14th Feb 2025

Sessional - III

Computational Complexity Theory :-

→ Branch of CS
→ deals with categorization of problems into different complexity classes.
→ Classes :

→ P	→ NP - complete
→ NP	→ NP - Hard.

Definitions :

1) → P : set of problems that are decision problems which can be solved using Deterministic Algorithm in polynomial time.
i.e. $O(n^k)$ where k is constant.
→ Decision problem : yes/no problems.
eg. SumOfSubset Problem
→ Deterministic Algorithm : practically possible Algos which can be ran on the deterministic machines and can obtain an output that algos are deterministic algorithms.
→ Deterministic machine : practical existing machines are all deterministic machines.
→ Efficient = polynomial time on deterministic machine.
= tractable.

Q) Sum-of-subset is which kind of Algo?
Ans NP
- There is no deterministic algorithm which can solve it in polynomial time.
- Given some solution, they can be verifiable in polynomial time.

Page No.
Date

2) → **NP**: It is a class(set) of decision problems which can be solved in polynomial time using non-deterministic algorithm.

Q Write non-deterministic Algo for sum of subset problem.

Ans. Algo: NP_Sum_of_subset(A, target):-

```
{
  total = 0
  for(i = 1 to A.length) ——————▷ O(n)
  {
    B[i] = choice(0,1)
    if (B[i] == 1)
      total += A[i];
  }

  if (total == target)
    success();
  else
    failure();
}
```

3 major functions for non-deterministic Algo:
- choice()
- success()
- failure()

- If every path of computation says failure() then finally it means that there is no sol^n But if atleast one path of computation says success() then sol^n do exists.

Page No.
Date

Q2] Write non-deterministic algo for searching.

Ans. Algo: ND_search(A, key) ⟶ O(1)

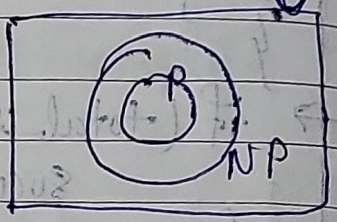
```
{
    j = choice(1, A.length)
    if (A[j] == key)
        success()
    else
        failure()
}
```

Q3] What is the need of ND Algorithms?

Ans. 1) To show that sum problem ∈ NP category.
2) useful for showing NP-complete problems.

Q4] Is P ⊆ NP?

Ans. for P ⊆ NP
- ∀x, x ∈ P ⇒ x ∈ NP.



Q5] Write non-deterministic algo for sorting.

Ans. Algo: ND_Sorting(A, n) ⟶ length, ⟶ array

```
{
    for (i=1 to n) ⟶ O(n)
    {
        j = choice(1, n);
        B[i] = A[j];
    }
    for (i=1 to n-1)
    {   if (B[i] > B[i+1]) failure();
    }
    success();
}
```

Q6] Consider a problem: Decision 0/1.
Given P[1...N], W[1...N], capacity m.
Is it possible to have soln of 0/1 knapsack such that total_profit ≥ x and total_weight ≤ m?

Ans. ND_knapsack(P, W, n, m, x):

```
{
  for (i=1 to N)  ------------------> O(n)
  {                                  ∈ NP
    total_w = 0, total_p = 0;
    x[i] = choice(0,1)
    if (x[i] == 1)
    {
      total_w += w[i]
      total_p += P[i]
    }
  }
→ if (total_w ≤ m AND total_p ≥ x)
      success();
  else
      failure();
}
```

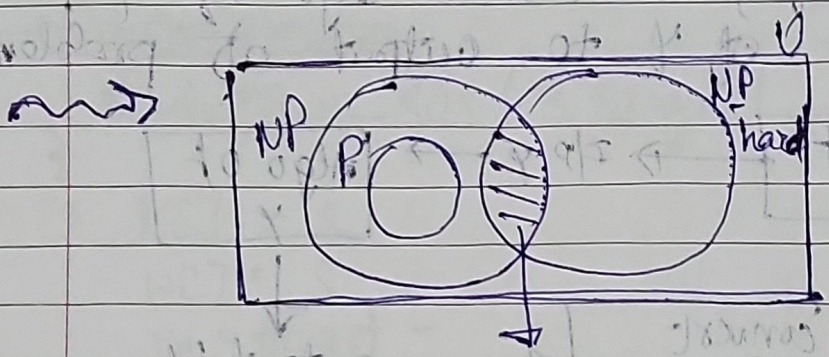
Q7] Show that optimization of 0/1 knapsack ∉ NP.

Ans:- Decision knapsack ∉ P
- Decision knapsack ∈ NP.
- optimization 0/1 knapsack ∉ NP:

→ ND-opt_knapsack(P, W, n, m)
{
// we get total_p and total_weight after for
for verification we need to compare all feasible possible solutions for success() or failure().
thus it does not have poly. time, thus optimization_0/1_knapsack ∉ NP. */
}

→ for NP -
- if finding soln is not in polynomial then atleast verification of soln should be in polynomial.

// optimization problems generally fall in the category of NP-Hard problems.
↳ Intractable.



→ NP ∩ NP-Hard ⇝ NP-complete

Defn.

3) NP-complete :- X is a NP-complete (NP-C) if
(i) X ∈ NP
(ii) X ∈ NP-Hard (NP-H).

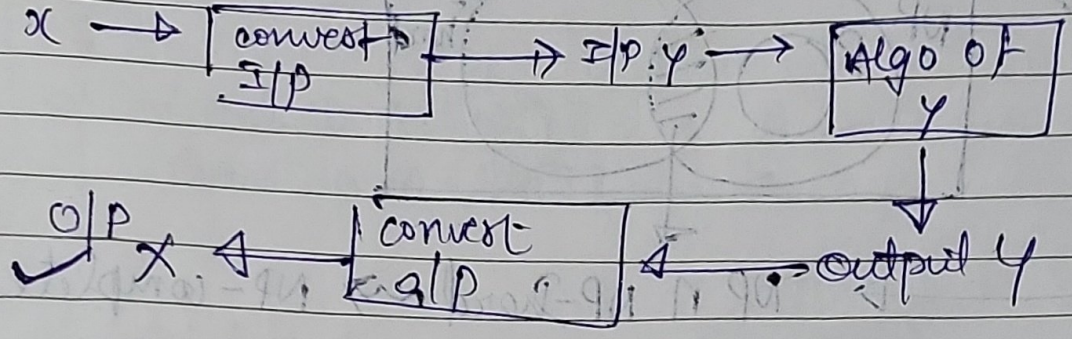
- Towers of Hanoi ∉ NP, ∈ NP-H

4) NP-Hard :- Algo. not in NP and has TC in exponential, factorial etc.

Reduction:

→ problem X reduces to Y in polynomial time.
symbol: $(X \leq_{PT} Y) \rightarrow$ PT: polynomial time.

→ How to reduce X into Y:
1) Convert input of X to Y.
2) Get output of Y as we have algo of Y.
3) convert output of Y to output of problem X.



→ If Y is called polynomial no. of times then it is called polynomial reduction.

Eg. Sum of Subset $\leq_{PT}$ 0/1 knapsack

Given

$S = \{10, 20, 30, 40\}$, target = 50

→ convert :

$P = \{10, 20, 30, 40\}$ $P = S$

$W = \{10, 20, 30, 40\}$ $W = S$.

C = target = 50.

→

Reduction_SOS (S, target)

{

P[] = S.

W[] = S.

m = target.

dp = 0/1_knapsack(P, W, m); // 0/1 knapsack returns dp table

if (dp[S.size][target] == target)

success()

else

failure()

}

→ So, it is a polynomial reduction problem.

Polynomial Time Reductions :-

(1) Say $X \overset{\text{P.T.}}{\leq} Y$

and X is known as NP-Hard problem then, Y is also NP-Hard problem.

(2) Say $X \overset{\text{P.T.}}{\leq} Y$

- and Y has deterministic poly. time algorithm, then X has also polynomial time algorithm.

(3) $X \overset{\text{P.T.}}{\leq} Y$

- If X is known NP-Hard problem then Y is atleast as hard as X.

Note: If $X \overset{\text{P.T.}}{\leq} Y$ then not necessarily Y is reducible to X.

$$Y \overset{\text{P.T.}}{\leq} X \cdot Y$$

$\rightarrow X \overset{\text{P.T.}}{\leq} Y$

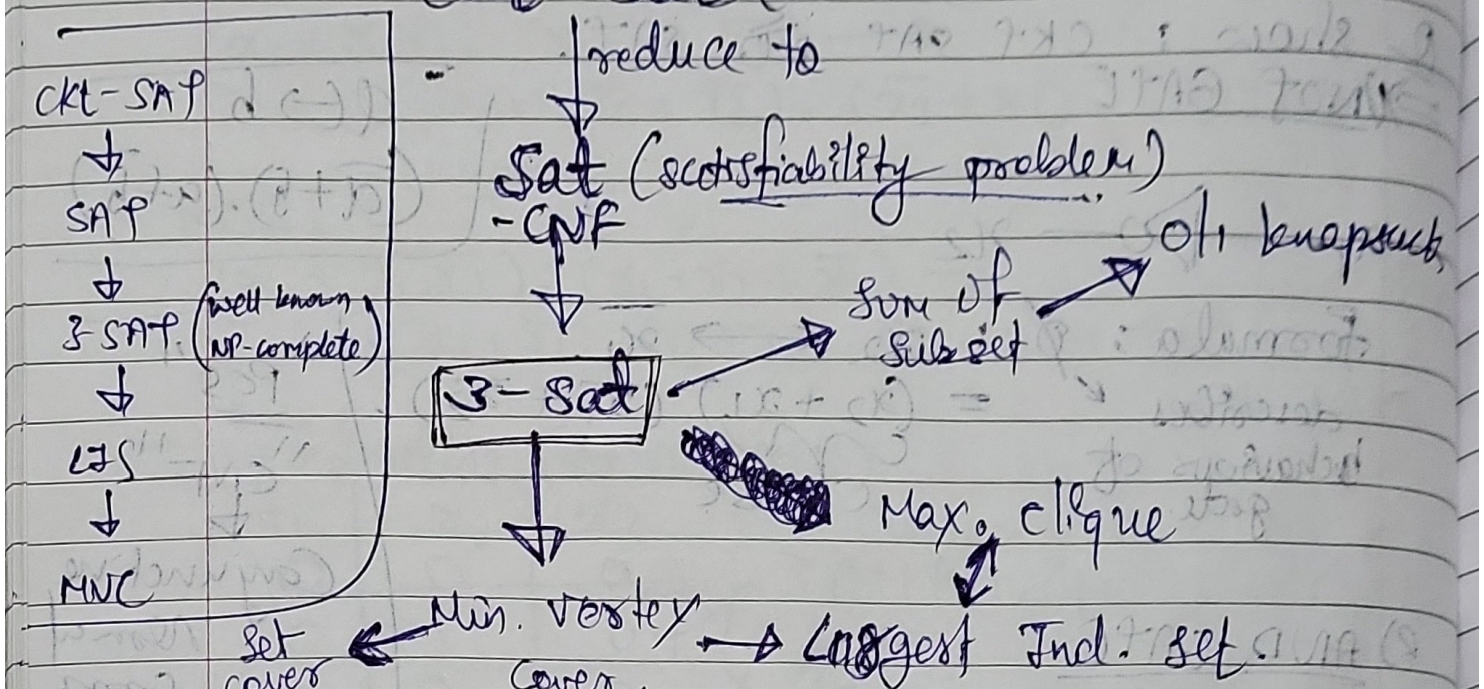
Given y is easy then x is also easy.

$\rightarrow$ Reverse Compatibility is never guaranteed while reduction.

Cook-Levin's theorem:-

⇒ Circuit-satisfaction is NP-Complete.

Plan:-



ckt-sat $\xrightarrow{P.T.}$ SAT ↳ SAT is also hard

↳ Known hard

SAT $\xrightarrow{P.T.}$ 3-SAT ↳ 3-SAT is also hard

↳ Known hard

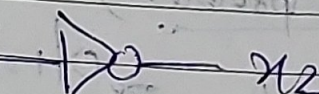
Note: Most people use 3-SAT as a known NP-Complete problem and thereby they can show that others problem is also NP-complete problem.

CKT-SAT problem (not tractable)

- Given combinational circuit, the goal is to check whether there exists input value which makes the output of the circuit "true" i.e. 1?

Q show: CKT SAT $\leq_P$ SAT.

⇒ NOT GATE



Formula: $\phi : x_2 \longleftrightarrow \overline{x_1}$.

$= (x_2 + x_1) \cdot (\overline{x_1} + \overline{x_2})$.

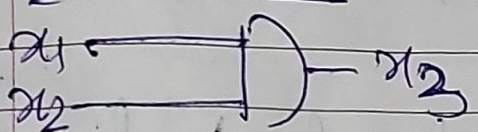
(describes behaviour of gate; each bracket: clause)

$a \leftrightarrow b$

$(\overline{a}+b).(a+\overline{b})$

"POS" = "CNF" ⇒ Conjunctive normal form

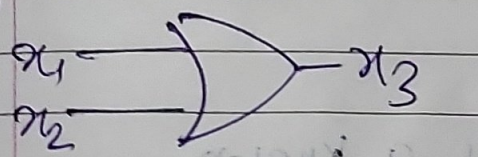
2) AND GATE



$\phi : x_3 \longleftrightarrow x_1 . x_2$.

$= (x_3 + \overline{x_1 x_2}) . (x_1 x_2 + \overline{x_3})$.

3) OR GATE



$\phi : x_3 \longleftrightarrow x_1 + x_2$.

$= (x_3 + \overline{x_1 + x_2}) . (x_1 + x_2 + \overline{x_3})$.

Page No.
Date

Pol. $x \to y = \bar{x} + y$

$y \to x = x + \bar{y}$

$x \leftrightarrow y = (x \to y) \cdot (y \to x)$

$= (\bar{x} + y) \cdot (x + \bar{y})$ ✓

→ AND:

$(x_3 + \bar{x}_1 x_2) \cdot (x_1 x_2 + \bar{x}_3) \longrightarrow$

$(x_3 + \bar{x}_1 + \bar{x}_2) \cdot (\bar{x}_3 + x_1) \cdot (\bar{x}_3 + x_2)$

→ OR:

$(x_3 + \bar{x}_1 \cdot x_2) : (x_1 + x_2 + \bar{x}_3) \longrightarrow$

$(x_3 + \bar{x}_1) \cdot (x_3 + \bar{x}_2) \cdot (x_1 + x_2 + \bar{x}_3)$

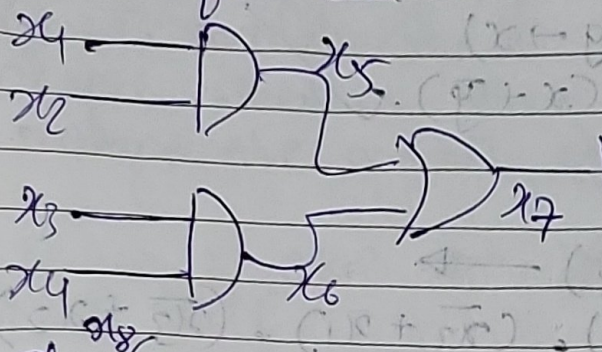
→ Verification (AND GATE)

x_1	x_2	x_3	
0	0	0	→ (0+1+0)·(1+0)·(1+1) = 1
0	1	0	
1	0	0	ϕ = 1
1	1	1	

0	0	0	
0	1	1	ϕ = 0
1	0	1	
1	1	0	

a →

Eg: Convert the following ckt to equivalent SAT formula.



Sol: $\phi = x_8 (x_5 \leftrightarrow x_1 . x_2) . (x_6 \leftrightarrow x_3 + x_4) .$
$(x_7 \leftrightarrow x_5 + x_6) . (x_8 \leftrightarrow \overline{x_7})$

→ This can be converted to POS.

$\therefore \phi = x_8 (\overline{x_5} + x_1) . (\overline{x_5} + x_2) . (x_5 + \overline{x_1} + \overline{x_2}) .$
$(\overline{x_6} + x_3) (x_6 + x_4) . (x_6 + \overline{x_3} + \overline{x_4}) .$
$(\overline{x_7} + x_5 + x_6) . (x_7 + \overline{x_5}) . (x_7 + \overline{x_6}) .$
$(x_8 + x_7) (\overline{x_8} + \overline{x_7}) .$

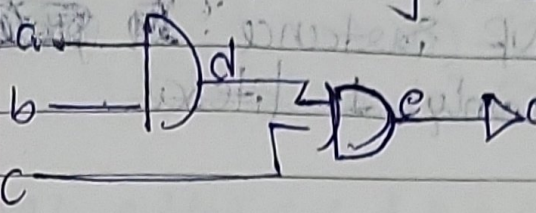
↓

CNF - conjuctive normal format
- AND of clauses.

Q When is ϕ in 3-SAT?

Ans. If each clause had exactly 3 values, then ϕ is in 3-SAT.

Q1] Consider following instance of CKT-SAT.



↳ Given a boolean circuit, can we have inputs such that o/p is 1?

Sol:- Here,

$$\phi = f(f \leftrightarrow \bar{e})(e \leftrightarrow d \cdot c)(d \leftrightarrow a \cdot b)$$
$$= f(f+e)(\bar{f}+\bar{e})(\bar{e}+d \cdot c)(\bar{e}+d \cdot c)$$
$$(d+\bar{a}+\bar{b})(\bar{d}+a \cdot b)$$
$$= f(f+e)(\bar{f}+\bar{e})(\bar{e}+d)(\bar{e}+c)(e+\bar{d}+\bar{c})$$
$$(d+\bar{a}+b)(\bar{d}+a)(\bar{d}+b).$$

↳ SAT CNF's instance.

★ SAT CNF: Given a boolean formula ϕ, is there any i/p that gives $\phi = 1$.

→ CKT-SAT can be solved by SAT-CNF

i.e. CKT-SAT $\leq_{P.T.}$ SAT-CNF

→ In SAT-CNF formula ϕ:

1) Literal: Variable or its complement
2) Clause: "OR" of literals
3) If every clause has exactly k literals then the formula ϕ is in k-SAT CNF formula.
4) 3-SAT CNF: every clause should have 3 literals.

Q2) Show that instance of SAT-CNF can be converted to 3-SAT-CNF instance?

Ans- Case I: Clause having only 1 literal.

Eg: clause has $\bar{a}$.

Given $\bar{a}$

$= \bar{a} + 0$

$= \bar{a} + d_1 \bar{d}_1$ → (where d_i is "dummy" variable)

$= (\bar{a} + d_1)(\bar{a} + \bar{d}_1)$

$= (\bar{a} + d_1 + d_2 \bar{d}_2)(\bar{a} + \bar{d}_1 + d_3 \bar{d}_3)$

$= (\bar{a} + d_1 + d_2)(\bar{a} + d_1 + \bar{d}_2)(\bar{a} + \bar{d}_1 + d_3)$

$(\bar{a} + \bar{d}_1 + \bar{d}_3)$.

Case II: clause having only 2 literals.

Eg: $a + \bar{b}$

$= a + \bar{b} + 0$

$= a + \bar{b} + d_1 \bar{d}_1$ → (where d_1 is "dummy" variable)

$= (a + \bar{b} + d_1) \cdot (a + \bar{b} + \bar{d}_1)$

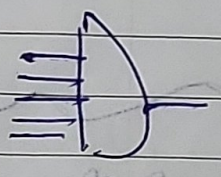
Case III: clause having more than 3 literals

Eg: $a + \bar{b} + c + d$

$= (a + \bar{b} + d_1)(c + d + \bar{d}_1)$

dummy ↑ ↑ complemented same dummy.

Eg:



Here,

If $a \cdot \bar{b} = 1 \Rightarrow d_1 = 0 \Rightarrow \bar{d}_1 = 1$

If $c \cdot d = 1 \Rightarrow \bar{d}_1 = 0 \Rightarrow d_1 = 1$.

If $a = \bar{b} = c = d = 0$ / $d \oplus \bar{d} = 1$ so true.

Eg2: $\bar{a}+b+\bar{c}+d+e$.

$= (\bar{a}+b+d_1)(\bar{c}+d+e+\bar{d}_1)$

$= (\bar{a}+b+d_1)(\bar{c}+d+d_2)(e+\bar{d}_1+\bar{d}_2)$

Eg3: $a+b+c+d+\bar{e}+f$.

$= (a+b+d_1)(c+d+\bar{e}+f+\bar{d}_1)$

$= (a+b+d_1)(c+d+d_2)(\bar{e}+f+\bar{d}_1+\bar{d}_2)$

$= (a+b+d_1)(c+d+d_2)(\bar{e}+f+d_3)(\bar{d}_1+\bar{d}_2+\bar{d}_3)$

→

∴ for $K > 3$

⇒ K-3 dummy's are needed to convert to 3-SAT CNF.

→ So, Any SAT instance can be converted to 3-SAT CNF.

(Cook Levin) ∀ 3-SAT-CNF is a well known NP-Complete problem.

→ Vertex Cover: $G = \langle V, E \rangle$ consider graph. Subset $V' \subseteq V$ is vertex cover if all edges $(u,v) \in E$ either $u \in V'$ or $v \in V'$

- finding minimum sized vertex cover is an NP-Hard problem.
- To show that:

known NP-Hard ≤ p.t. Min. vertex cover optimized (MVC).

Eg 3-SAT.

→ Decision version of MVC → NP-Complete

Q Given ∀ vtex graph G, does it have vertex cover of size K., If for K → true, it will be true ∀ K' > K.

Plan:

- Decision version of MVC is NP complete.

steps:

1) Decision MVC $\in$ NP
   - write non-deterministic algo. which runs in P.T.
2) Show that it is atleast as hard as some well known NP-Hard problem:
   Known hard $\leq_{P.T.}$ Decision MVC.

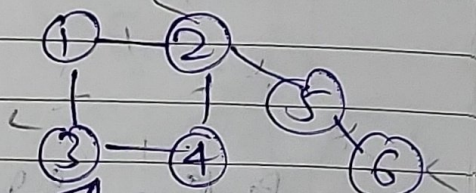
   3-SAT $\leq_{P.T.}$ Decision MVC

→ we do: 3SAT $\leq_{P.T.}$ Largest Independent set $\leq_{P.T.}$ Decision MVC

→ Independent largest set:

- $G = \langle V, E \rangle$, subset $V' \subseteq V$ is "IS" Independent set if no two vertices u, v from IS are connected by edge i.e. if $u \in IS$ and $v \in IS \Rightarrow (u,v) \notin E$.

Eg: {1,5} ✓ IS {1,4,5} IS ✓
{1,2} ✗ IS



→ Vertex V − IS = Vertex cover

Eg: {1,2,3,4,5,6} − {1,4,5}
= {2,3,6} → vertex cover

- we want to find largest size independent set (LIS).

→ In a graph G,
- If VC of size k exists then IS of size n-k exists.
- If IS of size k exists then VC of size n-k exists.
- So, VC and IS are reducible over each other.

$\Rightarrow IS \leq_{P.T.} VC$, $VC \leq_{P.T.} IS \Rightarrow IS \equiv_{P.T.} VC$.

Q1 Show that S is Ind. set of size K iff V-S is vertex cover of size n-K.

Sol:- Part - I

Let S be the "independent set" of size K, then set V-S will be of size n-K.

- now consider some edge $(u,v) \in E$,
here, S is "IS" $\Rightarrow$ either $u \notin S$ or
$v \notin S$ or both $\notin S$.
$\Rightarrow u \in V-S$ or $v \in V-S$ or both $\in V-S$ ... (A)

→ which means
V-S covers (u,v) as from (A)
∴ V-S covers all edges $(u,v) \in E$ and so
V-S is vertex cover of size n-K.

→ Part - II

Given V-S be VC of size n-K,
then set S will be of size K.

- now consider some edge $(u,v) \in E$
then, V-S is "VC" $\Rightarrow$ either $u \in V-S$ or
$v \in V-S$ or both $\in V-S$.
$\Rightarrow u \notin S$ or $v \notin S$ or both $\notin S$.
which means
S will never contains $u,v \in S$ where $(u,v) \in E$
and so S is an independent set of size K.

- So from part - I and part - II we can say that if S is IS of size K then V-S is VC of size n-K and

VC ≡ IS

Page No.
Date

Q: Show that decision version of Ind. set is NP-complete.

X: Is there an "IS" of size $\geq k$ &

→ Step 1: Show X is NP.

- Below is a non-deterministic soln / Algo for X:

```
ND_IS(V, G, K)
{
    IS = ∅.
    for(i=1 to 1
    {
        x = choice(1, n)
        IS = IS ∪ {x}  // set adds unique x only.
    }

    If( IS.size < K)
        failure();
    else
    {
        for every edge (u,v) ∈ G
        {
            if(u ∈ IS AND v ∈ IS)
                failure();
        }

        success();
    }
}
```

This Algorithm runs in poly. time on non-dete. machine and can also be verified in poly time.

∴ ND_IS ∈ NP ⇒ X ∈ NP ... (1)

→ Step 2: Show X ∈ NP-hard.

→ for that we show 3-SAT $\leq_{p.t.}$ IS.

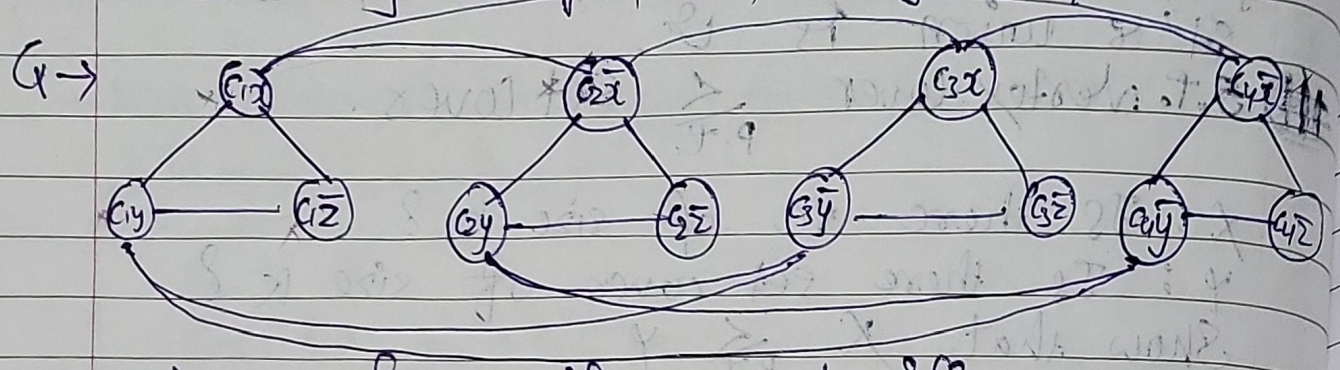
- Consider a 3-SAT formula ϕ:

$\phi = (x + y + \bar{z}) \cdot (\bar{x} + y + z) \cdot (x + \bar{y} + \bar{z}) \cdot (\bar{x} + \bar{y} + \bar{z})$

(C_1, C_2, C_3, C_4)

- n = no. of variables = 3
- m = no. of clauses = 4.

- Constructing a graph from given ϕ:



- ϕ is satisfiable if and only if graph G has IS of size m, here, m = 4.

- Let we consider IS = { $C_1\bar{z}$, $C_2\bar{z}$, $C_3\bar{z}$, $C_4\bar{z}$ }.
Since we can get IS of size m = 4,
ϕ is also having truth value which makes ϕ "true".

→ As ϕ is satisfiable that means X is atleast as hard as SAT
and SAT is NP-hard so X is also NP-hard. ... (2)

from (1) and (2):

X ∈ NP, X ∈ NP-hard

⇒ X ∈ NP-complete

Page No.
Date

Q3) Set Cover

Eg: $U = \{1, 2, 3, 4, 5, 6\}$.

$A_1 = \{2, 5, 6, 4\}$

$A_2 = \{1, 4, 6\}$

$A_3 = \{1, 3, 5\}$

$A_4 = \{2, 4, 5\}$

$A_5 = \{2, 4, 6\}$

} family of subsets of U.

Find min. size subsets of family such that it's union is U.

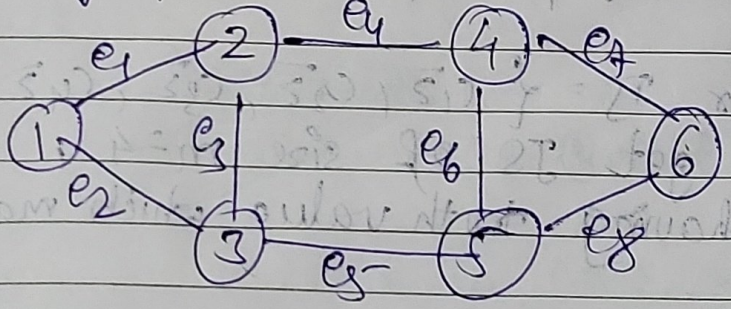
5.T. Vertex Cover $\underset{P.T.}{\leq}$ Set Cover.

X: Is there VC of size k?

Y: Is there set cover of size k?

Show that $X \underset{P.T.}{\leq} Y$.

Sol:- Given graph G:



K = 4

$U = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$;

$A_1 = \{e_1, e_2\}$

$A_2 = \{e_1, e_3, e_4\}$

$A_3 = \{e_2, e_3, e_5\}$

$A_4 = \{e_4, e_6, e_7\}$

$A_5 = \{e_5, e_6, e_8\}$

$A_6 = \{e_7, e_8\}$.

Where

A_i: edges covered by vertex i.

claim:

G has VC of size K ⇔ G has SC of size K

- So, here if you can find SC of size K then it makes that there are K vertices which can cover every edge in G
⇒ K is the size of vertex cover.

This shows that SC is atleast as hard as known hard problem ⇒ MSC is also NP-Hard.

→ Non-deterministic Algo for Set cover

∴ ND_SC(U, A, K)

```
{
    SC = ∅
    for( i=1 to K)
    {
        x = choice (1, n)
        SC = SU ∪ x
    }

    if ( SC == U)
        success();
    else
        failure();
}
```

This algo. runs in poly. time on non-det. machine and can be verified in poly. time

∴ ND-SC ∈ NP ⇒ P ∈ NP

→ As SC is NP as well as NP-hard thus it is NP-complete.

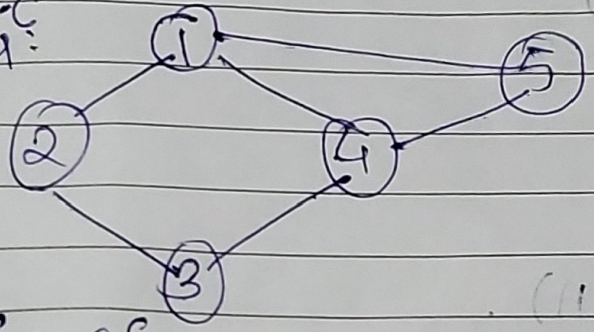
Page No.
Date

Q4) Clique :- (Max clique) [largest size complete subgraph]
clique of $G = \langle V, E \rangle$ is the subgraph of G which is complete.

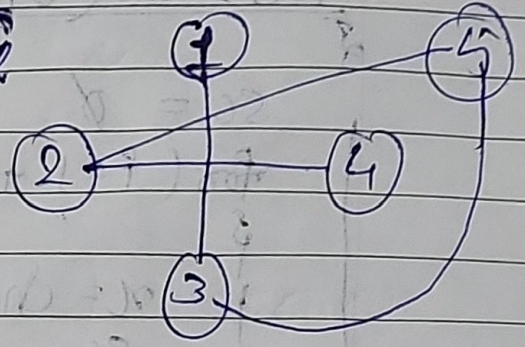
- Claim
G has IS of size k if and only if G^c has clique of size k. → IS $\leq_{P.T.}$ clique

IS: Is there an Ind. set of size k in G?
clique: Is there a clique of size k in G^c?

Eg: G^c:



G



for G^c:
$G^c(i,j) = 1$, if $G(i,j) = 0$
$G^c(i,j) = 0$, if $G(i,j) = 1$

→ k = 3
- Here,
In G^c, clique of size 3 exists i.e. {1, 4, 5}
∴ In G, IS of size 3 exists i.e. {1, 4, 5}.

Similarly, clique $\leq_{P.T.}$ IS
↓ G. ↓ G^c.

∴ IS ≡ clique. (equivalent).

Page No.
Date

Q5] Show that 2-SAT ∈ P.

Ans Eg: $\phi = (x+y).(\bar{x}+y).(\bar{x}+\bar{z}).(y+\bar{z})$.
↑ 2-SAT-CNF

n = no. of variables = 3
m = no. of clauses = 4.

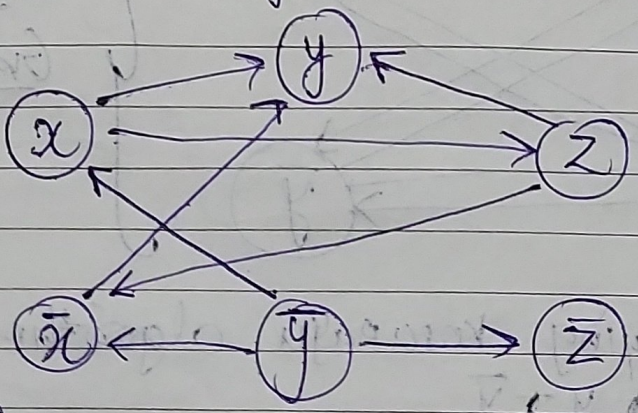
P	NP-complete
2-SAT	3-SAT
2-colouring	n-colouring
Eulers cycle	Hamiltonian cycle

$\bar{x}+y \equiv x \to y \equiv \bar{y} \to \bar{x}$.

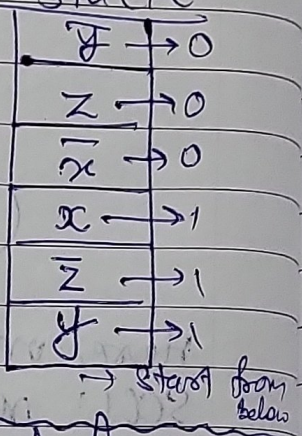
for
clause 1: $x+y \equiv \bar{x} \to y \equiv \bar{y} \to x$
clause 2: $\bar{x}+y \equiv x \to y \equiv \bar{y} \to \bar{x}$
clause 3: $\bar{x}+\bar{z} \equiv x \to \bar{z} \equiv z \to \bar{x}$
clause 4: $y+\bar{z} \equiv z \to y \equiv \bar{y} \to \bar{z}$

create Implication graph :-

- create 2n vertices
- create 2m edges



DFS stack



- Graph construction algo. can be created in poly. time. on practical machine.

- Apply Kasaraju's Algorithm and find strongly connected components.

SCC 1: $\bar{y}$ SCC 3: $\bar{x}$ SCC 5: $\bar{z}$
SCC 2: z SCC 4: x SCC 6: y.

- If a and $\bar{a}$ where a is variable both are present in any SCC. ⇒ ϕ is not satisfiable.
- If there is no SCC where a and $\bar{a}$ is present ⇒ ϕ is satisfiable.

Eg2 $\phi = (x+y) \cdot (x+\bar{y}) \cdot (\bar{x}+y) \cdot (\bar{x}+\bar{y})$

$n = 2$

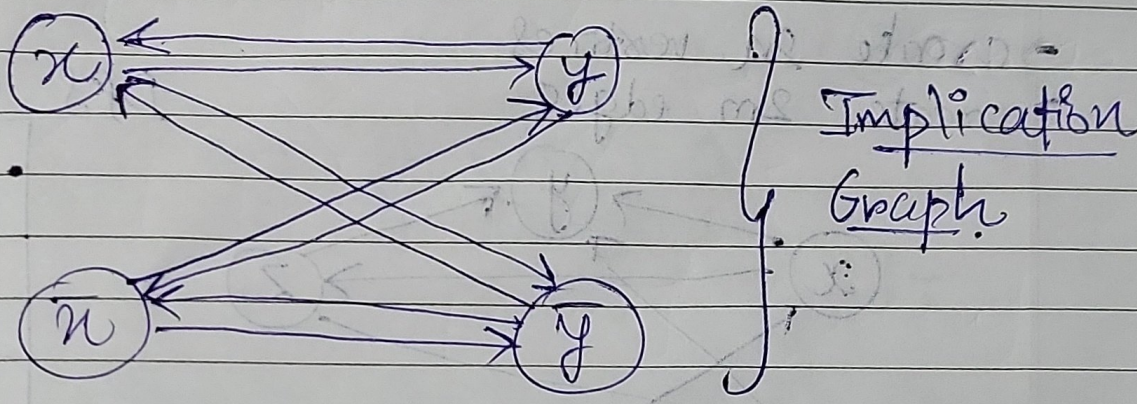
$m = 4$

Clause 1: $x+y = \bar{x} \to y \equiv \bar{y} \to x$

2: $x+\bar{y} \equiv \bar{x} \to \bar{y} \equiv y \to x$

3: $\bar{x}+y = x \to y \equiv \bar{y} \to \bar{x}$

4: $\bar{x}+\bar{y} = x \to \bar{y} \equiv y \to \bar{x}$



Here, on applying Kosaraju algorithm:

SCC1: $x \to \bar{x} \to y \to \bar{y}$

here, x and $\bar{x}$ are in same SCC

so, ϕ is not satisfiable.

→ $x \to y$ means if x is true y is necessary to be true

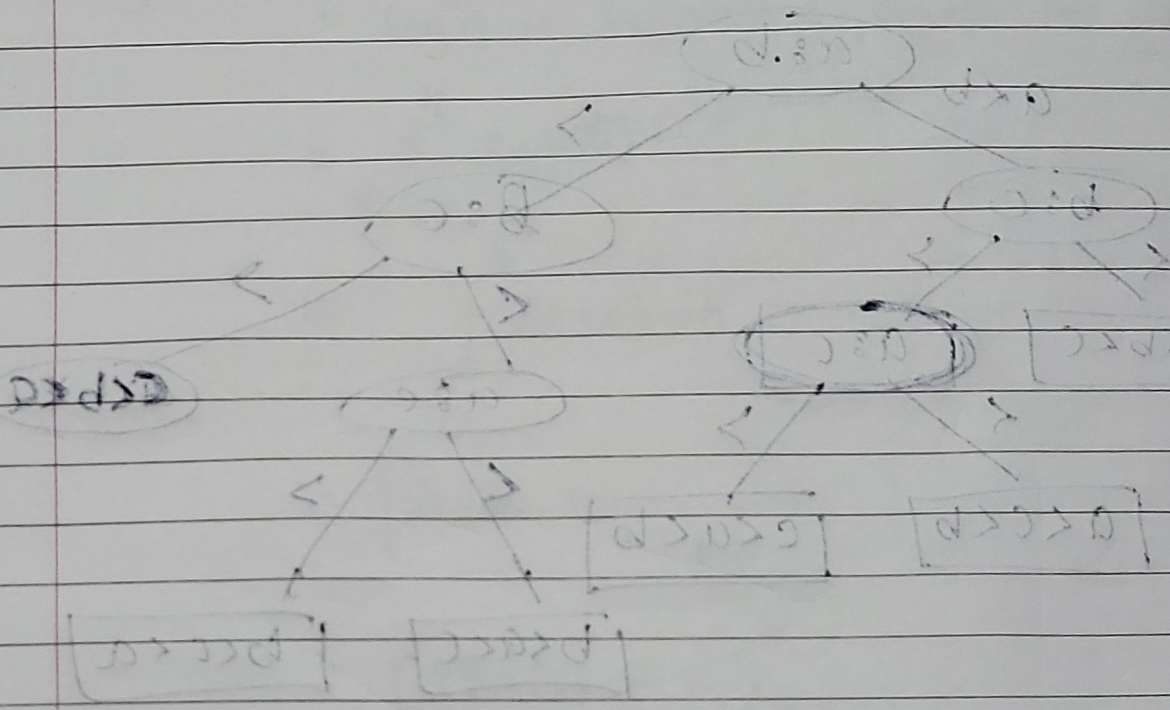
As 2-SAT can be done in poly. time, ∴ 2-SAT ∈ P.

Page No.
Date

Q6 Show that SOS reduces to 0/1 knapsack but reverse is not true.

Sol:- → Sum of subset $\propto$ 0/1 knapsack
P.T.

→ 0/1 knapsack $\not\propto$ Sum of subset.
P.T.



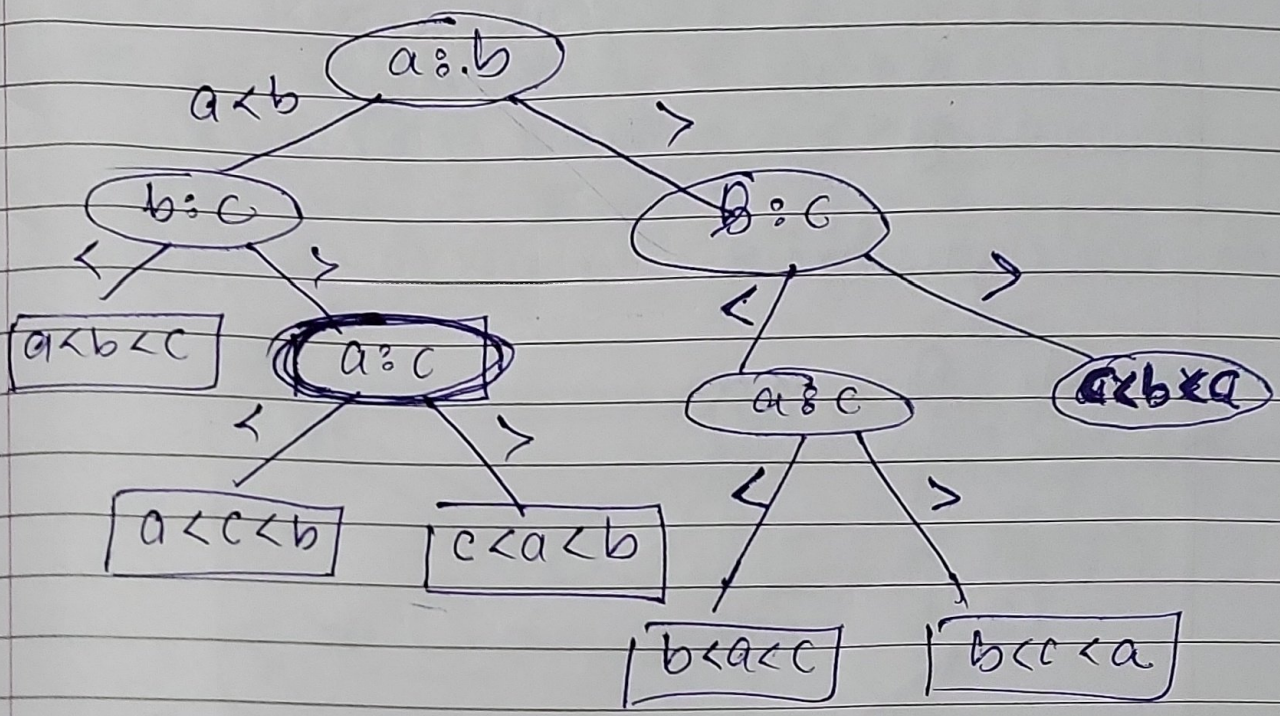
Lower Bound Theory :-

Q) Find out the lower bound for comparison based sorting.

Ans. Let us consider 3 distinct numbers, $n = 3$.

let numbers be a, b and c.

Decision Tree



- for n numbers, in decision tree, leaf nodes = n! (because any permutation could be the answer).

- no. of comparisons = height of the tree.

→ Say height is h to sort array of n integers then : leaf nodes = n!

at height h no. of nodes = 2^h. $h \geq 0$.

→ ∴ for height h we need:

→ $n! \leq 2^h$

$\therefore \log(n!) \leq h \log 2$

$\therefore \frac{\log(n!)}{\log(2)} \leq h \quad \cdots (1)$

→ Now, for $\log(n!)$

$\therefore \log(n!) = \log(n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdots 3 \cdot 2 \cdot 1)$

$= \log n + \log(n-1) + \log(n-2) + \cdots \log 2 + \log 1$

$= \sum_{x=1}^{n} \log(x)$

$\approx \int_{x=1}^{n} \ln(x)\, dx$

$\approx \left[\ln(x) \cdot x - \int_{x=1}^{n} dx \right]_{x=1}^{n}$

$\approx [x \ln(x) - x]_{x=1}^{n}$

$\approx [n \ln(n) - n] - [0 - 1]$

$\approx n \ln(n) - n + 1.$

→ Substitute $\log(n!)$ in (1).

$\therefore \frac{n\log n - n + 1}{\log 2} \leq h$

$\therefore (n\log n - n + 1)\log_2 e \leq h$

$\therefore n\log n \cdot \log_2 e - n\log_2 e + \log_2 e \leq h$

here, dominating term is $n\ln(n)$ and $\log_2 e$ is constant

$\therefore h \geq K \cdot n \cdot \ln(n)$

$\Rightarrow h = \Omega(n\ln(n))$

$\Rightarrow h = \Omega(n\log(n))$

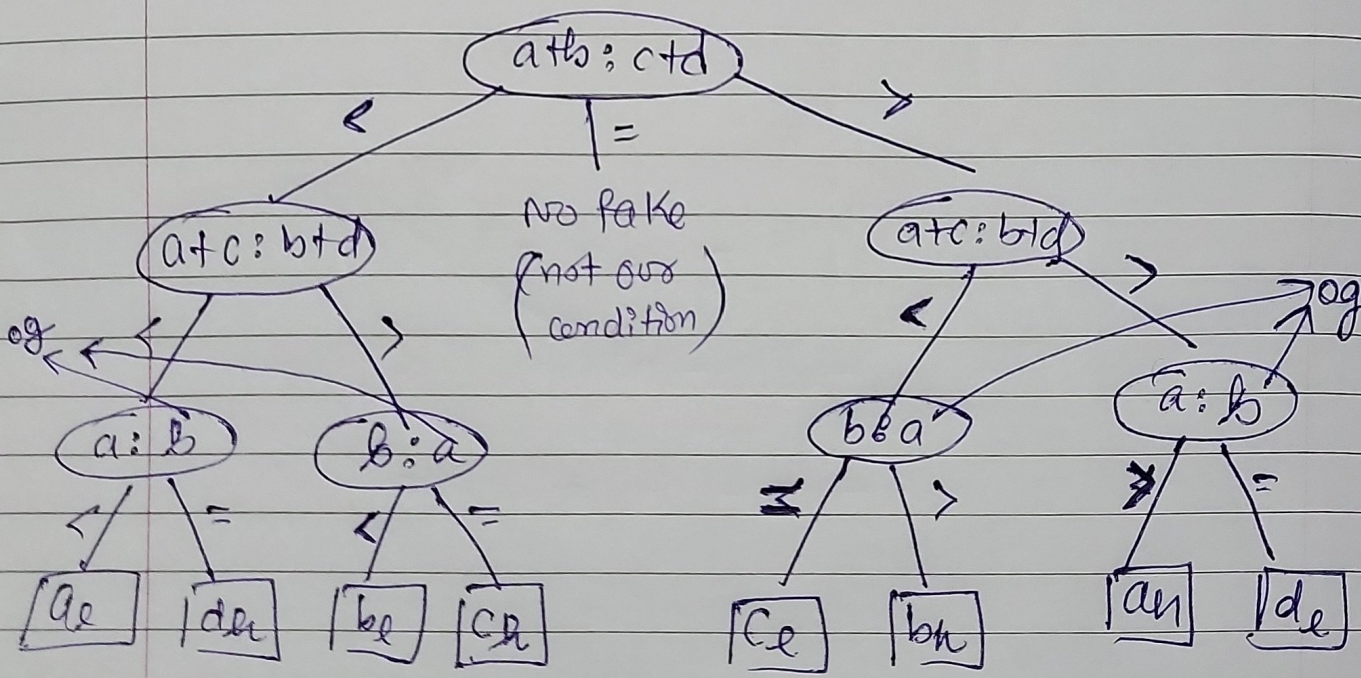
Fake Coin Detection :-

→ Here,
lower bound = $\lceil \log_3(2n) \rceil$.

where n is the no. of coins.

- Consider n = 4.
original coin will have m units weight and fake coin will be lighter or heavier.
- let coins name be a, b, c, d.
possible - a_h b_h c_h d_h
a_l b_l c_l d_l
∴ for n coins - total possibilities are 2n.

⇒ Decision Tree



- Max. no. of comparisons = h.
  needed

→ leaf nodes ≤ no. of nodes in height h

∴ $2n \leq 3^h$ (ternary tree)

∴ $h \geq \log_3(2n)$

⇒ $h = \lceil \log_3(2n) \rceil$