Que ⇒ Given a set of n integers. Determine an integer k and determine the kth smallest.

→ (A) Sort + Binary search $= O(n\log n) + O(\log n)$
(Quick s.)
$= O(n\log n)$

(B) Min-heap — $O(n)$ time. $(O(n + k\log n))$
↳ Remove smallest k elements from the heap.

(C) Using partition function of Quick Sort

Avoid Sorting

(i) Pick a random element R.
(ii) Split X into :-

U	V	W
$<R$	$=R$	$>R$.

⇒ Random number cannot be generated by Computer as it is a deterministic machine. (uses specific architecture - Not actualy random)

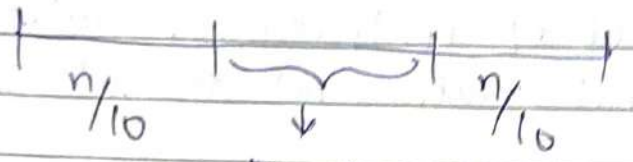
→ To solve this, everytime by luck "splitter" happens to be "median". [use Pseudomedian]

$T(n)$ = time to select k^{th} from n element.
$= kn + T(\frac{n}{2}) = O(n)$

Date
Page

* **Pseudo Median:-** [Use as a splitter]

⇒ P is a pseudomedian of X if atleast $\frac{n}{10}$ are smaller and $\frac{n}{10}$ are larger



$$T(n) = kn + T\left(\frac{9n}{10}\right) + T_p(n)$$

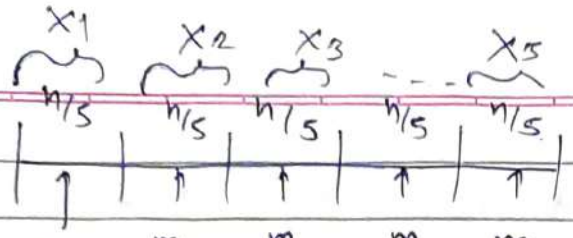
Algo:- Select (X, r)

```
    m = pseudomedian (X)
    U = { x | x ∈ X & x < m }
    V = { x | x ∈ X & x = m }
    W = { x | x ∈ X & x > m }.

    if ( r ≤ |U| return select (U, r)
else if ( r ≤ |U| + |V| ) return m
else return select (W, r - |U| - |V|)
```

```
pseudomedian (X)
    n = |X|
    S = ⌊ n/5 ⌋

    for i = 1 to S
        Xi = ith set of 5 elements from X.
        mi = median (Xi)
    end
    M = {mi}   (set of all medians)
return Median(M)
```



$\Rightarrow$ Medians of each part.

Using these medians we will determine the pseudomedian.

$\Rightarrow$ Claim : (i) $|U| < \frac{3n}{4}$ (ii) $|W| < \frac{3n}{4}$

$\rightarrow$ Proof :- Count the # of elements that cannot be in U.

(i)

= # of elements that are not smaller than M. (Median of entire set).

= # of medians (m_i) not smaller than M. $\geq \left\lfloor \frac{S}{2} \right\rfloor$ (S = # of parts of size $n/5$)

$\Rightarrow$ For each m_i which is not smaller than M, set X_i contains 2 other elements which are also not smaller.

$\therefore$ Total # of elements in X which are not smaller than M is atleast $\left(\frac{S}{2} + 2 * \frac{S}{2}\right) = \frac{3S}{2}$

Now, $S = n/5$, $= 3\left\lfloor \frac{\lfloor n/5 \rfloor}{2} \right\rfloor = 3\left\lfloor \frac{n}{10} \right\rfloor \geq n/4$

$\therefore$ # of elements which cannot be in U

$\therefore$ No. of elements in $U = n - \frac{n}{4} = \frac{3n}{4}$.

Date
Page

$\Rightarrow$ Claim :- Select (x, r) takes time $O(n)$

Proof :- $T(n)$ = time for select to Complete.

$\therefore T(n) = O(n) + T\left(\frac{n}{5}\right) + O(n) + T\left(\frac{3n}{4}\right)$

$T(n) = O(n) + T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right)$ Solve it using tree Method.

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

$\therefore$ k^{th} smallest / k^{th} largest $\Rightarrow$ $O(n)$ time (Divide & conquer technique).

$\Rightarrow$ $T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right) + O(n)$.

$T(n/5) \Rightarrow a=1, b=5$, $T\left(\frac{3n}{4}\right) \Rightarrow a=1, b=4/3$.

$f(n) = O(n)$ $\qquad$ $f(n) = O(n)$

$n^{\log_b a} \Rightarrow n^{\log_5 1} \Rightarrow n^0 = 1$ $\qquad$ $n^{\log_b a} = n^{\log_{4/3} 1} = n^0 = 1$

$\therefore f(n) > n^{\log_b a}$ $\qquad$ $\therefore n > 1$

$\Rightarrow O(n)$ $\qquad$ $\Rightarrow O(n)$

$\therefore T(n) = O(n) + O(n) + O(n) = O(n)$

$\therefore \boxed{T(n) = O(n)}$

Que:- Suppose n is a power of 2

Que:- Given positive integers y and z, Compute $x = yz$. Each number is of n bits and n is power of 2.

- Naive approach $= O(n^2)$
- Using Divide and Conquer :-

⇒ Divide y and z into two halves each with $n/2$ bits.

y	a	b
z	c	d.

$$y = a \cdot 2^{n/2} + b$$
$$z = c \cdot 2^{n/2} + d$$

$$\therefore yz = (a \cdot 2^{n/2} + b)(c \cdot 2^{n/2} + d)$$

$$= \underbrace{ac}_{n/2 \text{ bits}} 2^n + (\underbrace{ad}_{n/2 \text{ bits}} + \underbrace{bc}_{n/2 \text{ bits}}) 2^{n/2} + \underbrace{bd}_{n/2 \text{ bits.}}$$

4 multiplication of $n/2$ bits.

→ Complexity :- $T(1)$ = constant.

$$T(n) = 4T(n/2) + dn$$

$$= \underline{\underline{O(n^2)}}$$ { Still got the same Complexity }

Date
Page

Variation in previous technique:-

① $u = (a+b)(c+d) \Rightarrow$ Mul. of $(n/2+1)$ bit #.
② $v = ac$ } Multiplication of $n/2$ bit number
③ $w = bd$.

$\therefore x = v\,2^n + (u - v - w)2^{n/2} + w$

→ Substraction and Shifts. (2^n and $2^{n/2}$ can be done by shifts)

→ Complexity :- Treat the leading bits of $(a+b)$ & $(c+d)$ Seperately.

$(a+b)$ | a_1 | b_1 |
$(c+d)$ | c_1 | d_1 |

$\Rightarrow (a+b) = a_1 2^{n/2} + b_1$
$(c+d) = c_1 2^{n/2} + d_1$

$\therefore (a+b)(c+d) = a_1 c_1 2^n + (a_1 d_1 + b_1 c_1) 2^{n/2} + b_1 d_1$

= 3 multiplication of $n/2$ bits. and Constant number of addition & shifts.

$$\therefore T(n) = \begin{cases} \text{Constant} & \text{if } n = 1 \\ 3T\left(\frac{n}{2}\right) + dn & \text{Otherwise.} \end{cases}$$

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

Date
Page

Que:- Multiply two n^{th} degree polynomials :-

$\Rightarrow$ $P(x) = P_0 + P_1 x^1 + P_2 x^2 + P_3 x^3 + \dots + P_{n-1} x^{n-1}$
$Q(x) = Q_0 + Q_1 x^1 + Q_2 x^2 + Q_3 x^3 + \dots + Q_{n-1} x^{n-1}$

$R(x) = P(x) \cdot Q(x)$

Given $P_0, P_1, \dots P_{n-1}$ & $Q_0, Q_1, \dots Q_{n-1}$ return $r_0, r_1, \dots r_{2n-2}$.

$\rightarrow$ Brute-force :- $O(n^2)$

$\rightarrow$ Better solution :-

$\Rightarrow$ P is represented as $(a_0, a_1, a_2 \dots a_{n-1})$
$a_i = P(i)$

$\Rightarrow$ n points are sufficient to define a $(n-1)^{th}$ ~~dep~~ degree polynomial.

$\Rightarrow$ (Represent Co-efficient to point value) in $O(n \log n)$ time.

P, Q in Co-efficient Representation	$O(n^2)$ $\longrightarrow$	Product in Co-efficient form	final complexity
$O(n \log n)$ $\downarrow$		$\uparrow$ $O(n \log n)$	
P, Q in Point value representation	$O(n)$ $\longrightarrow$	Product in Point value representation	

Next :- Matrix Multiplication

* Co-efficient representation to point value representation :- $(O(n\log n))$ [Fast Fourier Transform]

⇒ In co-efficient representation the co-efficients of the polynomial are represented as vectors and thus while multiplying two polynomial we need to multiply these vectors which will ~~the~~ take $O(n^2)$ time.

⇒ But it can be converted to point value representation ~~[illegible]~~ in $O(n\log n)$ time.

⇒ Polynomial $A(x) = \sum_{k=0}^{n-1} a_k x^k$

Eg:- $A(x) = x^2+3x+2$, $B(x) = 2x^2+1$

$C(x) = A(x)\cdot B(x) = (x^2+3x+2)(2x^2+1)$

$C(x) = x^2(2x^2+1) + 3x(2x^2+1) + 2(2x^2+1)$

⋮ (continous multiplications).

⇒ Co-efficient Representation :-

$O(d^2)$

$A(x) = a_0 + a_1x + \dots + a_dx^d \Rightarrow [a_0, a_1 \dots a_d]$

$B(x) = b_0 + b_1x + \dots + b_dx^d \Rightarrow [b_0, b_1 \dots b_d]$

$C(x) = c_0 + c_1x + \dots + c_{2d}x^{2d}$

Theorem ⇒ Any polynomial with degree d can be represented by (d+1) points uniquely.

Proof :- ⇒ $\{(x_0, P(x_0)), (x_1, P(x_1)) \dots (x_d, P(x_d))\}$

$\Rightarrow P(x) = p_0 + p_1 x + \dots p_d x^d$.

$\therefore$ $P(x_0) = P_0 + P_1 x_0 + P_2 x_0^2 + \dots + P_d x_0^d$

$P(x_1) = P_0 + P_1 x_1 + P_2 x_1^2 + \dots + P_d x_1^d$

$\vdots$

$P(x_d) = P_0 + P_1 x_d + P_2 x_d^2 + \dots + P_d x_d^d$

DFT Matrix

$$\begin{bmatrix} P(x_0) \\ P(x_1) \\ \vdots \\ P(x_d) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^d \\ 1 & x_1 & x_1^2 & \dots & x_1^d \\ \vdots & \vdots & & & \\ 1 & x_d & x_d^2 & \dots & x_d^d \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_d \end{bmatrix}$$

$\hookrightarrow$ M is invertible for unique $x_0, x_1 \dots x_d$

$\Rightarrow$ Unique $p_0, p_1 \dots p_d$ exists

$\Rightarrow$ Unique polynomial $P(x)$ exists.

$\therefore$ <u>Value-Representation</u>:-

$\{(x_0, P(x_0)), (x_1, P(x_1)) \dots (x_d, P(x_d))\}$

Which generates a unique polynomial as proved in above theorem.

This is used to reduce the complexity from $O(d^2)$ to $O(d)$.

$\Rightarrow$ Let $A(x)$ and $B(x)$ be two polynomials and $C(x) = A(x) \cdot B(x)$. ~~Then~~ $A(x)$ and $B(x)$ both are of degree 2 and thus $C(x)$ = degree 4 and thus needs $4+1$ points. $= 5$. So, take 5 points ~~for~~ from $A(x)$ and $B(x)$ each and simply multiply those $d+1$ points and we will get a unique polynomial $C(x)$ which will be ~~the~~ result of multiplication of two polynomial.

$\hookrightarrow$ $d+1$ multiplication

$\Rightarrow$ ~~As it takes d~~ $\Rightarrow O(d)$.

⇒ we know that when $P(x)$ = even then $P(-x) = P(x)$ and when $P(x)$ = odd then $P(-x) = -P(x)$. So, to find the $d+1$ points for a unique polynomial we will only find half of the points and rest will be having same value & sign according to odd/even functions.

⇒ For a +ve, -ve pair we need a points which are n^{th} roots of unity [i.e. real and complex roots]

Algo:- $P(x) : [P_0, P_1, \ldots P_{n-1}]$, $\omega = e^{2\pi i/n} : [\omega^0, \omega^1 \ldots \omega^{n-1}]$

⇒ Base case ⇒ $\boxed{n=1 \Rightarrow P(1)}$

$P_e(x^2) : [P_0, P_2 \ldots P_{n-2}]$ — Even degree — $[\omega^0, \omega^2, \ldots \omega^{n-2}]$

$P_o(x^2) : [P_1, P_3, \ldots P_{n-1}]$ — Odd degree — $[\omega^1, \omega^3, \ldots \omega^{n-1}]$

Find $n/2^{th}$ root of unity

$\therefore y_e = [P(\omega^0), P(\omega^2) \ldots P(\omega^{n-2})]$, $y_o = [P_o(\omega^1), P_o(\omega^3), \ldots P_o(\omega^{n-1})]$

$$\therefore P(x_j) = P_e(x_j^2) + x_j P_o(x_j^2).$$
$$P(-x_j) = P_e(x_j^2) - x_j P_o(x_j^2). \quad , j \in \{0, 1, \ldots n/2 - 1\}$$

Now, $x_j = \omega^j$ and $-\omega^j = \omega^{j+n/2}$

$$\therefore P(\omega^j) = y_e[j] + \omega^j y_o[j] \qquad \Rightarrow y_e[j] = P_e(\omega^{2j})$$
$$P(\omega^{j+n/2}) = y_e[j] - \omega^j y_o[j] \qquad y_o[j] = P_o(\omega^{2j})$$

$\therefore y = [P(\omega^0), P(\omega^1), \ldots P(\omega^{n-1})]$.

Code:-
```
def FFT(P): # P - [P0, P1 --- Pn-1] ⇒ Co-eff. Representation
    n = len(P)
    if n == 1:
        return P
    ω = e^(2πi/n)
    Pe, Po = [P0, P2, ... Pn-2], [P1, P3 ... Pn-1]
    ye, yo = FFT(Pe), FFT(Po)
    y = [0] * n.
    for j in range(n/2)
        y[j] = ye[j] + ω^j yo[j]
        y[j+n/2] = ye[j] - ω^j yo[j]
    return y.
```

$O(n \log n)$.

Value → Co-eff :- Inverse a DFT Matrix :-

∴ For value → co-eff :- $\omega = \frac{1}{n}\omega^{-1} = \frac{1}{n} e^{-2\pi i/n}$

(Inverse FFT)

Date
Page

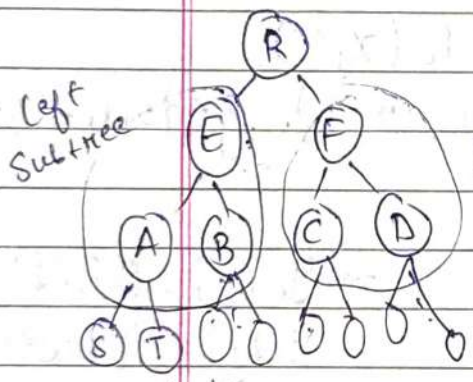
* Matrix Multiplication :-

Que :- Find a compact layout for a complete binary tree.

⇒ $N = 2^n$, height = h and all the leaf nodes are at level h.

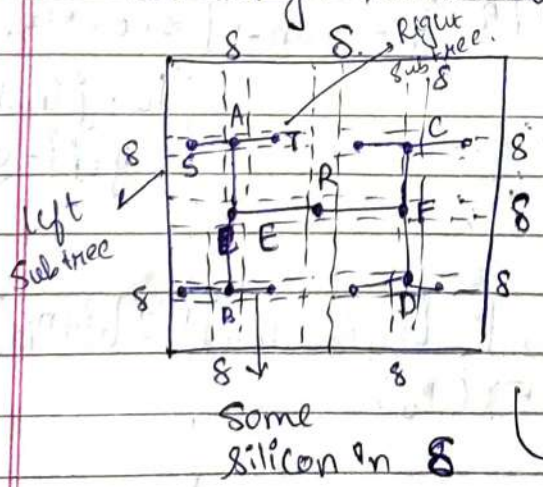
⇒ $N = 2^{h+1} - 1$, h = height.

⇒ Optimal design for binary tree :-



Solution :- Take a n×n rectangular slab.

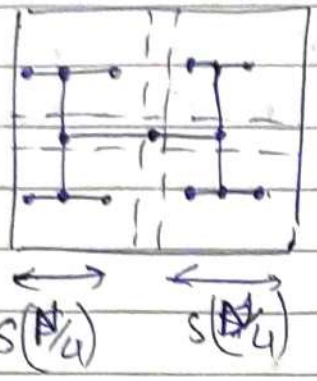
⇒ Here, we need to take care of all the conditions like we can't have a link between B and C directly. Similarly, for all others.



⇒ Here δ is the space where the silicon is removed ~~so that they will not be there~~

This way we can divide the space and place all the nodes.

⇒ Complexity:-



$\therefore S(N) = 2S\left(\frac{N}{4}\right) + 1$

$a = 2,\ b = 4$

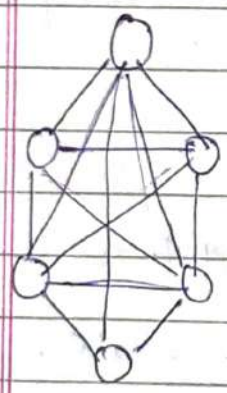
$n^{\log_b a} = n^{\log_4 2}$

$= n^{\frac{1}{\log_2 4}} = n^{1/2} = \sqrt{n}$

$\therefore \boxed{S(N) = O(\sqrt{n})}$

Que:- Layout design for a Complete Graph:-

⇒ All vertices are connected to every other vertices of the graph — Complete graph.



Floor ⟹ Representation.

* Greedy Algorithms :-

Eg:- Minimum spanning Tree, optimization problems, sceduling problems, profit maximization, Coin change problem etc.

① Scheduling problem :-

⇒ n jobs (1 to n).
⇒ Each job needs unit time for processing.
⇒ One processor available.
⇒ Profit on i^{th} job = P_i, Deadline for i^{th} job = d_i

⇒ Job i must execute in interval [1..d] otherwise no profit on i.

⇒ ~~Sced~~ Schedule the jobs to maximize the profit

Goal :- ① Which jobs to execute?
② When to execute?

Select jobs in order of decreasing profit.
Retain as much flexibility as possible.
Schedule selected job at latest possible time.

⇒ Theorem :- Greedy schedule is optimal
i.e. There does not exist another (non-greedy) schedule with higher profit.

Date
Page

~~Lemma~~ → Optimal schedule may not be unique.

Proof:- Consider some optimal schedule S. We will prove that profit of greedy strategy (G) = profit of optimal schedule (S).

⇒ S must contain the highest profit job. i^{th} most profitable job must be present if it's deadline is greater than or equal to 1.

Now, here there are other schedules which are possible and G is the greedy schedule.

$S = S_0$ $\quad P(S_0) = P(S)$

$= S_1$ $\quad P(S_1) = P(S_0)$

$= S_2$ $\quad \vdots$

$\vdots$

$G = S_i$ $\quad P(S_i) = P(S_{i-1})$

⇒ Properties of S_i: i most profitable jobs in S_i are also the i most profitable jobs in G and further ~~for that~~ this jobs are ~~represented~~ executed at the same time in S_i & G.

Lemma:- Suppose job j does not appear in G. Then at every instant in the interval $[1..d_j]$ a job with higher profit than j has been scheduled.

Lemma:- Suppose in G the ith most profitable job is scheduled at time t. Then a higher profit job is scheduled at every instant in $[t+1, d_i]$

Lemma:- S_i has the same profit as S. The i highest profit jobs in S_i are the same as the highest profit jobs in G. The i highest profit jobs are scheduled at the same time in S_i & G

→ There exists schedule S_i such that for all $i = 1 \ldots$ number of jobs in G.

⇒ Proof based on induction:-

→ Induction over i :-

→ Base case :- $i = 0$. Select $S_0 = S$. This satisfies all conditions.

→ Suppose the lemma holds for S_{i-1} then $\exists S_i$

Case-1 Profit (S) < Profit (g)

(i) S can't be from the most profitable i-1 jobs.

(ii) profit (S) can't be larger than profit (g) then S should be the nth most profitable job $S_{i-1} \Rightarrow S$

Case-2:- profit(s) > profit(g)

⇒ Does g occur in S_{i-1}? ⇒ g does not occur in S_{i-1}

⇒ we can replace s with g. Now ~~schedule~~ Schedule will have larger profit. Hence S_{i-1} is not optimal.

⇒ s is absent in G. let t = time at which s is scheduled in S_{i-1}

⇒ Greedy property :- Some task of higher profit was ~~were~~ scheduled.

⇒ i^{th} highest profit.

⇒ profit(g) > profit(s) → Contradiction

⇒ g = s.

⇒ Thus, whatever jobs are scheduled in s at time t, exactly same jobs are scheduled in g at time t.

⇒ Thus S is optimal Solution.

(2) Select jobs which do not overlap. Maximize their number.

⇒ n - jobs, s_i - starting time of job i
f_i - finishing time of job i

1 - processor [CPU scheduling]

Solution:- Arrange in ascending order of :-
(i) Smallest time duration $(f_i - s_i)$
(ii) Finishing time
(iii) Minimum number of overlaps
(iv) Select in order of given numbering.

(1) With Starting and finishing time :-

↳ ① Sort in whatever order (earliest finishing time)
② Mark first activity
③ Eliminate those activities that overlap with selected activity.
④ Repeat from ① & ② as appropiate.

Claim:- Algorithm returns an optimal solution.
$S(n) = S(n') \cup \{1\}$ is optimal.

Proof:- Assume algorithm works for problem size $\leq n-1$
⇒ Base Case :- $n=1$, solution is trivial $\{1\}$

⇒ Suppose $S(n)$ as discovered by the algorithm is not optimal.

Date
Page

⇒ Let O be an optimal solution to the problem P(n).

∵ # jobs in O > # of jobs in S(n).

O = # of jobs in S(n') + 1 — (1).

Case-1: O contains job-1

⇒ Let O' = O − {1}. So, O' must be a sol^n to P(n').

{O could not contain jobs overlapping with 1 ⇒ O' can also not contain jobs overlapping with 1}.

# of jobs in O' = # of jobs in O − 1.

> # of jobs in S(n') + 1 − 1

≥ # of jobs in S(n'). ⇓ (from (1))

Case-2:- O does not contain job-1.

⇒ If O ∪ {1} is a valid schedule then O cannot be optimal.

⇒ Hence a contradiction.

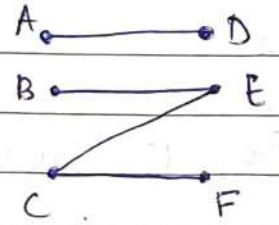
∴ O ∪ {1} is not a valid schedule.

Date
Page

(3) Maximum Weight Matching :-

⇒ Matching :- Set of edges such that no two share an end point.

Eg :-



⇒ Perfect Matching :- All vertices must be matched.

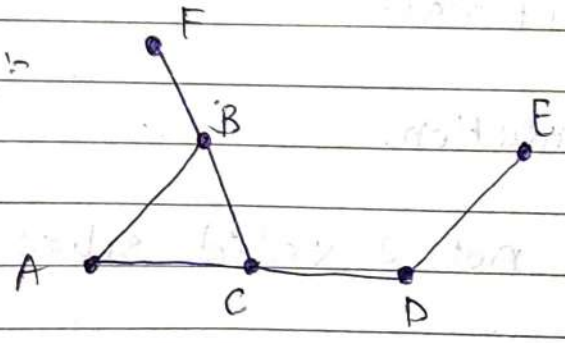
⇒ The problem of finding, in a weighted graph, a matching in which the sum of weights is maximized.

(4) Independent Set :-

In a graph $G = (V, E)$ we say that $V' \in V$ is independent if $u, v \in V'$, $(u, v) \notin E$

⇒ Maximum Independent set :- Given a graph G, find an independent set of maximum cardinality.

Eg :-



$V = \{F, B, A, C, D, E\}$

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

$G = (V, E)$.

$V' = \{F, C, E\}$
$V' = \{F, A, D\}$] → Independent set.

Date
Page

⇒ Usind BFS or DFS we can find thes independent set.

Like (1) start from F :- we can only visit B as there is an edge. So visit B but can't be included with F in independent set.

(2) From B, we can go to A or C. So we can include any one as A and C also contains an edge between them. So, Go to C and include in independent set :- {F, C}. And so on... Set = {F, C, E}.

OR If we go with A then Set = {F, A, D}.

⇒ Maximal Independent set :- V' is said to be maximal independent set of G = (V, E) if there does not exist any $v \in V - V'$ such that $V' \cup \{v\}$ is an independent set.

Eg :- Above example :- {B, D} is a maximal independent set, but not maximum.

⇒ Every Maximum independent set is also maximal but vice versa is not true.

↳ {F, C, E} and {F, A, D} are maximal but {B, D} is not maximum.

Date
Page

* **Graph Colouring:-** $G = (V, E)$. $c(v)$ is colouring if $(u, v) \in E$.

$\Rightarrow$ $c(u) \neq c(v)$

Minimize the number of colours.

$c : V \rightarrow \{1, 2, \ldots C\}$

16.1 - 3 (Solution) $\Rightarrow$ Coreman

* **Precedence Constrained Scheduling:-**

Input:- Directed acyclic graph

Vertex :- job (unit Time)

Arc (u, v) : u must execute before v.

No. of processors : p.

Output:- Schedule.

$T(u)$ = Time at which u executes such that

(i) (u, v) is an arc $T(u) < T(v)$

(ii) For all t $|\{u \mid T(u) = t\}| \leq p$

(iii) Max $T(u)$ is minimum

N = Number of nodes

h = length of the longest path.

Sol^n:- **Generic Greedy Algo:-**

(1) Select nodes without predecessors or whose predecessors have been executed.

Date
Page

(2) Select as many ready nodes but at most p and execute them.

(3) Repeat until necessary.

⇒ Select the vertices along the longest path

↳ Selection Strategy:-

(1) Select nodes which reduce length of longest path.

(2) Select set of nodes that results in maximum no. of nodes becoming ready.

Theorem:- Any greedy will finish in time at most $\left\lceil \frac{n}{p} \right\rceil + h$ where n = # of nodes, p = # of processor and h = length of longest path.

⇒ Time for best case. $\geq \max\left(\left\lceil \frac{n}{p} \right\rceil, h\right)$

↓

If all the nodes are at same height and every time we get p nodes for execution than height will be reduced by 1 and thus time will be $\left\lceil \frac{n}{p} \right\rceil$.

⇒ Time for greedy $\leq \left\lceil \frac{n}{p} \right\rceil + h$.

Now, Prove that Time for greedy $\leq 2 *$ time for best case.

<u>Proof</u>:- G is a graph with N-nodes and h length of longest path.

$\Rightarrow$ G' is the graph obtained after one iteration of GGS, N' nodes left, h'- length of longest path.

$G' = \{$ unexecuted nodes $\}$.

$\Rightarrow$ we show that either $h' = h - 1$ <u>or</u>.

$$\left\lceil \frac{N'}{P} \right\rceil = \left\lceil \frac{N}{P} \right\rceil - 1$$

$\Rightarrow$ <u>Two Cases</u>:-

(i) All processors are busy with given iteration:-
$N' = N - P$.

$$\therefore \left\lceil \frac{N'}{P} \right\rceil = \left\lceil \frac{N-P}{P} \right\rceil = \left\lceil \frac{N}{P} - \right\rceil = \left\lceil \frac{N}{P} \right\rceil - 1.$$

(ii) All processors are not busy :-

$\therefore$ There are $< P$ ready nodes.

$\Rightarrow$ Consider any longest path. The first node on it must be ready and will be executed.

$\Rightarrow$ Length of this path reduces by 1.

⇒ True for all the paths ⇒ $h' = h - 1$.

∴ Every iteration must reduce h or $\lceil \frac{N}{P} \rceil$ by 1.

So atmost $\lceil \frac{N}{P} \rceil + h$ iteration are required.

PCS :-

* Finding optimal schedule is considered to be hard in general.

⇒ (Maximum Independent set) (Previous Algo).

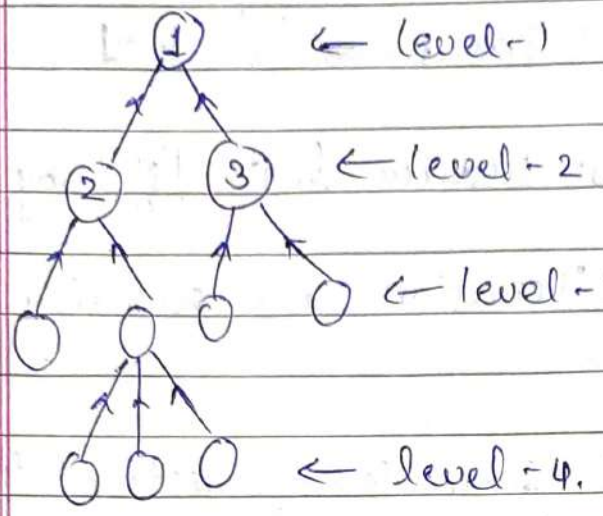
	MIS.	Colouring	PCS	MM. (Matrix Multip)
General Graphs	Hard	Hard	Hard	Polynomial time
Interval Graphs	Polynomial time	Polynomial time	—	—
Scheduling with deadline & profit				Easy.

⇒ Treelike directed graph :- Every node has exactly one successor except root node.

Theorem :- PCS can be solved in polynomial time if input graphs. as treelike.

⇒ Greedy strategy :- Give priority to nodes with highest level.

Eg:-



Theorem:- Every schedule for a tree shaped graph with height h has length atleast:-

$$\text{Max}_{i=1\ldots h} = i-1+\left\lceil \frac{N_i}{p} \right\rceil \text{ where}$$

N_i = # nodes in levels i or larger.

$i=1 = 0+\left\lceil 11/2 \right\rceil + 6$, $i=2 = 1+\left\lceil \frac{10}{2} \right\rceil = 6$.

and so on...

⇒ Here, Max = 6 ∴ length of 6 nodes will be executed first.

Proof:- Let v be the last node from levels i or larger, that is executed by the algo.

Time at which v is executed $\geq \left\lceil \frac{N_i}{p} \right\rceil$

Additional time $\geq i-1$

Date
Page

Theorem:- Greedy algo. finishes in same time as mentioned above.

Proof:- n_i = # of nodes in level i

L_i = # of nodes in level i or larger

$b_i = i - 1 + \lceil \frac{N_i}{p} \rceil$.

$B = \max\{b_1, \ldots b_n\}$.

→ Proof by induction on n:-

→ Base case:- Theorem is true time for all trees with number of nodes $< n$.

T = given tree, T' = Tree that remains after executing 1 iteration on T.

→ Induction hypothesis:-

→ B', N_1', n_i', b_i', L_i' corresponding to T'

→ Greedy algorithm finishes T' in time

$B' = \max\{b_1', b_2' \ldots b_n'\}$.

↳ suffices to prove that $B' + 1 \leq B$.

→ We show that $b_i' < \max(b_i, b_{i+1})$.

$b_i' = i - 1 + \lceil \frac{N_i}{p} \rceil$.

$$\therefore b_i' = i-1 + \left\lceil \frac{N_i - \min(L_i, p)}{p} \right\rceil$$

# of nodes executed in step 1 from level i or above $= \min(L_i, p)$

Now, case-1:- $\min(L_i, p) = p$.

$$\therefore b_i' = i-1 + \left\lceil \frac{N_i - p}{p} \right\rceil = i-1 + \left\lceil \frac{N_i}{p} \right\rceil - 1$$

$$= b_i - 1 < b_i$$

Case-2:- $\min(L_i, p) = L_i$ $[L_i \gg n_i]$

$$b_i' = i-1 + \left\lceil \frac{N_i - L_i}{p} \right\rceil$$

(Number of nodes at level and greater levels are much greater than current level nodes)

~~$\therefore$ minimum will be n_i $\therefore L_i = n_i$.~~

$$\therefore b_i' \leq i-1 + \left\lceil \frac{N_i - n_i}{p} \right\rceil = i-1 + \left\lceil \frac{N_{i+1}}{p} \right\rceil$$

$$\therefore b_{i+1}' - 1 < b_{i+1}.$$

* Other Greedy problems:-

⇒ Coin change problem, Knapsack problem, fractional knapsack, Human Resource scheduling

Date
Page

* Dynamic Programing :-

⇒ Recursion + Remembering whether you have solved a given problem earlier and if so, remembering the solution.

(1) Eg :- Counting Combinations.

⇒ To choose r things out of n, either
(i) choose the first item. Then we must choose the remaining $r-1$ items from the other $n-1$ items.

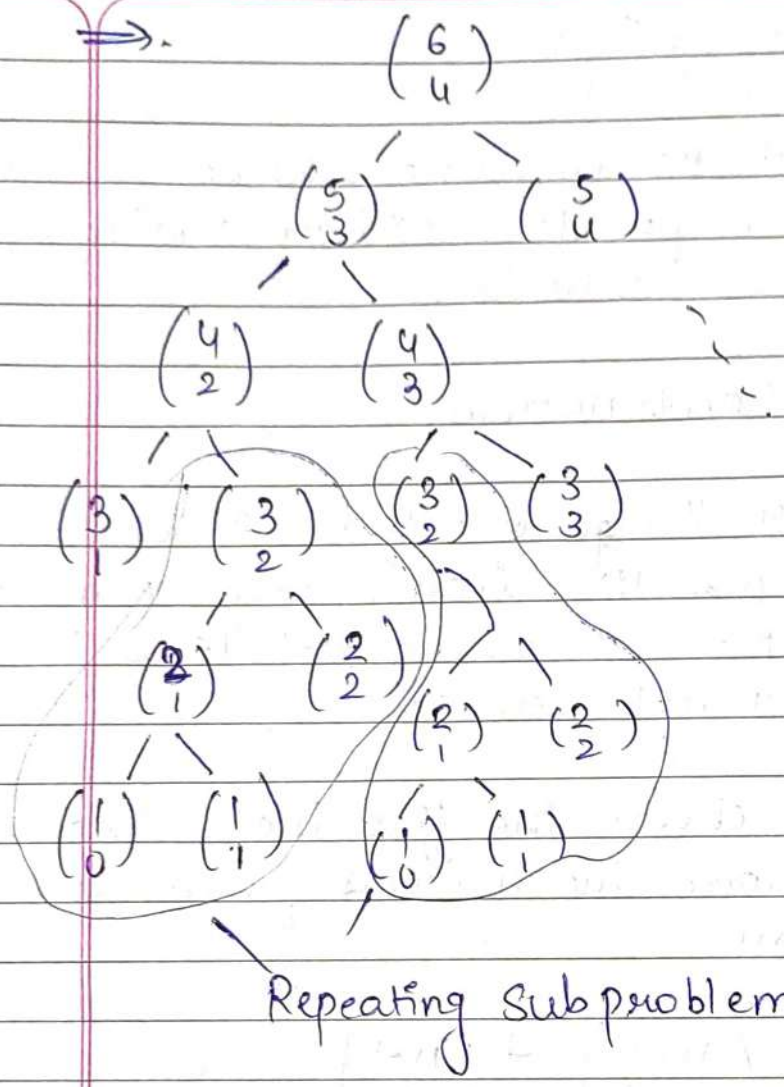
OR (ii) Don't choose the first item. Then we must choose the r items from the other $n-1$ items.

$$\therefore \binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}.$$

While, In Divide and Conquer :- ~~also~~

⇒ Choose(n, r)
If $r = 0$ or $n = r$ then return(1) else
return (Choose $(n-1, r-1)$ + Choose$(n-1, r)$)

↳ $T(n) = 2T(n-1) + d = \underline{\underline{O(2^n)}}$
↳ Too much time



Repeating Subproblems.

⇒ If we Store the value of $\binom{3}{2}$ in any table than we can reduce the operations when it Occurs next time.

⇒ Use a Table $[0 \ldots n, 0 \ldots r]$
$T[i, j]$ holds $\binom{i}{j}$

Choose (n, r)

$(r=0)$ for $i=0$ to $n-r$ do $T[i,0]=1$

$(n=r)$ for $i=0$ to r do $T[i,j]=1$

(else) ⇒ for $j=1$ to r do
  for $i=j+1$ to $n-r+j$ do
    $T[i,j] = T[i-1, j-1] + T[i-1, j]$

return $(T[n, r])$.

⇒ Entries in the table :-

$$(n-r+1)(r+1) = nr + n - r^2 + 1$$
$$\leq n(r+1) + 1$$

↳ Each entry takes $O(1)$ time and thus total time complexity $= O(n^2)$

↳ Space Complexity :- $O(nr)$ [Matrix of $n \times r$]

* Optimized Solution :- [Dynamic Programming]

⇒ Choose (n, r).

$(r=0)$ for $i = 0$ to $n-r$ do $T[i, 0] = 1$

$(r=n)$ for $i = 0$ to r do $T(i, i) = 1$

(else) for $j = 1$ to r do

for $i = j+1$ to $n-r+j$ do $\Rightarrow O(n^2)$.

[Generation of Table].

$T[i, j] = T[i-1, j-1] + T[i-1, j]$.

return $(T[n, r])$.

→ Then use the conquer part of divide and conquer to calculate the combinations.

② Knapsack problem :-

⇒ Given n items of length $s_1, s_2 \ldots s_n$. Is there a subset of these items with total length exactly S?

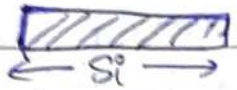
s_1 s_2 s_n

$\square \; \square \; \ldots \; \square$ ⇒ If subset of i items returns sum $= S$ then return true else return false.

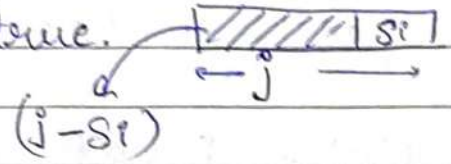
Date
Page

Que ⇒ When knapsack (i,j) is true?

Ans ⇒ (i) If the ith item is not used and knapsack (i,j) returns true



(ii) If the ith item is used & knapsack $(i-1, j-S_i)$ returns true.



Approach-1 Divide & Conquer.

⇒ knapsack (i, j)
return true if $S_1 \ldots S_i$ can fill j.
If $i = 0$ then return $(j = 0)$
else if knapsack $(i-1, j)$ then return (true)
else if $S_i \le j$ then
return (knapsack $(i-1, j-S_i)$)

$T(n) = 2T(n-1) + d \Rightarrow \Theta(2^n)$
↳ Too much time.

Approach-2 Dynamic programming

⇒ Store knapsack(i,j) in table $t[i,j]$
$t[i,j]$ = true iff either
$t[i-1, j]$ is true OR
$t[i-1, j-S_i]$ makes sense and is true.

⇒ <u>Algorithm</u>:- knapsack ($S_1, S_2 \ldots S_n, S$)

```
t[0,0] = true
for j=1 to S do t[0,j] = false
for i=1 to n do
    for j = 0 to S do
        t[i,j] = t[i-1, j].
        if j-si ≥ 0 then
        t[i,j] = t[i,j] OR t[i-1, j-si]
return(t[n,s]).
```

Eg:- $S_1 = 1$, $S_2 = 2$, $S_3 = 2$, $S_4 = 4$, $S_5 = 5$, $S_6 = 2$, $S_7 = 4$ and $S = 15$

⇒ $t[i,j] = t[i-1,j]$ or $t[i-1, j-s_i]$

$t[3,3] = t[2,3]$ or $t[2, 3-S_3]$.

$= t[2,3]$ or $t[2,1]$.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	t	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f
1	t	t	f	f												
2	t	t														
3	t															
4	t															
5	t															
6	t															
7	t															

⇒ $t[1,0] = t[1-1,0] = t[0,0] = t$

$t[2,0] = t[2-1,0] = t[1,0] = t$

⋮

$t[7,0] = t$.

$t[1,1] = t[1-1, 1]$ OR $t[1-1, 1-s_1]$

$= t[0,1]$ $\quad = t[0, 1-1]$

$= f$ $\quad = t[0,0] = t$

$\therefore t[1,1] = f$ OR $t = t$.

Similarly for others.

$\Rightarrow$ $t[7,15] = t$ then it is possible than using those 7 items we can get $S = 15$.

(3) Matrix products problem:-

$\Rightarrow M_1 = 10 \times 20$, $M_2 = 20 \times 50$, $M_3 = 50 \times 1$, $M_4 = 1 \times 100$

$\Rightarrow M = M_1 * M_2 * M_3 * M_4$

Que:- Which way we should multiply the matrices so that we can get minimum # of multiplication

Eg:- $M_1(M_2(M_3 \cdot M_4))$.

(1) $M_3 \cdot M_4 = (50 \times 1)(1 \times 100)$

$\therefore$ # of multi. $= 50 \times 100 = 5000$

(50×100)

(2) $M_2(M_{34}) = (20 \times 50)(50 \times 100)$

$\therefore$ # of multi. $= 20 \times 50 \times 100$

$= 20 \times 5000$

(20×100) $\quad = 100000$

③ $M_1(M_{234}) = (10\times20)(20\times100)$

\# of multi. $= 10\times20\times100 = 20000$

$\therefore$ Total $= 5000 + 100000 + 20000$

$= \underline{125000}$.

$\Rightarrow (M_1(M_2 * M_3))M_4$.

① $M_{23} = (20\times50)(50\times1)$

$= 20\times50$

$= 1000$.

20×1.

② $M_{123} = (10\times20)(20\times1)$

$= 10\times20$

10×1 $= 200$.

③ $M_{1234} = (10\times1)(1\times100)$

$= 10\times100$

$= 1000$

$\Rightarrow$ Total $= 1000 + 200 + 1000 = \underline{2200}$.

$\Rightarrow$ Let $X(n)$ be the number of ways parenthesizing the n matrices.

$\Rightarrow$ It is easy to say that $X(1) = X(2) = 1$ & for $n>2$:-

$$X(n) = \sum_{k=1}^{n-1} X(k)\, X(n-k)$$

Eg:- $X(3) = \sum_{k=1}^{2} X(k) \cdot X(n-k)$

$= X(1) \cdot X(2) + X(2) \cdot X(1)$

$= 1 + 1 = 2.$

$M_1 (M_2 M_3)$ & $(M_1 M_2) M_3$.

$X(4) = \sum_{k=1}^{3} X(k) \, X(n-k).$

$= X(1) X(3) + X(2) X(2) + X(3) X(1)$

$= 2 + 1 + 2$

$= 5.$

$\therefore \underline{\underline{X(n) \geq 2^{n-2}}}.$

<u>Approach - 1</u> Divide & Conquer:-

⇒ Let $Cost(i,j)$ be the minimum cost of computing $M_i \cdot M_{i+1} \cdot \ldots \cdot M_j$.

→ What is the cost of breaking the product at M_k?

⇒ $(M_i \cdot M_{i+1} \ldots M_k) \cdot (M_{k+1} \ldots M_j)$.

$$\therefore \boxed{Cost(i,j) = Cost(i,k) + Cost(k+1,j) + r_{i-1} * r_k * r_j}$$

~~Now, cost(i~~.

Date
Page

Algo:- Cost(i, j)
{
if $i = j$ then return(0)
else
return $\min_{i \le k < j}$ (cost(i, k) + cost(k+1, j) + $r_{i-1} * r_k * r_j$).

$T(n) = 3T(n-1) + d \Rightarrow O(3^n)$.

$\Rightarrow$ we can do better with dynamic programming:-

Algo: Store cost(i, j) in m[i, j].
m[i, j] = 0 if $i = j$ and

$$\min_{i \le k < j} (m[i,k] + m[i+1, j] + r_{i-1} r_k r_j)$$

Eg:- $r_0 = 10$, $r_1 = 20$, $r_2 = 50$, $r_3 = 1$, $r_4 = 100$

$\rightarrow$ (m[1,1]) (m[2,2])

$\therefore m[1,2] = \min_{1 \le k < 2} (m[1,k] + m[k+1, 2] + r_0 r_k r_2)$
$= r_0 r_k \cdot r_2 = 10 \times 20 \times 50$
$= 10000$.

$m[2,3] = r_1 \cdot r_2 \cdot r_3 = 1000$
$m[3,4] = r_2 \cdot r_3 \cdot r_4 = 5000$.

$m[1,3] = \min_{1 \le k < 3} (m[1,k] + m[k+1, 3] + r_0 r_k r_3$
$= \min(m[1,1] + m[2,3] + r_0 r_1 r_3,\ m[1,2] + m[3,3] + r_0 r_2 r_3)$

$= \min(1000 + 200,\ 10000 + 500).$

$= \min(1200,\ 10500) \quad = \underline{1200}$

$m[i,j] \Rightarrow$

	1	2	3	4
1	0	10k	1.2k	2.2k
2		0	1k	3k
3			0	5k
4				0

Minimum = 1000
(2, 3)

for $i = 1$ to n do $m[i,j] = 0 \rightarrow O(n)$

for $d = 1$ to $n-1$ do
  for $i = 1$ to $n-d$ do
    $j = i + d$
    $m[i,j] = \min_{i \le j < n} (m[i,k] + m[k+1,j] + r_{i-1} r_k r_j)$

$O(n^2)$

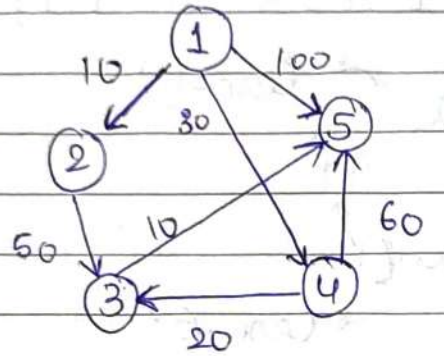
$O(n) \leftarrow$

return $(m[1,n])$

Combined $\Rightarrow \underline{O(n^3)} \rightarrow$ Just better than divide and conquer.

(4) All pair shortest path problem:-

⇒ Graph $G = (V, E)$



⇒

A_0 :-

	1	2	3	4	5
1	0	10	∞	∞	100
2	∞	0	50	∞	∞
3	∞	∞	0	∞	10
4	∞	∞	20	0	60
5	∞	∞	∞	∞	0

⇒ Given a labelled directed graph $G = (V, E)$, find for each pair of vertices $v, w \in V$, the cost of the shortest path from v to w.

⇒ Define: A_K to be an ~~A×A~~ $n \times n$ matrix with $A_K[i, j]$ the cost of the shortest path from i to j with internal vertices numbered $\leq K$

⇒ $A_0[i, j]$ equals

- if $i \neq j$ and $(i, j) \in E$, then the cost of the edge from i to j
- if $i \neq j$ and $(i, j) \notin E$ then cost $= \infty$
- if $i = j$ then cost $= 0$.

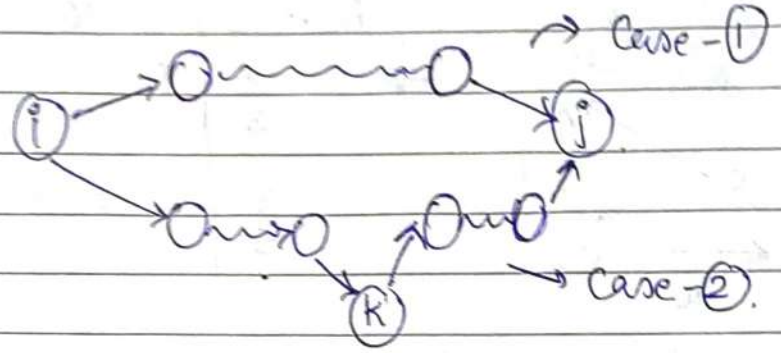
⇒ Computing A_K:-

- Consider the shortest path from i to j with internal vertices $1 \ldots k$

- Either (1) it does not go through k, in which place it costs $A_{K-1}[i, j]$

② It goes through k, in which case it goes through k only once.

So it costs $A_{k-1}[i,k] + A_{k-1}[k,j]$.



$\therefore A_k[i,j] = \min\{A_{k-1}[i,j],\ A_{k-1}[i,k] + A_{k-1}[k,j]\}$

$\Rightarrow$ Now, $A_k[k,j] = \min\{A_{k-1}[k,j],\ \underbrace{A_{k-1}[k,k]}_{\text{"0"}} + A_{k-1}[k,j]\}$

(k-Row)

$\therefore A_k[k,j] = \min\{A_{k-1}[k,j];\ A_{k-1}[k,j]\}$

$\therefore \boxed{A_k[k,j] = A_{k-1}[k,j]}$

$\Rightarrow$ Now, $A_k[i,k] = \min\{A_{k-1}[i,k],\ A_{k-1}[i,k] + \underbrace{A_{k-1}[k,k]}_{\text{"0"}}\}$

(k-Column)

$= \min\{A_{k-1}[i,k],\ A_{k-1}[i,k]\}$

$\therefore \boxed{A_k[i,k] = A_{k-1}[i,k]}$

⇒ So, based on this :- <u>Flyod's Algorithm</u> :-

```
for i = 1 to n do
    for j = 1 to n do
        if (i,j) ∈ E
            then A[i,j] = cost of (i,j)
        else
            A[i,j] = ∞.
        ~~A[i,j] = 0~~
A[i,j] = 0.
for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
            if A[i,k] + A[k,j] < A[i,j] then
                A[i,j] = A[i,k] + A[k,j].
```

Time Complexity = $O(n^3)$.

Eg:- Previous graph

A_0 :-

	1	2	3	4	5
1	0	10			
2		0			
3			0		
4				0	
5					0

$A_1[1,2] = \min\{A_0[1,2],\ A_0[1,1] + A_0[1,2]\}$

$= \min\{10,\ 0+10\} = \min\{10, 10\} = 10$

~~A0[1,2]~~ Similarly for others and then find A_2, A_3, A_4, A_5. A_5 will show the shortest path from each vertex v to another vertex w.

Date
Page

→ Warshall's Algorithm:-

→ Given a directed graph $G(V, E)$ find for each pair of vertices $v, w \in V$ whether there is a path from v to w.

→ $A[i,j] \neq \infty$

Algo:-

```
for i = 1 to n do
    for j = 1 to n do
        A[i,j] = (i,j) ∈ E
        A[i,j] = true.

for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
            A[i,j] = A[i,j] or (A[i,k] & A[k,j])
```

* Complexity Theory:-

① Reduction:-

⇒ Problem P belongs to complexity class $f(n)$ iff best algorithm solves P in time $f(n)$.

↳ Means, no algorithm with time $< f(n)$ is possible.

Problems

⇒ P: Class of problem which have an algorithm taking time $O(n^k)$, where k is any fixed integer and n is size of input.

Polynomial time.

⇒ Exponential time :- Problems for which algorithms take time $O(k^n)$.

P: Class of problems for which "fast" algorithm exist are "easy" problems.

⇒ Problems not in P are "hard" ⇒ fast algorithm do not exist.

⇒ Polynomial time Reduction :- Problem P can be polytime reduced to problem Q iff there exists polytime computable functions f, g such that given an instance x of P, $f(x)$ is instance of Q, given a solution z to an instance y of Q, $g(z)$ is the solution to x if $y = f(x)$.

$\Rightarrow$ x [instance of P] $\longrightarrow$ [Polytime k_1 function f] $\xrightarrow{f(x) = y}$ [Solves Q.] $\xrightarrow{z = \text{solution for } y}$ [Polytime k_2 function g] $\longrightarrow$ $g(x)$ [solution for x].

f, g must be polytime function

$\Downarrow$

Mandatory Condition

$\hookrightarrow$ $P \leq_p Q$. [P is polytime reducable to Q]

<u>Theorem</u>:- Suppose P can be polytime reduced to Q, then if there exists a polynomial time algorithm for Q, then there exists a polytime algorithm for P.

<u>Proof</u>:- $|s|$ = length of s in bits.

$\Rightarrow$ Time to compute f : $|x|^{k_1}$

$\therefore |y| \leq |x|^{k_1}$

and given that A solves Q in time n^{k_2}.

Time for solving $|y| = |y|^{k_2} \leq |x|^{k_1 k_2}$.

$(\because |y| \leq |x|^{k_1})$

$\Rightarrow$ Q is no easier than P.

$\Rightarrow$ Suppose we can prove that no polytime algo. exists for P $\Rightarrow$ No polytime algo for Q.

$\Rightarrow$ Suppose no polytime algo. exists for Q

$\not\Rightarrow$ no polynomial algo. exists for P

(does not imply).

* Decision Problems:- [Answer = yes or No]

Eg:- Can this graph be coloured in 3 colours?

⇒ P is a decision problem if every instance has a unique solution which is either 0 or 1.

Eg:- Can G be scheduled on P processors such that execution completes in d steps?

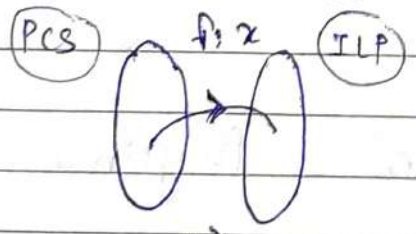
Problem-1

Eg: 0,1 - Integer linear programming:- [ILP].

⇒ Input : Matrix A → $m \times n$
Vector b → $m \times 1$.

⇒ Decision problem:- Does there exists a vector x $(n \times 1)$ such that $A \cdot x \leq b$ and $x \in \{0,1\}$.

(i) PCS reduces to ILP:-



$O(n^k)$.

PCS = Directed acyclic graph
- # of processors
- Deadline d.

⇒

$T(v) < T(u)$

⇒ T_u - time at which vertex u is scheduled.

⇒ x_{ut} - $\begin{cases} 1 & \text{if } u \text{ is schedule at } t \\ 0 & \text{otherwise} \end{cases}$

Date
Page

⇒ <u>Constraint</u>:- for every ~~ste~~ time step t, only P nodes can be scheduled.

$$\sum_{u} x_{ut} \leq P \quad \text{for all } t$$

O(n)

$\therefore x_{11} + x_{21} + x_{31} + \dots + x_{n1} \leq P$

↳ Every node must be scheduled exactly at one instant.

$$\sum_{t} x_{ut} = 1.$$

O(t)

$\therefore \sum_{t} x_{ut} \leq 1 \quad \& \quad -\sum_{t} x_{ut} = -1.$

⇒ Time at which u is scheduled

$= 1 \cdot x_{u1} + 2 \cdot x_{u2} + 3 \cdot x_{u3} + \dots + d \cdot x_{ud}$

[Deadline = d]

$\therefore 1 \cdot x_{u1} + 2 \cdot x_{u2} + 3 \cdot x_{u3} + \dots < 1 \cdot x_{v1} + 2 v_2 + \dots$

$\therefore 1 \cdot x_{u1} + 2 \cdot x_{u2} + 3 \cdot x_{u3} + \dots \leq 1 \cdot x_{v1} + 2 x_{v2} + \dots - 1$

O(d)

∴ Time complexity = O(n) ⇒ Polytime

∴ PCS is polytime reducible to ILP

↳ $\boxed{PCS \leq_p ILP}$

Problem-2 Clique reduces to PCS.

PCS : Task graph T. # processors = p, deadline = d.

Decision:- Can T be scheduled using P processors in time d?

Clique:- Graph G, integer k.

Decision:- Does G contain as a subgraph a clique of size k? clique of size k = Complete graph of k vertices.

Proof:- ⇒ clique instance PCS instance.

$$\begin{pmatrix} G \\ k \end{pmatrix} \xrightarrow[\text{function}]{f \Rightarrow \text{Polytime}} \begin{pmatrix} T \\ P \\ d \end{pmatrix} \Rightarrow \text{Clique} \leq_p \text{PCS}$$

⇒ For every G, k, d = 3.

$$p = 1 + \max\left(k,\ n - k + \frac{k(k-1)}{2},\ m - \frac{k(k-1)}{2}\right)$$

m = # edges in G, n = # vertices in G

⇒ $T = T_1, T_2$.

T_1 = vertex set = { One vertex $f(u)$ for every vertex u of G and.
One vertex $f(u,v)$ for every edge (u,v) of G.

T_2 edge set = { Divided edge from vertices $f(u)$ to $f(u,v)$ }

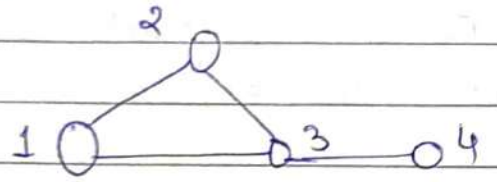
T_2 = 3 level graph

IIIrd level :- $P - \left(m - \frac{k(k-1)}{2}\right)$

IInd level :- $P - \left(n - k + \frac{k(k-1)}{2}\right)$

I^{st} level :- $P - k$ tasks.

Eg:-



$k = 3$ (clique)
$n = 4$
$m = 4$
↳ # of edges.

$\Rightarrow x = G$ (graph).
$\therefore f(x) = (T, p, d)$ Result

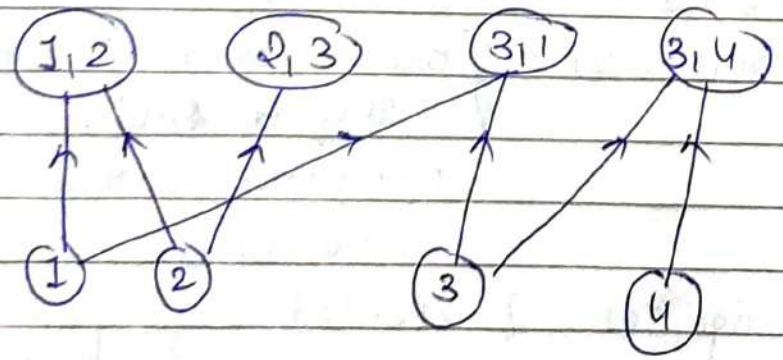
$d = 3 \Rightarrow p = 1 + \max\left(k,\ n - k + \frac{k(k-1)}{2},\ m - \frac{k(k-1)}{2}\right)$

$\therefore P = 1 + \max\left(3,\ 4 - 3 + \frac{3 \cdot 2}{2},\ 4 - \frac{3(2)}{2}\right)$

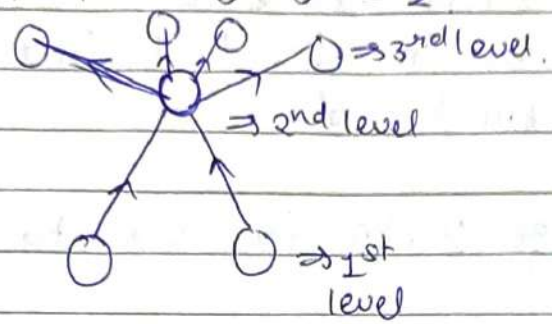
$\therefore P = 1 + \max(3, 4, 1)$

$\therefore P = 1 + 4 = \underline{\underline{5}}$

T_1 :-



T_2 :- $P - K = 5 - 3 = 2 \Rightarrow 1^{st}$ level.



2nd level :-

$$5 - \left(4 - 3 + \frac{3 \cdot 2}{2}\right)$$
$$= 5 - (1 + 3)$$
$$= 1.$$

3rd level :- $P - \left(m - \frac{k(k-1)}{2}\right)$

$$= 5 - (4 - 3) = 5 - 1 = 4$$

∴ Number of tasks in T_1 and T_2 :- $|T_1| + |T_2|$
$$= m + n + 3p - (m+n)$$
$$= 3p.$$

⇒ Schedule exists iff every processor is busy at every step

<u>Claim</u> Suppose a Schedule exists for T, P, d then G must contain a clique of size k.

<u>Proof</u>:- Number of nodes of T_2 scheduled in top level

$$= \underset{\text{Total}}{P} - \Big(P - \underbrace{m + \frac{k(k-1)}{2}}_{\text{nodes of } T_2}\Big).$$

⇒ $m - \frac{k(k-1)}{2}$ must all be edges nodes $\frac{k(k-1)}{2}$ edge nodes in level 2.

Date
Page

⇒ All vertices associated with these edge tasks must be in level 1 but level 1 has space for k tasks.

⇒ $\frac{k(k-1)}{2}$ edge tasks in level two are such that associated edges are incident only on k vertices.

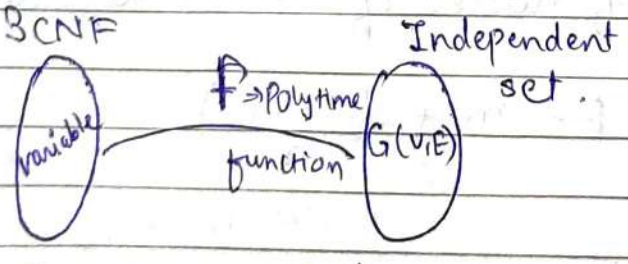
⇒ <u>k clique exists</u>

∴ $\boxed{\text{clique} \leq_p \text{PCS} \leq_p \text{ILP}}$.

<u>Problem-3</u> <u>Formula Satisfiability</u>.

⇒ Input : Formula
⇒ Output :- True if there exist an assignment of values to variable such that the formula evaluates true or false.

<u>Theorem</u>:- Formula Set (3CNF) $\leq_p$ independent set



<u>Proof</u>:- let input formula be $x = (x_{11} + x_{12} + x_{13})$
$(x_{21} + x_{22} + x_{23})$
$- - - -$
$(x_{n1} + x_{n2} + x_{n3})$

where x_{ij} is the literal.

⇒ $f(x)$: G will have 3n vertices.

⇒ Edges :- There are two kind of edges :-

(i) Triangle connecting x_{i1} x_{i2} x_{i3} $\forall i$

(ii) For every x_{ij} & x_{kD} such that x_{ij} is negation of x_{kl}, put an edge.

Eg :- $(A+\bar{B}+C)\ (C+\bar{D}+A)\ (G+\bar{A}+\bar{C})$

⇒ The resultant must be equal to 1, which means that $(A+\bar{B}+C)=1$, $(C+\bar{D}+A)=1$, $(G+\bar{A}+\bar{C})=1$.

↳ In each triangle graph, you have one contribution to independent set.

⇒ If $P(x)$ is true then $Q(f(x))$ is true.

Proof :- $P(x)$ = true implies that in each sum there exists a literal x_{ij_i} that is true.

Claim :- The set of vertices corresponding to x_{ij_i} is independent.

Proof :- Suppose there exists an edge (x_{ij_i}, x_{kj_k})

$\Rightarrow x_{ij_i} = \bar{x}_{kj_k}$

⇒ This is not possible. Hence the proof.

⇒ If $Q(f(x)) = \text{true}$ then $P(x)$ is true

Proof:- $Q(f(x)) = \text{true} \Rightarrow \exists$ independent set of size n.

⇒ Due to type I edges exactly one vertex x_{ij_i} from each triple must be in the independent set.

∴ Consider the assignment $x_{ij_i} = 1 \ \forall i$. Set other variables arbitarily.

⇒ Since x_{ij_i} & x_{kj_k} are never compliment, the 3CNF is true.

Problem-4:- Circuit Satisfiability:-

⇒ Given:- Combinational circuit made up of 2 input NAND gates with a single output.

⇒ Question:- Does there exist an input assignment such that output = True.

Theorem:- Circuit Sat. $\leq_p$ CNF. satisfiability

Proof:- x: Circuit → $f(x)$ = 3CNF formula.

⇒ $f(x)$ will assert that the input values and the output value for each gates are consistent with its type & output is a 1.

⇒ Express using 3CNF : x is NAND(C, D)

$$CD \rightarrow \bar{x}$$
$$\equiv \bar{C}\bar{D} + \bar{x}$$
$$\equiv \bar{C} + \bar{D} + \bar{x}$$

⇒ $f(x) = \underset{\text{output}}{\omega} \underset{\text{multiplication}}{\prod} \cdot \underbrace{\text{CNF expressions asserting that I/O to the } i^{th} \text{ gate are consistent with its type}}_{G_i}$

$\therefore f(x) = 1 \Rightarrow \omega = 1,\ G_i = \text{True}$

* <u>Verifier</u> :-

⇒ Let P be a decision problem. We will say that V is a verifier for (true instances of) P iff

① V. takes two assignments x and y and returns true or false.

② $\forall$ instances of x :- $P(x) = \text{true}$ implies that $\exists y$ such that $V(x, y) = \text{true}$.

$P(x) = \text{false} \Rightarrow \forall y \ V(x, y) = \text{false}$.

⇒ Every true statement must have some proof.

⇒ If a statement is false, it must not have a proof.

⇒ Proof checker says whether or not y is a proof for x.

Date
Page

⇒ Now, V is a polytime verifier if time taken by V is polynomial in length of the first input.

Define:- We will say that a decision problem belongs to the class NP if P has a polynomial time verifier.

Cook's theorem:- All problems in class NP can be reduced to circuit satisfiability. ⇒ [CLRT.B.]

Theorem:- Graph coloring decision problem is in NP.

Proof:- Graph = G and Colors = K.

⇒ By definition we need two variables x and y.

i.e. x: instance.
y: Colouring. [sequence of n numbers $c_1, c_2 \ldots c_n$ where c_i is colour of node i]

⇒ V(x,y): For every edge (i,j) in G do
If adjacent edges have same colour. — if $c_i = c_j$
return false. ⇒ O(E)
end.
⇒ For every i do.
Check if G can be K-colorable. — if $c_i > K$ or $c_i < 1$
return false. ⇒ O(E + i) — very small
end.
return true

O(E)

∴ Graph Colouring problem is in NP.

Date
Page

Theorem:- PCS is in N.P.

Proof:- $x : (T, P, d)$

y : time instances $t_1, t_2 \ldots t_n$, $n = \#$ of nodes

t_i = time at which node i is ~~calculated~~ executed.

$V(x, y)$: (i) For every edge (i, j) if $t_i \geq t_j$ return false. $\Rightarrow O(E)$

(ii) $P_T = \{ i \mid t_i = T \}$ $\Rightarrow O(V)$

Set of nodes executed at step T

if $P_T > P$ return false.

(iii) Check that $1 \leq t_i \leq d$ for all i.

$O(E+V) \Rightarrow$ polytime.

$\Rightarrow$ Hence, PCS is in NP.

Theorem:- Sorting decision problem is in NP

Proof: Input :- $(k_1\ k_2 \ldots k_n, s_1, s_2 \ldots s_n)$.

[i/p keys] [Sorted order]

$\Rightarrow$ Verifier :- (i) To check if s_i are in non-decreasing order.

(ii) Every number in k_i is present as many times in s_i

y = null.

$\Rightarrow V$: happens to generate the proof and check it.

Hence Sorting decision Problem is in NP

Date
Page

Theorem:- Shortest path decision problem is in NP.

Proof:- ⇒ (G, u, v, k) where G = Graph, u = source, v = dest., k = length.

instance

⇒ Does there exists a path of length k from u to v in graph G? ⇒ Decision problem.

⇒ Verifier:- x: (G, u, v, k)

y: path from u to v with weights $w_1 \ldots w_l$.

∴ $V(x, y)$: Check whether there is the path.

① ⇒ check (w_l, w_{l+1}) is an edge.

⇒ check $l \leq k$, $w_1 = u$ and $w_l = v$

OR

② ⇒ $V(x, y)$: Compute the shortest path from u to v:-

if length of shortest path $\leq k$

return true

otherwise

return false.

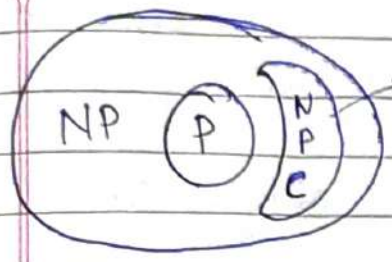
⇒ This takes polytime as we can apply any algo to find shortest path which are polytime algos.

⇒ Thus, $V(x, y)$ = polytime and hence the shortest path decision problem is in NP.

☆ Any decision problem solvable in polynomial time $\in$ NP.

⇒ Polytime Algo = Good
Exponential Algo = Bad.

Date
Page

⇒ 

Problems not in NP.

① ↳ ~~Not~~ Does the given problem P halt on input I?

② Is K the min. # of colors required to color G?

<u>Theorem</u>:- Independent set is in NP

<u>Proof</u>:- Does there exist an independent set of size K?

⇒ Instance :- (G, K)

⇒ x : (G, K)

y : independent set. $\{v_1, v_2 \ldots v_l\}$

$V(x, y)$: Check whether there is an edge between any vertices.

$\forall i, j$ $(v_i, v_j) \in E$ ⇒ return false.
$(v_i, v_j) \notin E$ ⇒ return true.
↳ $l = K$

⇒ Then the problem belongs to class NP.

⇒ <u>Other Theorems</u>:- Circuit SAT, 3-SAT, clique are in NP. [Prove this]., ILP.

Date
Page

Cooks Theorem :- [Helps to prove a Prblm belongs to class NPC]

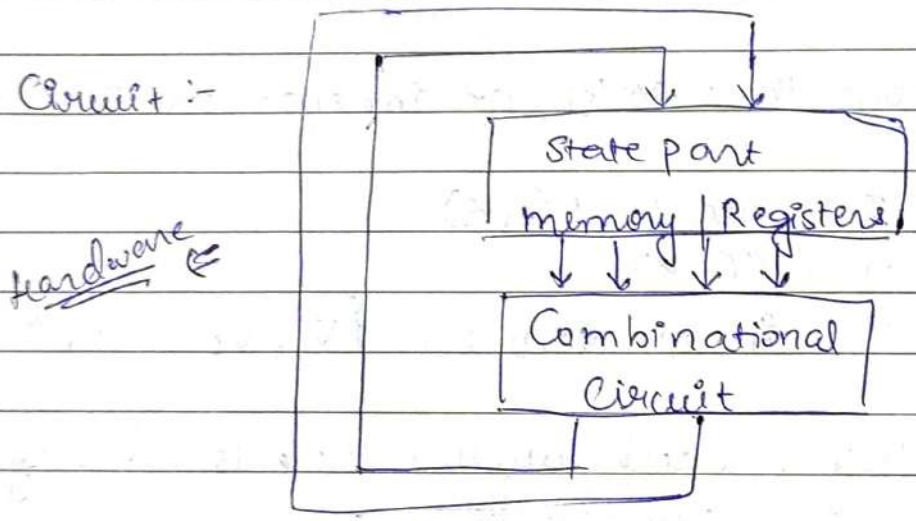
* All problems in NP can be reduced to circuit sat

⇒ F simulates the action of a polynomial time algorithm for A.

⇒ y is encoded in assignment of F

⇒ The formula is satisfiable if any ∈ A.

⇒ Circuit :-



⇒ Suppose we have a program A for solving a certain problem P.

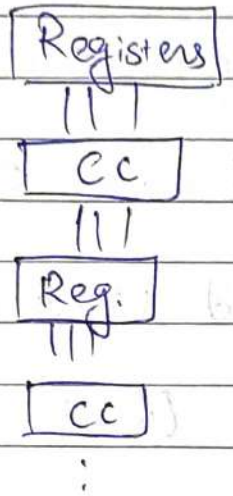
⇒ we can construct a Combinational circuit family C where C(n) solves the problem for inputs of length n.

⇒ With the size of the circuit being polynomial in time taken by the program

⇒ In hardware :-

Begining : Entire State fed to Combinational circuit. [CC]

End : Entire State latched into registers.



⇒ ∀ Instances x of P $f(x)$ is a circuit $C(x)$ Such that if $P(x) = \text{true}$ ⇒ $C(x)$ is satisfiable.
↳ (implies).

⇒ $f(x) = C_{V_x}$ is circuit for instance x of P where $n = |x|$.

$V_x(y) = V(x, y)$ [verifier of x, y].

⇒ Input to $C_{V_x}(n) = y$. If $P(x) = \text{true}$ ⇒ there exist y such that $V(x, y) = \text{true}$ ⇒ $C_{V_x}(n)$ is satisfied with inputs set to the given y.

If $P(x) = \text{false}$ ⇒ $\forall y$ $V(x, y) = \text{false}$ ⇒ $C_{V_x}(n)$ is ~~not satis~~ always false

⇒ Size = # CC · size + Reg. T ⇒ $O(T^2)$

If $T(n) = \text{poly}(n)$, Size = poly(n).

↳ $P \leq_p$ circuit

⇒ NP :- Circuit Sat = Hardest problem.
⇒ NP - Hard.

* Co-NP [Complement - NP] :

⇒ Class of decision problems whose complement problems is in NP. ⇒ Co-NP problems.

Eg:- Complement of colouring decision problem.

⇒ Input : Graph, k = Number of colours.

Problem :- Can G not be coloured using atmost K colour?

* NP Complete problems :- [NPC]

⇒ Problem Q is NP Complete if :

(i) $Q \in NP$. (ii) $Q' \leq_p Q$, $\forall Q' \in NPC$

Why this logic is sufficient?

⇒ Since Q' is NPC, all problems in NP is reducible to Q.

⇒ Show Q' is reducible to Q, then all problems in NP is reducible to Q.

∴ Q is NP Complete.

Eg:- PCS is NPC :-

Proof :- PCS ∈ NP

⇒ $x = (T, P, d)$ ⇒ $V(x, y)$: $t_i \geq t_j$ - false

⇒ $y = t_1 \ldots t_n$.

$P_T = \{ i \mid t_i = T \}$

if $P_T > P$ → false

$1 \leq t_i \leq d$ - true

Poly

$\therefore$ (i) PCS $\in$ NP.

$\rightarrow$ Now, (ii) Circuit-Sat $\leq_p$ PCS — ①.

- Circuit-Sat $\in$ NPC.

$\forall Q' \in NP$, $Q' \leq_p$ Circuit-Sat. — ②.

From ① & ②, $\forall Q' \in NP$, $Q' \leq_p$ PCS

Hence PCS is NPC.

* General Strategy for proving Q is NPC:-

$\Rightarrow$ (i) Find a polytime verifier for Q

(ii) Find some problem R such that R is NPC & $R \leq_p Q$.

Eg:- The directed Hamiltonian Cycle is known to be NPC. Use this fact to prove that undirected hamiltonian Cycle is NPC.

$\Rightarrow$ Proof:- (i) Prove undirected hamiltonian cycle $\in$ NP.

$\Rightarrow$ We have a polytime verifier $V(x,y)$ for HC and thus HC $\in$ NP. — Ⓐ

(ii) Directed HC $\leq_p$ Undirected HC — Ⓑ

Define the transformation T.

$\Rightarrow$ let $G = (V, E)$ be a directed graph. Define G to the undirected graph $G' = (V', E')$ by the following transformation:-

$u \in V \rightarrow v^1, v^2, v^3 \in V'$ and $(v^1, v^2), (v^2, v^3) \in E'$

$(u, v) \in E \rightarrow (u^3, v^1) \in E'$

$\Rightarrow$ If $v_1, v_2, v_3 \ldots v_n, v_1$ has a HC then $v_1^1, v_2^1, v_3^1 \ldots v_n^1, v_n^2, v_n^3, v_1^1$ is an undirected HC for G'.

$\therefore$ we can say that, $v_{i_1}^1, v_{i_1}^2, v_{i_1}^3, v_{i_2}^1, v_{i_2}^2, v_{i_2}^3, \ldots$
$\ldots v_{i_n}^1, v_{i_n}^2, v_{i_n}^3, v_{i_1}^1$

Complexity $= 3(V + E)$

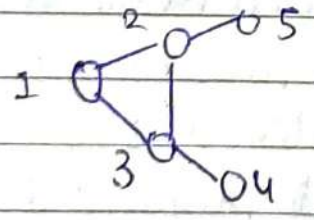
From (A) & (B) undirected HC is NPC.

[H.W] Eg:- Show that Subset sum is NPC.

Eg:- <u>Vertex Cover</u>:-

$\Rightarrow$ I/P :- Undirected graph G, integer k.
O/P :- Yes iff G has a vertex cover of size k.

$\Rightarrow$ Vertex Cover : $G = (V, E)$, $V' \subseteq V$ a vertex cover if every edge in E has atleast one endpoint in V'



$\Rightarrow$ $\{2, 3\}$, $\{1, 3, 5\}$ $\rightarrow$ vertex cover

$\Rightarrow$ ~~[illegible]~~

Eg:- **Subset Sum:-**

⇒ I/P :- $S = \{s_1, s_2 \dots s_n\}$, t : integer target.
O/P :- True if $\exists s' \subseteq s$ such that sum of number in s' is exactly t.

Eg:- $S = \{5, 17, 3, 2\}$, $t = 11$ ⇒ No
$S = \{5, 17, 3, 2\}$, $t = 10$ ⇒ Yes.

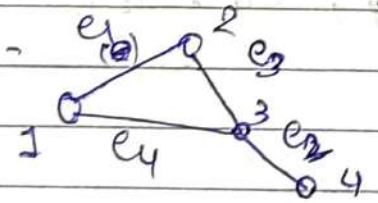
Theorem:- (Vertex Cover) $\leq_p$ (Subset sum).

⇒ $G = (V, E)$, K. $\xrightarrow{f}$ $S = \{x_1, x_2 \dots x_n, y_1, y_2 \dots y_m\}$.

m = # edges
n = # vertices.

⇒ **Edge incidence Matrix:-**

Eg:-



⇒

	e_1	e_2	e_3	e_4
v_1	1	0	0	1
v_2	1	0	1	0
v_3	0	1	1	1
v_4	0	1	0	0

⇒ Interpret each vertex as a radix 4 number N:
$x_i = N_i + 4^m$.
$\therefore x_1 = 4^0 \cdot 1 + 0 * 4 + 0 \cdot 4^2 + 1 * 4^3 + 256$
$x_1 = 321$

y_i = plane "1" is column corresponding edge e_i & "0" elsewhere

SS = subset sum.
VC = vertex cover.

$y_4 = 64,\ y_3 = 16,\ y_2 = 4,\ y_1 = 1.$

$t = k4^m + 2 \cdot 4^{m-1} + 2 \cdot 4^{m-2} + \dots + 2 \cdot 4^0$

Claim:- If vertex-cover(x) = true then SS(x) = true

Proof:- Suppose VC is vertices $v_1, v_2 \dots v_k$.

Subset: $S' = \{x_{v_1}, x_{v_2} \dots x_{v_k} \dots\}$
$k \cdot 4^m$ + For every edge position we have either 1 or 2.

Include y_i in S' if VC covers edge i exactly once.

Claim:- If SS(x) = true then VC(x) = true

Proof:- Suppose we know S', can we know suitable V'?

$S' = \{\underbrace{x_{i_1}, x_{i_2}, \dots x_{i_k}}_{\text{atleast } k\ x_i\text{'s}}, y_{i_1}, y_{i_2} \dots\}$

$V' = \{i_1, i_2, i_3\}$.

$\Rightarrow$ Suppose V' does not cover a certain edge $e = (u, v) \Rightarrow x_u$ & x_v are not in S'.

$\Rightarrow$ Consider column corresponding to e. Sum of any subset $S = \{x_u, x_v\}$ will not produce 2 in this position.

Eg:- Set Cover:-

⇒ Input: Sequence of sets $S_1, S_2 \ldots S_n$
$S_i \subseteq X$ ⇒ Universal set.
(ii) Integer k.

⇒ Output: yes iff $\exists k$ sets $S_{i_1}, S_{i_2} \ldots S_{i_k}$ such that
$S_{i_1} \cup S_{i_2} \cup \ldots \cup S_{i_k} = X$.

Eg:- $X = \{1, 2, 3, 4, 5, 6\}$
$S_1 = \{1, 2, 3\}$, $S_2 = \{2, 3, 4\}$, $S_3 = \{4, 5\}$,
$S_4 = \{4, 5, 6\}$, $S_5 = \{6\}$.

$\therefore \{S_1 \cup S_3 \cup S_5\} = \{1, 2, 3, 4, 5, 6\} = X$
$\{S_1 \cup S_4\}$, $\{S_1 \cup S_2 \cup S_3 \cup S_4\}$.
↳ Minimize the size of set cover.

⇒ Weighted set cover - Minimize total weight of the set cover.

Theorem:- Set cover is NPC [H.W].

Proof:- Find a verifier.
Check (i) Does there exists a set cover of size k?
$V(x, y) \Rightarrow$?
(i) Does C cover all elements? } $O(V+E)$
(ii) $|C| = k$? ⇓ Polytime

$\therefore$ Set cover is NPC

Date
Page

Eg:- Set Partition is NPC

Proof:- Subset Sum decision problem:-

Given a positive integer c, and set $S = \{s_1, s_2 \ldots s_n\}$ of positive integers s_i for $i = 1, 2 \ldots n$. Is there a $J \subseteq \{1, 2 \ldots n\}$ such that

$$\sum_{i \in J} s_i = c \ ?, \quad w = \sum_{i=1}^{n} s_i \geq c.$$

Set partition decision problem:-

Given a set S of numbers. Can S be partitioned into two sets A and $\bar{A} = S - A$ such that

$$\sum_{x \in A} x = \sum_{x \in \bar{A}} x \ ?$$

Proof:- Let S be an instance of subset sum with $w = \sum_{s_i \in S} s_i$ and the target c

$\Rightarrow$ Define the set S' as follows:- $S' = S \cup \{u, v\}$ ↳ edge

where $u = 2w - c$, $v = w + c$

$\Rightarrow$ Let $J \subseteq S$ and the elements in J sum to c. Then $J \cup \{u\}$ sum to $2w$.

$\Rightarrow$ Note that the elements in $\bar{J} = S - J$ sum to $w - c$. Hence, $\bar{J} \cup \{v\}$ also sums to $2w$.

$\therefore$ S' can be partitioned into $J \cup \{u\}$ and $\bar{J} \cup \{v\}$

where both partitions sums to $2w$. Hence the Subset sum is polytime reducible to set partition decision problem. [Subset sum $\leq_p$ set partition decision problem].

$\therefore$ Subset sum is in NPC

* **NP-Hard**:- Problem Q is NP-H if $Q' \leq_p Q$ for every $Q' \in NP$.

$\Rightarrow$ [NPC = NP $\cap$ NP-Hard]

* **Approximation algorithms for NPC problems**:-

Problem:- Minimize the size of the set cover.

$\Rightarrow$ Find a cover with size nearly the best possible (Goal) [Apply greedy algorithm]

Theorem:- There exists a polytime algorithm that gives a set cover of size atmost $\log|X|$ * size of optimal cover.

$\Rightarrow$ **Greedy algorithm**:- Pick sets that cover maximum number of currently uncovered elements.

Proof:- Let C_{opt} denote the size of the optimal cover
# of elements covered by greedy in first iteration is atleast $\geq \frac{|X|}{C_{opt}}$

$\therefore$ # of elements left after first iteration $= |X| - \frac{|X|}{C_{opt}}$

$$= |X| \left(1 - \frac{1}{C_{opt}}\right).$$

$\Rightarrow$ Let N_i denote the number of uncovered elements after i iterations.

$\therefore X_i$ = Set of uncovered elements.

X_i must have a cover of size C_{opt}.

$\therefore$ There exists a set which covers $\frac{|X_i|}{C_{opt}}$ uncovered elements

$\Rightarrow$ Greedy will choose a set that covers atleast these many.

$\therefore$ # of uncovered elements at the end of step i $= N_i - \frac{N_i}{C_{opt}} = N_i\left(1 - \frac{1}{C_{opt}}\right).$

$$\therefore N_T \leq \underset{\substack{\downarrow \\ |X|}}{N}\left(1 - \frac{1}{C_{opt}}\right)^T$$

$$\therefore N_T \leq N\left(e^{-1/C_{opt}}\right)^T$$

$$= N \cdot e^{-T/C_{opt}}.$$

$$\therefore T \leq \underbrace{C_{opt}}_{\text{optimal cover size}} \cdot \log_e N \qquad [N = |X|].$$

$$\therefore T \leq C_{opt} \cdot \log |X|$$