UNIT - 5

NP-Hard and NP-Complete problems

→ **Based on Time-complexity:**

→ The Algorithms are classified into two types

1. polynomial Time
2. Exponential Time

Polynomial Time Algorithm Examples: (P)

Linear Search - n

Binary Search - $\log n$

Insertion Sort - n^2

Merge sort - $n \log n$

matrix multiplication - n^3

Exponential Time Taking Algorithm Examples: (NP) Non-polynomial

0/1 knapsack problem - 2^n

Travelling salesman problem - 2^n

Sum of Subsets " - 2^n

Graph coloring - 2^n

Hamiltonian cycle - 2^n

→ we want the ~~order~~ Algorithms which are faster i.e order of ~~A~~ time. (polynomial time)

→ we want the exponential Algorithms to be solved in polynomial time. (2^n (or) n^n is much bigger than polynomial time)

→even n^{10} is Smaller than 2^n for Some large values of n. These are time composuming Algorithms.

we want polynomial time Alg for this.

P-class problem: P is a set of problems that can be Solved (deterministic) in polynomial (P) time.

Ex: linear search $O(n)$, Binary Search $(O(\log n))$ etc.

Deterministic Alg:

→we know the working of the Algorithm.

$$\begin{tikzcd} & q_1 \\ \text{Start} \; q_0 \arrow[ur, "1"] \arrow[dr, "0"'] & \\ & q_2 \end{tikzcd}$$

} diff I/p's & diff o/p's

→The Algorithm in which every operation is uniquely defined is called "Deterministic Algorithms".

NP class-problem: NP is set of problems that can be solved (non-deterministic) in exponential (NP) time

→But these kind of problems can be verified in polynomial time.

Ex: TSP, 0/1 knapsack etc, Graphcoloring $-O(n \cdot m^n)$

$\to O(n^2 2^n)$ $\to O(2^{n/2})$

Non-deterministic Alg:

$$\begin{tikzcd} & q_1 \\ q_0 \arrow[ur, "0"] \arrow[dr, "0"'] & \\ & q_2 \end{tikzcd}$$

} Same I/p's but diff O/p's

Alg will take more than path, we cannot determine next step of execution. we cannot predict the correct path.

→ In Non-deterministic Alg → we get only approximate solutions, but ~~n~~ not exact solutions.

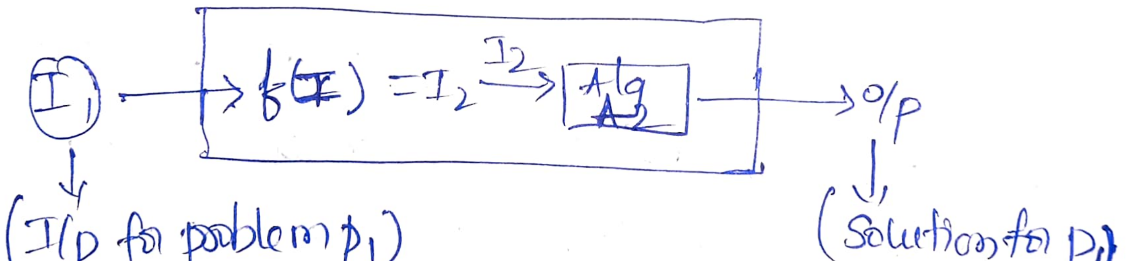
Reduction:-

→ Consider we have 2 decision problems - P_1 & P_2

$$\begin{tikzcd}
 & \text{Input}(I_1) & & & \text{Input}(I_2) \\
P_1 \arrow[ru] \arrow[rd] & & & P_2 \arrow[ru] \arrow[rd] & \\
 & \text{Algorithm}(A1) - \text{unknown} & & & \text{Algorithm}(A2) \ (\text{known})
\end{tikzcd}$$

→ If P_1 can be solved with the help of A2, then we convert I/p I_1 into I_2 and find solution for P_2.

→ P_1 is reducible to P_2.



→ objective 1: If we are unable to solve exp time problems + i.e polynomial time for them then ~~so~~ we have ~~to~~ try to find out the similarities between exptime Alg, finding the relationship between them such a way that if one problem is solved, then other is also solved based on their similar properties.

objective 2 :- when we are unable to write deterministic Algorithm (P) for Exponential time Alg ~~why~~ ~~do~~ ~~not~~ we can write non-deterministic Alg

Non-deterministic

```
Algorithm   NSearch(A, n, key)
{
        j = choice();          — 1
        if (key = A[j])
        {  write(j);
           Success();          — 1
        }
        write(0);
        Failure();             — 1
}                              ----
                               O(1) → constant time.
```

A

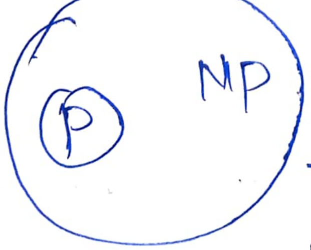
10	8	6	9	4	2
1	2	3	4	5	6

key = 9

→ choice(), success(), failure() stmts are non-deterministic, these are taking order of 1 time.

Relationship between P and NP class problems

$P \subseteq NP$



→ Every problem which is p-class is also in NP-class

→ Every problem which is a NP-class is not in p-class.

→ 'p' class problems can be solved efficiently.

→ 'NP' class problems cannot be solved efficiently

NP-class: → Solved in Non-polynomial Time
→ Verified in polynomial Time
→ Intractable problem

P-Class: → solved in polynomial Time
→ verified in polynomial Time
→ Tractable problem.

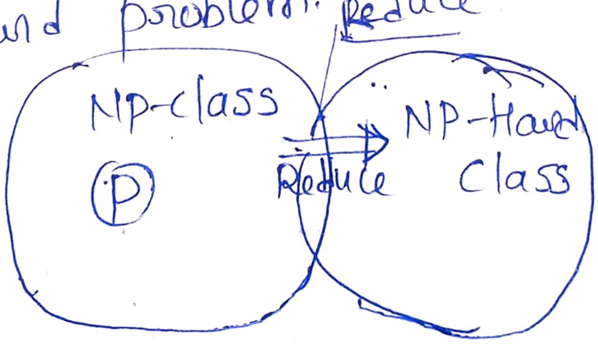
→ P class problems are subset of NP-class problem.

→ It is not known whether P = NP

→ P != NP still research is going on this.

NP-Hard problem :-

→ Every problem in NP class can be reduced into other set using polynomial time, then it is called NP-Hard problem.

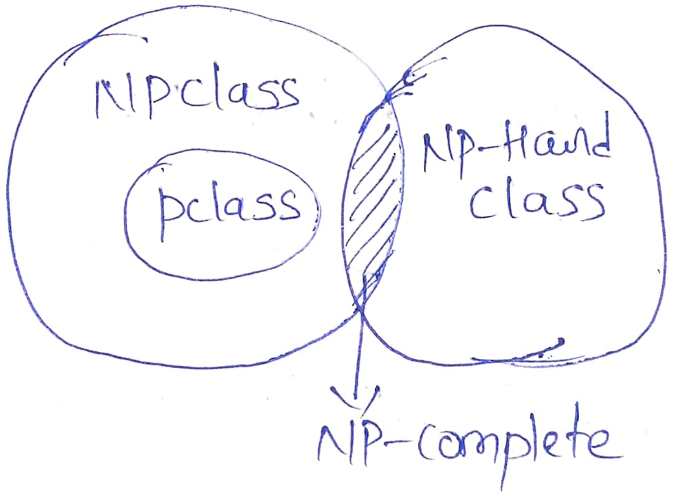


→ NP-Class problems are able to ~~solved~~ solve ~~with the~~ by the Deterministic Algorithms in polynomial time, then those kinds of problems are called as NP-Hard problems

NP-Complete problem:

→ the group of problems which are both in NP and NP-hard are known as NP-Complete problem.

→ All NP-Complete problems are NP-Hard but not all NP-Hard problems are not NP-Complete problem.



→ A problem L_1 is NP-Complete if and only if satisfies two conditions.

(i) L_1 is in NP i.e $L_1 \in NP$.

(ii) Every problem L_2 in NP is polynomial time reducable to L_1 $(L_2 \propto L_1)$

→ If L_1 is solved in polynomial time, then L_2 is also solved.

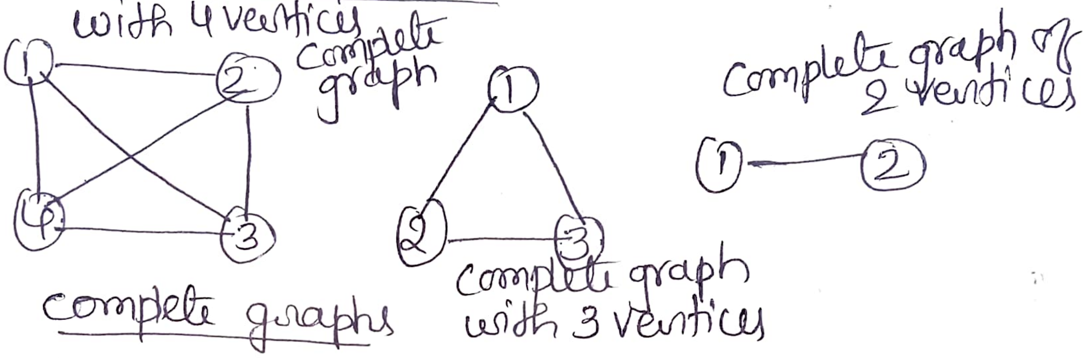
Ex: Determining whether a graph has a

Hamitonian cycle.

Ex: Determining whether a boolean formulae is Satisfiable.

clique - Decision problem

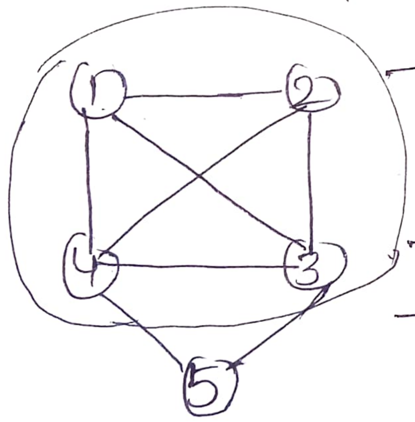
Ex



→ From every vertex, there is an edge connecting to all other vertices. These Graphs are called as complete graphs.

$|V| = n$

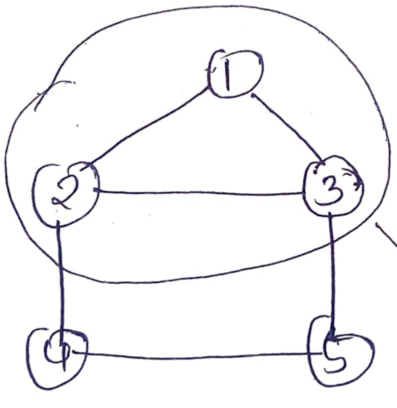
$|E| = \frac{n(n-1)}{2}$



→ subgraph of a graph which is complete. This is called clique.

→ clique

→ This is not a complete graph but having clique.

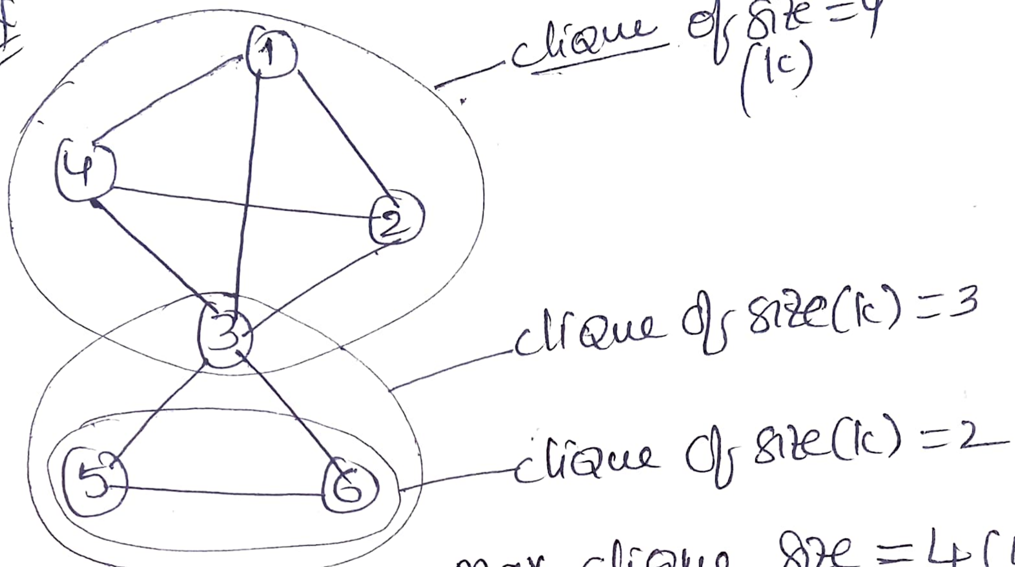


→ This is not a complete graph.

→ this is a complete graph with 3 vertices.

→ clique → ~~com~~. Subgraph of Given graph

Ex



max clique size = 4 (k)

Decision problem: Is there is any clique of size 4 (or) not is a Decision problem (Yes (or) no)
Finding if a graph is having a clique of size k.

optimization problem: what is the maximum clique size in the graph? ($k=4$) finding the maximum clique is a optimization problem.

→ we have to prove that clique-Decision problem as NP-Hard. ~~proof~~

→ If any problem $P = L_2$

→ we have to prove that problem L_2 is NP-Hard

→ For proving this, we select any problem L_1 which is already known as NP-Hard and we have to show that $\boxed{L_1 \propto L_2}$ $L_1 \propto L_2$

$L_1 \propto L_2$
I_1 $\quad I_2$

} I_1 & I_2 are example problems of L_1 & L_2

→ If I_2 is solved in polynomial time, then I_1 is also solved in polynomial time.

satisfiability problem

Sat $\propto$ CDP	
$L_1 \propto L_2$	
I_1	I_2

→ Satisfiability is the known NP-Hard problem.

→ we have to take example of satisfiability i.e Conjuctive Normal form formula, and from this formula, we should prepare a graph having some clique. So, we have to prove that

Sat $\propto$ CDP (Satisfiability reduces to CDP & hence prove that CDP is also NP-Hard

→ let us take 3 variables $x_1\ x_2\ x_3$

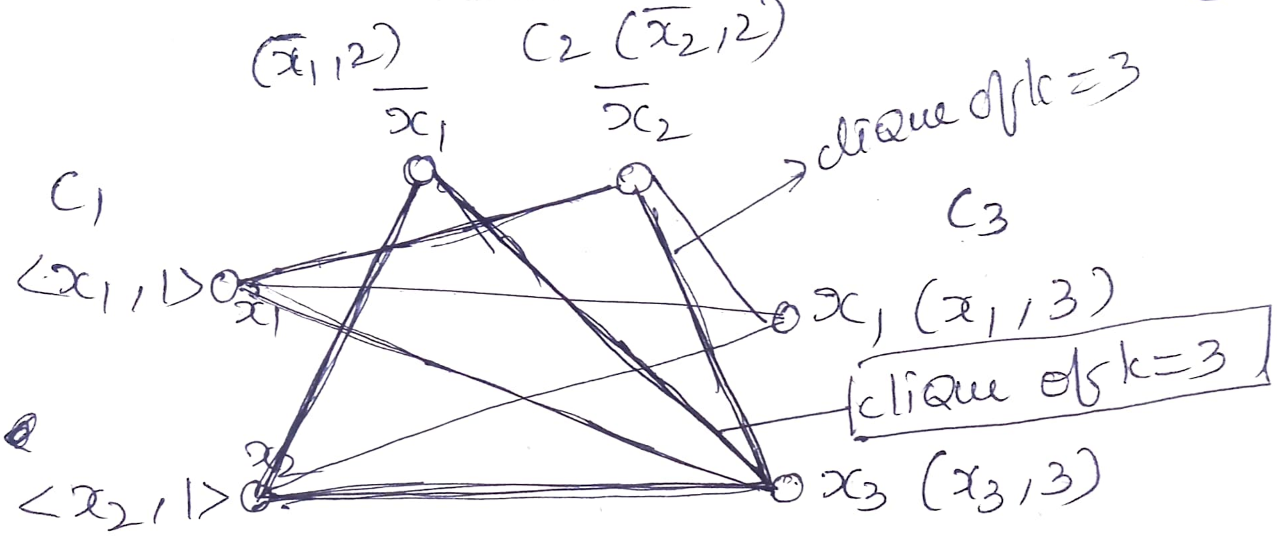
$$F = \underbrace{(x_1 \vee x_2)}_{\text{clause1}} \wedge \underbrace{(\bar{x}_1 \vee \bar{x}_2)}_{\text{clause2}} \wedge \underbrace{(x_1 \vee x_3)}_{\text{clause3}}$$

$$\boxed{F = \bigwedge_{i=1}^{K} C_i}$$

K=3 clauses

→ From this formula, we have to prepare a graph, such that which is having clique.

→ All clauses are doing with AND ($\wedge$) operation



$$\boxed{G = V\{<a, i> \mid a \in C_i\}}$$

→ we should not connect the vertices of same clause.

→ we cannot connect the clause to its Negation.

$x_1 \longrightarrow$ is not connected to $\bar{x}_1$ (Negation)

Forming an edge

$$E = \{(<a, i>, <b, j>) \mid i \neq j \text{ and } b \neq \bar{a}\}$$

⇒ If this problem is solved in polynomial time. then other CDP is also solved in polynomial time.

x_1	x_2	x_3
0	1	1

$$F = (\underset{0}{x_1} \vee \underset{1}{x_2}) \wedge (\underset{1}{\bar{x}_1} \vee \underset{0}{\bar{x}_2}) \wedge (\underset{0}{x_1} \vee \underset{1}{x_3})$$

$$(0 \vee 1) \wedge (1 \vee 0) \wedge (0 \vee 1)$$

$$\Rightarrow 1 \wedge 1 \wedge 1 = 1 \text{ (True) formula is}$$

→ From this graph, if we take another clique then will also becomes true with formula

→ Finally, this is proved that CDP is also NP-Hard.

→ CDP is not proved as NP-complete. In order to prove this, we have to write Non-deterministic ~~Algorithm~~ polynomial time Algorithm for CDP, then it can be proved as NP-Complete.