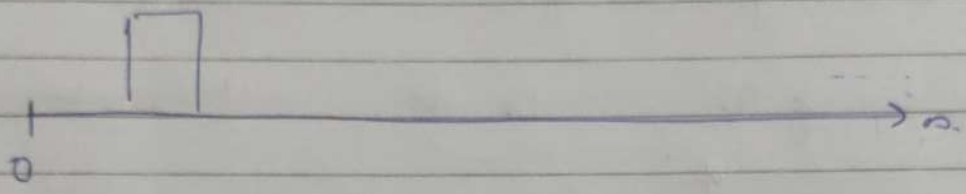
Base case: $n=1$



greedy rule 2 is working

I.H. · Greedy Rule-2 is best for $<n$ houses.

I.S. Show it for n houses.



Assume: $\{o_1 \ldots o_k\}$ $k \geq 1$

is optimal set

↓

is also an optimal sol^n by prev claim. $\{g_1 \ldots g_k\}$

Remove g_1 & its covered houses.

At least one houses is gone by now.

⇒ greedy is optimal for $k-1$

& gives

$\{g_2, \ldots g_k\}$

optimal for left over houses.

Put back $\{g_1, \ldots g_2 \ldots \sim \ldots g_k\}$

greedy rule 2 soln. (really?)

covers all inputs of size = k since optimal is of size = k then this is also optimal.

$g_2 \ldots g_k$ is ~~same as~~ same whether come from optimal on n-1 or after removing g_1

∵ i.e., how the Rule-2 algo works.

∴ done.

ToDo

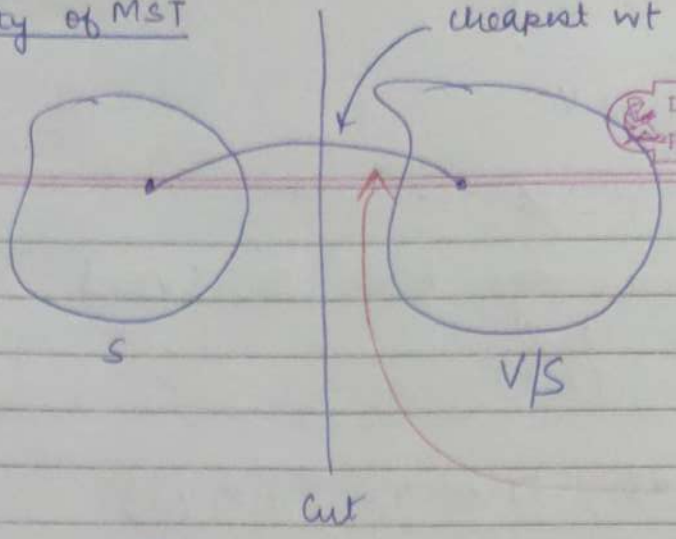
- Ask Sruthi abt Kruskal pf

Exercise

- ~~Start~~ Find a counter example to Rule-2 greedy in Interval scheduling.

⇒
- Implementation of the soln & learnt.
- Verify Interval scheduling pf was also kind of like an exchange argument

Cut Property of MST



$G = (V, E)$

this may not be unique either.

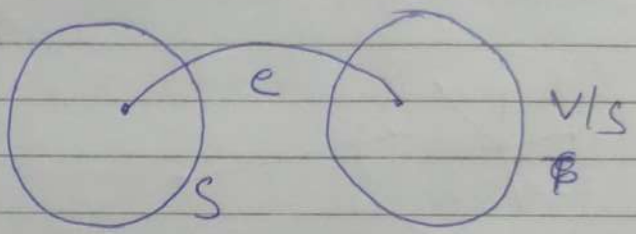
T

$\exists$ an MST of G s.t. T includes e $\forall S$

(MST is not unique)

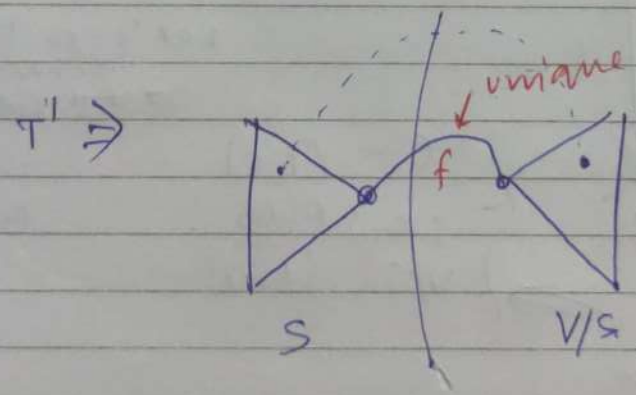
Hand-Wavy Pf (why is it hand-wavy)

Exchange argument



T' be some MST

(look at subtrees in $S \& V/S$)



cut property

$T \cup \{e\} \setminus \{f\}$

↓ connected & has got $n-1$ edges

↓

valid MST

$wt(e) \leq wt(f)$

∵ e was min. wt

but T' was MST

$\Rightarrow T' \cup \{e\} \setminus \{f\}$ must be MST too.

Algo (Naturally leads to)

(1) Pick any $v \in V$

(2) $S = \{v\}$, $T = \emptyset$

(3) pick lightest edge from $S \times \{V \setminus S\}$

↓

(x, y)

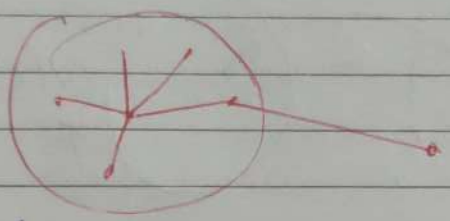
(4) $S \leftarrow S \cup \{x\}$, $T \leftarrow T \cup \{(x, y)\}$

(5) Go back to step 3 until ~~$|S| = n$~~.

$n-1$ edges are put in T

Note: Tree will connected & $n-1$ edges

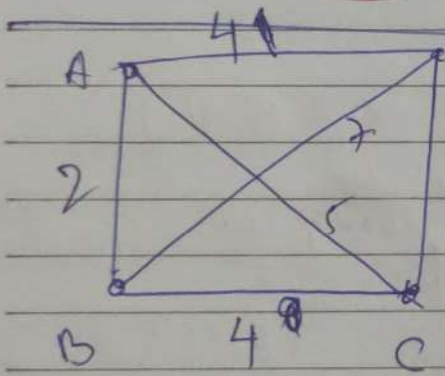
⇒ Spanning Tree.



why MST?

↓

will see



Example.

Let's go through exchange argument

↓

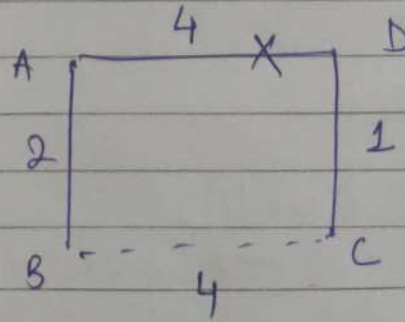
cut property

$e = (B, C)$

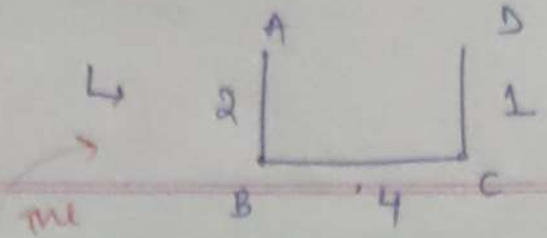
$S = \{A, B\}$

$V \setminus S = \{C, D\}$

4 edges now ⇒ 2 min



Can DO Kruskal to verify it is indeed MST



The cut property

In Prim's algo we can start with any node.

$S = \{A\}$, $V \setminus S = \{B, C, D\}$

↓ 1st Iteration

$S = \{A, B\}$, $T = \{(A,B)\}$

~~↓ 2nd iteration~~

~~$S = \{A, B\}$~~ $V \setminus C = \{C, D\}$

↓ 2nd iteration.

$S = \{A, B, D\}$ $T = \{(A,B), (A,D)\}$

$V \setminus S = \{C\}$

↓ 3rd Iteration.

$S = \{A, B, C, D\}$, $T = \{(A,B), (A,D), (C,D)\}$

$V \setminus S = \emptyset$

So Spanij Tree = $\{(A,B), (A,D), (C,D)\}$

(indeed an MST too)

(at least in this example)

Correctness

Spanning Tree ✓

Minimum?

↳ Pf by Ind^{n}

Claim: Let T be the grown tree at some intermediate iteration. ~~Theorem~~

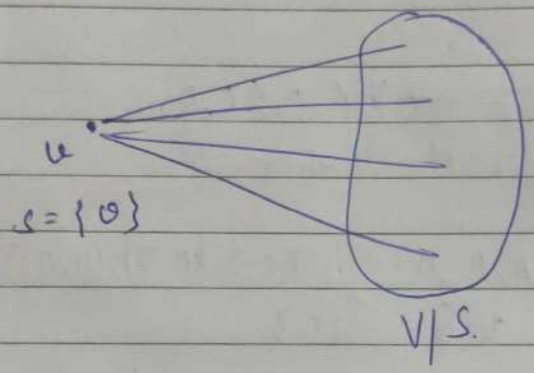
(jth iteration)

(So at last iteration we would be done)

⇒ ∃ an MST of G that includes T_j as subtree

↳ MST but smaller

Pf. j=1



e - cheapest wt edge. selected.

∃ an MST that includes e (By cut property.)

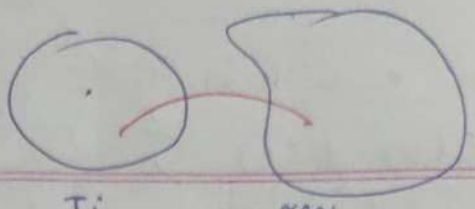
∴ Base Case Done.

IH

Suppose the claim is true for T_j

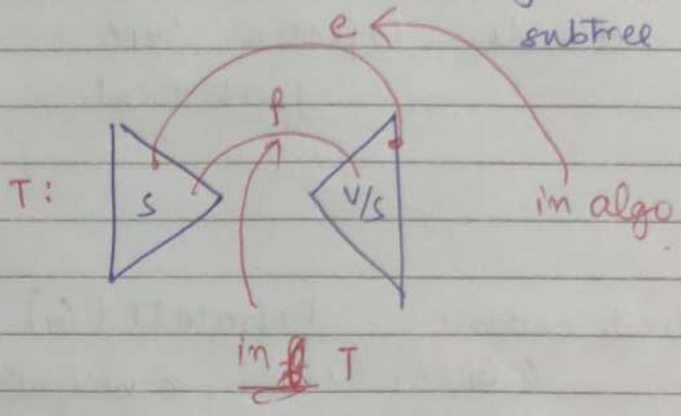
I.S

WTS for T_{j+1}



$S = T_j$ $\quad V/S = rest$

Let T be the MST that includes T_j as a subtree



$T|\{f\} \cup \{e\}$

↓

connected ∄ n-1 edges

↳ sparsity ∄ due to algo's choice

$wt(e) \leq wt(f)$

~~$wt(e) = wt(f)$~~

$wt(T \cup \{e\} \setminus \{f\}) \leq wt(T)$

But T was MST

$\therefore T|\{f\} \cup \{e\}$ is a MST too.

Also $T_j \cup \{e\}$ is a subtree of $T|\{f\} \cup \{e\}$.

$=$

T_{j+1}

which is MST

so done

∴

T.C.

⇒ go over all edges in each iteration, take ones that cross cut ep are minimum. $\theta(V) \cdot \theta(E)$ TS.

was Pkg.

↓

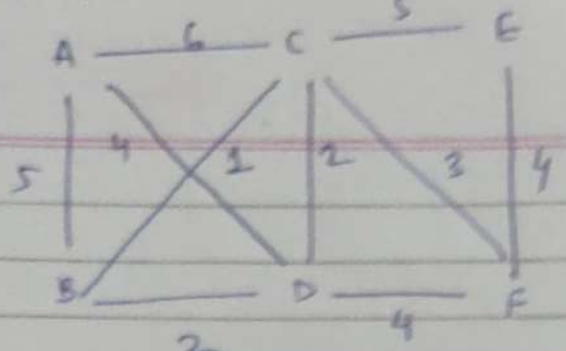
let's do Dijkstra type just that $d(u) + wt(u,w)$ won't come.

Procedure PrimMST (G)
// $G = (V, E)$ is a weighted undirected graph.
PQ ← priority queue of size $|V|$ with all keys = ∞
u ← any node in V
Decrease_key (PQ, u, 0)
π ← all NULL array of size of $|V|$
T ← ∅
while (PQ ≠ empty)
u ← extract_min (PQ)
for each $(u,w) \in E$ do
if ($wt(u,w) < key(w)$)
decrease_key (PQ, w, $wt(u,w)$)
π(w) ← u
endfor
T ← T ∪ {(u, π(u))}
endwhile
return π

T.C. → $O((V + \theta E) \log V)$

Add a condn for not edge

Example



	A	B	C	D	E	F
π	NULL	D NULL	B NULL	A NULL	C NULL	E NULL
Key	0	2	1	4	4	3

After Iteration-1

A is extracted

B, C, D will update

B	C	D	E	F

After iteration-2

B is extracted

B, C, F will be updated.

{(A, D)}

B	C	E	F

After iteration-3

B is extracted

C is updated

{(D, B), (A, D)}

C	E	F

After Iteration-4

C is extracted

E & F are updated

{(B, C), (D, B), (A, D)}

After iteration-5

F is extracted

E is updated

E
F
0 4

~~{C, E}~~ {(D, B), (B, C), (A, D), (C, F)}

After iteration - 6

E is extracted.

{(A, B), (B, C), (A, D), (C, F), (F, E)}

(Check this MST by running Kruskal or seeing the o/p of kruskal's algo this is same as this.)

4/10/22

lec-

Problem → which Paradigm?
↳ which Data structure

Calculator
(Data Structure):

$8 + 3 * 5 \wedge 2 - 9 * 67 = ?$
(exponentiation)

$\wedge$ > BDMAS

$= 8 + 3 * 25 - 9 * 67$
$\equiv 8 + 75 - 9 * 67$
$\equiv 8 + 75 - 603$
$\equiv -520$

~~O(n x~~
n = total no. of symbols or input number.

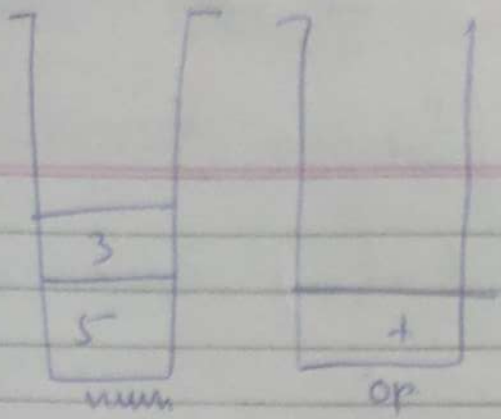
Time? $\theta(n \cdot n)$
↑ each time we scan the whole string again

Better?
store old results in a stack.

Idea: 2 stacks: - numbers
- operators.

(assumptions: ✓ no ()
← $2 \wedge 3 \wedge 5$ [0]
- well defined input

2 operators next to one another



5+3

All operations we are dealing with are binary operators.

8+3*5^2-9+67

who do we start with?

Procedure PopNOperate.

// numstack, opstack

Operator ← Opstack.pop()

number1 ← numStack.pop()

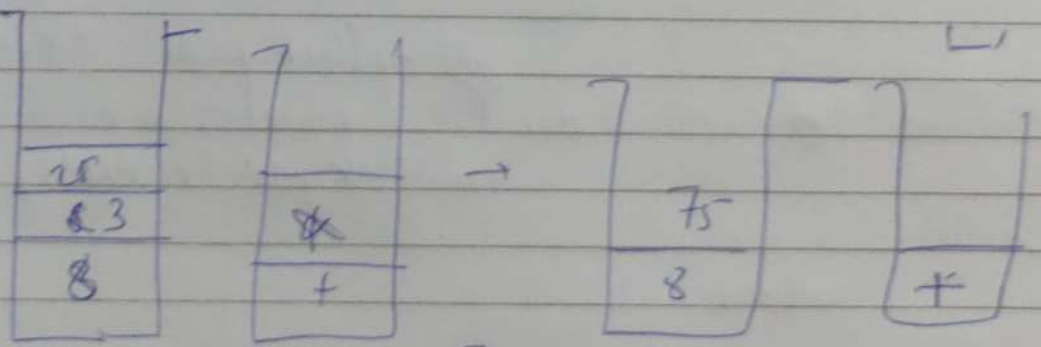
number2 ← numstack.pop()

Operate (number1, number3, operator)

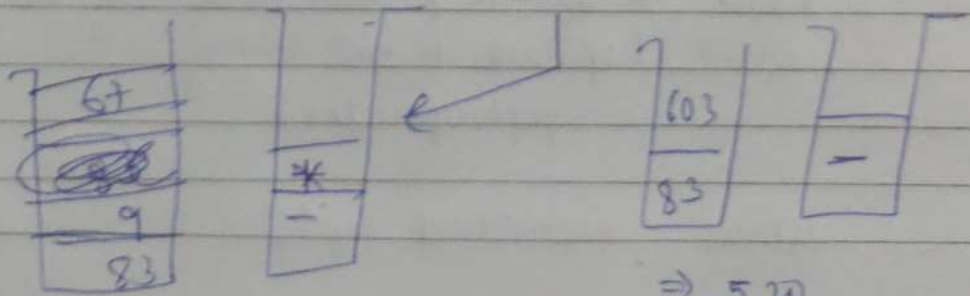
DIY: Write Operate fn code

There is a comparison btw operator to be added in op current top in stack.

'-'



'67'



⇒ -520

lesson: pop & operate whenever next ~~input~~ symbol, is ≤ the opstack top symbol.

Priority (- +) (-, *) same.

Procedure Calculator (Input x)
// x.next() gives us next operator for
opstacky ← empty stack, numstack ← ∅ number.
while (x.next() ≠ NULL) (i.e. input left)
if (x.next() is an operator)
if priority (x.next())
≤
priority (opstak.peek())
popNoperate.()

check
In, out.

else
opstack.push (x.next())
else if (x.next() is a number)
numstak.push (x.next())

endwhile

// opstack & numstack may not be empty
while (opstack is not empty)
popNoperate()

return numStack.top()

// Note : OpStack is sorted in Descendig order.

Keep on popping ← Date : Page :

of operator.

What is Parenthesis?

↳ Zone of self expression

is

separate ~~instance~~ instance of the problem

'(' must get the lowest priority

Idea : (≡ serves as a new bottom of stack.

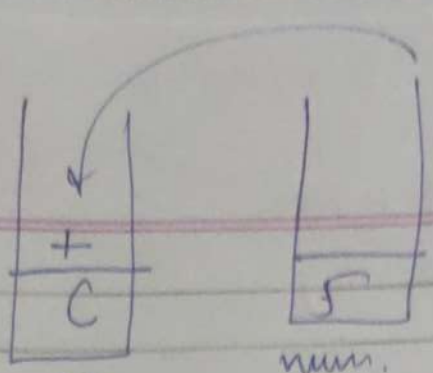
) ≡ serves as a new end of input.

---

else if (X.next is '(')

check.

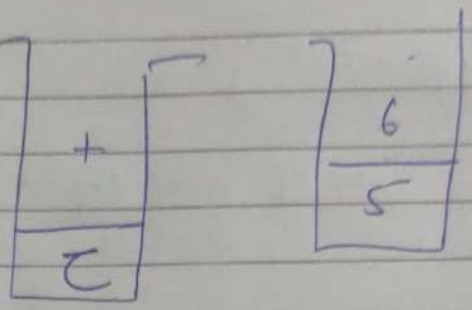
(5+6)



( ) is a new environment for calculator.

∴ C has ~~lowest~~ lowest priority ~~priority~~ y

check



check

Else if (S.next ~~is~~ is ~~not~~ '(')
write (S.next() is not d')')
while loop pass prev pseudo code will be replaced here!

if (S.next is ')')

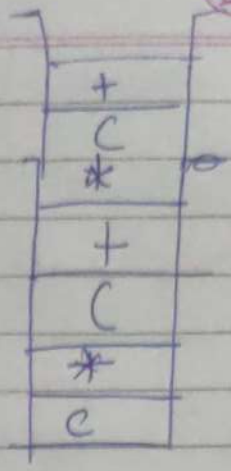
check

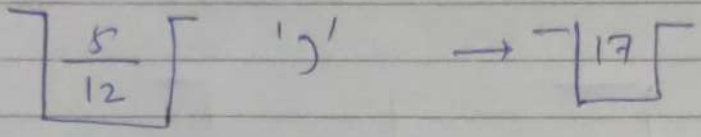
If u have nested paranthis

(.. (. . (.) .))

Date :
Page :

(5* (3+10) * (2+5))





Associativity

$2 \wedge 3 \wedge 4$ → $((2 \wedge 3) \wedge 4)$ 8^4

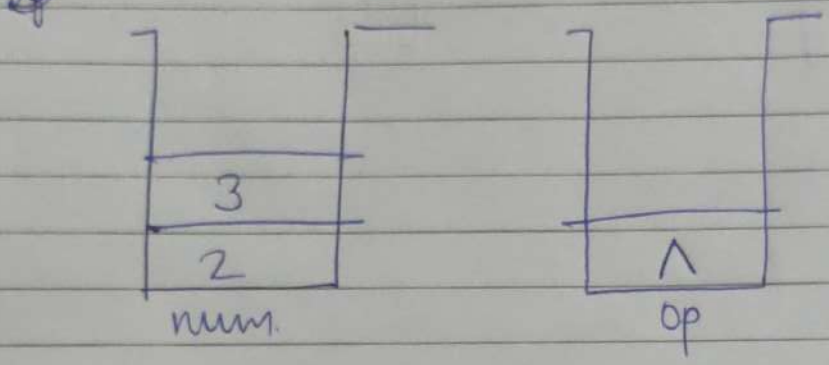
Right associative → $(2 \wedge (3 \wedge 4))$ 2^{81}

$2 | 3 | 4$ left associative → $(2|3)|4$

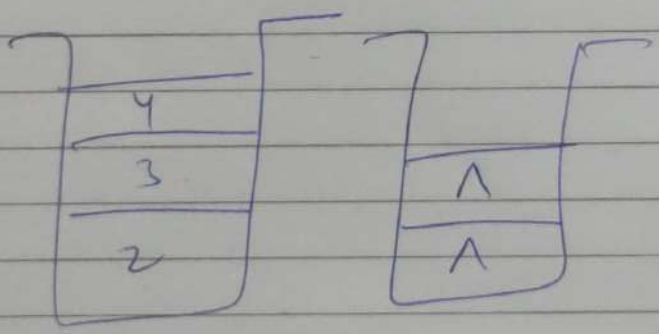
→ $2|(3|4)$

For +, *, - order doesn't ~~matter~~ matter.

$2 \wedge (3 \wedge 4)$ → $2 \wedge 3 \wedge 4$



if outside > inside for ∧.



O/W $(2 \wedge 3) \wedge 4$

We'll maintain 2 priorities of operator

↙

In & out priority.

^: out > In , /: out < In

	In	out
C	0	-
+, -	1	1
*	2	2
/	4	3
^	5	6
)	-	7

Correctness: Imitates human behaviour.

Time complexity: O(n), n = # input

"Use of right Data structure is very imp"

↳ sir will put ques on Hello IITK.

Now, Advanced Algorithm.

- Randomized algo (Quick sort etc.)

- ML backpropogation

- Cryptographic.

Is subtraction associative? Nope.

Dijkstra transformation works? Nope If length of paths are diff^n the no!