Fixed universe Succession

Maintain a dynamic set (S) of n elements

(Fixed: universe) — we know the elements

universe $u = \{0, \ldots, u-1\}$

$S \subseteq u$ (elements of S belongs to u)

Dynamic operations:

* Insert ($x \in u \setminus S$) Insert an element from u in S
* Delete ($x \in S$) Delete.

---

* 'Succession ($x \in u$)' (next element after x in S)

* Find the next element in S, longer than x

* predecessor ($x \in u$) — previous element of x in S.

Data Structure

Say: $u = \{0, 1, \ldots 15\}$

$S = \{1, 9, 10, 15\}$

Successor of $2 \Rightarrow 9$

Successor of $1 \Rightarrow 9$

'universe is fixed', $S \subseteq u$

options

I) Array

S	1	9	10	15

Not sorted

Insert : Shift (need to) : $O(n)$

Delete : $\Theta(n)$

Successor (x) $\Rightarrow O(n)$ → Scan the array.

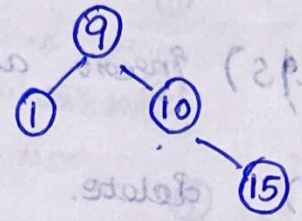
II) Linked list — Unordered.

II) S: [1] → [9] → [10] → [15]

'Insert / Delete' — Search for the element' → O(n)

Successor / predecessor ⇒ O(n)

III) Balanced binary search tree:

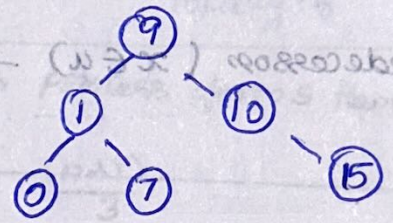


Insert: log n times

Delete: log n

Successor:

Size: n



= Successor (7) [In order traversal]

= 9 (go to height)

= log n

What we need: O(log (log u))

m = log n elements (Sorted)

↳ Binary search

↳ O(log log n)

'Any recurrence' can give this order.

$T(u) = T(\sqrt{u}) + O(1)$

$= O(\log(\log u))$

$\sqrt{u} = \log u$

$\frac{1}{2}\log u$

1975 → Invented by van peter emde boas

Suggested a DS - successor query / insertion

in the order of $\log(\log u)$

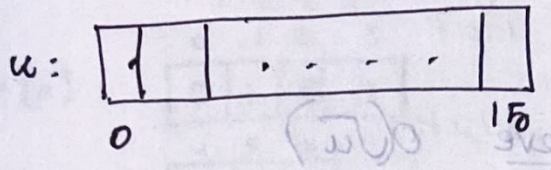
$$T(u) = T(\sqrt{u}) + O(1) \longrightarrow O(\log(\log u))$$

I) other type of arrays:

'Bit vector'

↓

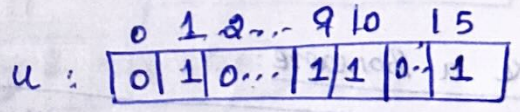
maintains array of u (size)



$$u(x) = \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{otherwise.} \end{cases}$$

If elements in array ↳ 1 else → 0.

$S = \{1, 9, 10, 15\}$



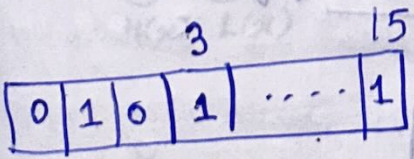
Insert:

'switch on' : switch on'

u → size (0 to u-1)

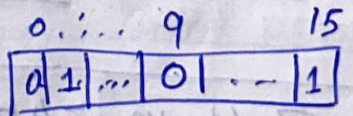
Insert (3)

1. u[3] ← 1



Delete (9)

1. u[9] ← 0



How much fast?

Insert / Delete → O(1)

Succesor: ?

Successor (x)

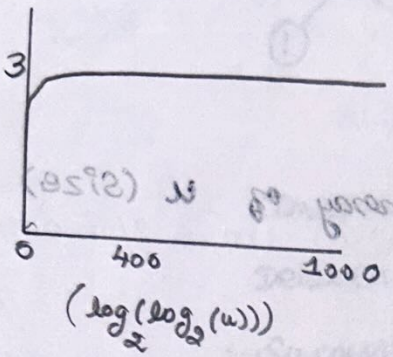
All the element are 0 except $u-1$

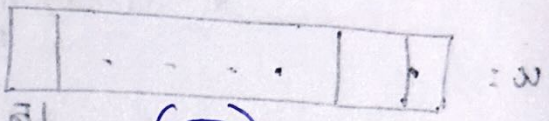
0 0 0	1

'Scan whole array' $\Rightarrow O(u)$ = worst case

---

we want: $\log(\log u)$





Achieve $O(\sqrt{u})$

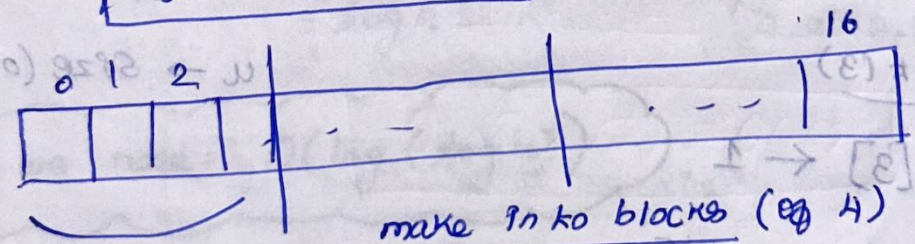
$O(\log \log u) \rightarrow$ Goal

$T(u) = T(\sqrt{u}) + O(1)$

$\sqrt{u} = 4$

Take a Square:

4 by 4 matrix = 16



make into blocks (eg 4)

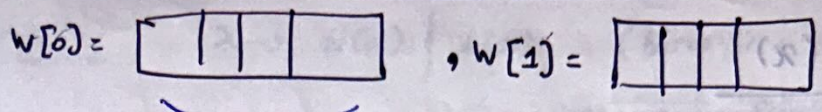
Blocks/ vessel

4 - blocks

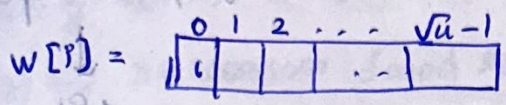
$\therefore \sqrt{u}$ blocks = 4 blocks

2 dimensional

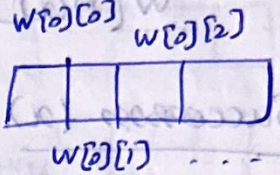
Van Emde boas DS



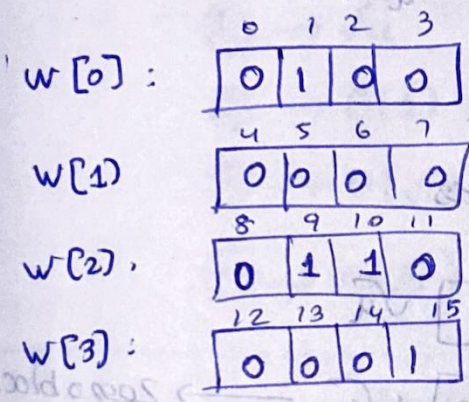
$\sqrt{u}$ = size of each block.



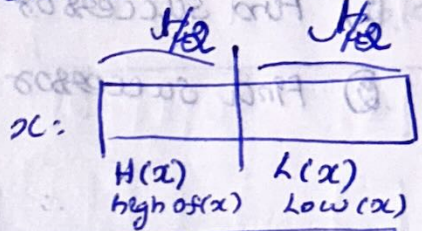
"2-d array"



$S = \{1, 9, 10, 15\}$



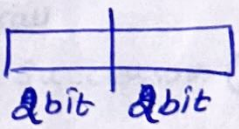
Insert (x)



$x = H(x)\sqrt{u} + L(x)$

Convert x to binary & half size

∴ ~~1000~~ ⇒ then half bits



∴ 10 | 1

$x = 9 = 1\underbrace{0}_{H(x)}\underbrace{0\,1}_{L(x)}$

10 | 0 1 → position of number

↓

Block number = 2

∴ $W[H(x)][L(x)]$

$x \to H(x)\sqrt{u} + L(x)$

Insert (x)

$W[H(x)][L(x)] \leftarrow 1$

Delete (x)

$W[H(x)][L(x)] \leftarrow 0$

$5 \Rightarrow 0\ 1 \mid 0\ 1$

1st block, 1 elem

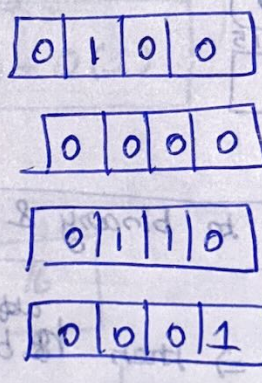
Successor (x)

$S = \{1, 9, 10, 15\}$

$x = 2 \Rightarrow 0\ 0 \mid 1\ 0$

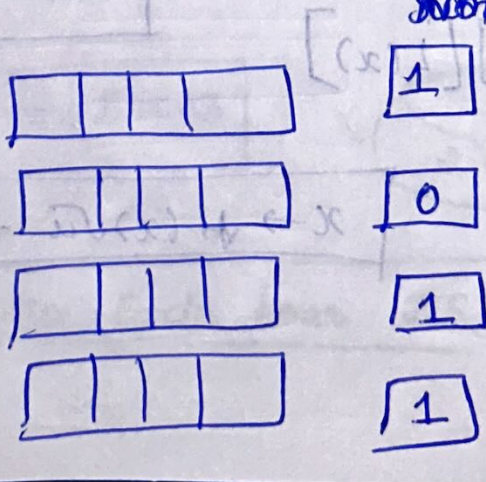
0 block, 2 element

1. Find successor in its block
2. Find successor in next blocks



$\therefore O(\sqrt{u} \times \sqrt{u}) = O(u)$

'Augmentation' → Search in 0 block - waste of time - Avoid searching (Add some extra info)



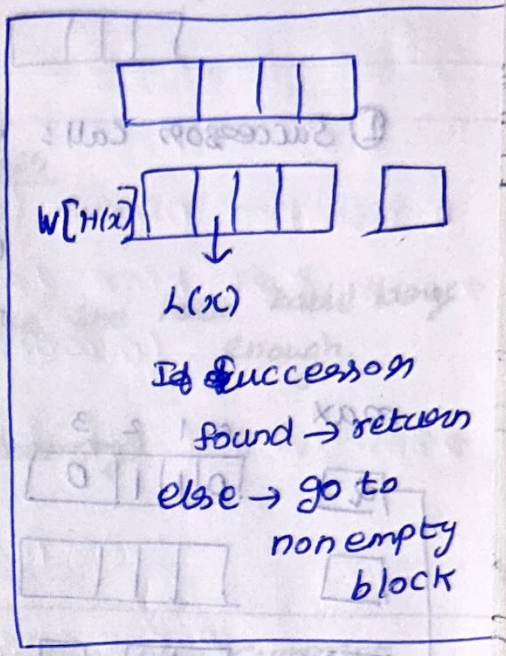
Successor (x) $x \to H(x) \mid L(x)$ (binary)

1) Look at the successor of x within widget $W[H(x)]$ starting after $L(x)$

2) If successor found return

3) else find the next non-empty widget $W[i]$

4) Find the successor in $W[i]$ starting from -1



① $\to \sqrt{u}$

② $\to O(1)$

③ $\to O(\sqrt{u})$

④ $\to O(\sqrt{u})$

Recursive

successor $(x, W[\,][\,])$

① call successor $(L(x), W[H(x)])$

$i \leftarrow$ successor $(W[x], \text{nonempty}[0, 1, \ldots \sqrt{n}-1])$ → successor call.

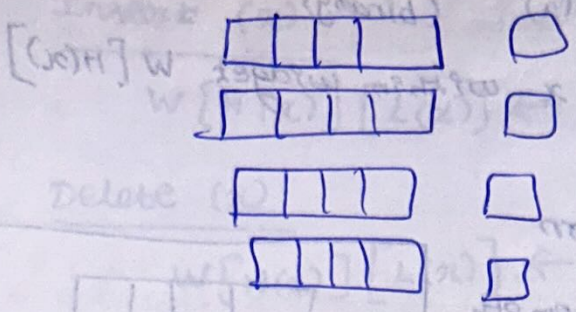
successor $(-\infty, W[i])$

worst case: go for 3 successor calls

we want? – 2 successor calls

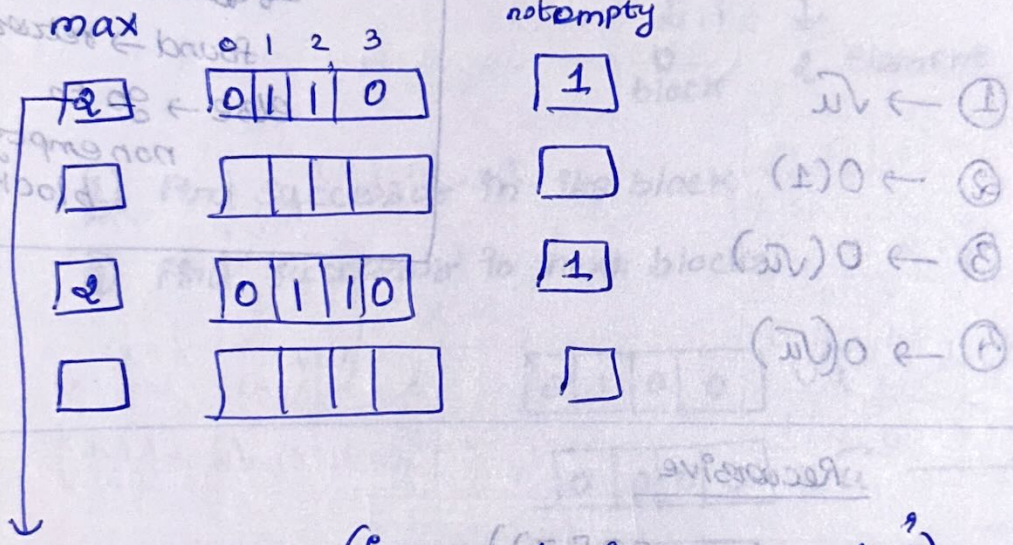
$T(u) = 3T(\sqrt{u}) + \Theta(1)$.

we want: $2T(\sqrt{u})$



① Successor call: Successor (L(x), W[H(x)]) → Avoid this call.

'Add another Augmentation'

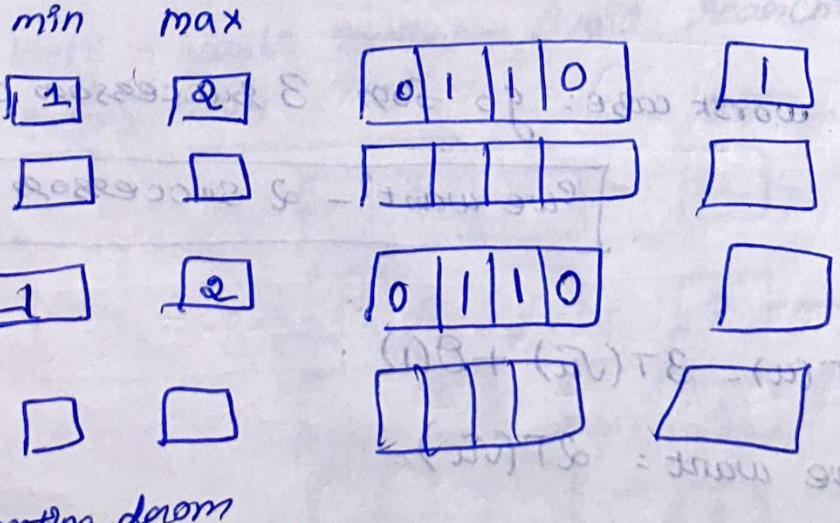


After 2 No 1 ('Save 1 Successor call')

Max < L(x)

Do! else Don't

'Adding Augmentation — Save time'



1 Starting from index 1.

'Save more' — successor — recursive call!

$T(u) = T(\sqrt{u}) + \theta(1)$

↳ 'Avoided' — 2 recursive call

$= O(\log(\log u))$

Amortized Analysis

How large should a hash table be?

Goal: 'No overflow' → make the hash table large enough.

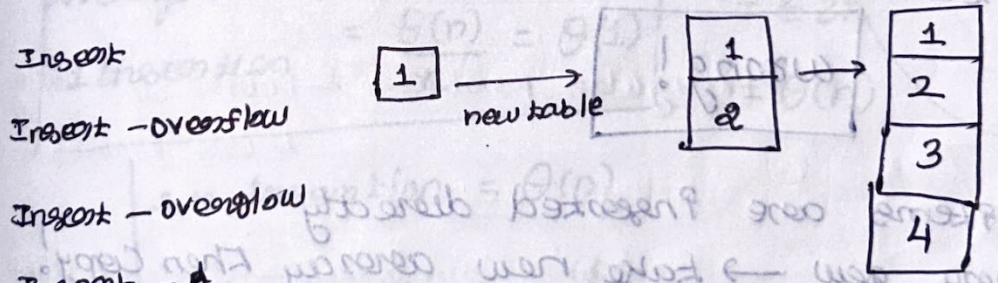
'Dynamic table' — overflow (allot new memory)

malloc / calloc!

Allocate new larger table: Then position all over current elements to the new larger table.

---

eg: dynamic table



'Strategy: double the older one'

No need to reallocate frequently!

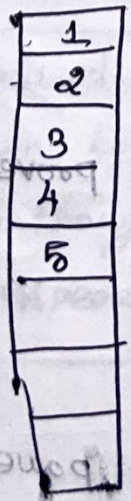


Table size = 8

→ overflow

(Copy all items)

↑ over time Complexity.

most items? → just allocating $O(1)$

few items → creating $O(n)$ time

↳ no. of elements in the table (prev) $+ O(1)$

---

worst case analysis

n insertion : n items (Consider sequence of n insertion)

* worst case time to execute one insertion is $O(n)$.

(overflow → $O(n)$ → shift to new one)

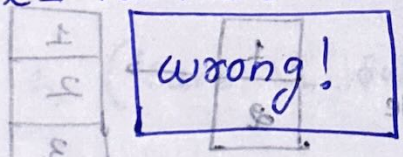
* Best case (Just insert → $O(1)$)

∴ Assuming

worst case time for n insertions

$n \cdot \theta(n) = \theta(n^2)$

each time new table?



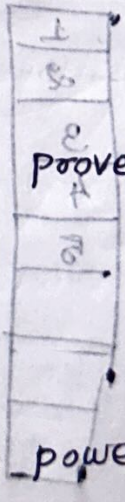
∴

* some items are inserted directly
* only very few → take new array then copy!

↓

Amortized analysis – Avg but there is no probability!

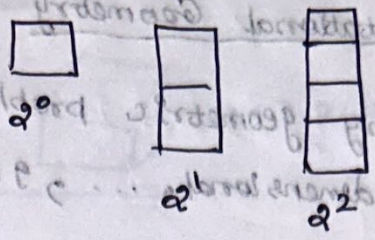
$$\text{n insertions} = \theta(n)$$ → To poove!



Tighten analysis:

c_i = cost of the ith insertion.

$$= \begin{cases} i & \text{if } i-1 \text{ is exact power of } 2 \\ 1 & \text{otherwise.} \end{cases}$$



overflow: when reaches (2^0+1)

i	1	2	3	4	5	6	7	8	9	10
Size i	1	2	4	4	8	8	8	8	16	16
c_i	1	2	3	1	5	1	1	1	9	1

most case: 1

Few case: $\theta(n)$: new table (copy old items)

---

cost of n insertion: $\sum_{i=1}^{n} c_i$

$= n + \sum 2^j$ ($\log(n-1)$ → Amortized.)

$= \leq 3n$

$= \theta(n)$

Average cost of 1 insertion $= \frac{\theta(n)}{n} = \theta(1)$

$\therefore$ n insertion $= \theta(n)$

doing $i = 5$, 2^2+1 (Here).

No probability involved → Amortized analysis.

Amortized analysis → strategy for analysing sequence of operation to show that average cost/operation is even though the single operation with in them may experience some expensive!

Types of Amortized analysis

- Aggregate (discussed)
- Accounting
- Potential method.

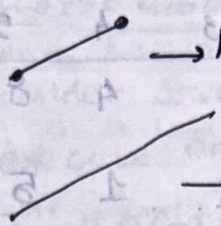
dynamic table. ←

Computational Geometry

* Algorithm for solving any geometric problem!

1 dimensional, ... , 9 dimensional.

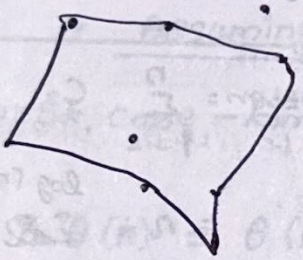
Fundamental objects ⇒ "." → point (any dimensional point)



→ line segment

→ line

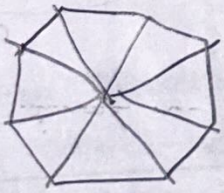
Any dimensional points:



→ Small region covering all the points.

Polygon.

'Triangular problem' — Form Δ^{le} using these points



Triangulation

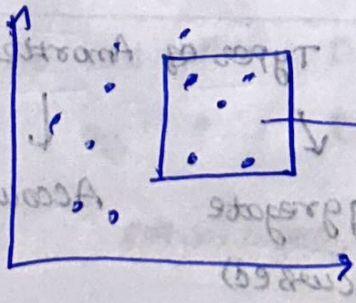
(or) Convex hull — Closed region (Any two points inside the region)

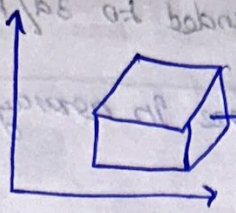
'So they have geometric structures'.

Orthogonal range search — more applications

Idea: n points : d dimension : range (orthogonal range)

query:





→ 3D range. / higher dimension also possible.

③ queries

1) Is there any points are there in this range!

2) If yes then how many?

3) Give the points.

Handle!

d-dimensional

* Is there any points? → Search

2D → * check x coordinates, y coordinate fitted in this rect/not

3D → x, y, z

↳ 'one sort solution'

↳ $O(n.d)$

$d = 1$

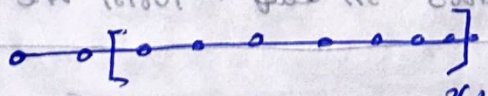
$O(n)$

'static!'

↳ Static DS (Fast search).

orthogonal range search.

1D range search

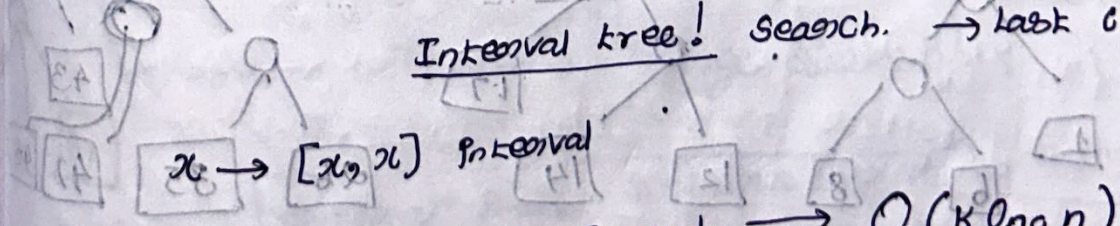


n points: Static set.

interval

* Find the points lies b/w here. → how?

Interval tree! search. → last class.



interval – search → $O(k \log n)$

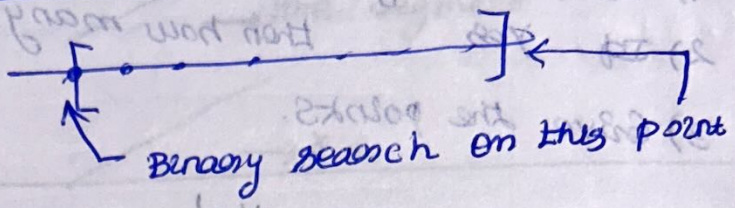
* Problem: This idea can't be extended to 3d/higher dimensions

Sort this! → Store in array!

→ Sort the point (static) - easy

→ Store in an array.

2 points (given) (interval)



Binary search on this point

So we get the positions of these two : Now the intermediate points are the solution!

Sorting: O(n log n)

Binary Search: 2 (log n) ≈ O(k + log n)

Problem: can't be extended to 2d!

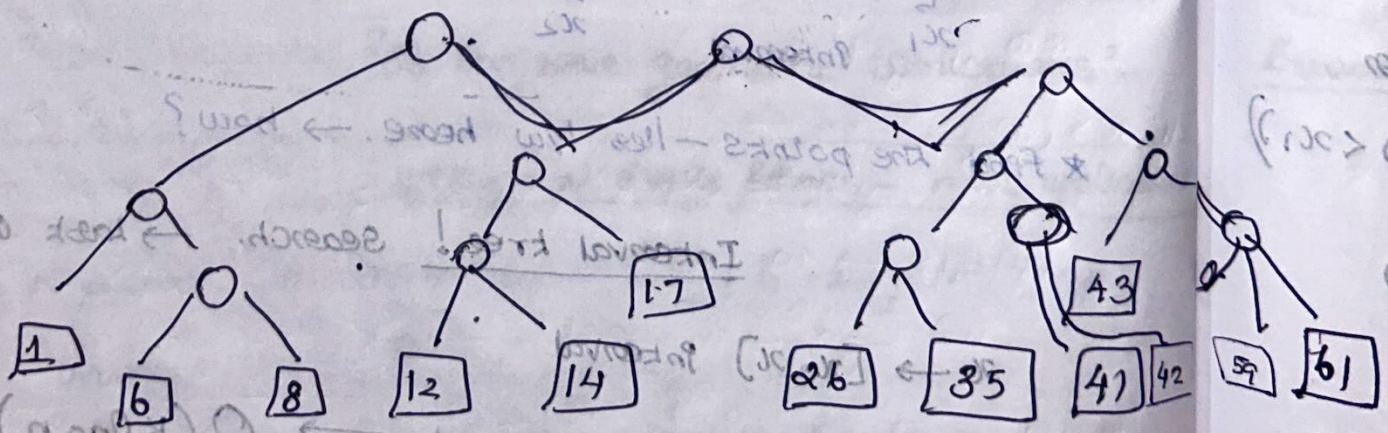
2 coordinates

which one to sort!

Solution: Tree

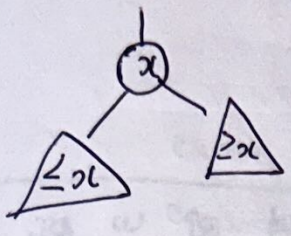
1-d - range - Search tree

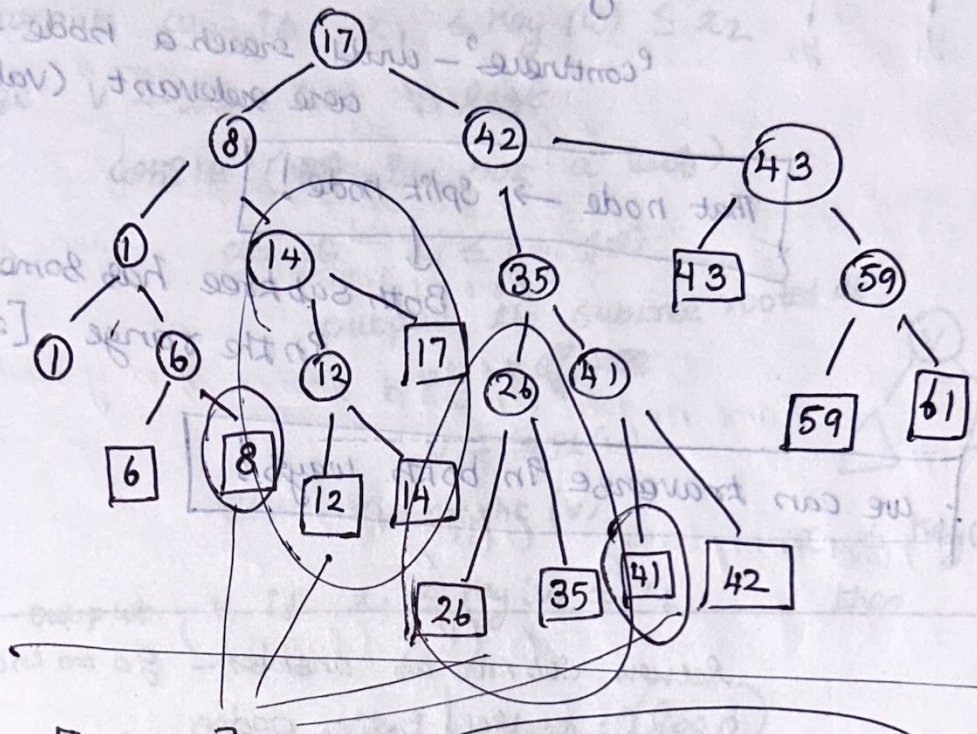
"put - all the points in leaf - form the tree"



"Balanced Search tree"

put some value: In empty nodes: To make it binary ST!





eg: [7, 41]

Return these roots

All the leaf nodes in these roots.

leaf of 6 → 8

leaf of 14 → 12, 14, 8

leaf of 26 → 26, 35

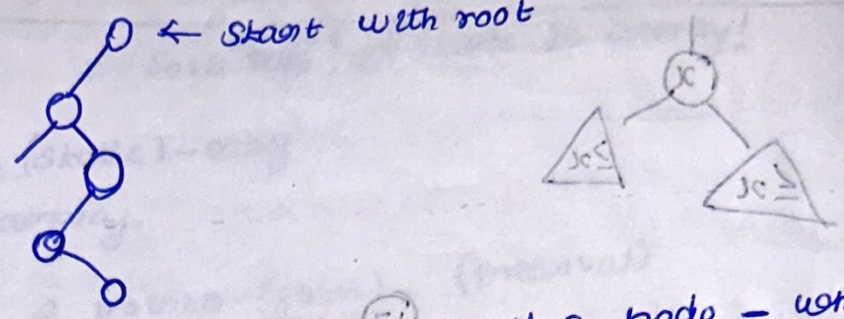
leaf of 41 → 41

Answer: Leaves in the subtree!

one-d-range Search!

General idea: return leaves

1) D range query $[x_1, x_2]$



← Start with root

"continue" – until reach a node – which are relevant (values)

That node → split node!

Both subtree has some nodes in the range $[x_1, x_2]$

∴ we can traverse in both ways!

Return subtree as answer – go to wider subtree

get indiv. nodes

"k-nodes" (say)

★ ∴ $(\log k)$ subtree (Intuitively)

1D – range – query $(T, [x_1, x_2])$

1. $w \leftarrow$ root (T)
2. while w is not a leaf and
   $(x_2 \leq$ key (w) or
   (key $(w) < x_1$))
3. do if $x_2 \leq$ key (w)
4. then $w \leftarrow$ left (w)
5. else $w \leftarrow$ right (w)

// when the while loop enters w is a leaf / split node.

$[x_1, x_2]$

$\rightarrow$ key (w)

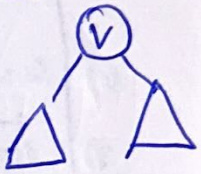
If $x_2 <$ key (w)
our focus: left side

else ($x_1 >$ key (w))
completely right side

Split node: 

Interesting here (return each elements).

6. If w is leaf
7. output w if $x_1 \leq key(w) \leq x_2$
8. else $v \leftarrow left(w)$ // left
9. while (v is not a leaf)
   do if $x_1 \leq key(v)$
      output the subtree rooted at right(v)
      $v \leftarrow left(v)$
   else ($v \leftarrow right(v)$.



$x_1 \leq key(v)$ then

output v if $x_1 \leq key(v) \leq x_2$

Time: Search from root → go to leaf : $O(\log n)$

"K → leaves" to reach leaves $O(K + \log n)$

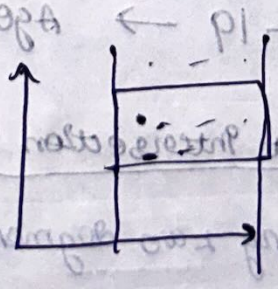
Space!

$O(n \log n)$

↳ For n tree

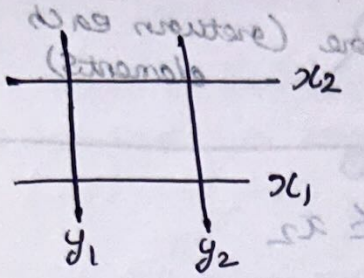
Extend this idea for n-dimensions

2-d range-tree



Primary : 1D range tree based on x-axis.

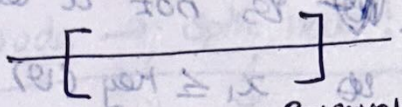
* secondary 1 dimensional tree based on the Y coordinates



x axis → n points

y axis → n points

Tree 1



Take x Interval

Tree 2

Take y Intervals

Same nodes: do 1d search for y coordinates.

Time complexity: $O((\log n)^2)$

$O(K + (\log n)^2)$

↓

No of points lying in rectangle.

Pre processing time = $O(n \log n)$

d dimension:

query time: $O(K + (\log n)^d)$

eg: Student records eg Age, CGPA, etc.

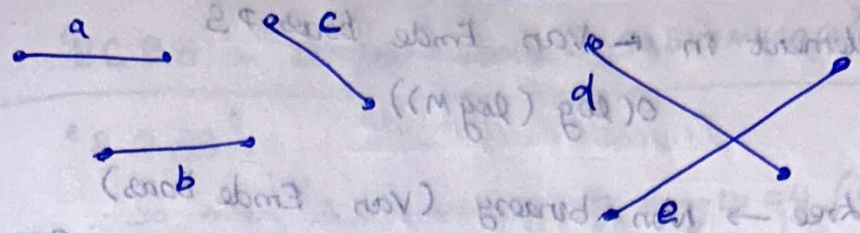
7.9 – 9.9 → CGPA

17 – 19 → Age

Line segment Intersection

* Few line segments – Any two segments are

Intersect / not → does any pair intersect.

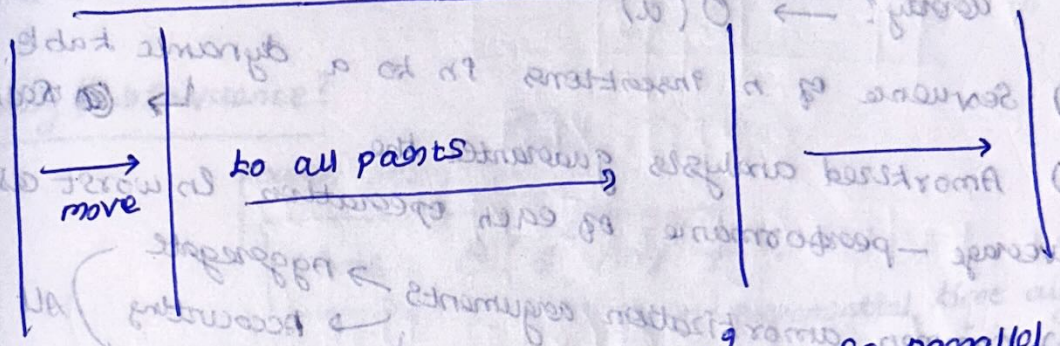


'Line segments'

Coordinate wise : $O(n^2)$ ∴ $^nC_2$ pairs

'Sweep line algorithm'

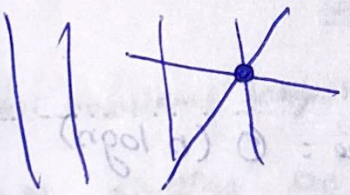
Use vertical line - Just move that vertical line.



'So if two intersections' → so parallel.

3 intersections (same point)

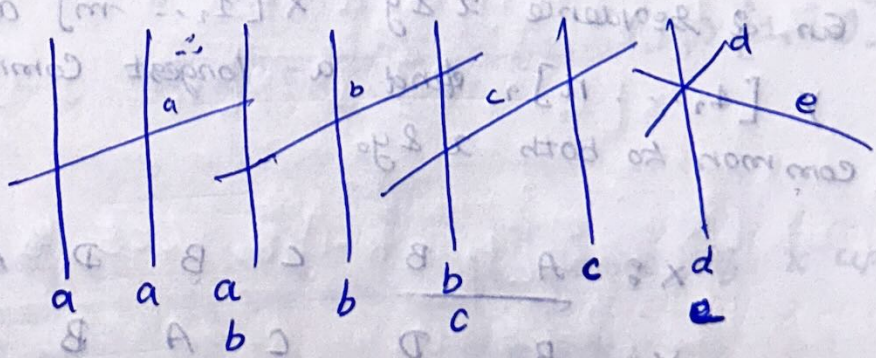
↳ Intersecting two lines



Interested in these intersecting points

maintain a DS (dynamic) : with this intersecting points

order on the value of x axis, y axis.



1) Search element in → Van Emde boas DS

$O(\log(\log n))$

2) Type of tree → Non-binary (Van Emde boas)

3) Abstract data type: Emde boas tree perform oper: Associative (2d) array

4) Van Emde → Universe size = 16), how many blocks?

4 blocks $(4^2 = 16)$

5) Worst case run of successor / predessor in bit vectors array? → $O(u)$

6) Sequence of n insertions in to a dynamic table → keenotes

7) Amortized analysis guarantee the average-performance of each operation in worst case

8) Common amortization arguments → Aggregate, Accounting, potential } All

9) $T(n) = T(\sqrt{n}) + O(1)$

$O(\log\log n)$

10) Space Complexity: 2D range tree: $O(n \log n)$

Dynamic programming

Design technique (Divide & conquer) →

Longest common subsequence problem

* Gn 2 sequence x & y $x[1, \ldots m]$ and $y[1, \ldots n]$, find a longest common subsequence common to both x & y.

X: A B C B D A B

Y: B D C A B A

"BCB" — sub sequence of x & y.

'BCBA' — Longest Common Subsequence

'BDAB'

∴ So many subsequence of length = 4 (Longest = 4)

Naive approach

X [1, ... m]

Y [1, ... n]

Naive: Check every subsequence of X, then check whether it is a subsequence of Y or not! → O(n) 'length of Y'

How many Subsequence?

power set : 2^m

Time = $\theta(2^m \times n)$ → exponential time algo. (very expensive)!

Do better!

Given: problem: Longest Common Subsequence (X & Y)

Instead of finding the longest C S — do find length of it

'Find the length of the longest Common Subsequence'

↳ 'variable'

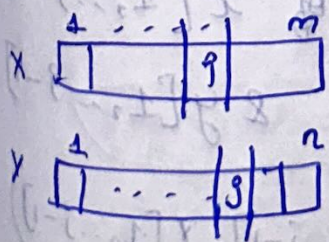
'Dynamic programming' technique:

Prefixes

$C[i,j] = |LCS(X[1,..i], Y[1,..j])|$

↓

Length



So we take subsequence of X upto ith index

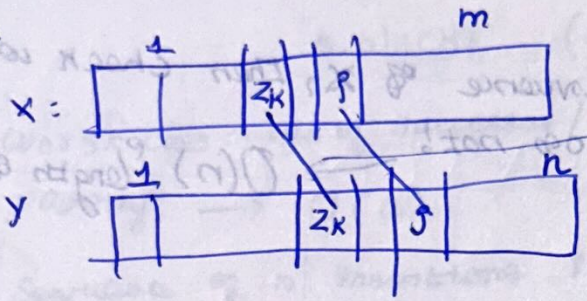
C[i,j] → Length LCS of X upto i and Y upto j

c[m,n] → we need (through x & through y)

Recursive formula

$$c[i,j] = \begin{cases} c[i-1, j-1] + 1 & \text{if } x[i] = y[j] \\ \max\{c[i-1,j], c[i,j-1]\} & \text{if } x[i] \neq y[j] \end{cases}$$

Proof:



$x[i] = y[j]$

Let $k = |LCS(x[1,\ldots i], y[1,\ldots j])|$

$z[1,\ldots k] = LCS(x[1,\ldots i], y[1,\ldots j])$

Claim $z[k] = x[i] = y[j]$

$z[1,\ldots k] \,||\, x[i] = z[1,\ldots k] \,||\, x[j]$

(concatenate)

Assume : k^2 → length

prove : k+1

Take

$z[k] = x[i] = y[j]$

$z[1,\ldots k-1]$ is a common subsequence b/w

$x[1,\ldots i-1]$ & $y[1,\ldots j-1]$

or

$z[1,\ldots k-1]$ is LCS of $x[1,\ldots i-1]$ & $y[1,\ldots j-1]$

$LCS(x[1,\ldots i-1], y[1,\ldots j-1])$

$Z[1, \ldots k-1] = LCS(X[1, \ldots, i-1], Y[1, \ldots, j-1])$

$\boxed{C[i-1, j-1] = k-1}$

$C[i,j] = k = (k-1)+1$

$= C[i-1, j-1] + 1$

$\boxed{X[i] = Y[j]}$

If $X[i] \neq Y[j]$

$C[i,j] = \max\{C[i-1, j], C[i, j-1]\}$

$$\therefore C[i,j] = \begin{cases} C[i-1, j-1] + 1 & \text{if } X[i] = Y[j] \\ \max\{C[i, j-1], C[i-1, j]\} & \text{If } X[i] \neq Y[j] \end{cases}$$

$LCS(X, Y, i, j)$

If $X[i] = Y[j]$

then $C[i,j] \leftarrow LCS(X, Y, i-1, j-1)$

else $C[i,j] \leftarrow \max\{LCS(X, Y, i-1, j),$
$LCS(X, Y, i, j-1)\}$

Longest common subsequence

Dynamic Programming

Hall mark #1

optimal substructure

"An optimal solution to a problem contains optimal solutions to subproblems."

LCS – recursive algorithm

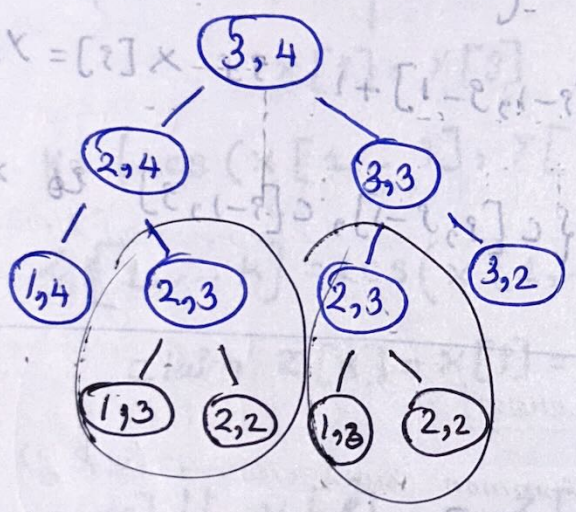
LCS (x, y, i, j)

1) If $x[i] = y[j]$

then $c[i,j] \leftarrow LCS(x, y, i-1, j-1) + 1$

else $c[i,j] \leftarrow \max\{LCS(x, y, i-1, j),$
$LCS(x, y, i, j-1)\}$

worst case if $x[i] \neq y[j]$ → need 2 recursive calls



m+n

Same thing, – same subproblems (repitation)

Hall mark of dynamic prog problem – many repitation of the overlapping subproblems.

Dynamic programming – Hall mark # 2

overlapping subproblems!

★ A recursive solution contain a small # of distinct subproblems repeated many times

↓

Indication: go for dynamic solving.

memorization Algorithm

LCS (X, X, i, 0)

1) If c [i, j] = NIL

Then If X[i] = Y[j]

Then c[i, j] ← LCS (X, Y, i-1, j-1) + 1

Else c[i, j] ← max { LCS (X, Y, i-1, j).
LCS (X, Y, i, j-1) }

Power: Calculated, won't calculated again.

		X →	A	B	C	B	D	A	B
	Y ↓								
empty ←			0	0	0	0	0	0	0
B			0	0	1	1	1	1	1
D			0	0	1	1	2	2	2
C			0	1	1	2	2	2	
A			0						
B			0						
A			0						

c[1, 1] = max { c[0, 1], c[1, 0] }

c[2, 1] = c[1, 0] + 1 → Not same (max + 1)

[B, C] = [4, 3]

max D → 1

max B → 1

many-repitation — Avoided?

Refer: notes

Graphs

G = (V, E)

↳ Edges

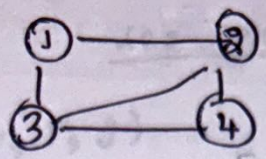
↳ vertices

V = { V1, V2, V3, V4 }

= { 1, 2, 3, 4 }

Directed.

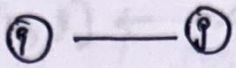
E = { (1,3), (2,1), (2,3), (4,3) }



undirected

'complete' graph - Any two vertex - there is a edge.

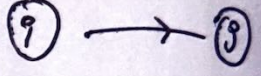
$V = \{1, \ldots, n\}$



$|E| = O(|V|^2)$

$^nC_2 = O(n^2)$

'Adjacency matrix'



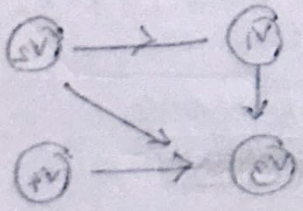
$$a(i,j) = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

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

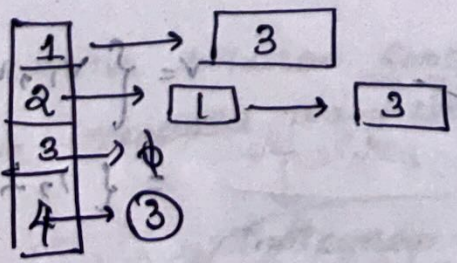
(prev. graph)

Drawback: Sparse graph (less no. of edges)

100 vertex, 10 vertex → No need for such matrix! (wasting memory)



Adjacent list



minimum Spanning tree

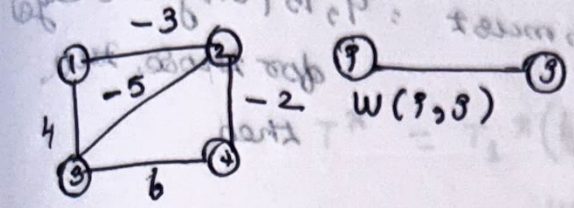
$G = (V, E)$

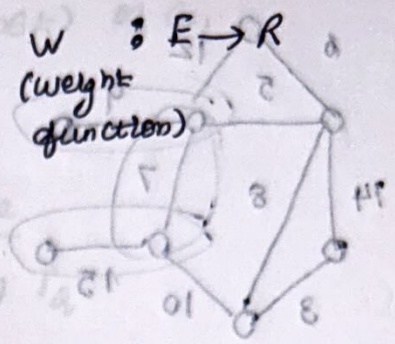
↓

undirected graph

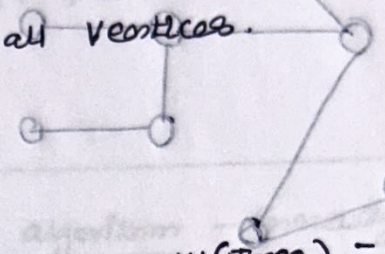
$w : E \to R$

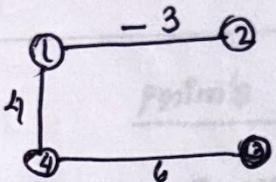
(weight function)





"minimum Spanning tree" ∴ Shouldn't Contain any cycle, Spanning - Covers all vertices.

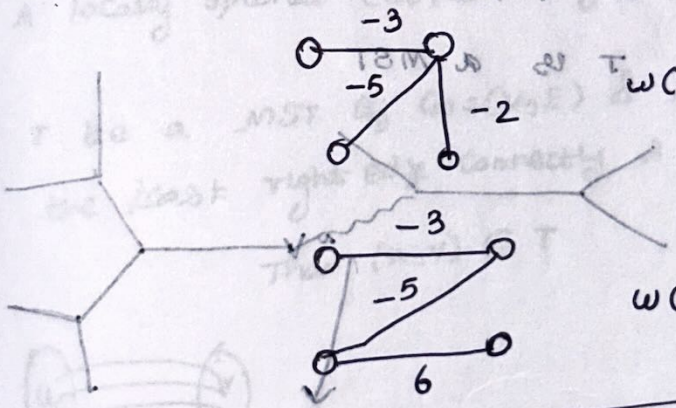




"Tree"

$w(Tree) = \sum w(9,3)$

$= 7$



$w(T) = -10$ → minimum one (Spanning)

$w(T) = -2.$

"Find a Subgraph - Spanning tree"

↓

Find the minimum Spanning one

MST Problem

Input : $< G = (V, E), w : E \to R >$

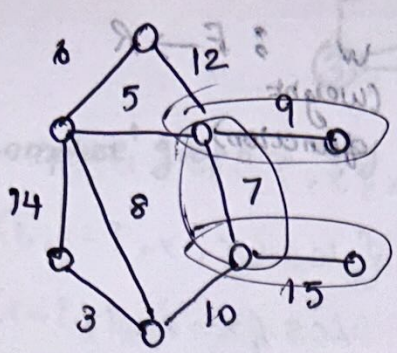
O/P : A spanning tree T

$w(T) = \sum w(u, v)$

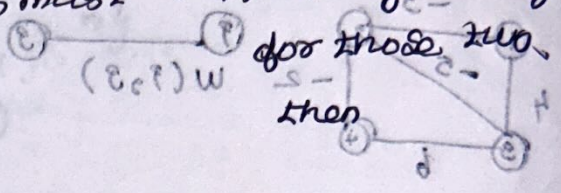
$(u, v) \in T$

$\min \{T | \ldots\}$

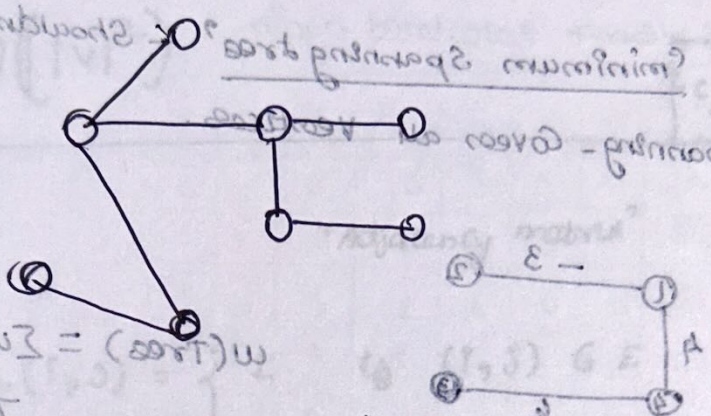
example of MST



which must : 9, 15 (only one edge for those two.



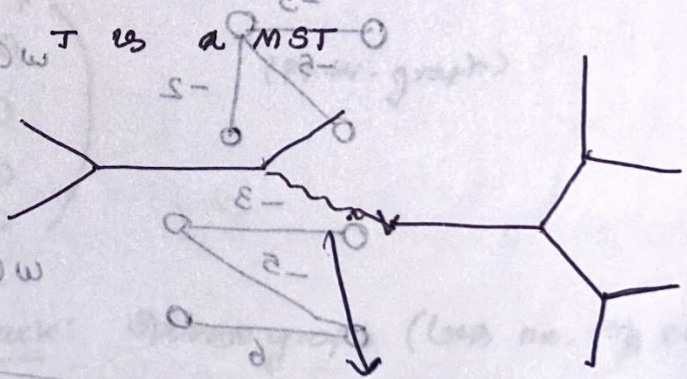
then



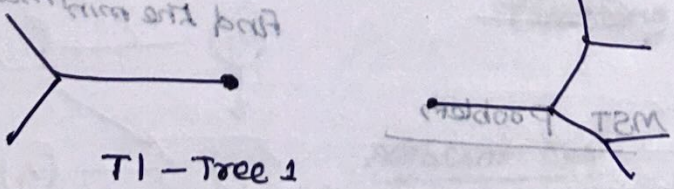
optimal substructure

Let $G_1 = (V, E)$

T is a MST



T: we remove this edge



T1 – Tree 1

claim T_1 is the MST of G_1 (induced by T1)

G_1 has all the edges connecting with the vertices by T_1.

$G_2 \rightarrow$ " " " T_2.

If T_1 – Not a MST of G_1

$\therefore T_1'$ (Another tree – MST) of G_1,

$$\boxed{w(T_1^*) < w(T_1)}$$

Take new one

$$T^* = T_1^* \cup (u,v) \cup T_2$$

Union

$$w(T^*) = w(T^*) + w(u,v) + w(T_2)$$

$$< w(T)$$

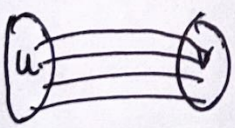
Prim's algorithm – Greedy approach

Hallmark of greedy algorithm – Greedy choice prop.

A locally optimal choice is globally optimal

Let T be a MST of $G=(V,E)$ & $A \subseteq V$. Suppose $(u,v) \in E$ is the least right edge connecting A & V–A. Then $(u,v) \in T$

Then $(u,v) \in T$



'Red notes'