⊙ Learn LaTeX.

* moodle - https://10.5.18.110/moodle/course/id:282

Name- Algorithms - II (CS 31005)

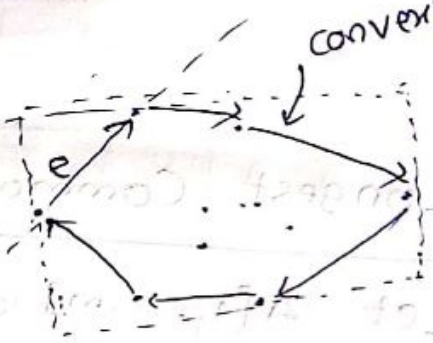
- 2019-20 Autumn

Password for student registration

↳ STU_ALG2

* Find min. perimeter polygon containing all the given points

→ Observations: $[V(P) \subseteq S]$



1. The vertices of the polygon are a subset of the input point set.

2. The min. axis parallel rectangle contains the min. perimeter polygon.

3. If the edges of the polygon are made directed in clockwise direction, every point lies to the right of every directed edge.

$$\downarrow \left[S \cap \text{Left Half}(e) = \emptyset \quad \forall e \in P \right]$$

Let $p, q \in S$

$\Rightarrow \overrightarrow{pq}$ is an edge of P iff $S \cap LH(\overrightarrow{pq}) = \emptyset$

S = input point set having n points.

P = smallest perimeter polygon containing all points of S.

LH(e) = Left Half plane w.r.t. 'e'.

RH(e) = Right Half plane w.r.t 'e'.

* Gift wrapping algo – $O(nh)$

h = # vertices of P

* Longest Common Subsequence (LCS):

Let A[1, ... m] and B[1, ... n] be two strings containing 'm' and 'n' characters respectively.

C[1, ... p] is said to be a/the LCS of A[1, ... m] and B[1, ... n] if:

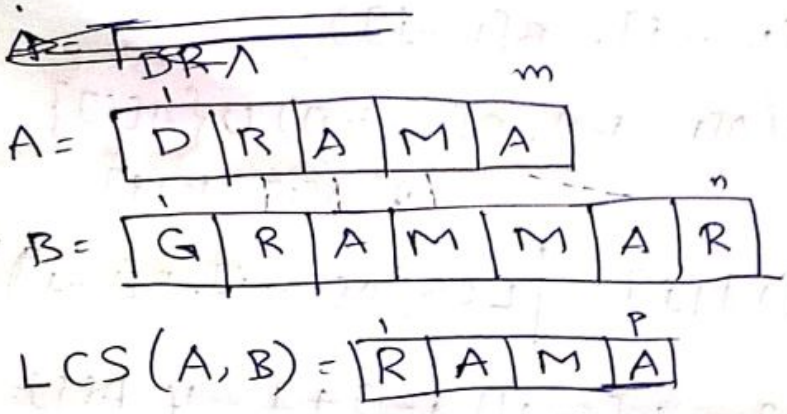
1. $C[k] = A[i] = B[j]$ for $1 \le k \le p$

for some $i \in [1, m]$ and some $j \in [1, n]$, and

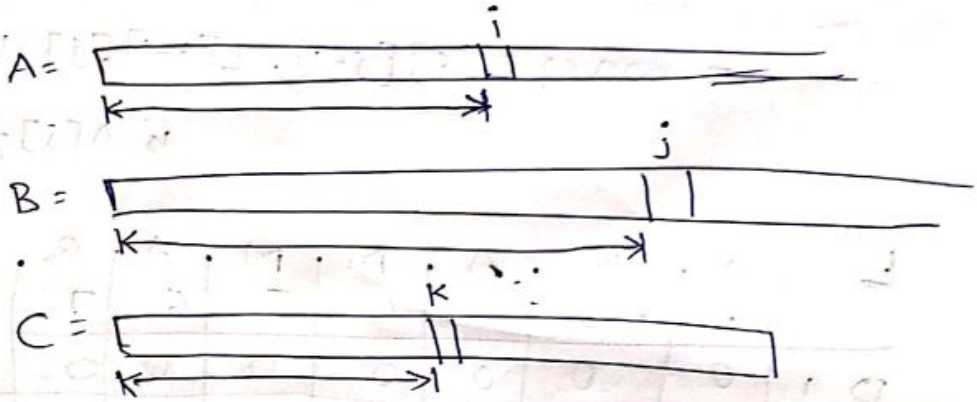
$C[1, \ldots, k-1] = LCS(A[1, \ldots i-1], B[1, \ldots j-1])$

2. Value of P is maximum.

Ex:-



*



$C[1, \ldots k-1] = LCS(A[1, \ldots i-1], B[1, \ldots j-1])$

if $C[k] = A[i] = B[j]$

* Dynamic Programming (DP):-

1. Optimal Substructure:-

$C[1, \ldots p-1] = LCS(A[1, \ldots m-1], B[1, \ldots n-1])$

→ $C[p] = A[m] = A[n]$ if $A[m] = B[n]$

→ If $A[m] \neq B[n]$ ⇒ One of the following will be true:

1. $C[p] = A[m] \Rightarrow C = LCS(A[1, \ldots m], B[1, \ldots n-1])$
2. $C[p] = B[n] \Rightarrow C = LCS(A[1, \ldots m-1], B[1, n])$
3. $C[p] \notin \{A[m], B[n]\}$

↳ $C = LCS(A[1, \ldots m-1], B[1, \ldots n-1])$

2. Overlapping Subproblems:-

$LCS(A[1, \ldots i], B[1, \ldots j])$

$= LCS(A[1, \ldots i-1], B[1 \ldots j-1]) \cup$

$LCS(A[1,\ldots i], B[1,\ldots j])$

$= LCS(A[1,\ldots i-1], B[1,\ldots j-1]) \cup \{A[i]\}$ if $A[i] = B[j]$

Let $L[i][j] = |LCS(A[1,\ldots i], B[1,\ldots j])|$

$\Rightarrow L[i][j] = L[i-1][j-1] + 1$ if $A[i] = B[j]$

$= \max\{L[i][j-1], L[i-1][j], L[i-1][j-1]\}$ if $A[i] \neq B[j]$

L		G	R	A	M	M	A	R
		1	2	3	4	5	6	7
D	1	0	0	0	0	0	0	0
R	2	0	1	1	1	1	1	1
A	3	0	1	2	2	2	2	2
M	4	0	1	2	3	3	3	3
A	5	0	1	2	3	3	4	4

* Tutorial Problem 1 :

Let $A[1,\ldots m]$ and $B[1,\ldots n]$ be two 1D arrays containing m & n integers respectively, where $m \leq n$. We have to construct a sub-array $C[1\ldots m]$ of B such that $\sum_{i=1}^{m} |A[i] - C[i]|$ is minimized.

Develop the ~~resource~~ recurrences needed for DP with clear arguments. Design the algorithm & write the pseudo code. Demonstrate your algorithm on a

few input instances.
Derive its time & space complexities.

* Matrix Chain Multiplication

$A_1 \cdot A_2 \cdot A_3 = (A_1 A_2) A_3$
$= A_1 (A_2 A_3)$

let $A_1 : 20 \times 10$
$A_2 : 10 \times 5$
$A_3 : 5 \times 30$

$A_1 A_2 : 20 \times 5 \Rightarrow$ No of Multiplications
$= 20 \times 5 \times 10$
$= 1000$

$A_{123} = 20 \times 30 \Rightarrow$ # Multiplications
$= 20 \times 30 \times 5$
$= 3000$

$\Rightarrow$ Total # Multiplications = 4000 for $(A_1 A_2) A_3$

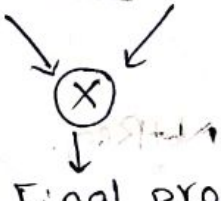
$\Rightarrow$ Total " $= 10 \times 5 \times 30 + 20 \times 10 \times 30$
for $A_1 (A_2 A_3)$

$= 1500 + 6000$
$= 7500$

$\therefore$ Min. cost multiplication is for $(A_1 A_2) A_3$

$\rightarrow$ n matrices $A_i : r_{i-1} \times r_i$

$(A_1 A_2 A_3 \cdots\cdots A_k)(A_{k+1} \cdots\cdots A_n)$



$k = ?$ To minimize the # multiplications.
$(1 \le k \le n-1)$

* Define $m(i,j)$ = min # multiplications for ~~$(A_i A_{i+1} \cdots\cdots A_j)$~~
$(A_i A_{i+1} \cdots\cdots A_k A_{k+1} \cdots\cdots A_j)$

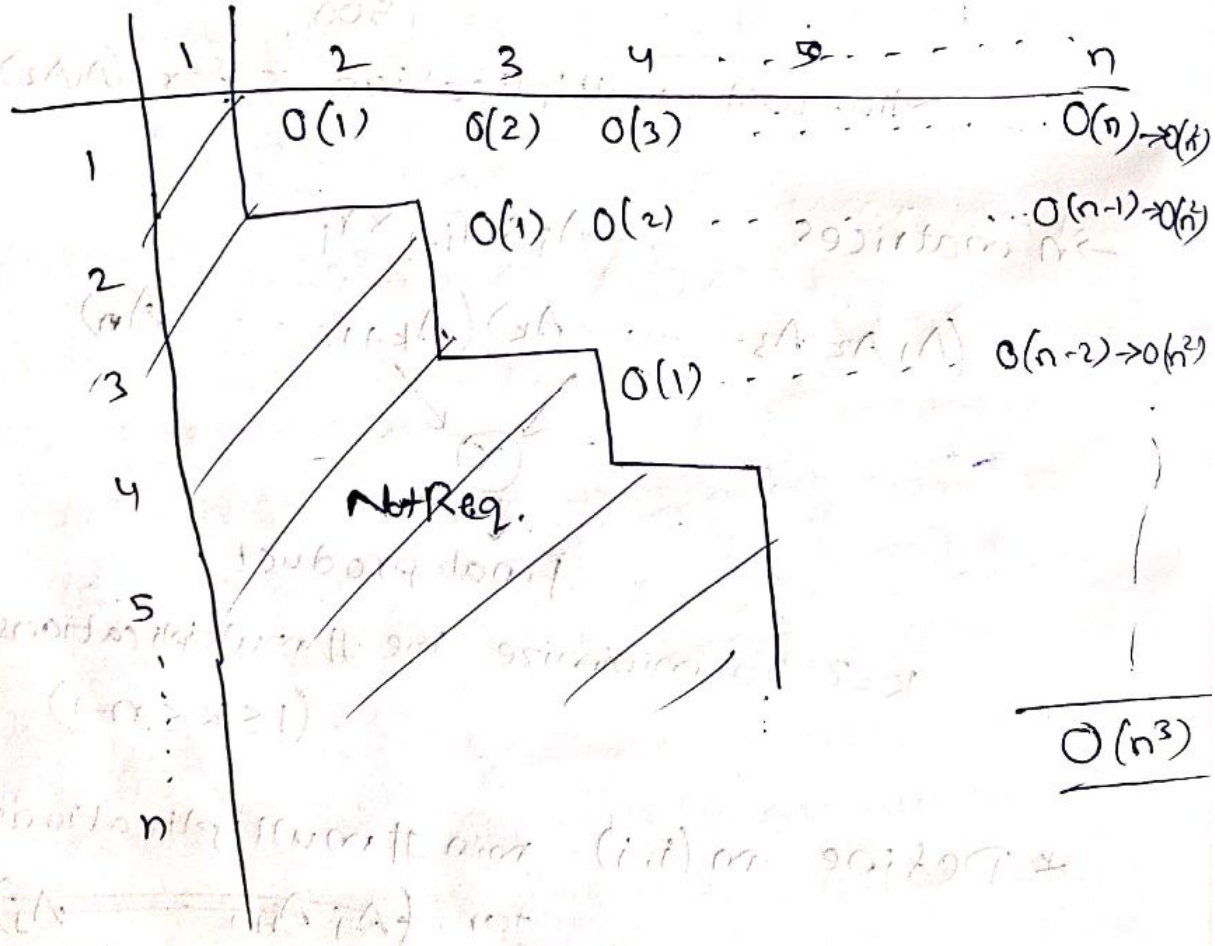
$\therefore m(i,j) = \{m(i,k) + m(k+1,j) + r_{i-1} \times r_k \times r_j\}$

$\therefore m(i,j) = \min_{i \le k \le j-1} \{m(i,k) + m(k+1,j) + r_{i-1} \times r_k \times r_j\}$

* $A_1 = 5 \times 10$
  $A_2 = 10 \times 5$
  $A_3 = 5 \times 5$
  $A_4 = 5 \times 20$
  $A_5 = 20 \times 10$

2D array m

i \ j	1	2	3	4	5
1	0	250			
2	×	0	250		
3	×	×	0	500	
4	×	×	×	0	1000
5	×	×	×	×	0

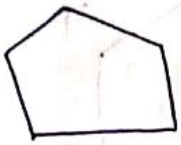


$O(n^3)$

* Minimum-ink triangulation of a convex Polygon (P):

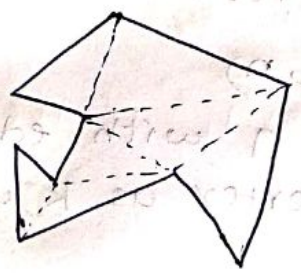
Def: A Polygon is said to be CONVEX if the line segment joining any two vertices of the polygon lies inside the polygon.

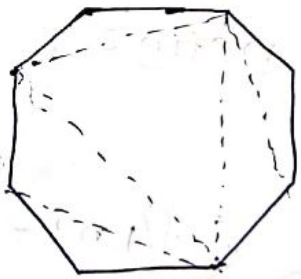
Ex:



* Triangulation:

Triangulation of a polygon is the collection of triangles obtained by adding diagonals such that the intersection between the interiors of any two triangles is empty.





* Minimum Ink

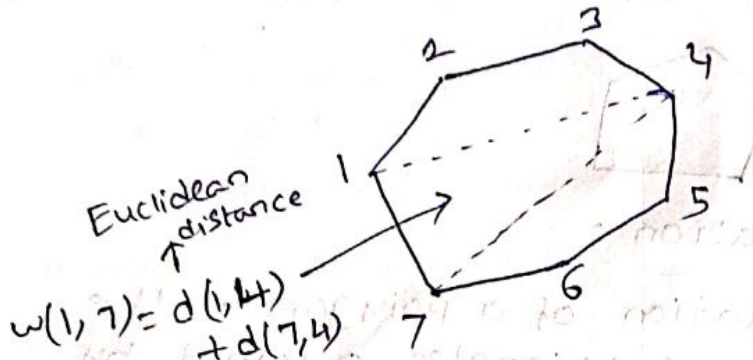
→ sum of lengths of the diagonals is minimum.

Observations:

→ # Δ'les = $n-2$ for any triangulation where 'n' is the no. of vertices of 'P'.

→ # diagonals = n-3

→ Each edge of P will be an edge of some triangle 't' of the Triangulation (T) whether T is min-ink or not



→ Every diagonal of T is incident on exactly two triangles of T

→ Vertex 1 & vertex n

→ Edge (1, n)

∴ A triangle in T with edge (1,n) has the third vertex as k, where

$2 \le k \le n-1$

* Define : $f(1,n)$ = Sum of length of the diagonals in T (min-ink)

$$\therefore f(1,n) = \min_{2 \le k \le n-1} \{ f(1,k) + f(k,n) + w(t(1,k,n)) \}$$

$f(1,2) = 0$

$f(n-1,n) = 0$

optimal sub-structure of DP

triangle with vertices 1, k & n.

Overlapping sub-problems.

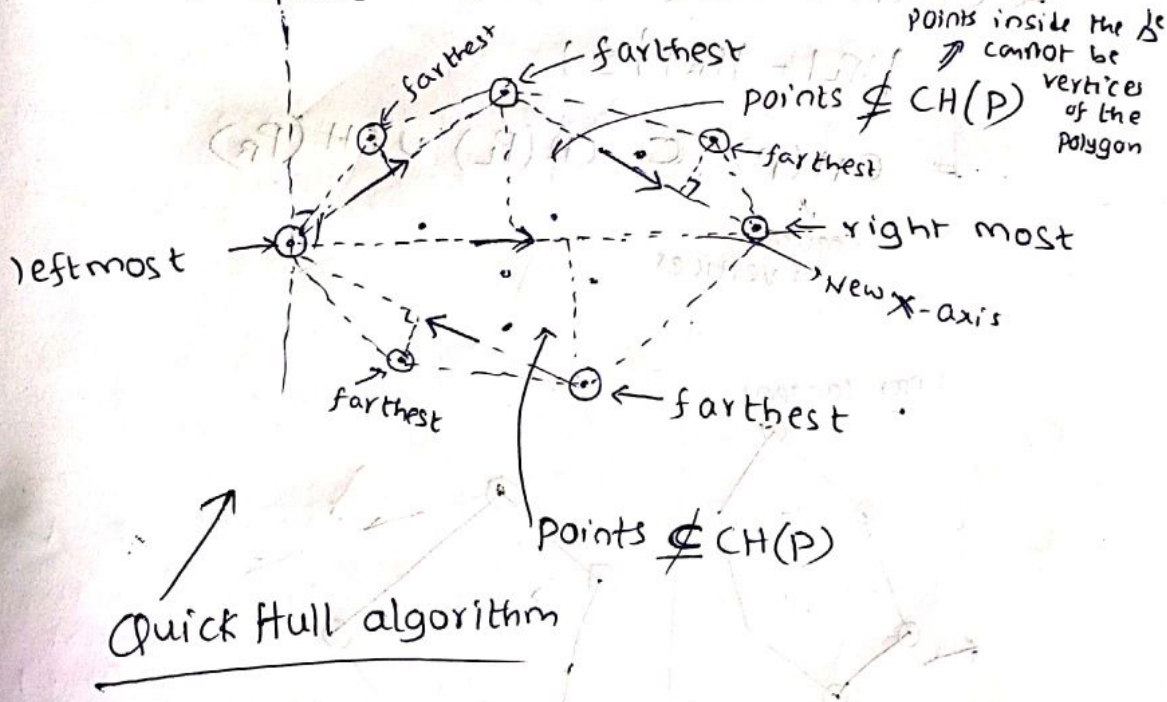
$$f(i,j) = \begin{cases} 0 & \text{if } j-i \le 1 \\ \min\limits_{i+1 \le k \le j-1} \{ f(i,k) + f(k,j) + w(t(i,k,j)) \} \end{cases}$$

$w(t(i,k,j))$ ← Sum of diagonals in the Δ^{le}

* Convex Hull :-

Let 'P' be a set of 2-dimensional points. The convex hull CH(P) of P is the smallest-perimeter polygon that contains all points of P.

→ The vertices of CH(P) are described in a specific order, say clockwise.



Quick Hull algorithm

Time: $T(n) = T(\alpha n) + T((1-\alpha)n) + O(n)$

$O(n)$ → to partition.

$T(\alpha n) + T((1-\alpha)n)$ → for balanced partition

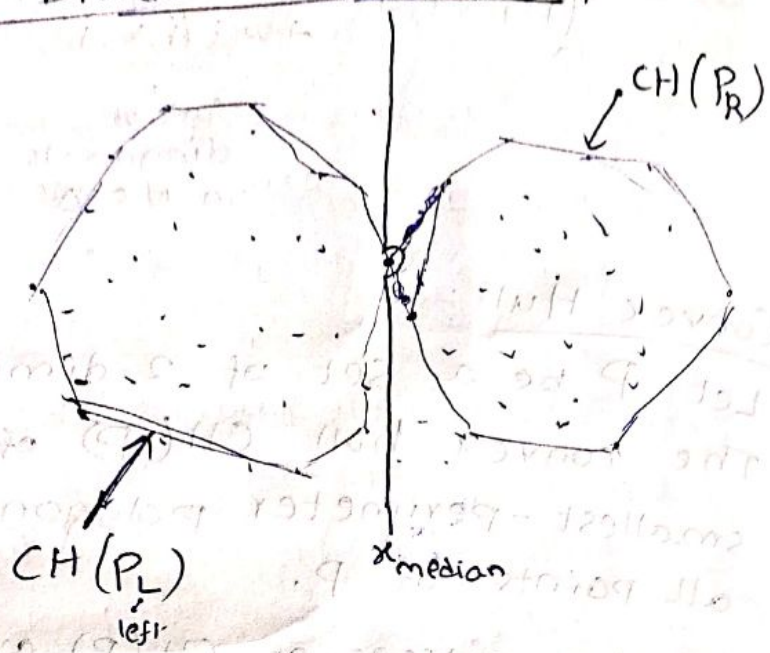
Best case :- α is a constant

$\Rightarrow T(n) = O(n \log n)$

Worst case : $T(n) = T(1) + T(n-2) + O(n)$

$\Rightarrow T(n) = O(n^2)$

* **CH(P) by Divide and Conquer :-**



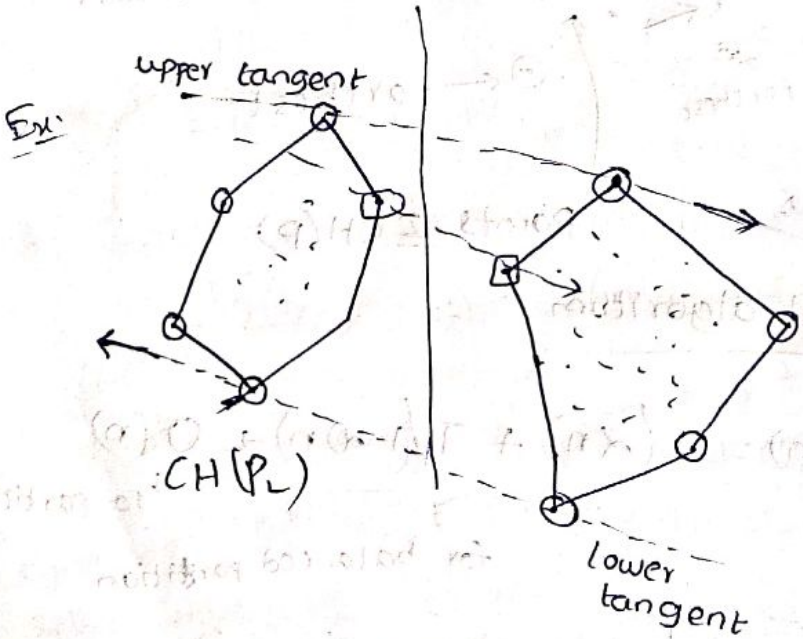
* $CH(P) = CH(P_L \cup P_R)$

$||P_L| - |P_R|| \leq 1$

$\therefore CH(P) \subseteq CH(P_L) \cup CH(P_R)$

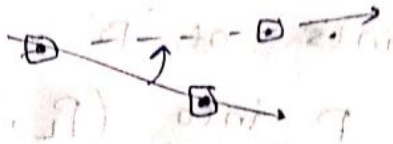
↓ sequence of vertices

Ex:

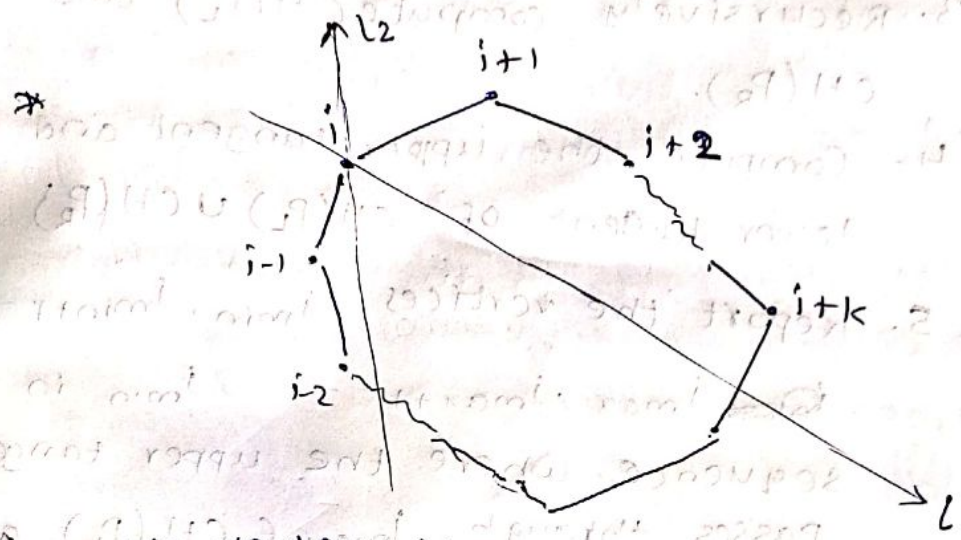


→ Consider the directed line joining the rightmost point of $CH(P_L)$ & leftmost of $CH(P_R)$. If any point in $CH(P_R)$

lies to the left of it, move the line to one of those points in $CH(P_R)$. [or next vertex to that vertex]



→ similarly, if any point in $CH(P_L)$ lie to left, move the line.



No vertex lies to the left of 'l' if the vertex 'i+1' lies to the right of 'l'. (or) vertex (i-1)

⇓

finding if any points lie to the the left of line in the above algo. takes $O(1)$ time.

⇒ No. of such lines to check = $O(n)$

$$\therefore T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$\Rightarrow \boxed{T(n) = O(n \log n)}$$

→ Steps :

1. Find the point whose x-coordinate is the median of the x-coordinates of all points of 'P'.
2. Partition P into (P_L, P_R) s.t. x of each point of P_L is less than x of each point of P_R.
3. Recursively compute $CH(P_L)$ and $CH(P_R)$.
4. Compute the upper tangent and the lower tangent of $CH(P_L) \cup CH(P_R)$
5. Report the vertices $i_{min}, i_{min+1}, \ldots, i_{max}, j_{max}, j_{max+1}, \ldots, j_{min}$ in sequence where the upper tangent passes through $i_{max} \in CH(P_L)$ and $j_{max} \in CH(P_R)$ and lower tangent passes through $i_{min} \in CH(P_L)$ and $j_{min} \in CH(P_R)$.

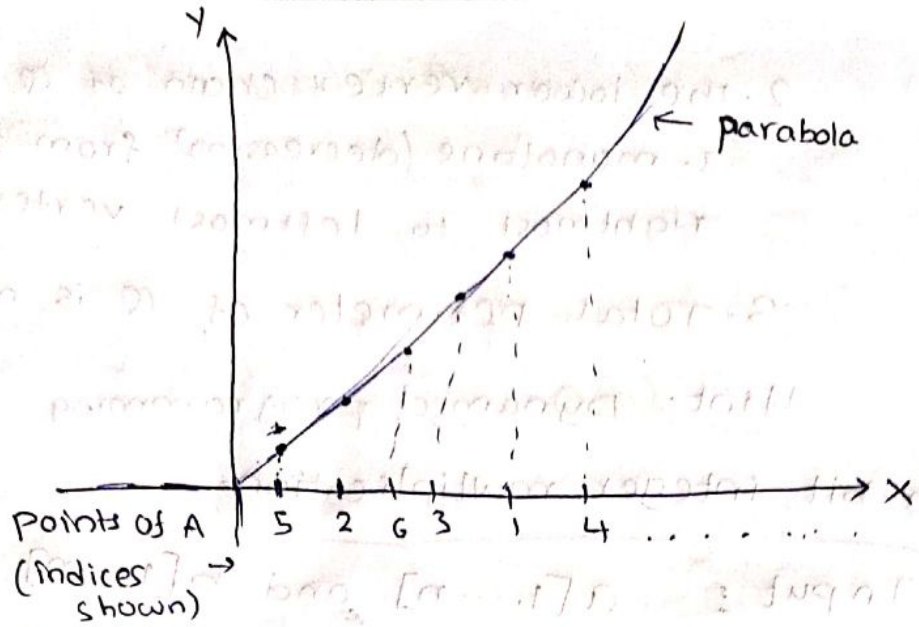
* Sorting of 'n' numbers cannot be done in less than $O(n \log n)$ time [in the worst case]

↳ Convex hull of 'n' points cannot be completed in $O(n \log n)$ time.

Why?

Let A[1...n] be an array of 'n' numbers,

Define P[1...n] be a set of 2D points in which $x(P[i]) = A[i]$ and $y(P[i]) = (A[i])^2$



So, all points of P will lie on a parabola whose axis is parallel to y-axis.

As a parabola is convex from its interior, all points of 'P' will be vertices of CH(P).

A convex-hull algorithm will report these vertices (i.e, all points of P) in clockwise ~~direct~~ order and hence we can find the sorted sequence of A.

If the convex hull algo ~~of~~ runs in less than O(nlogn) time, then A gets sorted in less than O(nlogn) time

↳ A contradiction

* Tutorial T2:

Input - P = a set of 'n' 2D points

output - Q = a polygon whose vertex set is 'p' in an order such that

1. The upper vertex chain of Q is x-monotone increasing from the leftmost ~~&~~ vertex to the rightmost vertex of P.

2. The lower vertex chain of Q is x-monotone (decreasing) from the rightmost to leftmost vertex of P.

3. Total perimeter of Q is minimum.

Hint:- Dynamic programming

* n-bit integer multiplication:-

Input : $a[1 \ldots n]$ and $b[1 \ldots n]$ are two integers in binary number system, each comprising 'n' bits.

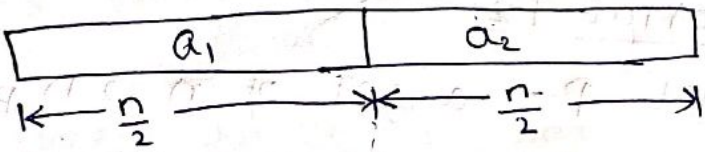
Output : The product of multiplication, $c = a \cdot b$.

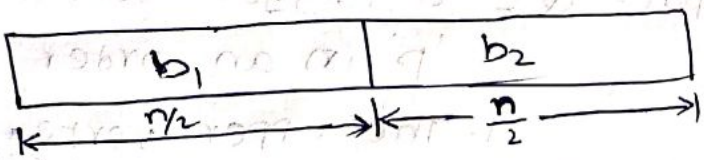
Ex:- $a = 1101$

$b = 1010$

```
    0000
   1101X
  0000XX
 1101XXX
--------
10000010
```

$\rightarrow T(n) = O(n^2)$

* $a =$ 

$b =$ 

Assume 'n' is a power of 2.

$a = a_1 \cdot 2^{n/2} + a_2$

$b = b_1 \cdot 2^{n/2} + b_2$

$$ab = (2^{n/2}a_1 + a_2)(2^{n/2}b_1 + b_2)$$

$$= 2^n a_1b_1 + 2^{n/2}(a_1b_2 + a_2b_1) + a_2b_2$$

Let $T(n)$ be the time complexity for multiplication of two n-bit numbers.

Then, $T(n) = 4T\left(\frac{n}{2}\right) + O(n)$

$$\Rightarrow T(n) = O(n^2)$$

Karatsuba ↙

$$a_1b_2 + a_2b_1 = (a_1 + a_2)(b_1 + b_2) - a_1b_1 - a_2b_2$$

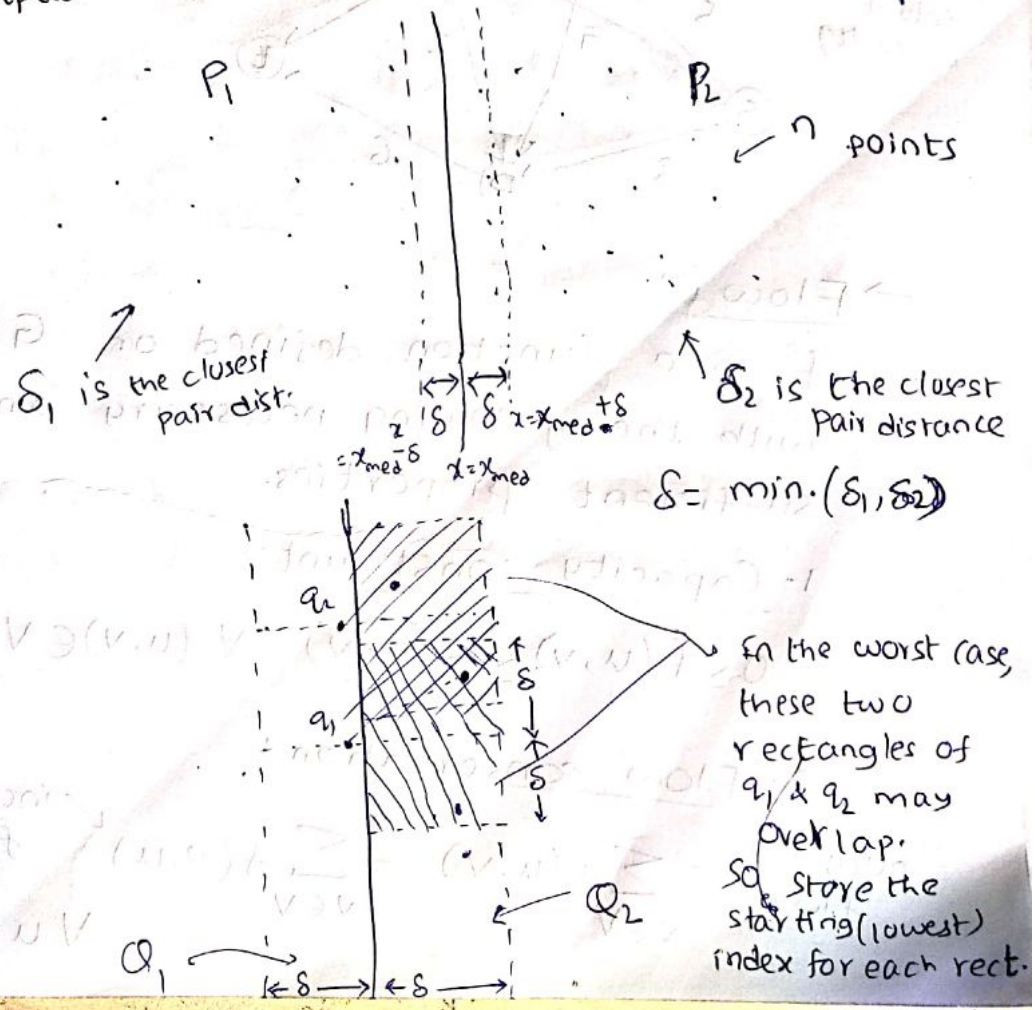
$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$

$$\Rightarrow T(n) = O(n^{\log_2 3}) \approx O(n^{1.59})$$

* Closest pair in a 2D point set :

Input - P = a list of 'n' 2D points

Output - The closest pair of points in P



* Time taken to process the points of Q_2 corresponding to ~~each~~ all points of Q_1

$= O(n)$.

∴ Total time complexity

$= O(n \log n)$ for sorting (w.r.t x & y) $+ T(n)$.

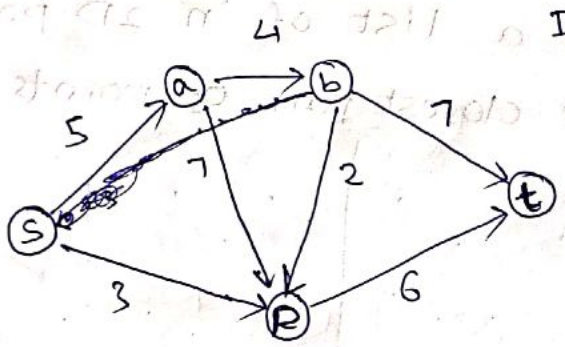
where $T(n) = 2T(\frac{n}{2}) + O(n)$

$\Rightarrow T(n) = O(n \log n)$

* Flow Network

A directed graph $G(V, E)$ with ~~the~~ two distinct vertices called the source (s) and the sink (t) and each edge (u,v) having a capacity $c(u,v) \geq 0$.

[Source can have edges into it as well]



If $(u,v) \notin E$, $c(u,v) = 0$

→ Flow (f):-

f is a function defined on $G(V, E)$ with the following necessary and sufficient properties.

1. Capacity constraint:-

$$0 \leq f(u,v) \leq c(u,v) \quad \forall (u,v) \in V^2$$

2. Flow conservation:-

$$\text{outgoing flow} \leftarrow \sum_{v \in V} f(u,v) = \sum_{v \in V} f(v,u) \leftarrow \text{incoming flow}$$

$\forall u \in V \setminus \{s,t\}$

* Value of a flow f :

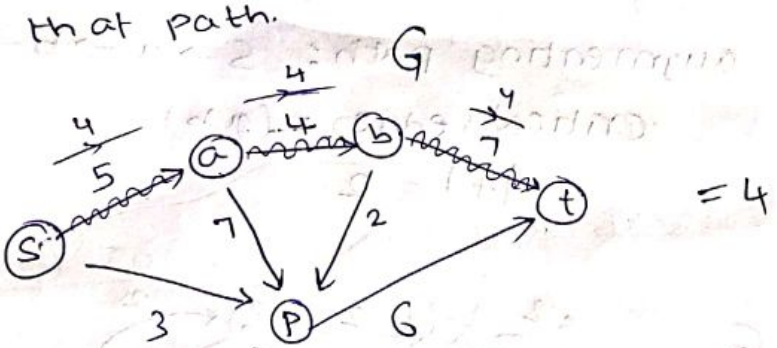
$$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s)$$

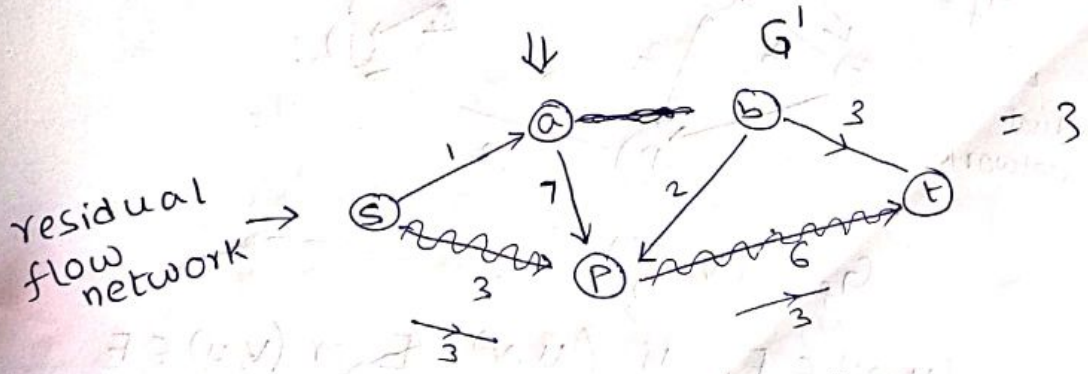
* Augmenting path

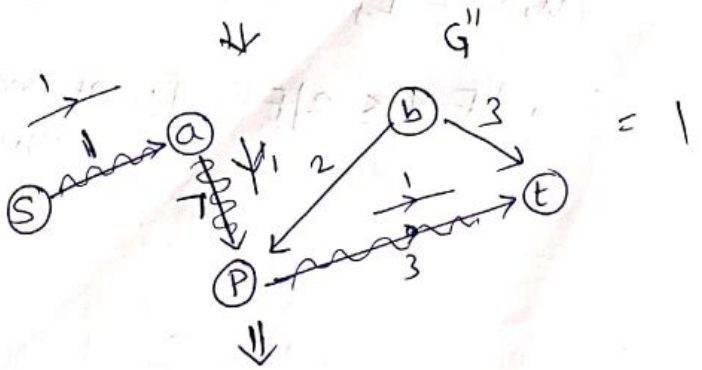
A path from 's' to 't' along which a flow is possible.

critical edge → edge which has the minimum capacity in the augmenting path.

Maximum flow in an augmenting path equals the critical edge capacity in that path.



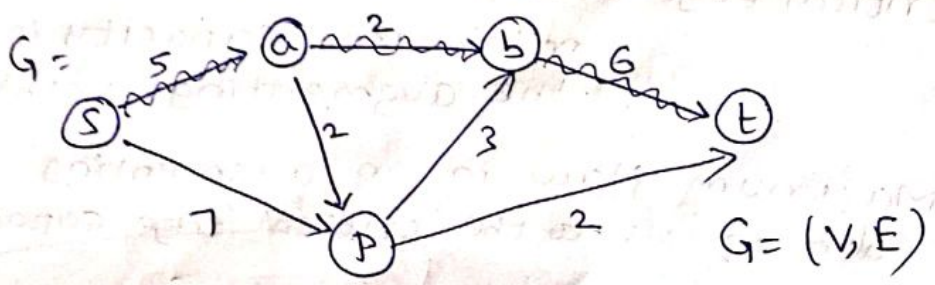




$\therefore |f|_{max} = 4+3+1 = 8$

* If f_1 is a flow in G & f_2 is a flow in G_1 (residual graph of G)
then $\underbrace{f_1 \uparrow f_2}_{\| \; f_1 + f_2}$ is a flow in G

*

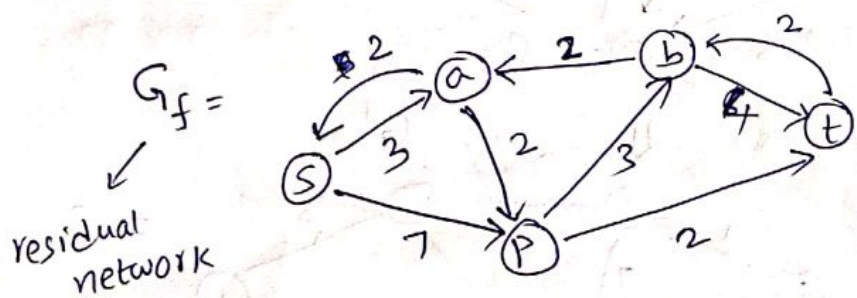


$G = (V, E)$

augmenting path: $s \to a \to b \to t$

critical edge = (a, b)

$|f| = 2$



$G_f = (V, E_f)$

$(u, v) \in E_f$ if $(u, v) \in E$ or $(v, u) \in E$

$\Rightarrow |E_f| \leq 2|E|$ [∵ at max., the augmenting path may be the whole graph]

* Theorem:- Let "f" be a flow in G and f' be a flow in G_f.

Then $f \uparrow f'$ is a flow in G and its value is $|f \uparrow f'| = |f| + |f'|$

→ Let $(u,v) \in V^2$

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \\ f(v,u) & \text{if } (v,u) \in E \\ 0 & \text{otherwise} \end{cases}$$

Residual capacity

$f(u,v) \leq c(u,v) \quad \forall (u,v) \in V^2$

$f'(u,v) \leq c_f(u,v) \quad \forall (u,v) \in V^2$

→ Proof :-

Capacity constraint :- Let $u, v \in V$

$(f \uparrow f')(u,v) = f(u,v) + f'(u,v)$

~~$\leq f(u,v) + f'(u,v)$~~

If $(u,v) \in E$,

Then $f'(u,v) \leq c_f(u,v)$

$(f \uparrow f')(u,v) \leq f(u,v) + c_f(u,v)$

$= f(u,v) + c(u,v) - f(u,v)$

$\therefore (f \uparrow f')(u,v) \leq c(u,v)$

If $(u,v) \notin E$ but $(v,u) \in E$

$f'(u,v) \leq c_f(u,v) = f(v,u)$

$\therefore (f \uparrow f')(u,v) \leq f(u,v) + f(v,u)$

$(u,v) \notin E \Rightarrow c(u,v) = 0 \Rightarrow f(u,v) = 0$

$\therefore (f \uparrow f')(u,v) \leq f'(v,u) \leq c_f(u,v)$

* Flow conservation : $u \in V - \{s, t\}$

$$\sum_{v \in V} (f \uparrow f')(u,v) = \sum_{v \in V} f(u,v) + \sum_{v \in V} f'(u,v)$$

$$= \sum_{v \in V} f(v,u) + \sum_{v \in V} f'(v,u)$$

$$= \sum_{v \in V} (f \uparrow f')(v,u)$$

* value of $(f \uparrow f')$:

$$|f + f'| = \sum_{v \in V} (f + f')(s,v) - \sum_{v \in V} (f + f')(v,s)$$ by defn

Let $V_1 = \{v \in V \mid (s,v) \in E\}$ and

$V_2 = \{v \in V \mid (v,s) \in E\}$

Then $V_1 \cap V_2 = \emptyset$ [$\because (u,v) \in E \Rightarrow (v,u) \notin E$]

$$\Rightarrow |f + f'| = \sum_{v \in V_1} (f + f')(s,v) - \sum_{v \in V_2} (f + f')(v,s)$$

$$= \sum_{v \in V_1} f(s,v) + \sum_{v \in V_1} f'(s,v) - \sum_{v \in V_2} f(v,s) - \sum_{v \in V_2} f'(v,s)$$

$$= \left(\sum_{v \in V_1} f(s,v) - \sum_{v \in V_2} f(v,s)\right) + \left(\sum_{v \in V_1} f'(s,v) - \sum_{v \in V_2} f'(v,s)\right)$$

$$= \left(\sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s)\right) + \left(\sum_{v \in V} f'(s,v) - \sum_{v \in V} f'(v,s)\right)$$

$$= |f| + |f'|$$

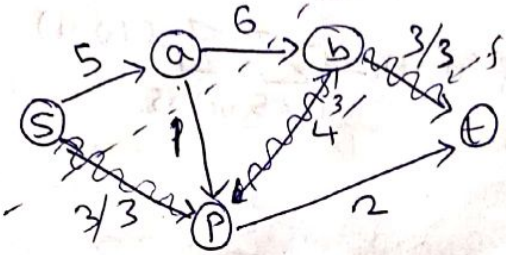
* Cut in a flow network:-

- s = source, t = sink

$S = \{s\} \cup \{\text{some vertices of } V \text{ excepting } t\}$

$T = V \setminus S$.

$\Rightarrow (S, T)$ is a cut in $G(V, E)$.



$\Rightarrow S = \{s, a\}$, $T = \{b, p, t\}$

$f(S, T) = 3 + 0 + 0 = 3$ flow of cut
(3 edges going out)
only one edge has flow

$c(S, T) = 3 + 1 + 6 = 10$ capacity of cut

* Lemma:-

For any cut (S, T) and for any flow f,

$f(S, T) = |f|$.

~~Fact~~: $f(S, T) \leq c(S, T)$

$\Rightarrow |f|_{max} = \min_{(S,T)} \{c(S, T)\}$

* Max-flow min-cut theorem:-

The following statements are equivalent:

1. f_{max} is maximum flow in G.
2. $G_{f_{max}}$ (residual network) has no augmenting path.

3. $|f_{max}| = \min_{(S,T)} \{C(S,T)\}$

→ Proof of lemma:

$f(S,T)$ = flow ~~total~~ outgoing − flow incoming

$$\therefore f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{u \in S} \sum_{v \in T} f(v,u)$$

$$= \sum_{u \in S} \sum_{v \in V-S} f(u,v) - \sum_{u \in S} \sum_{v \in V-S} f(v,u)$$

$$= \sum_{u \in S} \sum_{v \in V} f(u,v) - \sum_{u \in S} \sum_{v \in S} f(u,v)$$

$$- \sum_{u \in S} \sum_{v \in V} f(v,u) + \sum_{u \in S} \sum_{v \in S} f(u,v)$$

$$= \sum_{u \in S} \sum_{v \in V} (f(u,v) - f(v,u))$$

$$+ \sum_{u \in S} \sum_{v \in S} (f(u,v) - f(v,u))$$

$$= \sum_{u \in S} \sum_{v \in V} (f(u,v) - f(v,u))$$

$$= |f|$$

* Ford and Fulkerson Algorithm

Based on augmenting paths

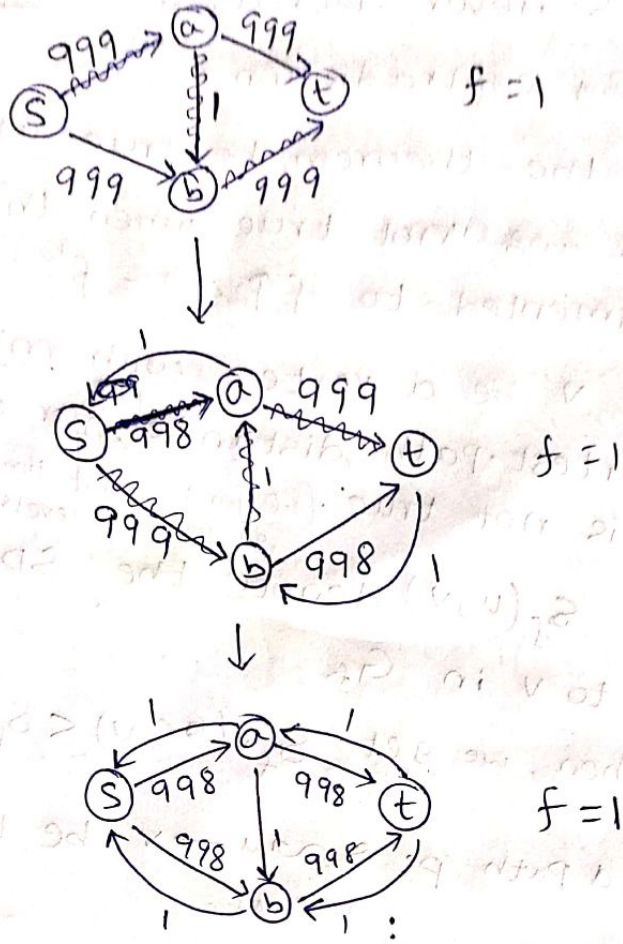
$G = (V, E)$

$G_f = (V, E_f)$

$|E_f| \leq 2|E|$

T(finding an aug. path) = O(|E|).

#aug. paths = ? ≤ |f_{max}|



⇒ T(finding flow) ≐ O(|E| |f_{max}|).

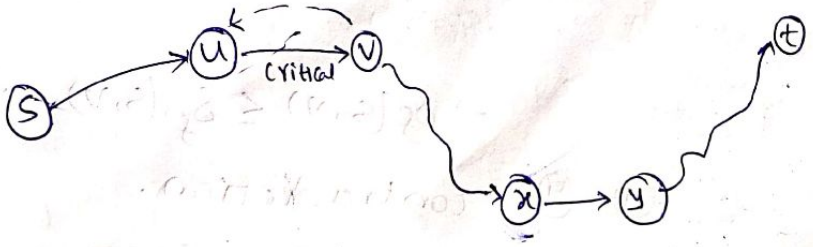
* Contribution by Edmonds and Karp

Use shortest augmenting path (in terms of no. of edges)

↙

can be found by using BFS.

→ Distance (level) of 't' from from 's' never decreases over successive flow augmentations.



*Theorem:- Over successive flow of augmentations the shortest-path distance of a vertex 'v' from 's' never decreases.

(for augmentations ~~Any augmentation~~ ~~not only~~ by the algorithm above)

Proof :- By contradiction

Let the theorem be true upto a flow f and ~~not~~ not true when the flow is augmented to $f \uparrow \Delta f = f'$

Let 'v' be a vertex with minimum shortest-path distance$^{(SD)}$ for which it is not true. (i.e for the first time when we go down the levels in BFS)

The theorem may fail for many vertices in $G_{f'}$. But 'v' has the min. shortest-path distance among them. Since 'u' has lesser S.D. than 'v'.

Let $\delta_f(u,v)$ denote the SD from u to v in G_f

Then, we get $\delta_{f'}(s,v) < \delta_f(s,v) \longrightarrow (1)$

Let a Path $p: s \rightsquigarrow u \rightarrow v$ be the shortest (path, edge)

path in $G_{f'}$.

Then $\delta_{f'}(s,u) \geq \delta_f(s,u) \longrightarrow (2)$

[∵ for 'u', the theorem holds.]

and $\delta_{f'}(s,u) = \delta_{f'}(s,v) - 1 \longrightarrow (3)$

Let $(u,v) \in E_f$

~~Then $\delta_f(s,u) \leq \delta_f(s,v)$~~

Then $\delta_f(s,v) \leq \delta_f(s,u) + 1 \leq \delta_{f'}(s,u) + 1$ [from (2)]

$\Rightarrow \delta_f(s,v) \leq \delta_{f'}(s,v) \longrightarrow (4)$

(1) & (4) $\Rightarrow$ contradiction.

$\Rightarrow (u,v) \notin E_f$

But $(u,v) \in E_{f'} \Rightarrow (v,u) \in E_f$, $\Delta_f(v,u) > 0$

~~$\Rightarrow \delta_f(s,u) \leq \delta_f(s,v) + 1$~~

~~from (3) $\Rightarrow \delta_{f'}(s,v) = \delta_{f'}(s,u) + 1$~~

~~$\therefore \delta_{f'}(s,v) \geq \delta_f(s,u) + 1$~~

∴ The shortest path from s to v includes s to v. since that is the augmenting path for Δf

i.e (v,u) is an edge in the augmenting path through which Δf has been pushed

$\therefore \delta_f(s,u) = \delta_f(s,v) + 1$

from (3) $\Rightarrow \delta_{f'}(s,v) = \delta_{f'}(s,u) + 1$

$\delta_{f'}(s,v) \geq \delta_f(s,u) + 1$

$\Rightarrow \delta_{f'}(s,v) \geq \delta_f(s,v) + 2$

∴ (5) which is a contradiction.

*Theorem: Any edge of a flow network can become "critical" for at most $O(V)$ times, where V = vertex set. ↳ same for above algo.

~~Proof~~: So #augmenting paths = $|E|O(V) = O(VE)$

Proof: Let (u,v) be any edge @ in $G(V,E)$

~~Let f' be the flow when (u,v) becomes critical.~~

Let f be the flow upto which (u,v) is not critical and ~~$f \uparrow \Delta f$ be the~~ $f \uparrow \Delta f = f'$ be the augmented flow when (u,v) is critical.

$\Rightarrow (u,v) \notin E_f$ and $(v,u) \in E_f$

$\therefore \delta_f(s,v) = \delta_f(s,u) + 1 \longrightarrow ①$

$\delta_{f'}(s,u) \leq \delta_{f'}(s,v) + 1 \longrightarrow ②$ $[\because (v,u) \in E_{f'}]$

As $(u,v) \in E_{f'}$, (u,v) can reappear as an edge in ~~the~~ the very next ~~some~~ subsequent flow augmentation only if (v,u) is an edge of the corresponding aug. path.

$\Rightarrow \delta_{f'}(s,v) = \delta_{f'}(s,u) - 1$

$\Rightarrow \delta_{f'}(s,u) = \delta_{f'}(s,v) + 1 \longrightarrow ③$

we have, $\delta_{f'}(s,v) \geq \delta_f(s,v)$ (from Previous theorem).

$\therefore \delta_{f'}(s,u) \geq \delta_f(s,v) + 1$

$\delta_{f'}(s,u) \geq \delta_f(s,u) + 2$ [from ①]

Shortest path

$\Rightarrow$ Distance of 'u' from 's' increases by atleast '2' from its one critical step to the next.

$\therefore$ (u,v) can become critical for atmost $\frac{|V|}{2}$ times.

→ Time Complexity of Edmonds-Karp algorithm:

$O(E)$ to find each aug. path (S.P. from 's' to 't') by using BFS

(∵ it's connected graph, $O(V+E) \equiv O(V)$)

$O(VE)$ ~~time to find~~ # of augmenting paths

$\Rightarrow O(VE^2)$

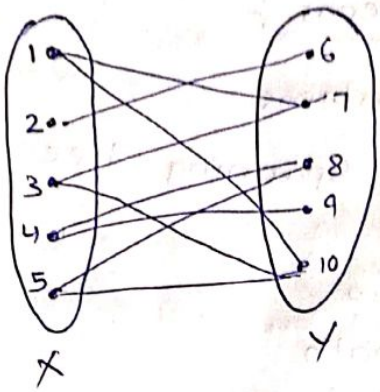
* Bipartite Graph Matching:

$G = (V, E)$

$V = X \cup Y$

$X \cap Y = \emptyset$

$\forall (u, v) \in E$, $u \in X$ & $v \in Y$



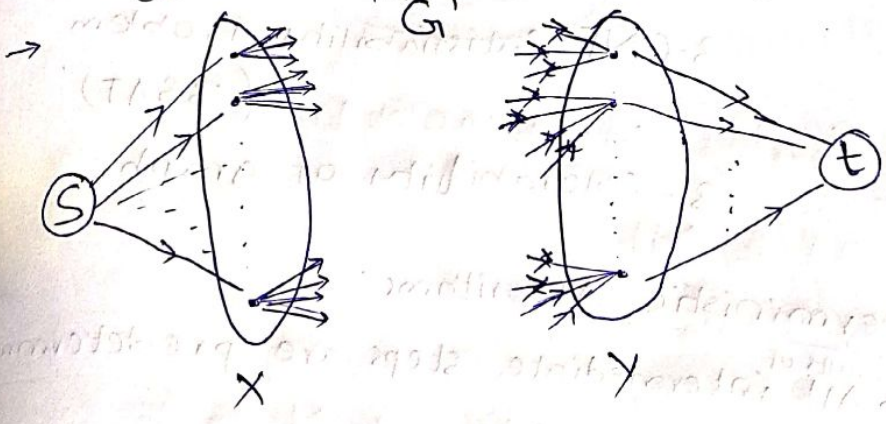
Matching, $M = \{(x, y) : x \in X, y \in Y\}$ and

- if $(x, y) \in M$ & $(u, v) \in M$ then

$\{x, y\} \cap \{u, v\} = \emptyset$.

Q. Find M such that $|M|$ is maximum

constructed from G ← G' where Given G' → 'G' is a Bipartite graph ⇒ it can have multiple edges from vertices in X.



capacity = 1 for each edge.

Theorem: 'f' is a max flow in G'

⇔ Max. matching in G has size $|f|$.

Integrality theorem: If all the edges have integer capacities in a flow network, the max flow is an integer

~~Augmenting~~

Alternating path: $S \to X \to Y \to X \to Y \to t$

(or) $S \to X \to Y \to X \to Y \to X \to Y \to t$

* Perfect matching: $|X| = |Y| = |M|$

* NP-Complete Problems :-

→ Optimization Problems

Ex:- Minimum Spanning Tree
Min. cut
Max Flow
Shortest path

→ Decision Problems

Ex:- Circuit Satisfiability Problem (SAT)

Formula Satisfiability Problem ($\emptyset$-SAT)

3-CNF Satisfiability Problem (3SAT)

3-Colorability of graph

* Deterministic Algorithms :-

→ All intermediate steps are pre-determined. (Results of)

* Nondeterministic Algorithms :-

→ For the same input, the intermediate steps are not the same in different runs.

→ Not for implementation, but to know the difficulty of the problem.

→ Procedure → O(1)

→ CHOICE(S) → returns an arbitrary element of S

→ SUCCESS → terminates a program on successful completion

→ FAILURE → " " " unsuccessful completion.

* Nondeterministic search

Input: A[1,...n], key x.

Steps:-

1. i ← CHOICE({1, 2, ..., n})
2. if A[i] = x
3. SUCCESS
4. FAILURE

Time complexity: O(1)

because Nondeterministic algorithms have an extraordinary power of choosing the correct path to the solution.

* Non-deterministic sorting (A[1,....n], of positive numbers)

1. for i ← 1 to n
2. B[i] ← 0

Permutation:

3. for i ← 1 to n
4. j ← CHOICE({1, 2, ..., n})
5. if B[j] ≠ 0
6. FAILURE
7. B[j] ← A[i]

Verification:

8. for i ← 1 to n-1
9. if B[i] > B[i+1]
10. FAILURE
11. Print B
12. SUCCESS

* Problem class P:

Comprises of all decision problems that can be solved by deterministic algorithms in polynomial time (w.r.t input size).

* Problem class NP:

Comprises of all decision problems that can be solved by non-deterministic algorithms in polynomial time.

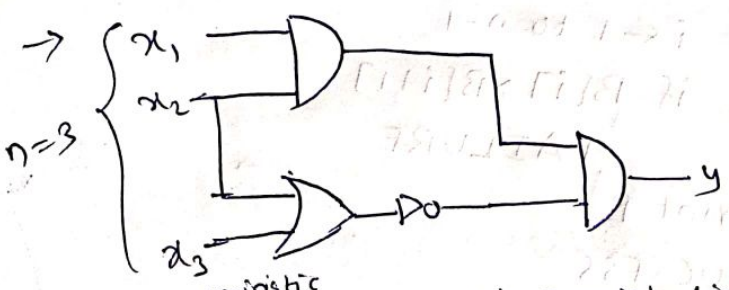
$P \subset NP$ (or) $P \subseteq NP$ is still debatable.

* SAT (Circuit Satisfiability Problem):-

→ SAT ∈ NP.

→ Problem statement

→ Given a ~~Proof~~ Boolean circuit made of AND, OR, NOT gates, properly connected by wires, we have to determine whether there exists an input certificate for which the output is TRUE.



→ No deterministic algorithm of polynomial time is given till date

* Cook-Levin Theorem

P = NP if and only if SAT ∈ P

* ∅-SAT

→ Given a Boolean formula ~~made of~~ with 'n' Boolean variables and AND, OR, NOT, ↔ Operators, determine whether there exists an input certificate for which the formula is TRUE.

Ex: $\emptyset = \left((x_1 \vee x_2) \wedge (\neg x_1 \wedge x_3)\right) \vee (x_1 \vee \neg x_2)$

* NP-Complete Class

→ A decision Problem X belongs to this class if:

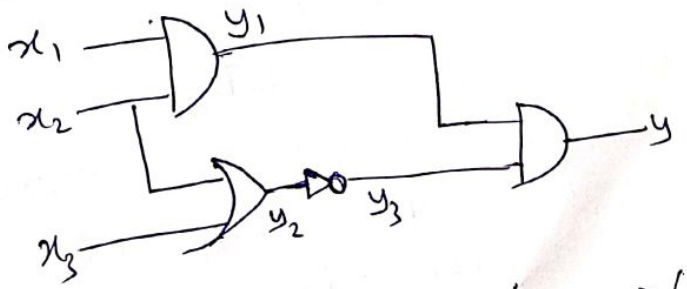
1. $X \in NP$
2. $SAT \leq_p X$

SAT can be reduced to X in Polynomial time.

→ SAT ∈ NP-complete

→ ∅-SAT ∈ NP-complete [Because ∅-SAT ∈ NP ~~reducible~~ and $SAT \leq_p \emptyset\text{-SAT}$]

* Reducing SAT to ∅-SAT :



$$\emptyset = y \wedge (y_1 \leftrightarrow x_1 \wedge x_2) \wedge (y_2 \leftrightarrow (x_2 \barwedge x_3)) \wedge (y_3 \leftrightarrow \neg y_2) \wedge (y \leftrightarrow y_1 \wedge y_3)$$

* Clique → completely connected subgraph of a graph.

* 3-SAT → Given a Boolean formula in conjunctive normal form (CNF) in which each clause consists of exactly 3 literals, we have to decide whether there exists an input certificate for which the Boolean formula is satisfiable.

Ex) $\phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$

↗ It has 3 variables and 4 clauses.

* 3-SAT is NP-Complete

Proof:- (1) SAT $\leq_P$ 3-SAT

(2) 3-SAT $\in$ NP.

#clauses, $k = \text{poly}(n) = (2n)^3$

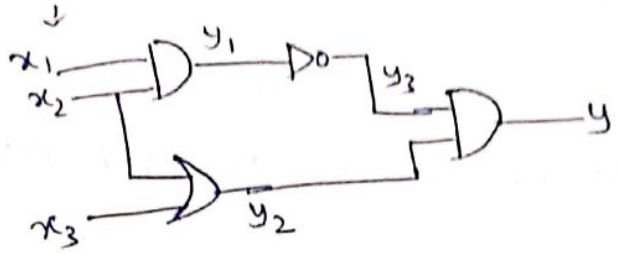
#input vars $= n$

Verification can be done in poly(n) time.

$\Rightarrow$ 3-SAT $\in$ NP $\longrightarrow$ (2) proved.

To prove (1), we have to propose a deterministic poly(n) time algorithm that can reduce any instance of SAT problem to some instance of 3-SAT problem.

SAT - instance



Input vars: x_1, x_2, x_3

output : y

1. Intermediate vars: y_1, y_2, y_3

#gates = poly(n)

#inputvars = n

2. Equivalent Boolean formula

$$\phi = y \wedge (x_1 \wedge x_2 \leftrightarrow y_1) \wedge (x_2 \vee x_3 \leftrightarrow y_2) \wedge (\neg y_1 \leftrightarrow y_3).$$

(ϕ_1: y; ϕ_2: $x_1 \wedge x_2 \leftrightarrow y_1$; ϕ_3: $x_2 \vee x_3 \leftrightarrow y_2$; ϕ_4: $\neg y_1 \leftrightarrow y_3$)

$$\phi = \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_{Poly(n)}$$

$$\phi_2 = x_1 \wedge x_2 \leftrightarrow y_1$$

3.

x_1	x_2	y_1	ϕ_2
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

~~$\phi_2 = \neg((x_1 \wedge x_2) \vee y_1)$~~

$$\Rightarrow \neg\phi_2 = (\neg x_1 \wedge \neg x_2 \wedge y_1) \vee (\neg x_1 \wedge x_2 \wedge y_1) \vee (x_1 \wedge \neg x_2 \wedge y_1) \vee (x_1 \wedge x_2 \wedge y_1)$$

$$\Rightarrow \phi_2 = (x_1 \vee x_2 \vee \neg y_1) \wedge (x_1 \vee \neg x_2 \vee \neg y_1) \wedge (\neg x_1 \vee x_2 \vee \neg y_1) \wedge (\neg x_1 \vee \neg x_2 \vee \neg y_1)$$

$\phi_4 = (\neg y_1 \longleftrightarrow y_3)$

↓

we get in the form of

$x_i \vee x_j$ or $\neg x_i \vee x_j$ or $x_i \vee \neg x_j$ or $\neg x_i \vee \neg x_j$

$\phi_4 = (\neg y_1 \longleftrightarrow y_3)$

↓

we get in the form of

$x_i \vee x_j$ or $\neg x_i \vee x_j$ or $x_i \vee \neg x_j$ or $\neg x_i \vee \neg x_j$

y_1	y_3	ϕ_4
0	0	0
0	1	1
1	0	1
1	1	0

$\neg\phi_4 = (\neg y_1 \wedge \neg y_3) \vee (y_1 \wedge y_3)$

~~$\phi_4 = (y_1 \vee y_3) \wedge (\neg y_1 \vee \neg y_3)$~~

Dummy, z.

$\Rightarrow \neg\phi_4 = (\neg y_1 \wedge \neg y_3 \wedge z) \vee (\neg y_1 \wedge \neg y_3 \wedge \neg z)$
$\vee (y_1 \wedge y_3 \wedge z) \vee (y_1 \wedge y_3 \wedge \neg z)$

$\phi_4 = (y_1 \vee y_3 \vee \neg z) \wedge (y_1 \vee y_3 \vee z) \wedge (\neg y_1 \vee \neg y_3 \vee z)$
$\wedge (\neg y_1 \vee \neg y_3 \vee \neg z)$

$\phi_1 = y$

Use two dummy variables z_1 & z_2.

~~$\phi_1 = (y \wedge z_1 \wedge z_2) \vee$~~

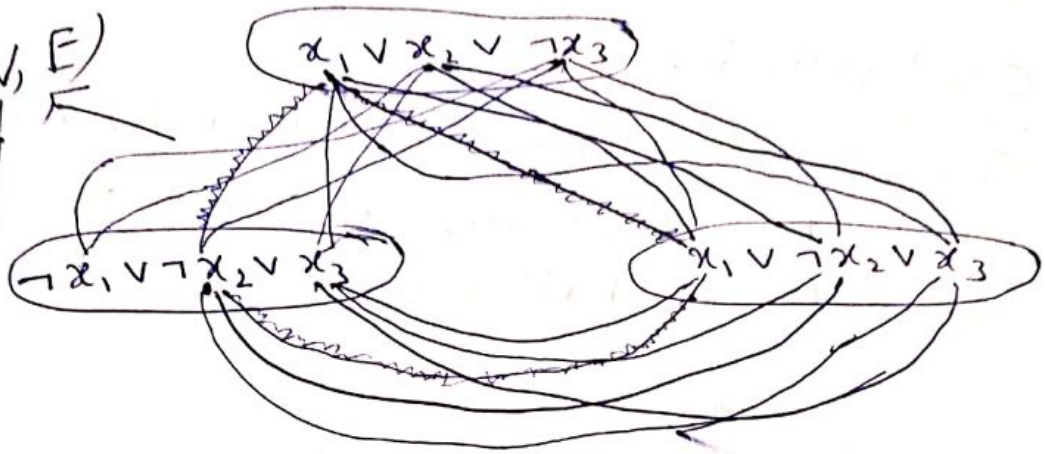
$\neg\phi_1 = (y \wedge z_1 \wedge z_2) \vee (y \wedge z_1 \wedge \neg z_2)$
$\vee (y \wedge \neg z_1 \wedge z_2) \vee (y \wedge \neg z_1 \wedge \neg z_2)$

$\phi_1 = (y \vee \neg z_1 \vee \neg z_2) \wedge (y \vee \neg z_1 \vee z_2)$
$\wedge (y \vee z_1 \vee \neg z_2) \vee (y \vee z_1 \vee z_2)$

* 3-SAT instance

$$\phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$$

$G(V, E)$



$x_1 = 1$
$x_2 = 0 \Rightarrow \phi = 1$
$x_3 = *$

There is a cycle from
$x_1 \to x_1 \to \neg x_2 \to x_1$

$$\phi = \bigwedge_{i=1}^{k} C_i$$

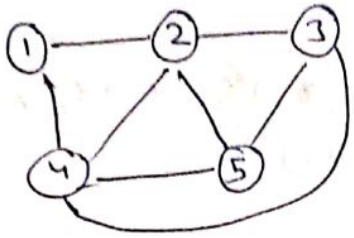
$$C_i = \bigvee_{j=1}^{3} y_{ij}, \quad i = 1, 2 \cdots k$$

$$V = \{ y_{ij} : 1 \le i \le k, \ 1 \le j \le 3 \}$$

$$E = \{ (y_{ij}, y_{uv}) : i \neq u, \ y_{ij} \neq \neg y_{uv} \}$$

→ ϕ is satisfiable if and only if G has a clique of size k, where k is the number of clauses in ϕ.

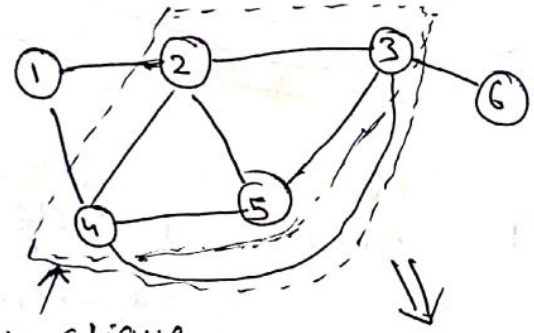
Clique:- For an undirected graph $G(V, E)$, $V' \subseteq V$ is a clique if $(u, v) \in V' \times V'$ is an edge in G for all $(u, v) \in V' \times V'$

Ex: 

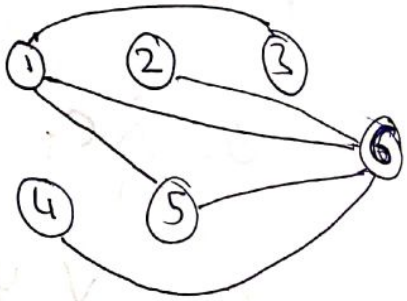
{2,3,4,5} is a clique

* CLIQUE(G, K)

Given a graph G and a number K, determine whether G contains a clique of size K.
CLIQUE is NP-Complete.

G = 

4-clique

$\bar{G}$ = 

$K_{max} = \{2,3,4,5\}$

K_{max} is max-clique in G ⇔ K_{max} is max-independent set in $\bar{G}$

independent set → A subset of vertices having no edge.

*
SAT
↓
∅-SAT
↓
3-SAT
↓
CLIQUE
↓
Independent Set

* For every edge of $\bar{G}$, at least one vertex is outside the independent set.

⇒ Every ~~vertex~~ edge of $\bar{G}$ is ~~adjacent to~~ incident on some vertex outside the independent set (I)

⇒ The complement of independent set $(V \setminus I)$ covers all the ~~vertices~~ edges of $\bar{G}$

↓

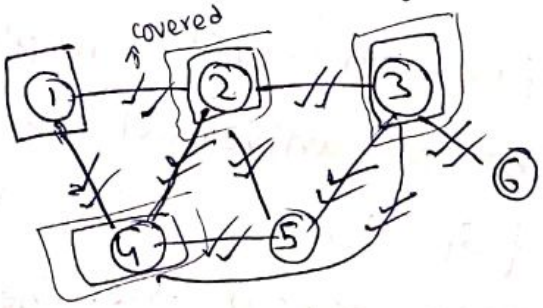
a vertex covers all ~~the adjacent vertices~~ the edges through it.

⇒ $V \setminus I$ is the vertex cover of $\bar{G}$.

* General statement :-

I is an independent set in $G(V, E)$ if and only if $V \setminus I$ is a vertex cover of G. So, $\boxed{\max I \Rightarrow \min V'}$

[V' is a vertex cover of G if for every $(u, v) \in E$, $\{u, v\} \cap V' \neq \emptyset$].

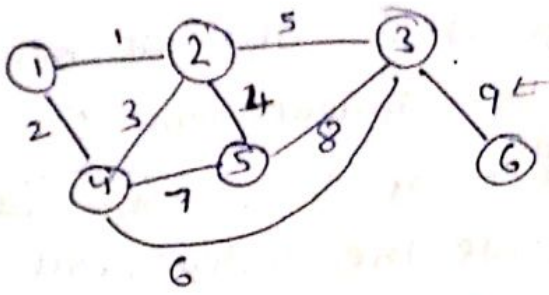


$V' = \{1, 3, 2, 4\}$

(or) $V' = \{2, 3, 4\}$ → minimum vertex cover

~~→ Incidence matrix~~

*



← edge numbers.

Incidence matrix

B → set of sets → called Family

A Vertices	B Edges
1	1, 2
2	1, 3, 4, 5
3	5, 6, 8, 9
4	2, 3, 6, 7
5	4, 7, 8
6	9

← Choose min. number of sets s.t their union equals the union of all the sets

Set Cover problem

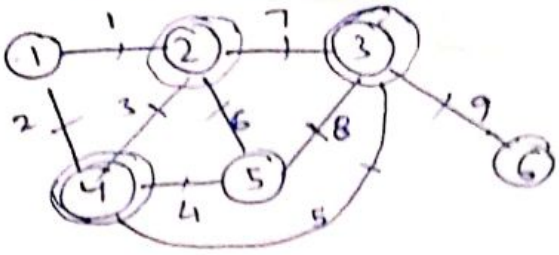
* **Set Cover problem :- (NP- Complete)**

→ Let U be a universal set.

→ Let $F = \{S_1, S_2, \ldots, S_n\}$ be a ~~facility~~ family with each member $S_i \subseteq U$ for $i = 1, 2, \ldots, n$. Let k be a given number.

→ Decide whether there exists 'k' members of F such that their union is U.

*



An instance of vertex cover

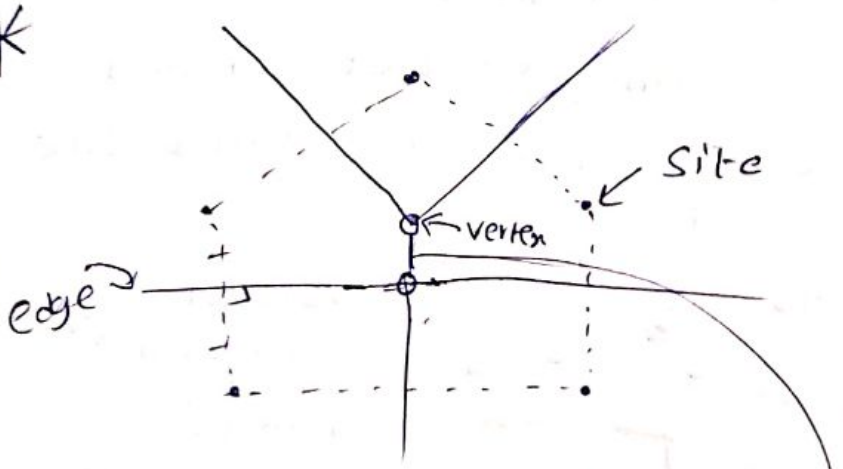
$V' = \{2, 3, 4\}$

vertex	Incident edges
1	1, 2
2	1, 3, 6, 7
3	5, 7, 8, 9
4	2, 3, 4, 5
5	4, 6, 8
6	9

$U = E = \{1, 2, \ldots, 9\}$

$F = \{\{1,2\}, \{1,3,6,7\}, \{5,7,8,9\}, \{2,3,4,5\}, \{4,6,8\}, \{9\}\}$

*



No. of sites = 5

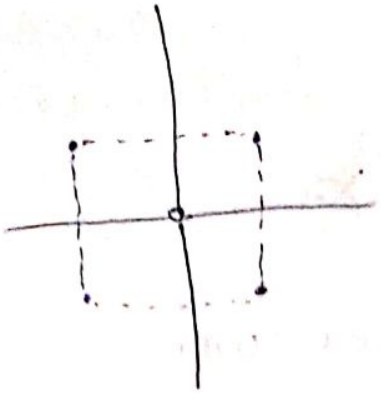
No. of edges = 6 [∵ This is also edge]

No. of vertices = 2

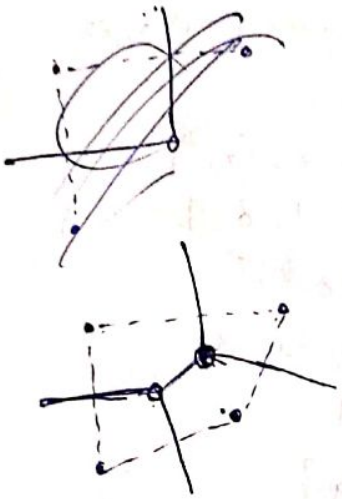
All vertices are circumcenters but all circumcentres are not vertices.

All edges are perpendicular bisectors but all ⊥ bisectors are not edges.

*

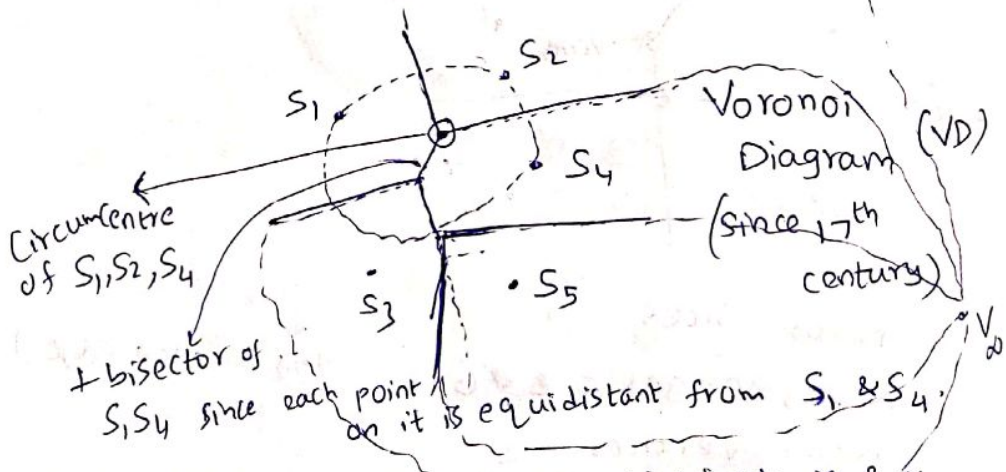


move the point in Q_4



* $S = \{S_1, S_2, \ldots\ldots, S_n\}$ is a set of sites on xy-plane. We have to subdivide the xy-plane into n-regions:

$R_1, R_2, \ldots\ldots, R_n$ such that for any point $P \in R_i$, S_i is the nearest site



Assume: 1. All sites have distinct x & y co-ordinates

2. No four sides are concyclic.

p is a vertex $\Leftrightarrow$ largest empty circle centered at p is a circumcircle of three sites

$\Leftrightarrow$ p is a circumcenter of three sites and the corresponding circumcircle doesn't contain any site in its interior.

→ How many vertices and edges are present in VD?

regions = n.

Add a new vertex V_∞ very far away.

$\Rightarrow$ #vertices = $n_v + 1$

∴ Now it becomes a planar graph..

Euler's formula: $(n_v + 1) - n_e + n = 2 \longrightarrow (1)$

(for planar graph)

Sum of degrees of all $(n_v + 1)$ vertices.

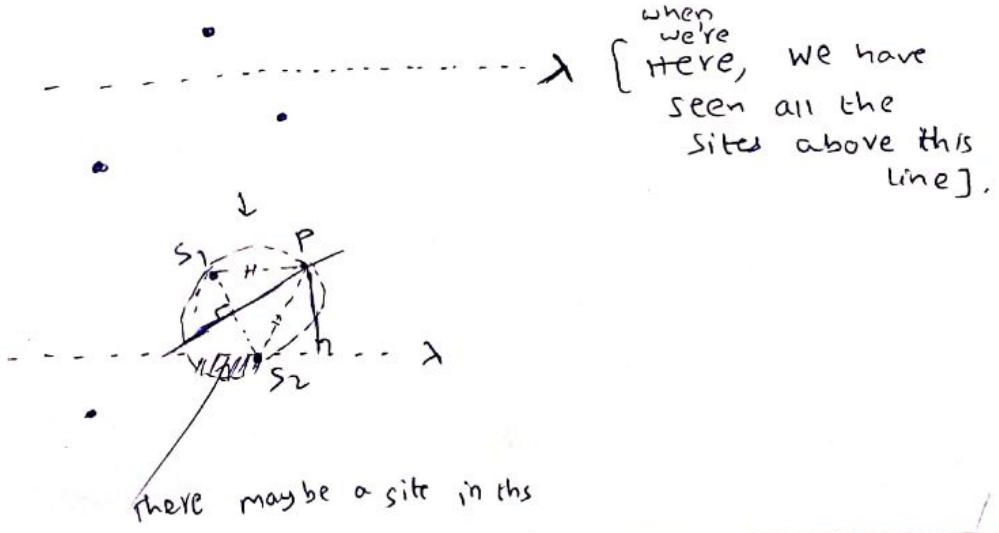
$$\geq 3(n_v + 1)$$

$$\Rightarrow 2n_e \geq 3n_v + 3 \longrightarrow (2)$$

(1) & (2) $\Rightarrow$ $2(n_v + n - 1) \geq 3n_v + 3$

$$n_v \leq 2n - 5 = O(n)$$

Similarly, $n_e = O(n)$.



∴ distance from S_1 to P must be equal to the distance from

∴ distance from S_1 to P must be equal to the distance from P to λ

" " S_2 to P "

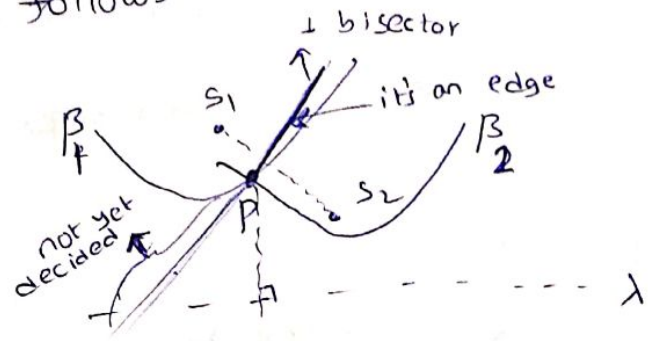
" " " "

" " " "

⇓

P follows two parabolas

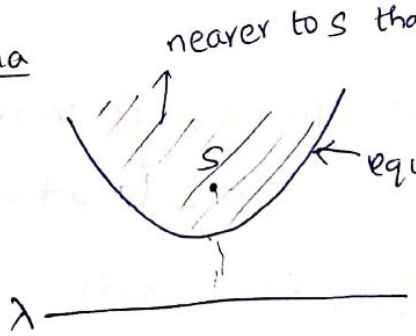
$S_1P = S_2P = P\text{-}\lambda$ ⇐



β_1 = parabolic arc with S_1 as focus

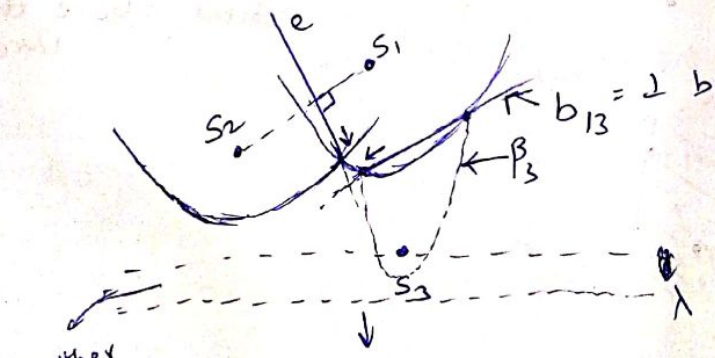
β_2 = parabolic arc with S_2 as focus.

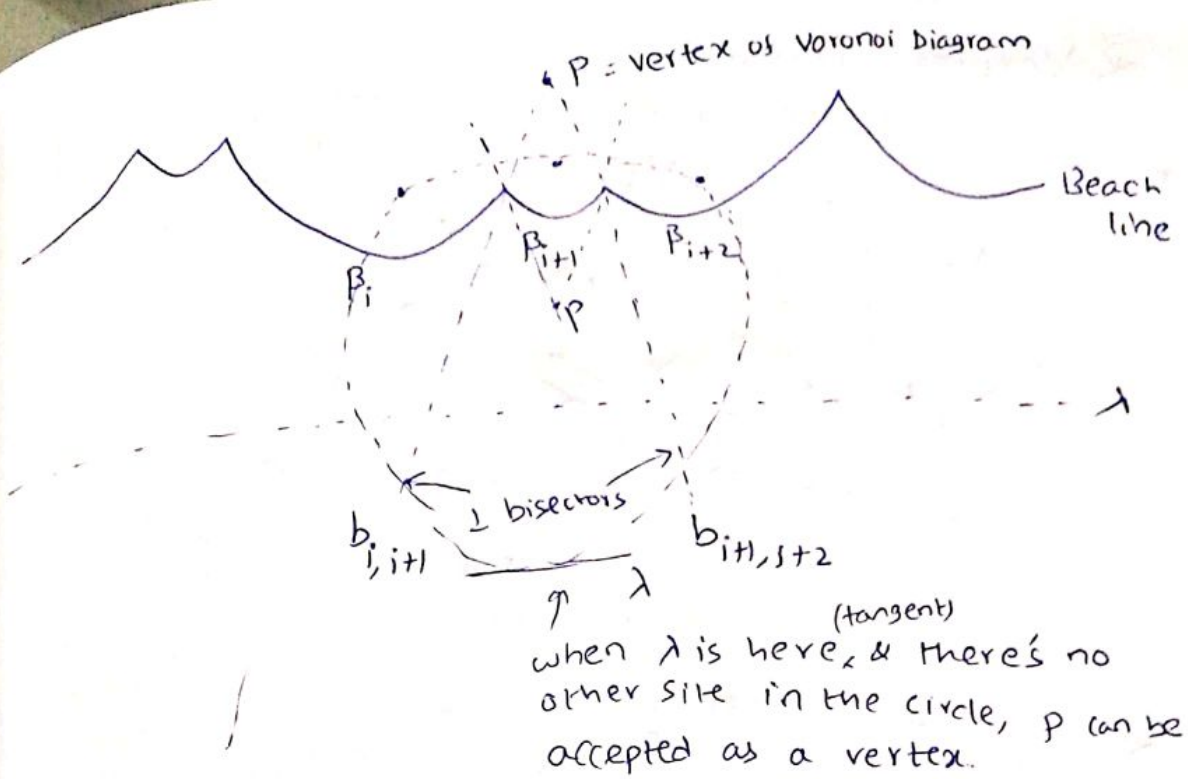
→ Parabola



→ For any point p lying in the union of the interiors of β_1 & β_2, either S_1 or S_2 is nearer than λ.

⇒ S_1 or S_2 is the nearest among all n sites for P.





when λ is here, (tangent) & there's no other site in the circle, p can be accepted as a vertex.

→ Let 'p' be the point of intersection between $b_{i,i+1}$ and $b_{i+1,i+2}$. p becomes eventually a vertex of VD if and only if $b_{i,i+1}$ and $b_{i+1,i+2}$ remain consecutive until β_{i+1} disappears from the beach line.

→ Site event (All known in the beginning)

→ A new site is encountered by λ.

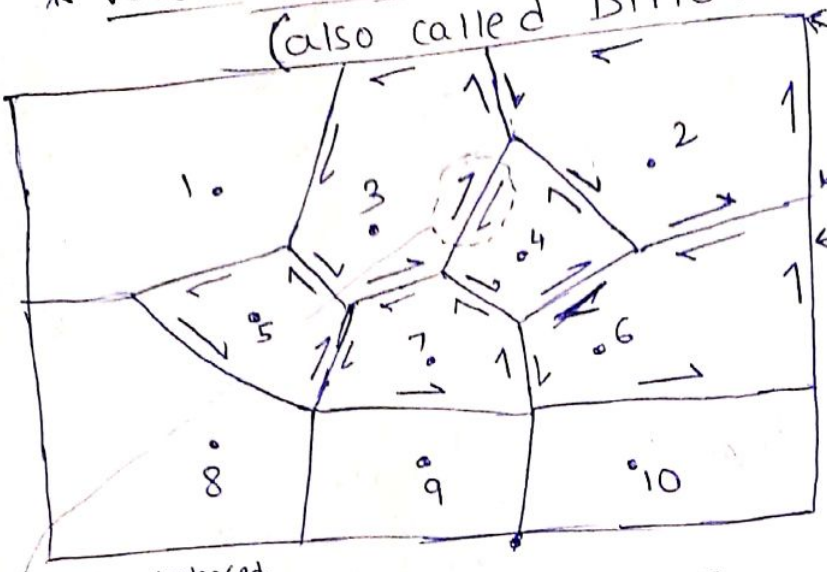
→ Circle event (Gradually known as λ comes down)

→ A new possible vertex is found.

⇒ A new triplet of parabolic arcs has appeared in Beach line.

Lowest point of the circumcircle.

* Voronoi Diagram (also called Dirichlet Tesselation)



Extra vertices of VD

Bounding box

height-balanced BST [priority queue Q - in decreasing order of y-co-ordinates]

height-balanced BST, T (increasing x-coordinates)

*: Another data structure to store beach line

Data structure? stores site events (initially) and circle events (on the fly)

Doubly Connected Edge List (DCEL) to store the VD and finally produce/construct the diagram.

* Doubly Connected Edge List (DCEL):

Used to store a planar subdivision or a planar graph in which faces need to be explicitly represented.

edges in a face are taken in anti-clockwise d

→ vertex records → ID(v) | (x, y)

→ Edge records → ID(e) | twin(e) (ID) | source Vertex(e) (ID) | incident face(e) (ID)

→ Face records → ID(f) = ID(site)

[e = a twin edge (directed)

Every undirected edge ≡ a pair of "twin edges" (directed, opposite to each other)

→ Vertex records → ID(v) | (x,y) co-ords | ID of some edge incident at v.

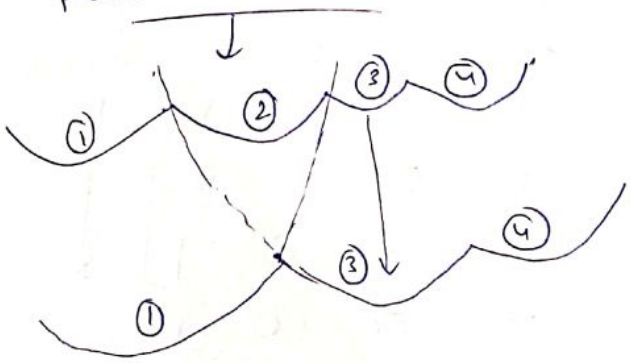
→ edge records → ID(e) | twin(e) ID | source vertex(e) ID | incident face(e) ID | next edge ID | previous edge ID (optional)

→ Face records → ID(f) = ID(site) | ID of some edge incident at f | Aux. info (optional) ↓ like color of a face

[e = a twin edge (directed)]

* Event queue size = O(n) } at any instant
* Beach line size = O(n) }
* Total # ins. / # del = O(n) → each O(log n) time
↓
Total time = O(n log n).

In Beach line, deletion includes an existing parabolic arc disappears (or) new parabola comes so that the arcs change



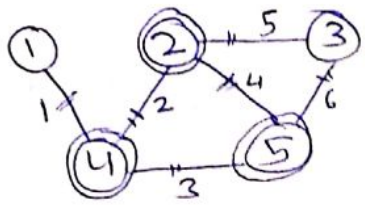
* NP-Complete problems :-

→ Vertex Cover Problem :-

Given an undirected graph G(V, E) and a natural number k, determine whether G has a vertex cover of size k.

[Def: $V' \subseteq V$ is a vertex cover of $G(V, E)$ if each edge $(u, v) \in E$ has either u or v (or both) in V'].

Ex:



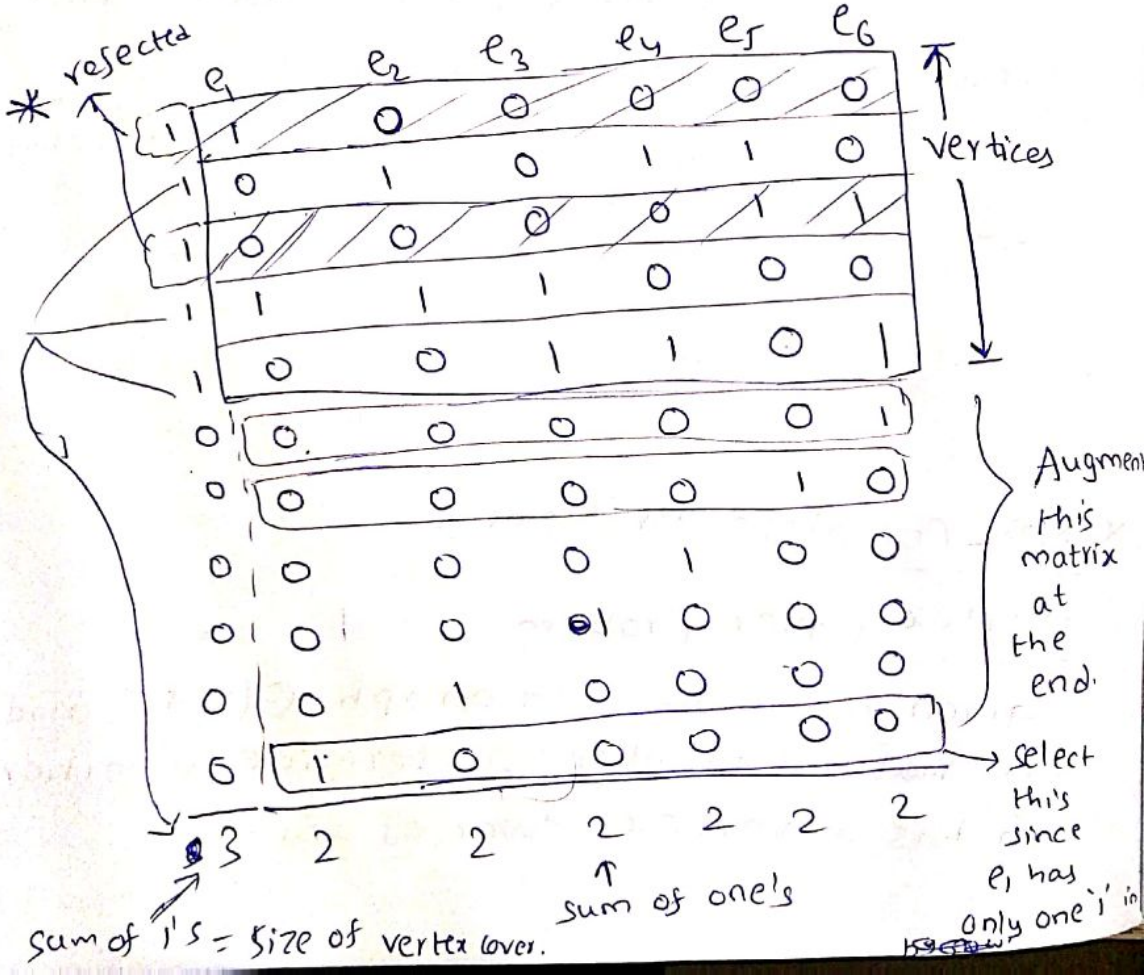
$K = 3$

$V = \{1, 2, 3, 4, 5\}$

$E = \{e_1, e_2, \ldots, e_6\}$

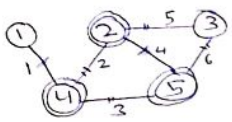
$V' = \{2, 4, 5\}$

	e_1	e_2	e_3	e_4	e_5	e_6
1	1	0	0	0	0	0
2	0	1	0	1	1	0
3	0	0	0	0	1	1
4	1	1	1	0	0	0
5	0	0	1	1	0	1



[Def: $V' \subseteq V$ is a vertex cover of $G(V,E)$ if each edge $(u,v) \in E$ has either u or v (or both) in V'].

Ex:



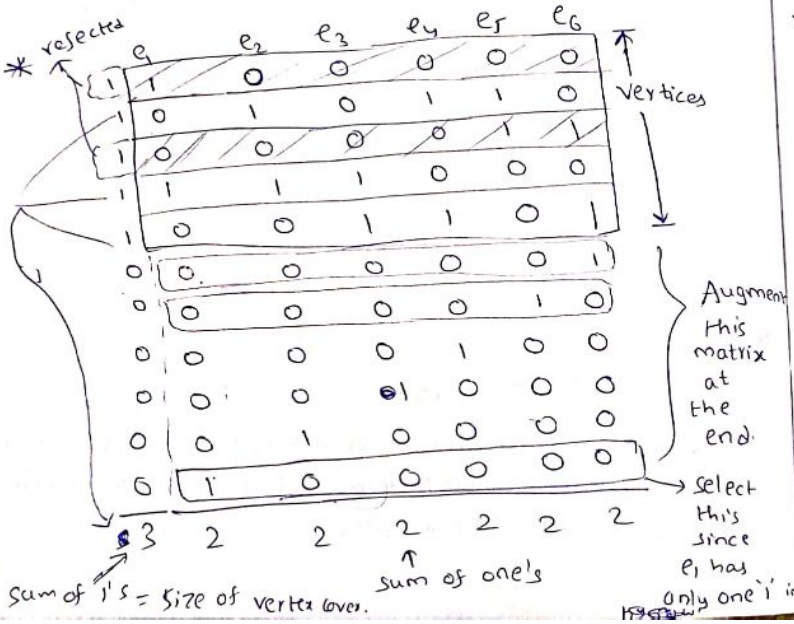
$K = 3$

$V = \{1, 2, 3, 4, 5\}$

$E = \{e_1, e_2, \ldots, e_6\}$

$V' = \{2, 4, 5\}$

	e_1	e_2	e_3	e_4	e_5	e_6
1	1	0	0	0	0	0
2	0	1	0	1	1	0
3	0	0	0	0	1	1
4	1	1	1	0	0	0
5	0	0	1	1	0	1



select in such a way that no of 1's in each column is '2'.

consider each row to be a number in base 4:

	decimal value	
row 1 → 1100000	= 5120	4^{K+1}
row 2 → 1010110	= 4352 ✓	
row 3 → 1000011	= 4101	
row 4 → 1111000	= 5440 ✓	
row 5 → 1001101	= 4177 ✓	
row 6 → 0000001	= 1 ✓	
row 6 → 0000010	= 4 ✓	
row 7 → 0000100	= 16	
row 8 → 0001000	= 64	
row 9 → 0010000	= 256	
row 10 → 0100000	= 1024 ✓	
row 11 → 3222222	= 15,018	

sum of ✓ rows = 15018

*Subset Sum problem:

Given $S \subset \mathbb{N}$ and $t \in \mathbb{N}$, determine whether some elements of S add up to t (Merkle-Hellman knapsack problem)

select in such a way that no of 1's in each column is '2'.

Consider each row to be a number in base 4.

		decimal value	
row 1 →	1 1 0 0 0 0 0	= 5120	K+1
row 2 →	1 0 1 0 1 1 0	= 4352 ✓	
row 3 →	1 0 0 0 0 1 1	= 4101	
row 4 →	1 1 1 1 0 0 0	= 5440 ✓	
row 5 →	1 0 0 1 1 0 1	= 4177 ✓	
row 6 →	0 0 0 0 0 0 1	= 1 ✓	
row 6 →	0 0 0 0 0 1 0	= 4 ✓	
row 7 →	0 0 0 0 1 0 0	= 16	
row 8 →	0 0 0 1 0 0 0	= 64	
row 9 →	0 0 1 0 0 0 0	= 256	
row 10 →	0 1 0 0 0 0 0	= 1024 ✓	
row 11 →	3 2 2 2 2 2 2	= 15,018	

sum of ✓ rows = 15018

* Subset Sum problem :

Given $S \subset \mathbb{N}$ and $t \in \mathbb{N}$, determine whether some elements of S add up to 't' (Merkle-Hellman knapsack problem)

* NP-Complete Problems :

→ Hamiltonian cycle problem (Ham cycle)

→ Travelling Salesman problem (TSP)

Ham cycle :

→ Given a (directed or undirected) graph $G(V, E)$, determine whether G contains a cycle passing through all its vertices

incidence matrix

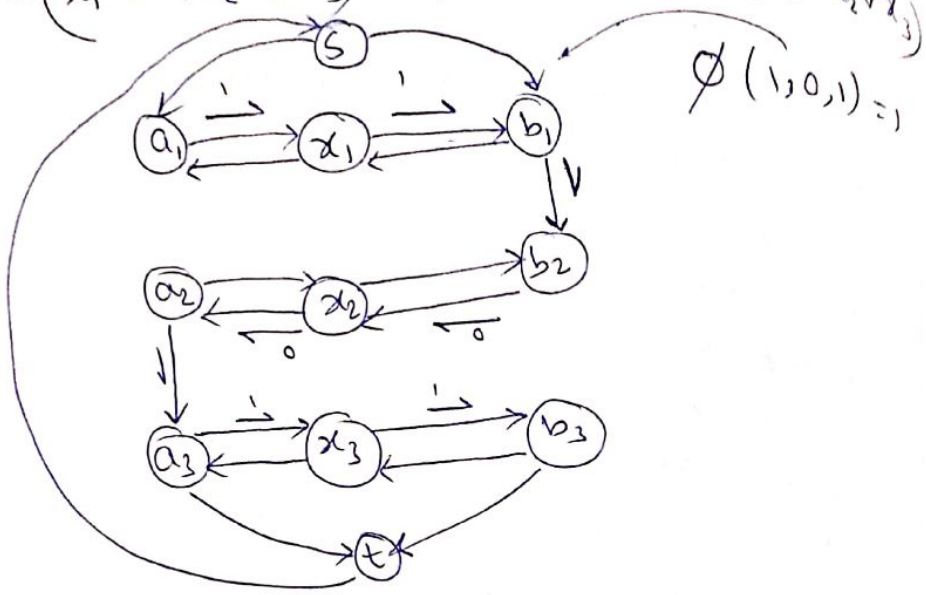
Consider 3-SAT :→ Given ϕ, determine in 3-CNF, determine whether all clauses are TRUE for some input.

↓ reduce to a

Hamcycle instance $G(V, E)$

Ex:

$\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_3)$



If $x_i = 1$, cycle passes through x_i from left to right

If $x_i = 0$, cycle passes through x_i from right to left.

So, add a_i, b_i as new vertices.

Ham Cycle

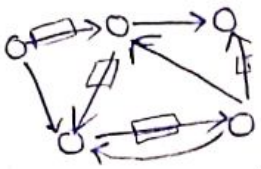
* 3-SAT $\leq_p$ Ham Cycle.

Reduction strategy

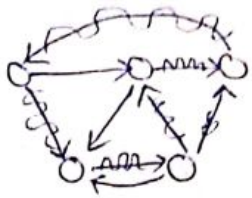
Let $\phi(x_1, x_2, \ldots, x_n)$ be any instance of 3-SAT. Let ϕ have k clauses: $C_1, C_2 \ldots C_k$. Construct a graph (directed) in which there are some nodes corresponding to 'n' variables and some nodes corresponding to 'k' clauses of ϕ.

Let this graph be $G(V, E)$.

G should be such that ϕ is TRUE for some input certificate if & only if G has a Hamiltonian cycle.



No Ham-cycle
It has Ham-path



Yes, Hamcycle.

1. Whatever be the formula ϕ, G should have a Ham-path passing through all the nodes corresponding to the variables $x_1, x_2, \ldots, x_n$.

2. A Ham-path will contain all the nodes corresponding to $C_1, C_2, \ldots, C_k$ only if ϕ is satisfiable.

3. G should have an edge to complete the Ham-cycle in case there is a path passing through all the vertices of G. (edge from t to s).

4. Need a few more nodes to satisfy (3). (s and t).

Ex: $\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$

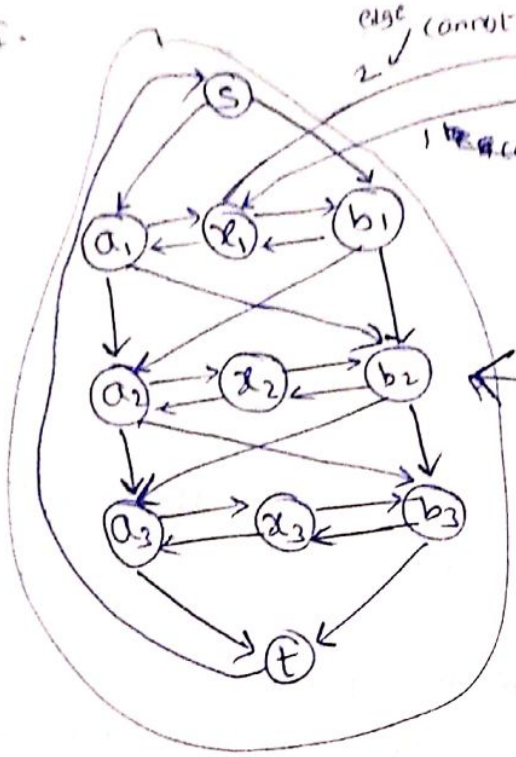
5. G should have as many Ham-cycles as the no. of input certificates for which ϕ is satisfiable.

$\phi(1, 1, 1) = 1 \Rightarrow$ G has a Ham-cycle

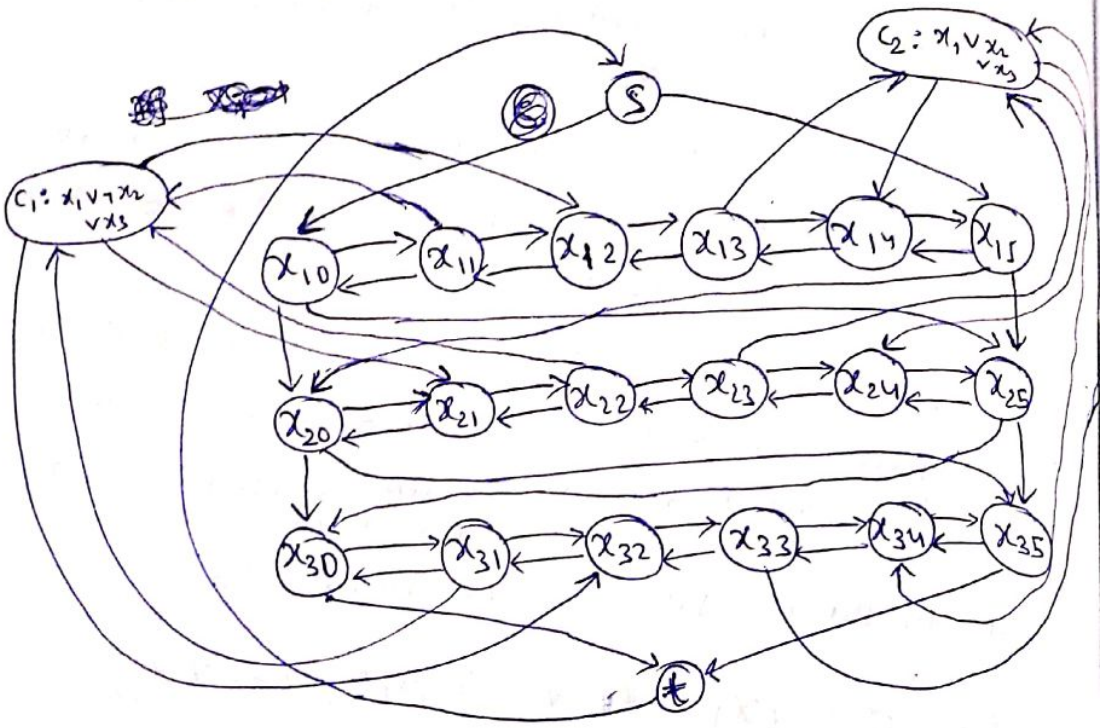
$\phi(0, 1, 0) = 0 \Rightarrow$ G has no Ham-cycle

6. If ϕ is FALSE, for some input certificate I, the desired Ham-cycle will just

miss those clauses whose value is FALSE for I.



Subgraph of G satisfying (1). We expand this to satisfy other strategic points.



* ϕ(n vars, k clauses) $\rightarrow$ G(V, E).

V
1) Include $2K+2$ vertices for every x_i in its row. ['2' vertices as beads (start & end) & '2' vertices each for a clause]
2) Include K vertices for k clauses.
3) Include s and t

E

4) Apply left-to-right and right-to-left edges between successive vertices in each row.

5) Apply an edge from the first vertex of each row to the last one of its next row, and an edge from the last of each row to the first of its next row. Also from first to first & last to last.

6) Drop edges from S to first & last vertices of first row

7) Drop edges from first & last vertices of the last row to t.

8) Add (t, s)

~~For every clause C~~

* Proof of Correctness :-

To be proved: $\phi(I) = \text{TRUE} \Leftrightarrow G(V,E)$ has a Ham-Cycle.

I denotes an input certificate.

For every clause, some literal (L) value is 1 for I. [$\because \phi(I)$ is TRUE]

$L = x_i \Rightarrow x_i = 1$ in I $\Rightarrow$ left-to-right path in the i-th row of G will include that clause (with a provision of including other clauses in which x_i is present as a literal).

In the clauses with $\neg x_i$ as a literal, there are some other literals whose value is 1 for I. (not x_i or $\neg x_i$)
Those clauses will be included in the paths for the corresponding rows.

As all clause vertices are included, we get a Ham-cycle.

→ converse is similar

(9) For every clause C_j $(1 \le j \le k)$ ~~for~~ every

for every literal $l_{j,p}$ $(1 \le p \le 3)$,

if $(l_{j,p} == x_i)$ then

apply edges $(x_{i,2j-1}, C_j)$

and $(C_j, x_{i,2j})$

else

apply edges $(C_j, x_{i,2j-1})$

and $(x_{i,2j}, C_j)$

As all clause vertices are included, we get a Ham-cycle.

⟶ converse is similar

(9) For every clause C_j $(1 \le j \le k)$ ~~for~~ every

for every literal $l_{j,p}$ $(1 \le p \le 3)$,

if $(l_{j,p} == x_i)$ then

apply edges $(x_{i,2j-1}, C_j)$ and $(C_j, x_{i,2j})$

else

apply edges $(C_j, x_{i,2j-1})$ and $(x_{i,2j}, C_j)$

* Travelling Salesman Problem (TSP):-

Given a weighted and completely connected undirected graph $G(V, E, w)$, find a/the cycle of minimum cost that passes through every vertex exactly once.

[cost of a cycle = sum of weights of the edges constituting the cycle]

This is an optimisation problem.

Decision version of TSP:

Given $G(V, E, w)$ and a value k, determine whether G ~~has~~ contains a cycle of cost 'k' that passes through every vertex once.

TSP $\in$ NP.

Ham cycle $\le_p$ TSP

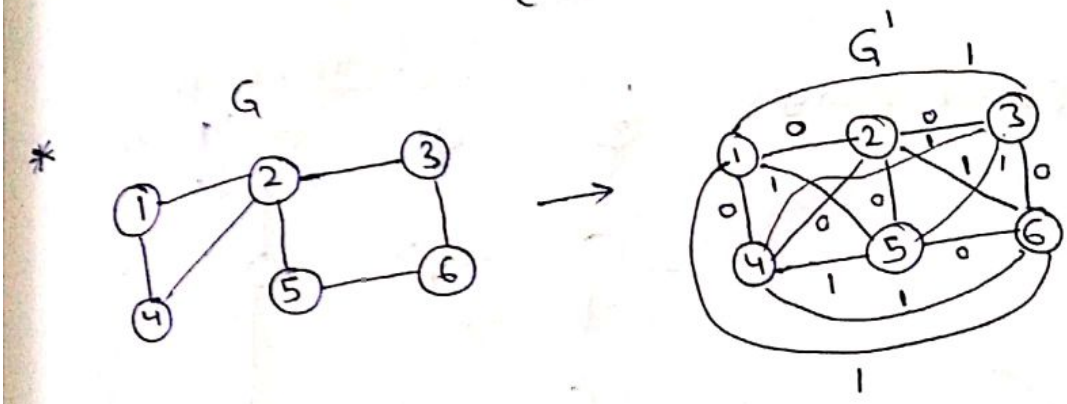
$\Rightarrow$ TSP $\in$ NP-Complete Class

Proof:-

Let $G(V, E)$ be an instance of Ham-cycle.

Construct $G'(V', E', w)$ as follows:

1. $V' = V$
2. $E' = V \times V$
3. $w(u,v) = \begin{cases} 0, & \text{if } (u,v) \in E \\ 1, & \text{otherwise} \end{cases}$



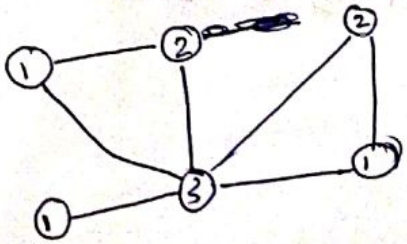
If G has a Ham Cycle, then that cycle in G' has cost '0'. $\Rightarrow$ G' has a '0-cost' cycle.

If G' has a cycle of cost 0, then that cycle is made of edges in E

$\Rightarrow$ G has a Hamcycle.

* 3-coloring problem

Given an undirected graph $G(V, E)$, determine whether its vertices can be colored by three colors such that no two adjacent vertices ~~ha~~get the same color.

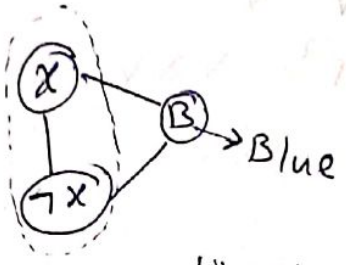


$\Rightarrow$ 3-colorable

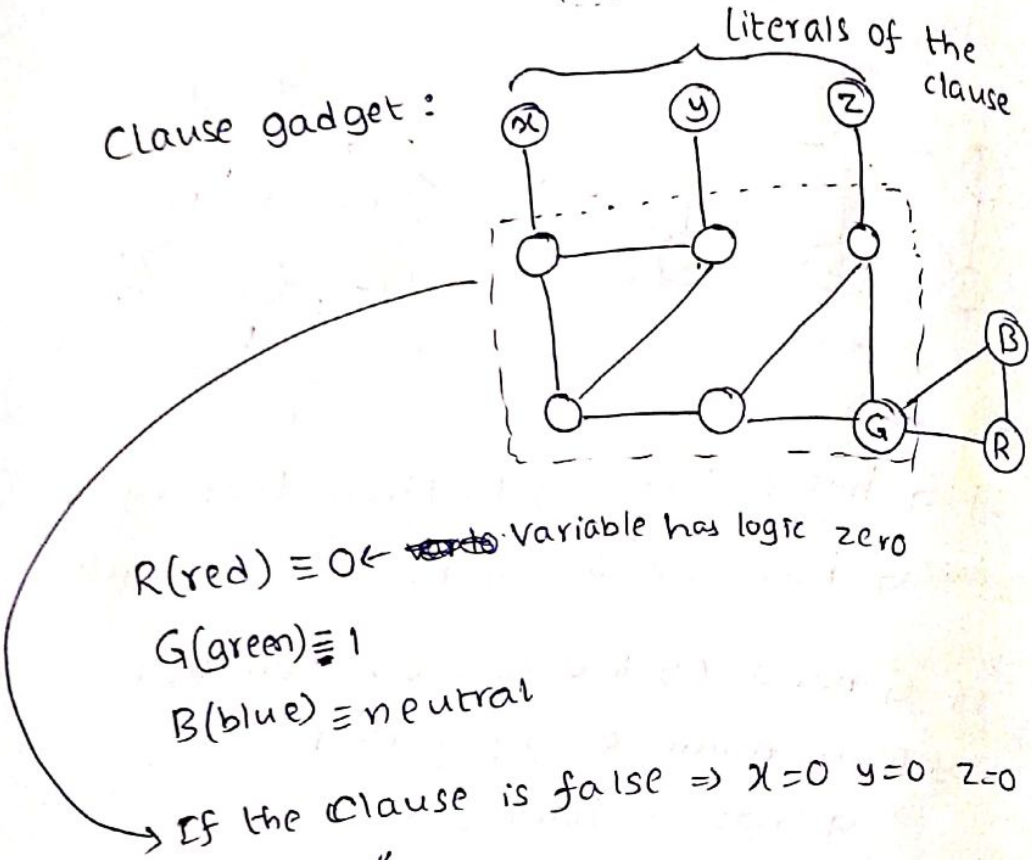
3-SAT $\leq_p$ 3-coloring

* Construction of G(V, E) from Ø (3-SAT instance)

Variable gadget:



Clause gadget:



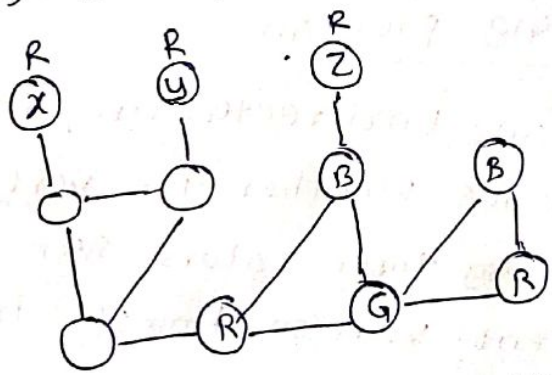
R(red) ≡ 0 ← ~~word~~ variable has logic zero

G(green) ≡ 1

B(blue) ≡ neutral

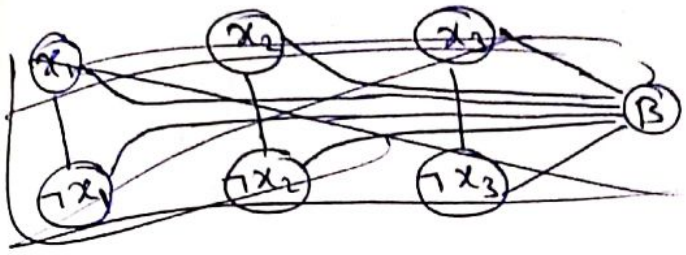
If the clause is false ⇒ x=0 y=0 z=0

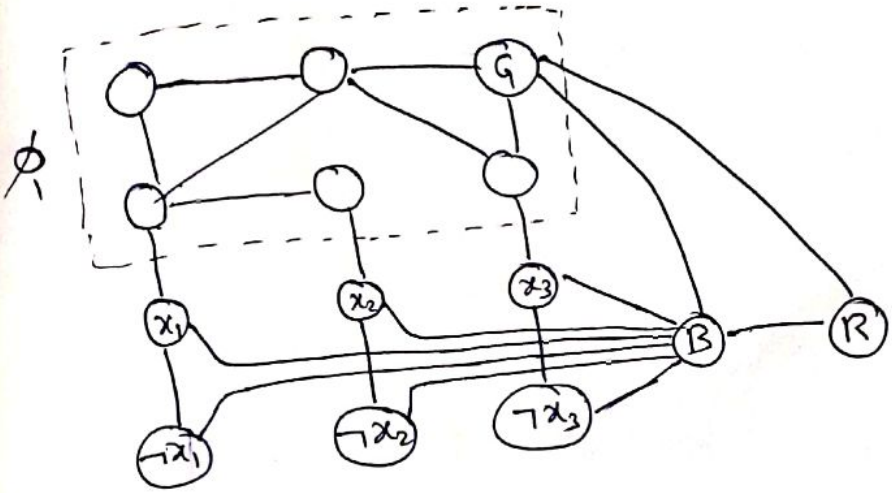
assigning colors,



∴ 3-coloring is not possible

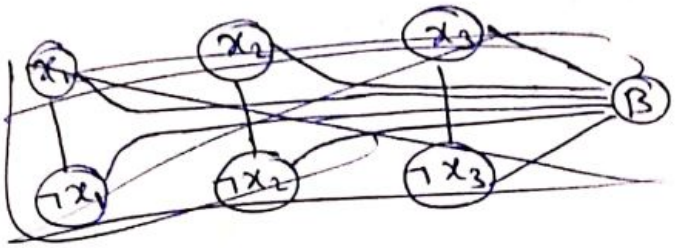
Ex: $\phi = \underbrace{(x_1 \vee x_2 \vee x_3)}_{\phi_1} \wedge \underbrace{(x_1 \vee \neg x_2 \vee x_3)}_{\phi_2} \wedge \underbrace{(x_1 \vee x_2 \vee \neg x_3)}_{\phi_3}$

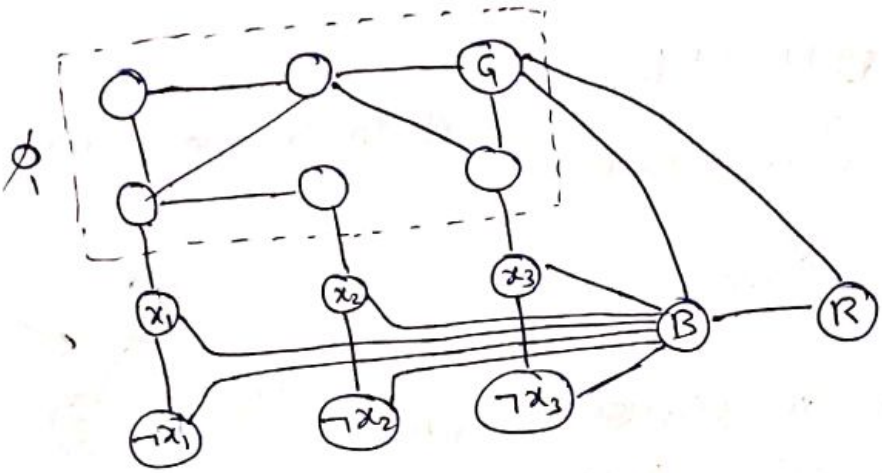




*

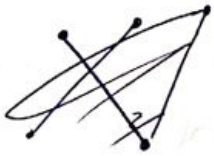
Ex: $\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$
(ϕ_1, ϕ_2, ϕ_3)

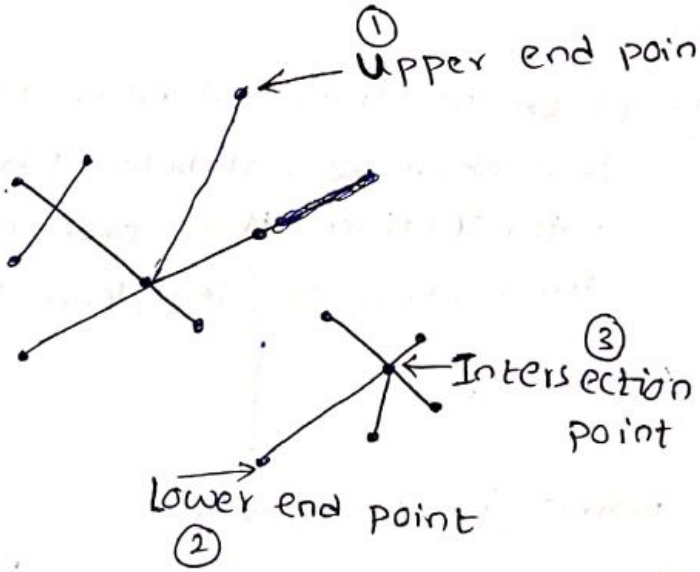




* Line segment Intersection:-

Given a set of n line segments we have to find their points of intersection



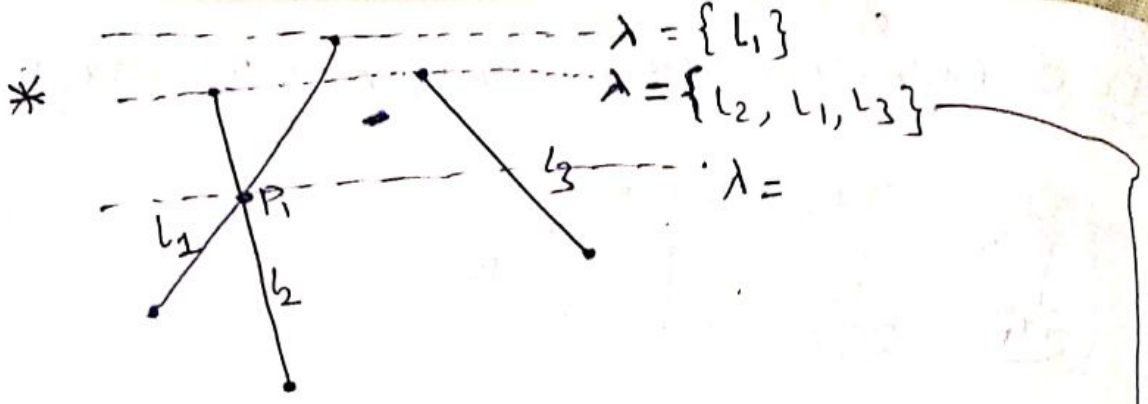


Three types of ~~points~~ event points.

U, L, I.

Data structures:-

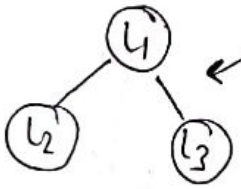
1. Event points — priority queue (y-coordinate)
2. Sweep line — balanced BST.



$Q \leftarrow \mathcal{U} \cup L$ → set of lower end points

$\mathcal{U}$ ↓ set of upper end points

sweep line ↔ $\lambda \leftarrow \emptyset$ (null)



when l_2 is first inserted into $\lambda = \{l_1\}$, l_2 appears to the left of l_1 (adjacent)

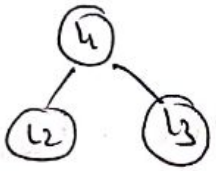
So, there may be an intersection b/w l_1, l_2.

$\Rightarrow P_1$ is inserted in Q.

[∵ If two lines intersect at a point, then just above & just below the point of intersection, the ordering of those two lines changes (left, right)].

when λ_0 reaches P_1

$\lambda =$ old

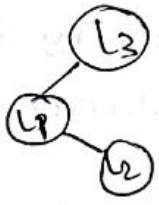


Delete l_1, l_2.

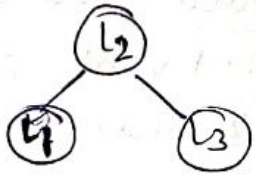
Insert l_1, l_2 such that l_1 is left of l_2.

We can't just swap l_1, l_2 because there may be many lines intersecting at P_1

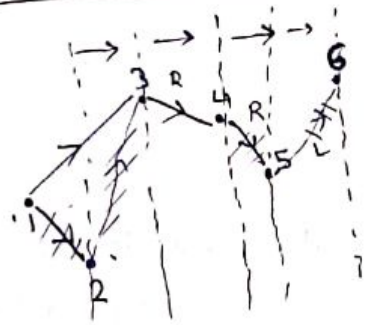
If P_1 is not lower endpoint of l_1 & l_2



rotation



* Convex Hull by Plane Sweep (Graham Scan):-



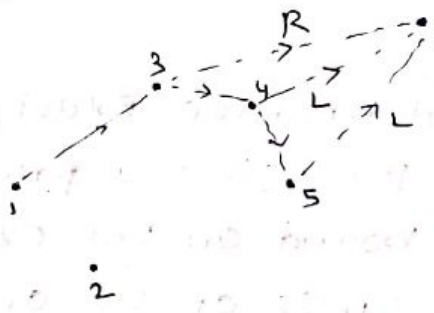
From $\overrightarrow{12}$ to $\overrightarrow{23}$, we took a left turn ⇒ 2 is not a vertex

L = left turn ⇒ not a vertex of CH in upper chain.

R = right " ⇒ is a vertex of CH in upper chain.

Points seen in scan	Vertices
1, 2	1, 2
1, 2, 3	1, 3
1, 2, 3, 4	1, 3, 4
1, 2, 3, 4, 5	1, 3, 4, 5
1, 2, 3, 4, 5, 6	

5 is not a vertex



⇒ 3 is a vertex but not 4.

→ **Observation:-**

Let u_i denote the upper hull chain for $P_1, P_2, \ldots P_i$ where ~~$x[P_i]$~~

$x[P_1] \le x[P_2] \le \ldots \ldots < x[P_n]$.

1. $P_i \in U_i$, $P_i \in U_i$, $U_i \subseteq \{P_1, P_2, \ldots, P_i\}$

2. $U_{i+1} \subseteq U_i \cup \{P_{i+1}\}$.

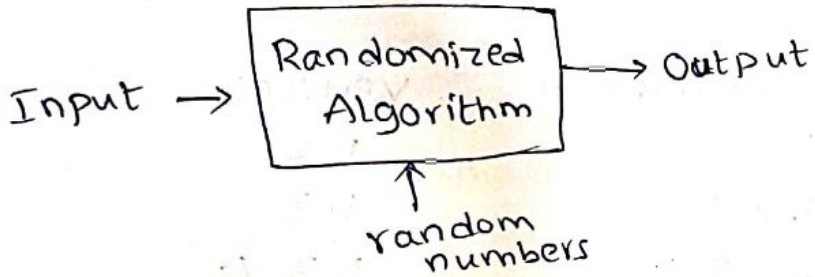
→ Stack for U_i.

* Time complexity :-

1. Sorting $P_1, P_2, \ldots, P_n$ by increasing 'x'
→ $O(n \log n)$.

2. Total # push = (Total # pop) (asymptotically) for the stack.

* Randomized Algorithms :-



Types of randomized algorithms

→ Las Vegas

→ Always produces correct/exact output

→ Has an expected time complexity.

→ Monte Carlo

→ May not produce exact/correct result. But gives a probabilistic lower bound on the exactness /correctness of the output.

→ Has an exact/deterministic time complexity.

* Quicksort :-

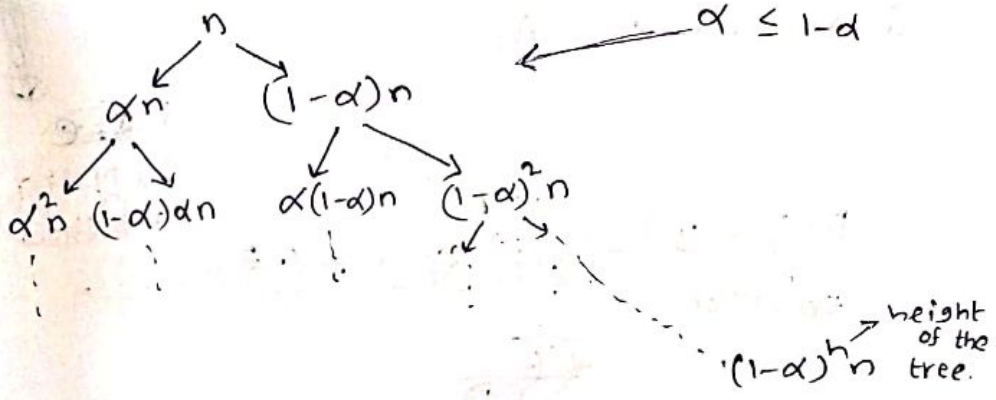
* For randomized quick sort, we choose the pivot randomly at each stage and ensure the the partition is nearly balanced.

$n \rightarrow$ partition : $\left(\frac{an}{a+b}, \frac{bn}{a+b}\right)$, $\alpha \le \frac{a}{a+b} \le \beta$

α, β are constants

Ex: $\alpha = 0.4$, $\beta = 0.6$

partition ratio $\rightarrow$ $\frac{\#\text{elements on left}}{\#\text{elements on right}}$

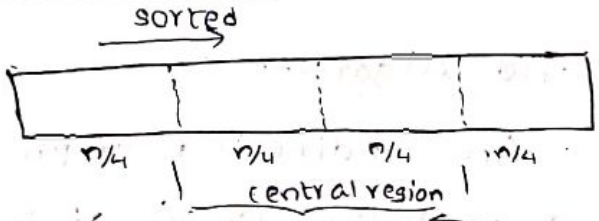


$(1-\alpha)^h n = 1$

$\Rightarrow n = \left(\frac{1}{1-\alpha}\right)^h$

$h = \log_{\frac{1}{(1-\alpha)}} n = O(\log_2 n)$ if α is a constant

* Las Vegas Quicksort :-



A randomly chosen pivot falls in this region with probability $\frac{1}{2}$.

x = random element from $A[1 \ldots n]$

$P(x \in \text{central Region}) = \frac{1}{2}$.

1. Choose 'x' randomly from the input array $A[1 \cdots n]$: $n > 1$

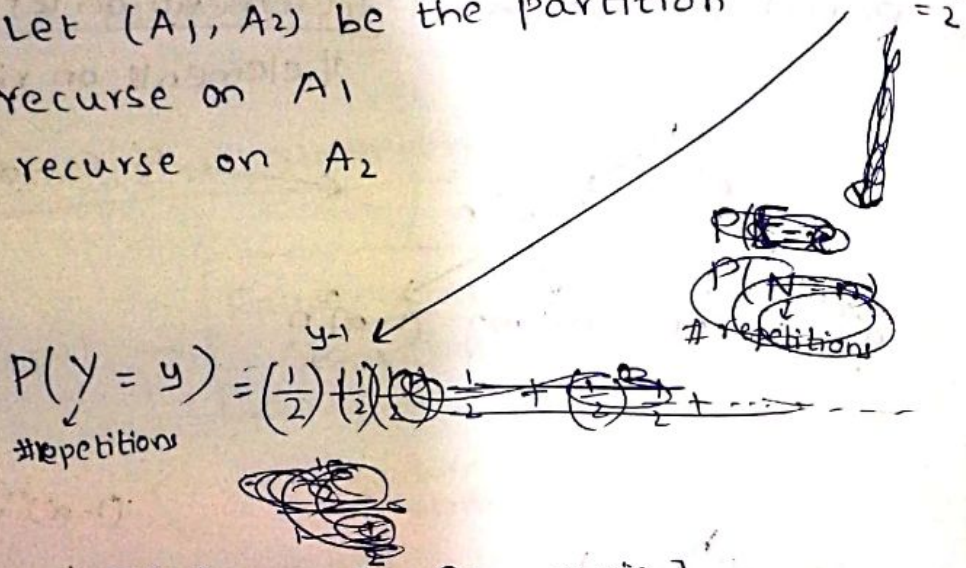
2. If x is a "central splitter" $\rightarrow O(n)$ then use x as the pivot for partitioning $A[1 \cdots n]$

3. else repeat from step 1. $\rightarrow$ expected no. of repetitions = 2

4. Let (A_1, A_2) be the partition

5. recurse on A_1

6. recurse on A_2



$P(Y = y) = \left(\frac{1}{2}\right)\left(\frac{1}{2}\right)^{y-1}$

repetitions

$\therefore E(Y) = 2$ [$\because$ Geometric distrib.]

$T(n)_{expected} = 2T(n/2)_{expected} + O(n)$

* **Min Cut :-**

Monte Carlo Algorithm :-

→ Input → An undirected graph $G(V, E)$

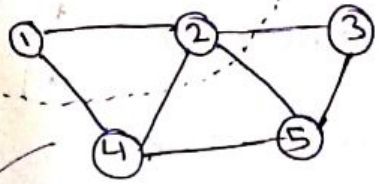
→ Cut = a partition (X, Y) of V.

$(X \cup Y = V, \; X \cap Y = \emptyset)$

Size of a cut = No. of crossing edges between X & Y.

$= |\{(x, y) : x \in X, y \in Y\}|$

* Min cut = a cut of smallest size.



$X = \{1, 2\}$

$Y = \{3, 4, 5\}$

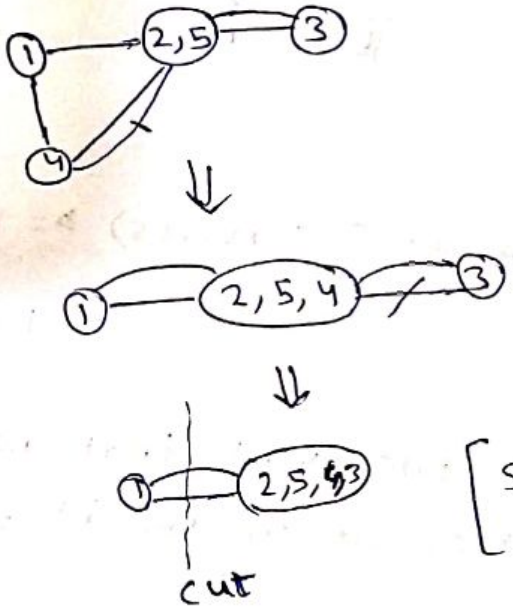
$Cut = \{(1,4), (2,4), (2,5), (2,3)\}$

$\Rightarrow$ cut size = 4

* Operation → edge contract ⇒ choose an edge randomly and contract the edge.

Considering edge (2,5)

two edges because (2,3) & (3,5)



[Stop when only two vertices are present]

→ Algorithm :-

1. Choose an edge randomly and contract it.
2. Repeat step 1 n-2 times ($n = |V|$)
3. Report the resultant cut.

* Lower Bound of the probability :-

Let 'C' be a particular min cut.

Let $k = |C|$ = size of min cut.

1st contract

$P(\text{an edge of } C \text{ is contracted}) = \frac{k}{|E|}$

$k \leq$ min-degree of a vertex

[∵ If k is greater, then choose the vertex as one set & the rest in other]

$\Rightarrow$ degree of each vertex $\geq k$

$\Rightarrow \sum$ degree of each vertex $\geq nk$

$\Rightarrow 2|E| \geq nk$

$$k \leq \frac{2|E|}{n}$$

$\Rightarrow \frac{k}{|E|} \leq \frac{2}{n}$

$\therefore P(C \text{ survives}) = 1 - \frac{k}{|E|}$

$\geq 1 - \frac{2}{n}$

Let E_i denote the event that no edge of C is contracted in the i^{th} contract, $1 \leq i \leq n-2$

$\Rightarrow P(E_1) \geq 1 - \frac{2}{n}$

2nd contract

$P(E_2 | E_1) \geq 1 - \frac{2}{n-1}$ [∵ degree of each vertex is still greater than or equal to k]

3rd Contract

$P(E_3 | E_1 \cap E_2) \geq 1 - \frac{2}{n-2}$

i^{th} contract

$$P\left(E_i \mid \bigcap_{j=1}^{i-1} E_j\right) \geq 1 - \frac{2}{n-i+1}$$

$\therefore P(C \text{ survives after } n-2 \text{ contracts})$

$$= P(E_1) \cdot P(E_2|E_1) \cdot P(E_3|E_1 \cap E_2) \cdots\cdots P\left(E_{n-2} \mid \bigcap_{j=1}^{n-3} E_j\right)$$

$$\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right) \cdots\cdots \left(1-\frac{2}{n-n+2+1}\right)$$

$$= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots\cdots \frac{2}{4} \cdot \frac{1}{3}$$

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

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

$$\Rightarrow P(C \text{ survives after } n-2 \text{ contracts}) > \frac{2}{n^2}$$

$$\Rightarrow P(C \text{ does not survive in one process}) \leq 1 - \frac{2}{n^2}$$

$$\therefore P(C \text{ does not survive after } m \text{ processes}) \leq \left(1-\frac{2}{n^2}\right)^m$$

$$\therefore P(C \text{ survives after } m \text{ processes}) \geq 1-\left(1-\frac{2}{n^2}\right)^m$$

$(1-x)^m \leq$ Const. for $m = ?$ $\quad |x|<1$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots\cdots$$

$$e^{-x} = 1 - x + \underbrace{\left(\frac{x^2}{2!} - \frac{x^3}{3!}\right)}_{+ve} + \underbrace{\left(\frac{x^4}{4!} - \frac{x^5}{5!}\right)}_{+ve \ [\because |x|<1]} + \cdots$$

$$\Rightarrow e^{-x} > 1-x \Rightarrow 1-x < \left(\frac{1}{e}\right)^x$$

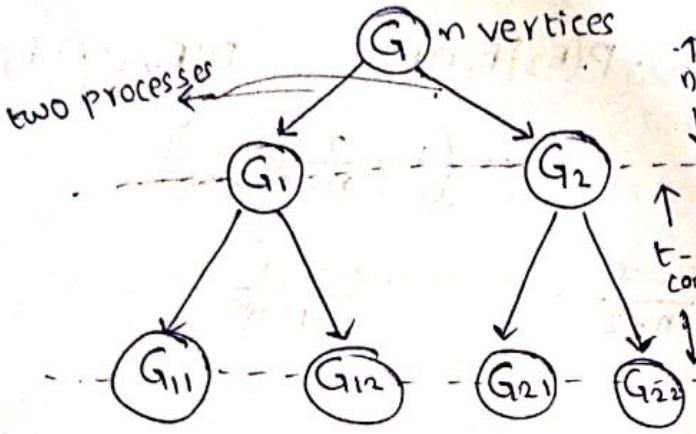
$$\Rightarrow (1-x)^{\frac{1}{x}} \leq \frac{1}{e}$$

$\therefore P(C \text{ survives after } m \text{ processes}) \geq 1-\left(1-\frac{2}{n^2}\right)^m$

$$\geq 1 - \frac{1}{e} \quad \text{for } m = \frac{n^2}{2}$$

$$\therefore T(n) = \underbrace{O(n^2)}_{\text{for one process}} \times \frac{n^2}{2} = O(n^4)$$

→ Further improvement :-



$n-t$ contractions $\Rightarrow O[n^2)$ time

t vertices such that C survives in G_1 with $P \geq \frac{1}{2}$ or in G_2 with $P \geq \frac{1}{2}$

$t-t'$ contracts

t' vertices s.t. C survives from G_1 / G_2 with $P \geq \frac{1}{2}$

P(C survives after $n-t$ contracts)

$$\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\cdots\cdots\left(1-\frac{2}{n-(n-t)+1}\right)$$

$$= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \cdots\cdots \frac{t}{t+2} \cdot \frac{t-1}{t+1}$$

$$= \frac{t(t-1)}{n(n-1)}$$

we want to make $\frac{t(t-1)}{n(n-1)} \geq \frac{1}{2}$

$$\Rightarrow t^2 - t \geq \frac{n(n-1)}{2}$$

$$t^2 - t \geq {}^nC_2$$

$$t^2 - t + \frac{1}{4} \geq {}^nC_2 + \frac{1}{4}$$

$$\left(t-\frac{1}{2}\right)^2 \geq {}^nC_2 + \frac{1}{4}$$

$$\boxed{t \geq \frac{1}{2} + \sqrt{{}^nC_2 + \frac{1}{4}}}$$

$$\Rightarrow \boxed{t = \left\lceil \frac{n}{\sqrt{2}} \right\rceil + 1}$$ → substitute & check

$\frac{1}{\sqrt{2}} = 0.732 \Rightarrow$ After 1st level → 70% vertices remaining

After 2nd level → 50% vertices remaining

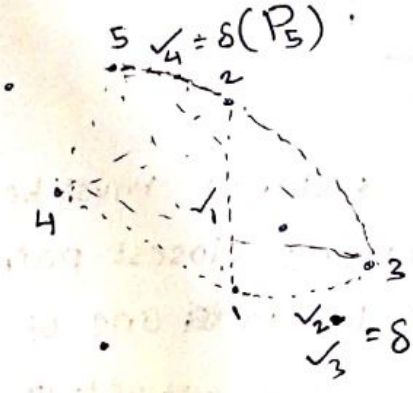
$\Downarrow h = O(\log n)$

$$T(n) = 2T\left(\frac{n}{\sqrt{2}}\right) + O(n^2) \Rightarrow T(n) = O(n^2 \log n)$$

* Closest pair of points in 2D :-

Las Vegas Algorithm :-

Randomly select the points & update the closest pair.



$\delta(P_1, P_2, P_3, P_4) = \delta(P_4)$

↓

first 4 random points.

Observation 1 :-

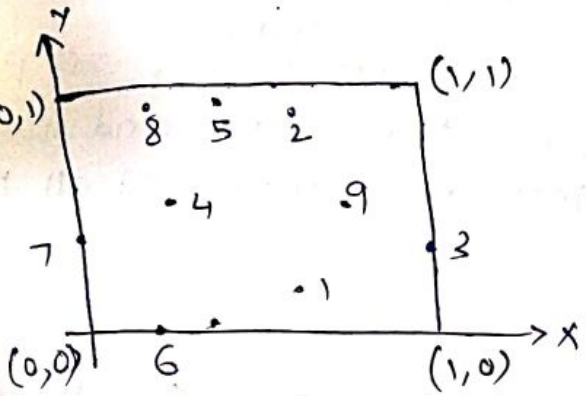
If $\delta(P_i) < \delta(P_{i-1})$ then P_i is a point of the closest pair in P_i

P_i = first 'i' random points $= \{P_1, P_2, \ldots P_i\}$

Observation 2 :-

For any scaling / translation / rotation on the point set $P = \{P_1, P_2, \ldots P_n\}$, the closest pair doesn't change.

[Transformation invariance]

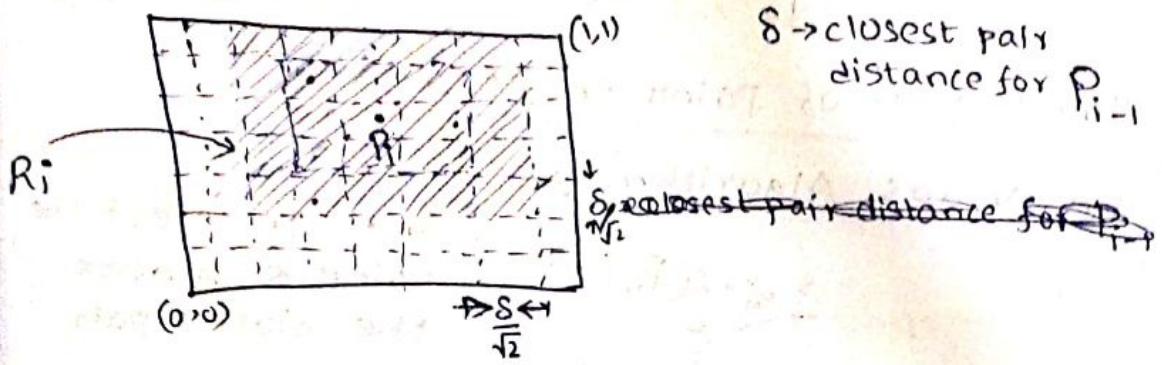


perform this transformation to get the points in the unit square as shown.

Observation 3 :-

Probability that P_i is a point of the closest pair in $P_i = \frac{i-1}{\binom{i}{2}} = \frac{i-1}{\frac{i(i-1)}{2}} = \frac{2}{i}$

Divide the unit square into $\left(\frac{2}{\delta^2}\right)$ small squares.



$\delta \rightarrow$ closest pair distance for P_{i-1}

~~$\frac{\delta}{\sqrt{2}}$ closest pair distance for P_{i-1}~~

The new smallest distance must be $\leq \delta$. So the point forming closest pair with P_i, if exists, must lie in one of the shaded 25 squares.

Reconstruction of Hash table for P_i (Assigns a value to each point P_j)

$(\text{if } \delta(P_i) < \delta(P_{i-1}))$ $\rightarrow O(i)$ time.

∴ Total expected time complexity,

$$T_{exp} = O(n) + \sum_{i=3}^{n} \frac{2}{i} O(i)$$

$$= O(n) + \sum_{i=3}^{n} O(1)$$

for scaling, translation $= O(n) + O(n)$

$$= O(n).$$

* Minimum Enclosing Disc :- (Las Vegas Algorithm) (M.E.D.)

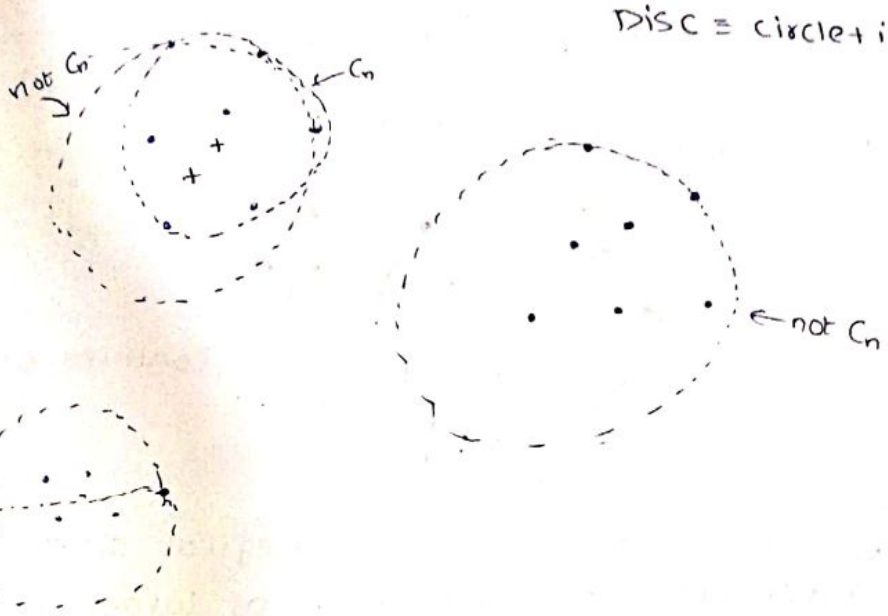
Given a set of n points on the 2D plane, we have to find the center and radius of the smallest disc containing all the n points.

Notations :-

Let D_n denote the M.E.D. for $P_1, P_2, \ldots, P_n$.

Let C_n denote the boundary of D_n i.e on or inside C_n.

Disc ≡ circle + interior



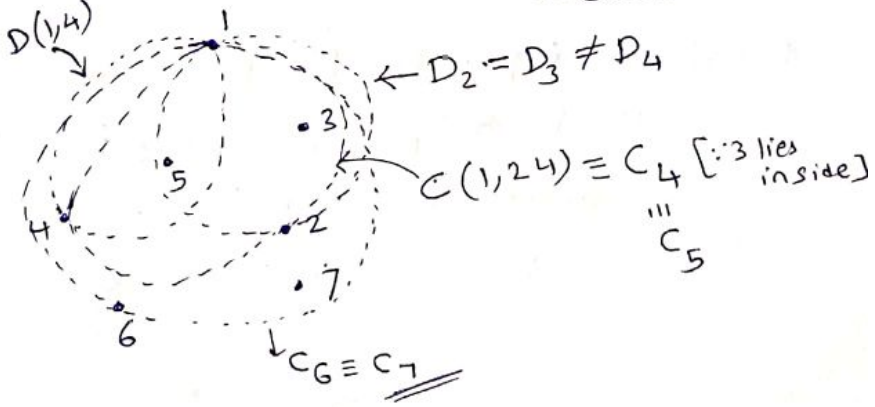
Observations :-

→ Atleast two points will lie on C_n.

→ If exactly two points lie on C_n, then they are the endpoints of a diameter of C_n.

→ If no four points are concyclic, then either two or three points will lie on C_n.

→ If three points lie on C_n, then the Δ^{le} with these three points as vertices cannot be an obtuse angled Δ^{le}.



$P_4 \notin D_3 \Rightarrow P_4$ will lie on C_4.

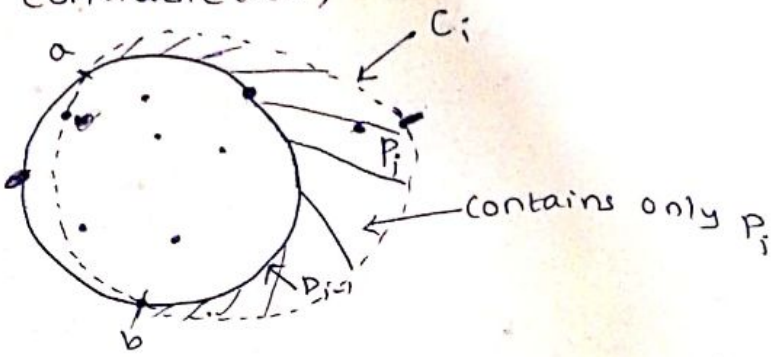
$P_i \notin D_{i-1} \Rightarrow P_i \in C_i$

$P_2 \notin D(1,4) \Rightarrow P_2 \in C(1,4,2)$.

$P_6 \notin D_5 \Rightarrow P_6 \in C_6$

Theorem: $P_i \notin D_{i-1} \Rightarrow P_i \in C_i$

Proof: By contradiction,



[we have if two circles of unequal sizes intersect, the smaller arc of larger circle obtained by the intersection, lies inside the smaller circle]

Let P_i be inside C_i and not on C_i, if possible. The larger arc ab of C_i will then have no point on it and hence subtends an angle greater than 180° at the centre of C_i.

This contradicts observation 3.

* Given i points $(P_1, P_2, \ldots, P_i)$ and two points P_j & P_k, $1 \le j < k \le i$, where P_j & P_k are ~~actua~~ already known to be lying on C_i, how to compute C_i?

$4 \in C(1,2,3,4,j,k)$

$\Rightarrow C(1,2,3,4,j,k)$

~~$C(j,k,4)$~~

= circumcentre of P_j, P_k, P_4

* MED with 2 points $(P_1, P_2, \ldots, P_i, j, k)$

1. Permute $P_1, \ldots, P_i$
2. $D_2 \leftarrow MED(P_j, P_k)$
3. for $h \leftarrow 1$ to i
4. if $P_h \in D_{h-1}$
5. then $D_h \leftarrow D_{h-1}$
6. else
7. $D_h \leftarrow$ Circumcircle (P_j, P_k, P_h)
8. return D_i

* Given i points $(P_1, P_2, \ldots, P_i)$ and one point P_j, $1 \le j \le i$, where P_j is already known to be lying on C_i, how to compute C_1?

* MED with 1 point $(P_1, P_2 \ldots, P_i, j)$

1. Permute $P_1, P_2, \ldots, P_i$ randomly
2. $D_1 \leftarrow MED(P_j, P_1)$
3. for $h \leftarrow 2$ to i
4. if $P_h \in D_{h-1}$
5. then $D_h \leftarrow D_{h-1}$
6. else // $P_h \in C_h$
7. $D_h \leftarrow$ MED with 2 points $(P_1, P_2 \ldots P_k, j, h)$
8. return D_i

* MED($P_1, P_2, \ldots, P_n$)

1. Permute $P_1, P_2, \ldots, P_n$ randomly
2. $D_2 \leftarrow$ MED(P_1, P_2)
3. for $i \leftarrow 3$ to n
4. if $P_i \in D_{i-1}$
5. then $D_i \leftarrow D_{i-1}$
6. else
7. $D_i \leftarrow$ MED with 1 point ($P_1, P_2 \ldots P_i, i$)
8. return D_n.

* Expected time Complexity :-

for MED with 2 points ($P_1, P_2, \ldots, P_i, j, k$)
where $1 \le j < k \le i$
& $P_j \in C_i$, $P_k \in C_k$

→ If $P_n \notin D_{n-1}$,

⇒ C_n = Circumcircle of P_n, P_j, P_k
↓ $O(1)$ time

⇒ C_i is computed in $O(i)$ time

for MED with 1 point ($P_1, P_2, \ldots, P_i, j$)
where $1 \le j \le i$
& $P_j \in C_i$

→ $P(P_n \in C_n) = \frac{{}^{n-1}C_1}{{}^{n}C_2} = \frac{2}{n} = O\left(\frac{1}{n}\right)$

No. of pairs with P_n as one element

No. of pairs

⇒ Expected time complexity is

$$= \sum_{n=2}^{i} O\left(\frac{1}{n}\right) \cdot O(n)$$

Time for MED with 2 points.

$$= \sum_{h=2}^{i} O(1) = O(i)$$

for $MED(P_1, P_2, \ldots, P_n)$

$$\rightarrow P(P_i \in C_i) = \frac{{}^{i-1}C_2}{{}^{i}C_3} = \frac{\frac{(i-1)(i-2)}{2}}{\frac{i(i-1)(i-2)}{6}}$$

$$= \frac{3}{i} = O\left(\frac{1}{i}\right)$$

So, expected time complexity

$$= \sum_{i=3}^{n} O\left(\frac{1}{i}\right) O(i) = O(n)$$

* Max cut - Monte Carlo Algorithm

Steps

1. $X \leftarrow \emptyset$, $Y \leftarrow \emptyset$. // (X, Y) is the cut
2. for each $v \in V$
3. randomly put v in X or Y
4. return (X, Y).

→ Expected cut size:

Size of a cut $(X, Y) = |\{(u,v) \in E : u \in X, v \in Y\}|$

$\forall e \in E$

$P(e \in C) = \frac{1}{2}$

Let X_i be a binary random variable associated with the ith edge (e_i) of the graph where $1 \le i \le |E|$, defined as:

$$X_i = \begin{cases} 0 & \text{if } e_i \notin C \\ 1 & \text{otherwise} \end{cases}$$

$$E(X_i) = \frac{1}{2} \times 1 = \frac{1}{2}$$

[∵ endpoints of the edge u, v can lie
→ both in X
→ both in Y
→ u in X, v in Y
→ u in Y, v in X]

let $X = \sum_{i=1}^{|E|} X_i$

$E(X) = E\left(\sum_{i=1}^{|E|} X_i\right)$

$E(X) = \sum_{i=1}^{|E|} E(X_i) = \frac{|E|}{2} = \frac{m}{2}$ [Where $m = |E|$]

~~$E(X) \geq \frac{m}{2}$~~

$P(\text{Size of random cut} \geq \frac{m}{2})$

i.e $P(X \geq \frac{m}{2}) \geq ?$

Let $P(X \geq \frac{m}{2}) = q$

Then $P(X \geq \lceil \frac{m}{2} \rceil) = q \Rightarrow P(X < \lceil \frac{m}{2} \rceil) = 1-q$

By definition,

$$E(X) = \sum_{i=1}^{m} i \cdot P(X=i) = \sum_{i=1}^{\lceil \frac{m}{2} \rceil - 1} i \cdot P(X=i) + \sum_{i=\lceil \frac{m}{2} \rceil}^{m} i \cdot P(X=i)$$

$$\leq \left(\lceil \tfrac{m}{2} \rceil - 1\right) \sum_{i=1}^{\lceil \frac{m}{2} \rceil - 1} P(X=i) + m \sum_{i=\lceil \frac{m}{2} \rceil}^{m} P(X=i)$$

$$= \left(\lceil \tfrac{m}{2} \rceil - 1\right)(1-q) + mq$$

$$= \lceil \tfrac{m}{2} \rceil - 1 - \lceil \tfrac{m}{2} \rceil q + q + mq$$

$$= \lceil \tfrac{m}{2} \rceil - 1 + q + \lfloor \tfrac{m}{2} \rfloor q$$

$\Rightarrow E(X) \leq \lceil \frac{m}{2} \rceil - 1 + q + \lfloor \frac{m}{2} \rfloor q$

$\Rightarrow \frac{m}{2} \leq \lceil \frac{m}{2} \rceil - 1 + q + \lfloor \frac{m}{2} \rfloor q$

$q\left(1 + \lfloor \frac{m}{2} \rfloor\right) \geq 1 + \frac{m}{2} - \lceil \frac{m}{2} \rceil$

$q\left(1 + \lfloor \frac{m}{2} \rfloor\right) \geq 1 - \left(\lceil \frac{m}{2} \rceil - \frac{m}{2}\right)$

$\lceil \frac{m}{2} \rceil - \frac{m}{2}$: m can be even or odd
⇒ equal to 0 or $\frac{1}{2}$

$\Rightarrow q\left(1+\left\lfloor\frac{m}{2}\right\rfloor\right) \geq \frac{1}{2}$

$$q \geq \frac{1}{2\left(1+\left\lfloor\frac{m}{2}\right\rfloor\right)}$$

$$q \geq \frac{1}{2\left(1+\frac{m}{2}\right)}$$

$$q \geq \frac{1}{m+2}$$

$\Rightarrow P\left(X \geq \frac{m}{2}\right) \geq \frac{1}{m+2}$

* How to get a constant lower bound of $P(X \geq \frac{m}{2})$?

→ Let us have `t' runs (in each run, we get a cut by the Monte Carlo algorithm).

For each run, $P\left(X \geq \frac{m}{2}\right) \geq \frac{1}{m+2}$

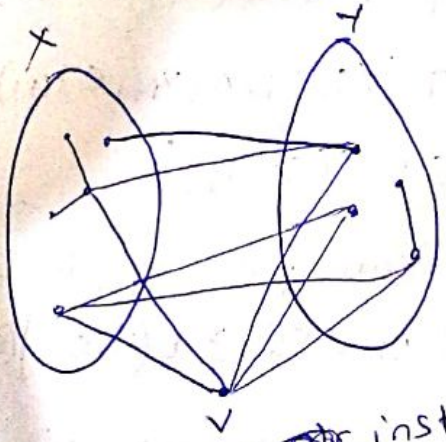
$\Rightarrow P\left(X < \frac{m}{2}\right) < 1 - \frac{1}{m+2}$

So, for all t runs, $P\left(X < \frac{m}{2}\right) \leq \left(1 - \frac{1}{m+2}\right)^t$

$\Rightarrow P\left(X \geq \frac{m}{2}\right)$ for one of the `t' runs $\geq 1 - \left(1 - \frac{1}{m+2}\right)^t$

If $t = m+2$ & $m \to \infty$ it is equal to $\frac{1}{e}$

$> 1 - \frac{1}{e}$



Now, instead of putting v randomly in X or Y, we put based on no. of edges it currently has in X or Y.

* Derandomization of Max cut Monte Carlo Algorithm :-

Let V_i be the set of first i vertices from V.

These i vertices define a cut (S_i, T_i) in which each vertex v_j $(1 \le j \le i)$ either belongs to S_i or belongs to T_i.

Let (S_i, T_i) be obtained by some deterministic manner.

Let (S, T) be the final cut on V. So, $S_i \subseteq S$, $T_i \subseteq T$.

Let $k(S,T)$ denote the cutsize of (S,T).

Let v_{i+1} has been randomly assigned to S_{i+1} or T_{i+1} with equal probability.

Then, ~~$E(k(S,T)) =$~~

$$E\left(k(S,T) \mid (S_i, T_i)\right)$$

$$= \frac{1}{2} E\left(k(S,T) \mid (S_i,T_i) \wedge (v_{i+1} \in S_{i+1})\right) + \frac{1}{2} E\left(k(S,T) \mid (S_i,T_i) \wedge (v_{i+1} \in T_{i+1})\right)$$

$$\Rightarrow \max\left\{E\left(k(S,T) \mid (S_i,T_i) \wedge (v_{i+1} \in S_{i+1})\right), E\left(k(S,T) \mid (S_i,T_i) \wedge (v_{i+1} \in T_{i+1})\right)\right\}$$

$$\ge E\left(k(S,T) \mid (S_i, T_i)\right)$$

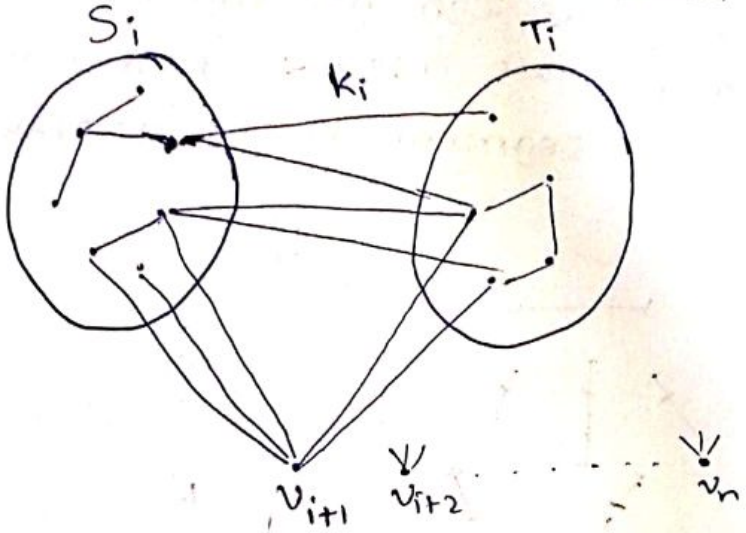
→ This means we can put v_{i+1} in S_{i+1} if

$$E\left(k(S,T) \mid (S_i,T_i) \wedge (v_{i+1} \in S_{i+1})\right) \ge E\left(k(S,T) \mid (S_i,T_i) \wedge (v_{i+1} \in T_{i+1})\right)$$

and in T_{i+1} otherwise. This doesn't worsen the expectation of the randomized

technique and gives a deterministic technique.

→ How to know which expectation is larger?



Let $m_S = \#\{(v_{i+1}, x) : x \in S_i\}$

$m_T = \#\{(v_{i+1}, y) : y \in T_i\}$

$K_i = \#\{(x,y) : x \in S_i,\ y \in T_i\}$

$m_i = \#\{(x,y) : x \in V - V_{i+1},\ y : (V - V_{i+1}) \cup S_i \cup T_i\}$

$$\max\{E(k(S,T) \mid (S_i,T_i) \wedge (v_{i+1} \in S_{i+1}),\ E(k(S,T) \mid (S_i,T_i) \wedge (v_{i+1} \in T_{i+1}))\}$$

$$= \max\left\{ K_i + m_T + \frac{m_i}{2},\ K_i + m_S + \frac{m_i}{2} \right\}$$

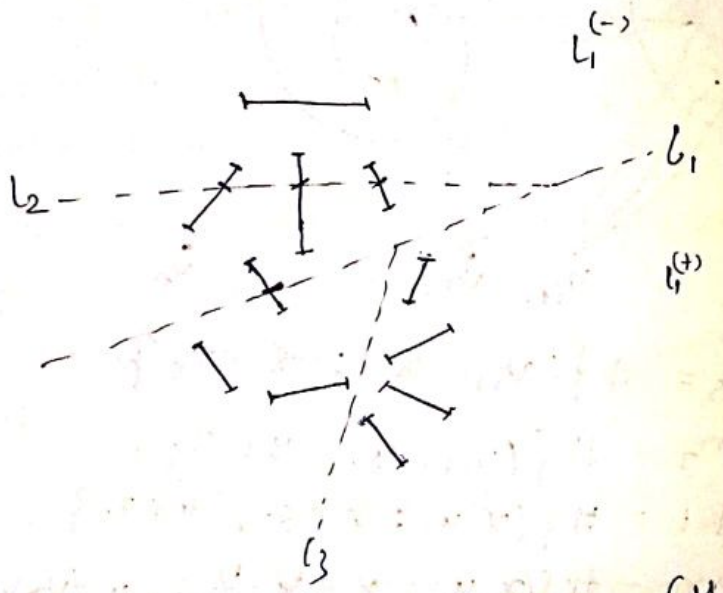
⇒ max. depends on m_S or m_T.

Pseudocode :

```
1. S ← {v1} , T ← ∅
2. for i ← 2 to m
3.     if (mS ≥ mT)
4.         T ← T ∪ {vi}
5.     else S ← S ∪ {vi}
6. return (S, T).
```

* Randomised Datastructures:- (Binary Space Partitioning (BSP) Tree)

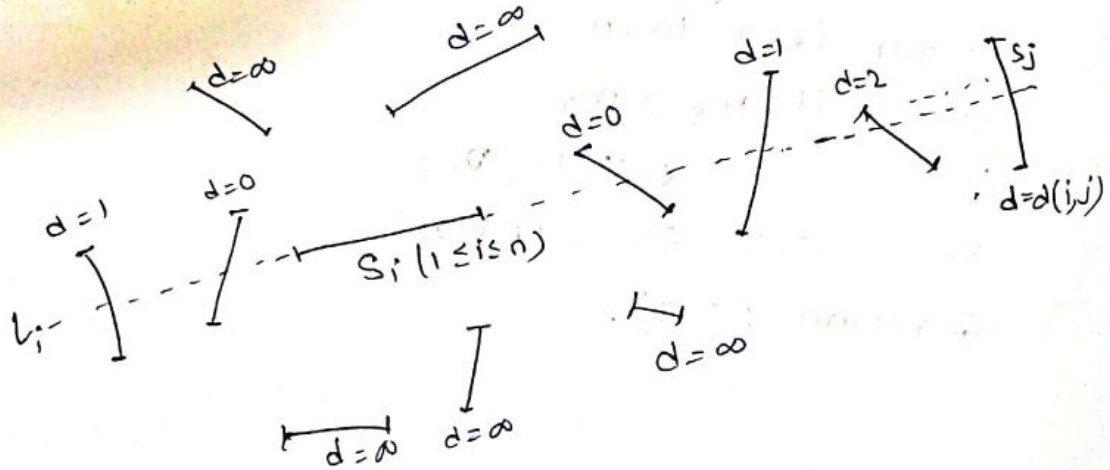
Given some line segments and a query line segment, we need to find out the nearest line segment to the query line segment



whenever a line (say l_1) cuts a line segment, it becomes two line segments, one on the left & the other on the right.

⇒ No. of line segments increases in that case.

Remedy:- choose one of the line segments randomly as the dividing line.



d → No. of lines, before the line segment, intersected by l_i

Expected number of line segments intersected by l_j = ?

$$P(l_i \text{ intersects } s_j) \leq \frac{1}{d(i,j)+2}$$

over the whole plane, l_i intersects s_j. But If any of the segments between s_i & s_j are chosen as lines before s_i, then s_i & s_j fall under different halves. So it can't intersect.

⇒ Expected number of line segments intersected by

$$l_j \leq \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{1}{d(i,j)+2} = O(\log n)$$

⇒ E(segments produced by l_i) $\leq 2 \cdot O(\log n) = O(\log n)$

⇒ E(segments produced by all lines) $= O(n \log n)$.

*APPROXIMATION ALGORITHMS :

→ Used mostly to solve NP-complete problems

→ provides sub-optimal solutions, i.e, the solution is close to the optimal with some guarantee.

→ The guarantee is given through approximation ratio (say, ρ).

→ Time complexity is polynomial in input size.

→ Job Scheduling Problem (NP-complete)

There are 'm' identical machines: $M_1, M_2, \ldots, M_m$.

There are 'n' jobs : $1, 2, 3, \ldots, j, \ldots, n$.

The time required for job 'j' = t_j

Let $A(i)$ = set of jobs assigned to M_i

Each ~~active~~ machine can process one job at a time. Once a job is started, it must be finish

Then total processing time for M_i is (total load)

$$T_i = \sum_{j \in A(i)} t_j$$

Task :- Minimize $\max_{1 \le i \le m} \{T_i\}$

T_{app} = approximation result (returned by the approximation algorithm).

T^* = optimal result

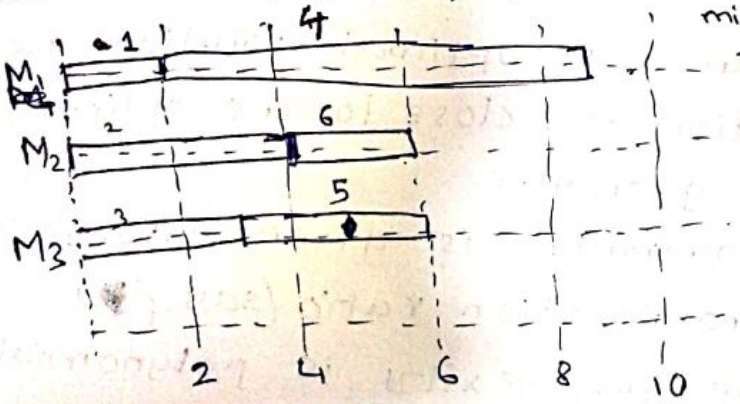
$$\rho = \frac{T_{app}}{T^*} \le ?$$

* Ex :- Say there are 3 machines : M_1, M_2, M_3.

~~Job times : 2, 3, 4,~~

Job times : 2, 4, 3, 7, 3, 2

Greedy technique
Assign the next job to machine with min. load.



$T_{app} = \max \{T_1, T_2, T_3\} = 9$

But $T^* = 7 \Rightarrow \rho = \frac{9}{7}$

*Observations:

m → # machines
n → # jobs

1. $T^* \geq \frac{1}{m}\sum_{j=1}^{n} t_j = \frac{1}{m}\sum_{i=1}^{m} T_i$

2. $T^* \geq \max_{1 \leq j \leq n} \{t_j\}$

3. Let M_L be the machine ~~on which job n was loaded (according to our Greedy technique)~~ with maximum load at the end

i.e $T_{app} = T_L$

Let 'p' be the last job assigned to M_L

⇒ Before 'p' was added to M_L, M_L has the least load among all machines.

If there are jobs after 'p', they will get assigned to the other machines since M_L has the highest load ever since 'p' was added.

⇒ $T_L - t_p$ is the minimum among all loads before p is added.

Since minimum is less than the average,

$$T_L - t_p < \frac{1}{m}\sum_{j=1}^{p-1} t_j$$

$$\Rightarrow T_L - t_p < \frac{1}{m}\sum_{j=1}^{n} t_j \leq T^* \quad [\because \text{from } 1]$$

$$\Rightarrow T_L - t_p < T^*$$

$$\Rightarrow T_{app} - t_p < T^*$$

$$T_{app} < T^* + t_p \leq T^* + \max_{1 \leq j \leq n} \{t_j\}$$

$$\Rightarrow T_{app} \leq T^* + \max_{1 \leq j \leq n} \{t_j\}$$

From (2),

$$T_{app} < T^* + T^*$$

$$\Rightarrow T_{app} < 2T^* \Rightarrow \boxed{\rho < 2}$$

20-3

2-factor Approximation algorithm (Greedy Technique)

1. $A_i \leftarrow \emptyset$ for $i = 1, 2, \ldots, m$, $T_i \leftarrow 0$
2. for $j \leftarrow 1$ to n
3. assign job j to M_i with min T_i
4. $A_i \leftarrow A_i \cup \{j\}$, $T_i \leftarrow T_i + t_j$
5. return $\{A_i : 1 \le i \le m\}$

Approximation ratio = 2

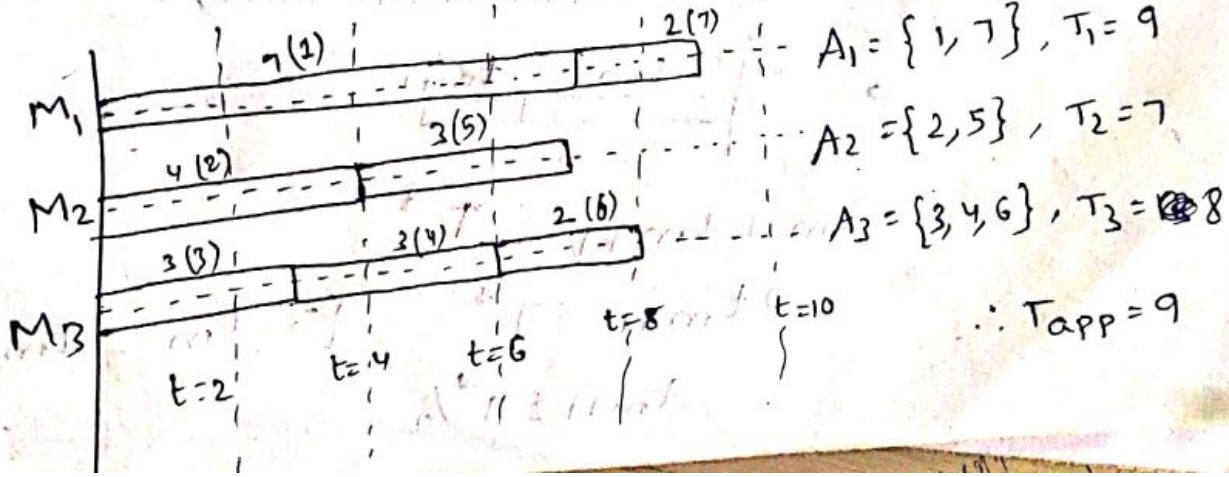
* Improved algorithm

Sort the jobs in non-increasing order of t_j and then assign them to machines in the same way (greedy) as previous.

Ex: $M = 3$, $n = 7$, $\{t_j\} = \{2, 3, 7, 4, 2, 3, 3\}$

↓ sort

$\{7, 4, 3, 3, 3, 2, 2\}$



$A_1 = \{1, 7\}$, $T_1 = 9$

$A_2 = \{2, 5\}$, $T_2 = 7$

$A_3 = \{3, 4, 6\}$, $T_3 = 8$

$\therefore T_{app} = 9$

*Observations :-

1. Let M_l be the machine with the highest load at the end.

So, load of M_l is T_l and $T_{app} = T_l$

Let p be the last job assigned to M_l.

Then, $T_l - t_p \leq$ min. of all loads

$\Rightarrow T_l - t_p \leq$ avg. of all loads

$$\Rightarrow T_l - t_p \leq \frac{1}{m}\sum_{i=1}^{m} T_i = \frac{1}{m}\sum_{j=1}^{p-1} t_j$$

$$\therefore T_l - t_p < \frac{1}{m}\sum_{j=1}^{n} t_j \leq T^*$$

$$\Rightarrow T_{app} \leq T^* + t_p \longrightarrow (1)$$

2. $t_p \leq \max_{j=1}^{n}\{t_j\}$

2. $t_p \leq \max\{t_j : 1 \leq j \leq n\} = t_1$

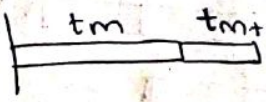
We will consider jobs $m+1, \ldots, n$, since assignment of jobs 1 to m is always optimal (because if it is from 1 to m, the algorithm always gives an optimal sol^n) each job to a machine

So, $t_p \leq \max\{t_j : m+1 \leq j \leq n\} = t_{m+1} \longrightarrow (2)$

$t_p \leq \min\{t_j : 1 \leq j \leq m\}$

$t_p \leq \text{avg}\{t_j : 1 \leq j \leq m\}$

When $(m+1)^{th}$ job was assigned, 1^{st} m jobs were assigned optimally. So $(m+1)^{th}$ job is also assigned optimally by the algorithm (since it's added to machine with min. load).

$\Rightarrow$ 

$\therefore t_m + t_{m+1} \leq T^*$

$2t_{m+1} \leq T^* \quad [\because t_m \geq t_{m+1}]$

$t_{m+1} \leq T^*/2$

② $\Rightarrow t_p \leq t_{m+1}$

$\Rightarrow t_p \leq \frac{T^*}{2}$

① $\Rightarrow T_{app} \leq T^* + \frac{T^*}{2} = \frac{3T^*}{2}$

$$\frac{T_{app}}{T^*} \leq \frac{3}{2}$$

$\therefore \rho = \frac{3}{2}$

* Set Cover problem: (NP-Complete)

Given:- $U = \{x_1, x_2, \ldots, x_n\}$

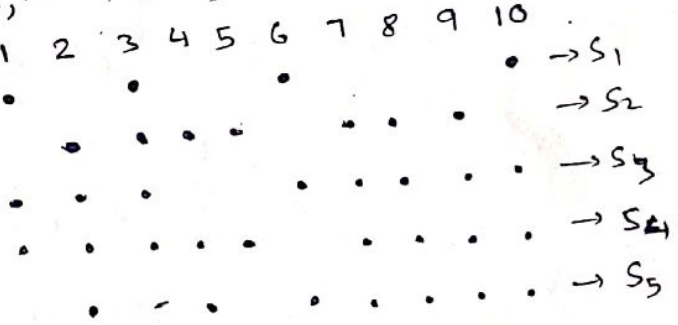
$\mathcal{F} = \{S_1, S_2, \ldots, S_k\}$

where $S_i \subseteq U$ for $1 \leq i \leq k$

Find the smallest sub-family of $\mathcal{F}$ such that its member sets unite to U.

Ex:- $U = \{1, 2, \ldots, 10\}$

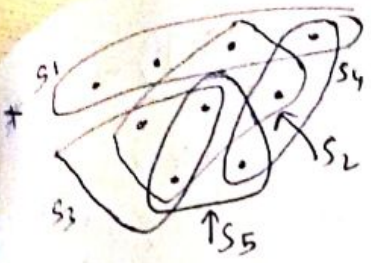
$\mathcal{F} =$



Greedy Algorithm

1. $T \leftarrow U$, $K \leftarrow \{\}$
2. while $T \neq \emptyset$
3. ~~find 'i' such that~~
   find i
   s.t. $|T \cap S_i| = \max\{|T \cap S_i| : i \in \{1,2,\ldots,k\}\}$
4. $T \leftarrow T - S_i$, $K \leftarrow K \cup \{i\}$
5. return K.

In every step, assign a cost of $\frac{1}{|T \cap S_i|}$ to each $x \in \{T \cap S_i\}$.



$\mathcal{F} = \{S_1, S_2, S_3, S_4, S_5\}$

$K = \{S_2, S_1, S_4\}$

cost = $5 \times \frac{1}{5}$ $3 \times \frac{1}{3}$ $1 \times \frac{1}{1}$

Total cost = $1+1+1 = 3$ = min # sets required

$= |K|$

* Let K^* be an optimal solution.

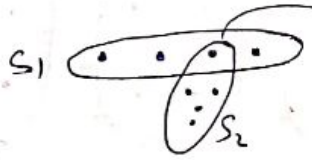
So, approximation ratio = $\frac{|K|}{|K^*|}$

$$|K| = \sum_{x \in U} cost(x)$$

* Let S_i denote the i^{th} set of $\mathcal{F}$.

Let x_{ij} denote the j^{th} element of S_i

i.e, covered as j^{th} element by any set.



This is x_{11} because among the 4 elements of S_1, it is first covered (by S_2).

Then $cost(x_{ij}) \leq \frac{1}{|S_i \cap T|}$

Since if the j^{th} element is covered by S_i itself, cost = $\frac{1}{|S_i \cap T|}$

if it is covered by $S_k (\neq S_i)$, then

$|S_k| > |S_i| \Rightarrow cost < \frac{1}{|S_i \cap T|}$

No. of uncovered elements

$|S_i \cap T| \geq |S_i| - (j-1)$

$\Rightarrow cost(x_{ij}) \leq \frac{1}{|S_i| - j + 1} \rightarrow (2)$

$\therefore |K| = \leq \sum_{S_i \in K^*} \sum_{x_{ij} \in S_i} cost(x_{ij}) \rightarrow (1)$

[∵ There will be some repetitions of elements & sum of all costs = 1]

$$\sum_{x_{ij} \in S_i} cost(x_{ij}) \leq \sum_{j=1}^{|S_i|} \frac{1}{|S_i|-j+1} = h(|S_i|) \rightarrow (2)$$

harmonic function $\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots\right)$

(1) & (3) $\Rightarrow$ $|K| \leq \sum_{S_i \in K^*} h(|S_i|)$

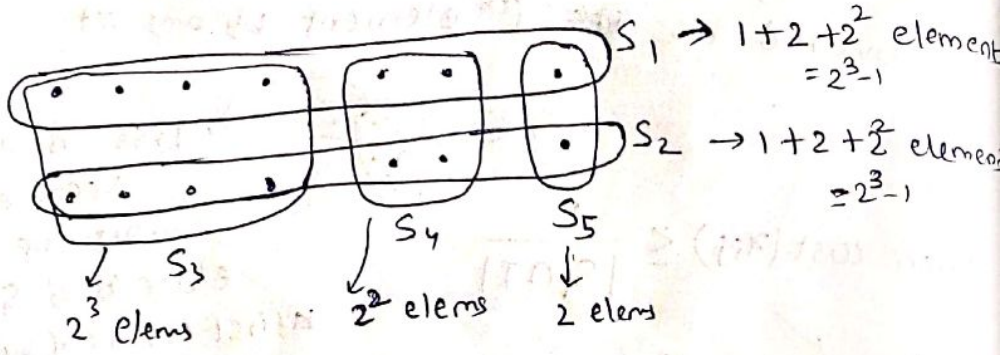
Let $M = \max\{|S_i| : S_i \in \mathcal{F}\}$

$\Rightarrow |K| \leq \sum_{S_i \in K^*} h(M)$.

$\Rightarrow |K| \leq h(M)|K^*| = h(O(n))|K^*|$

$\Rightarrow \frac{|K|}{|K^*|} \leq h(O(n)) = O(\log n)$ → By ~~integrating~~ limit of sum, integration

$\therefore \boxed{\rho = O(\log n)}$

* 

Let $n = 2(2^k - 1)$ ← Total # elements → From the algo, we get

$|K| = k$

But $|k^*| = 2$

$\Rightarrow \rho = O(\log n)$

* Vertex Cover