DAA

UNIT-5

(1) P and NP class problems :

There are two groups in which a Problem can be classified. the first group consists of the problem that can be solved in polynomial time. for example 'Searching' of an element from the list O(logn) sorting the elements O(logn).

The second group consists of problems that can be solved in NON-deterministic polynomial time. for example, knapsack problem and Travelling sales person problem.

* Any problem for which answer is either yes or NO is called "Decision problem". The algorithm for Decision problem is called 'Decision Algorithm'.

* Any problem that involves the Identification of optimal cost [minimum or maximum] is called "optimization problem". The algorithm for optimisation Problem is called "optimization Algorithm"

ex: Kruskal's Algorithm.

Definition of P :-

A problem that can be solved in polynomial time is called "P" and 'P' stands for polynomial time

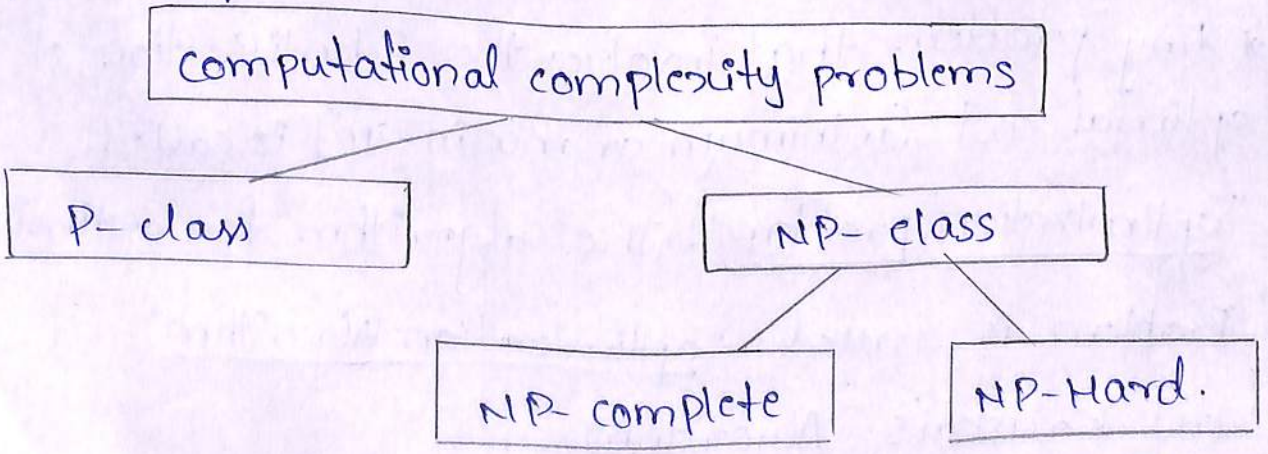
ex: 1. Searching of key element
2. Sorting of elements
3. All pair shortest path.

Definition of NP :

A problem that can be solved in NON-Deterministic polynomial time is called "NP" and NP stands for NON-Deterministic polynomial

ex:- 1. Travelling salesperson problem
2. Graph coloring problem
3. Knapsack problem
4. Hamiltonian circuit problem.

There are two types of computational complexity problems as follows.



A problem is said to be a NP-complete if

i) It belongs to class NP

ii) Every problem in NP can also be solved in Polynomial time.

* If an NP-Hard problem can be solved in polynomial time then all NP-complete problems can also be solved in polynomial time.

* All NP-complete problems are NP-Hard but All NP-Hard problems cannot be NP-complete.

* The NP-class problem are the Decision problem that can be solved by non-deterministic polynomial Algorithms.

Tractable problems:

The problems that can be solved in Polynomial time are called Tractable problems.

for example, 1. Searching an element from a list

2. sorting the list

3. performing matrix multiplication

* Intractable problem:

The problems that can not be solved in polynomial time are called Intractable problems

ex: 1. Towers of Hanoi
2. 8-queen's problem.

Difference between P and NP class problems

P class Problems	NP class Problems.
1. All the p class problems are basically deterministic	1. All the NP class problems are basically non-deterministic
2. Every problem which is a p class is also in NPclass	2. Every problem which is in NP is not the P class problem.
3. p class problems can be solved efficiently	3. NP class problems can not be solved efficiently as efficiently as p class Problems.
ex: Binary search, bubble sort	ex: Knap sack problem, Travelling sales person problem.

Non-Deterministic Algorithm:-

The algorithm in which every operation is uniquely defined is called Deterministic Algorithm

The algorithm in which every operation may not have unique result, such type of algorithms is called NON-Deterministic algorithm.

→ NON-Deterministic means that no particular rule is followed to make the guess.

→ The NON-Deterministic algorithm is "two stage" algorithm.

i) NON-Deterministic (Guessing) stage: It generate an arbitary string that can be though of as a candidate solution.

ii) Deterministic (verification) stage: In this stage, It takes as Input the candidate solution and the Instance to the problem and return. "yes" if the candidate solution represents Actual solution.

Non-Deterministic Algorithm:-

The algorithm in which every operation is uniquely defined is called Deterministic Algorithm

The algorithm in which every operation may not have unique result, such type of algorithms is called NON-Deterministic algorithm.

→ NON-Deterministic means that no particular rule is followed to make the guess.

→ The NON-Deterministic algorithm is "two stage" algorithm.

i) NON-Deterministic (Guessing) stage: It generate an arbitary string that can be though of as a candidate solution.

ii) Deterministic (verification) stage: In this stage, It takes as Input the candidate solution and the Instance to the problem and return. "yes" if the candidate solution represents Actual solution.

Algorithm:

```
Algorithm NON-Determin()
{
 for i=1 to n do
   A[i]:=choose(i);          } // Guessing stage.
  if (A[i]=x) then
   {
    write (i);
     success();               } // verification stage
    }
   write(0);
 fail()
}
```

In the above program given non-Deterministic algorithm there are three functions. used.

1. choose :- Arbitrarily choose one of the element from given input set
2. Fail :- Indicates the unsuccessful completion.
3. Success :- Indicates successful completion.

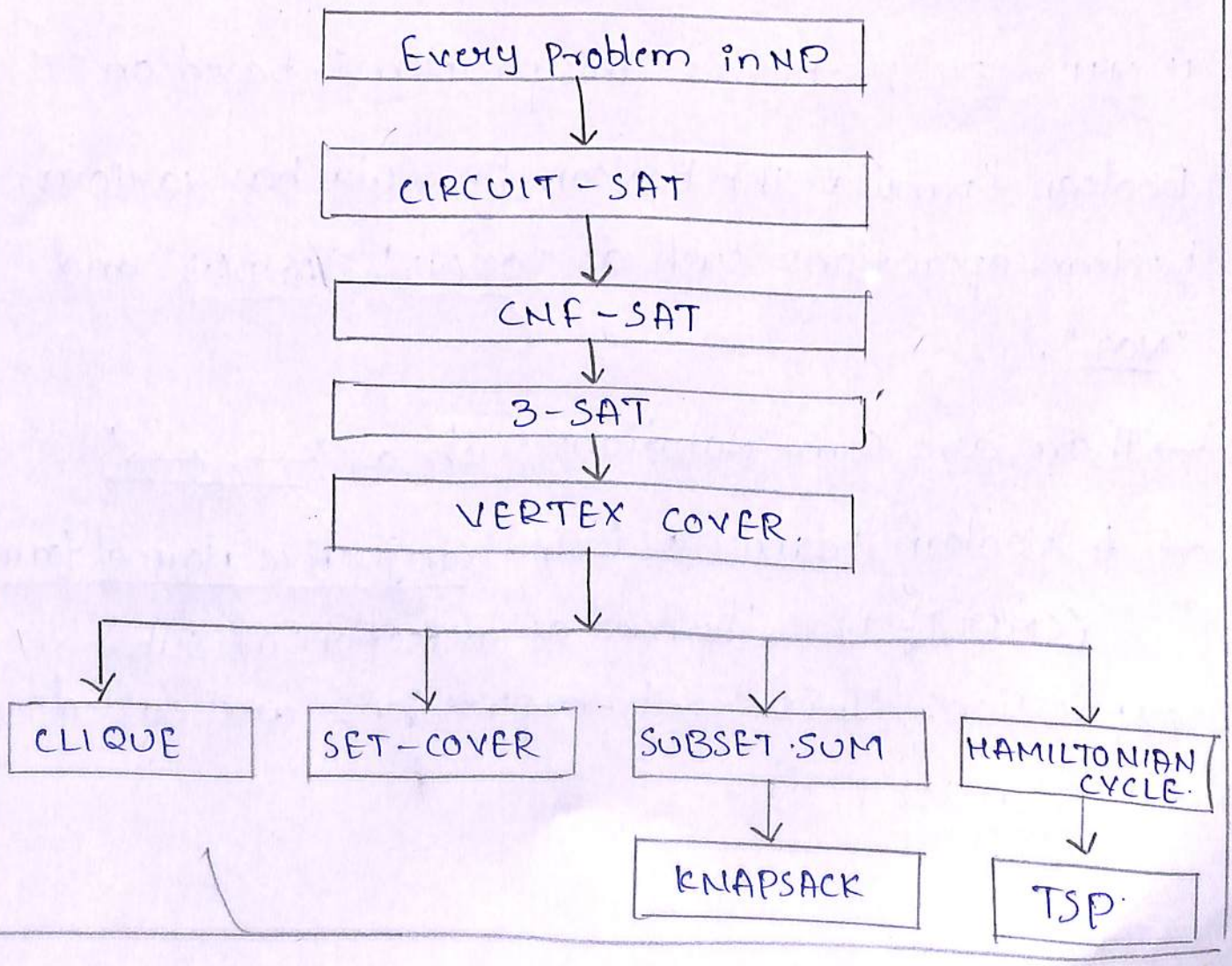
NP - complete problems:

To prove wether particular problem is NP complete or NOT; we use polynomial time. reducibility that means if

$A \xrightarrow{Poly} B$ and $B \xrightarrow{Poly} C$ then

$A \xrightarrow{Poly} C$

→ The reduction is an important task in NP completeness proofs. This can be illustrated by figure as show in below.



→ various types of reductions are

Local replacement: In this reduction A → B by dividing input to "A" in the form of components and these components can be converted to components of "B"

Component Design: In this reduction A → B by building special component for input "B" that enforce properties required by "A".

(i) THE SATISFIABILITY PROBLEM :-

1. CNF - SAT Problem :- This problem is based on Boolean formula. The boolean formula has various boolean operations such as "OR (+)", "AND (·)" and "NOT".

→ There are some Notations such as "→, ↔"

→ A boolean Formula is in "conjuctive Normal form" (CNF) if it is formed as collection of sub-expressions. Thise sub expressions are called as "clauses"

ex: $(\bar{a}+b+d+\bar{g})(c+\bar{c})(\bar{b}+d+\bar{f}+h)(a+c+e+\bar{h})$

$= (0+1+1+0)(1+0)(0+1+0+1)(1+1+1+0)$

$= (1)(1)(1)(1)$

$= 1$

NOTE: $A = 0$

$\bar{A} = 1$

$1+1 = 1$

$1+0 = 1$

$\boxed{CNF = (\ ) \wedge (\ ) \wedge (\ )}$

∴ The CNF-SAT is a problem which takes boolean formula CNF form and checks whether any assignment is there to Boolean values so that formula evaluates to "1"

Theorem: CNF-SAT is in NP-complete

Proof: let "S" be the Boolean formula for which we can construct a simple non-Deterministic Algorithm which can guess the values of variables in Boolean Formula and then evaluates each clause "S". if All the clauses evaluate S to 1 then "S" is satisfied.

Thus CNF-SAT is NP-complete

ii) 3SAT Problem:

A 3SAT problem is a problem which takes a boolean formula "s" in CNF form with each clause having exactly three literals and check whether "s" is satisfied or not. CNF can represent as (+) * (+) * (+) (or)

$(v) \wedge (v) \wedge (v)$ ... etc..

ex: $(\bar{a}+b+\bar{g})(c+\bar{e}+f)(\bar{b}+d+\bar{f})(a+e+\bar{h})$

(or)

$(\bar{a} \vee b \vee \bar{g}) \wedge (c \vee \bar{e} \vee f) \wedge (\bar{b} \vee d \vee \bar{f}) \wedge (a \vee e \vee \bar{h})$

Here "+" nothing but "v" symbol

" * "(.) nothing but "$\wedge$" symbol

Theorem: 3SAT is in NP complete

Proof: let "s" be the boolean formula having 3 literals in each clause for which we can construct a simple non-Deterministic Algorithm which can guess an Assignment of boolean values to "s". if the "s" is evaluated as "1" then "s" is satisfied. then we can prove that 3 SAT is in NP-complete

Some other NP complete problems:

* The 0/1 Knapsack problem: It can be proved as NP complete by reduction from "SUM OF SUBSET" Problem

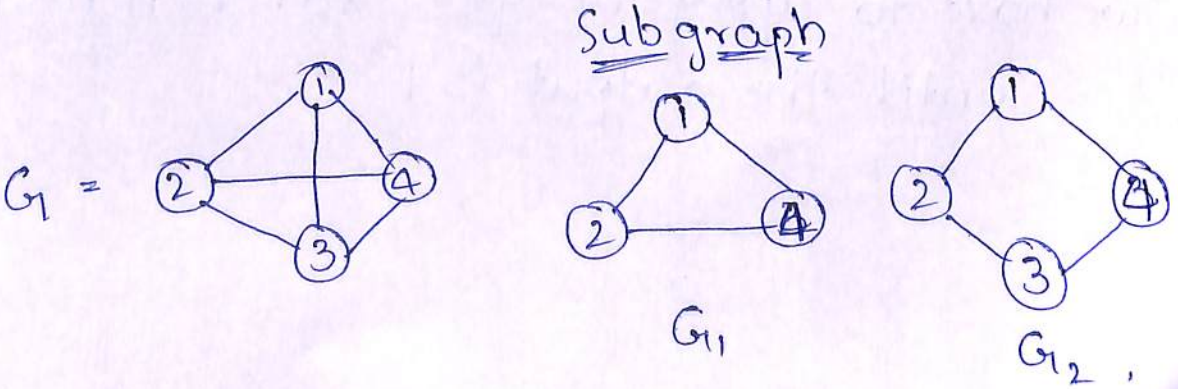
* The Hamiltonian cycle: It can be proved as NP complete, by reduction from vertex cover.

* Travelling salesperson problem: it can be proved as NP complete by reduction from Hamiltonian cycle.

iiii) Clique problem:

clique is nothing but a maximal complete subgraph of a graph G. The graph G is a set of (V, E) where 'V' is a set of vertices and 'E' is a set of edges. The size of a clique is number of vertices in a graph of subgraph.

ex:



→ The graph "G_1" contains three vertices

∴ clique size is = 3.

→ similarly, the graph "G_2" contains four vertices

∴ clique size is = 4.

→ Note that clique of size = 3 and 4 are complete subgraph of graph "G"

Max clique problem: it is an optimization Problem in which the largest size clique is to be obtained.

In the decision problem of clique we have to determine whether G has a clique of size at least K for given K

Let, Dclique (G, K) is a Deterministic Decision algorithm for determining whether graph G has clique or not of size atleast K. then we have to apply Dclique for K = n, n-1, n-2, n-3, ... until the output is 1

→ The max clique can be obtained in

$$\boxed{\text{time} \leq n \cdot f(n)}$$

where $f(n)$ = Time complexity of Dclique

→ If clique Decision problem is solved in polynomial time then Max clique problem can be solved in polynomial time.

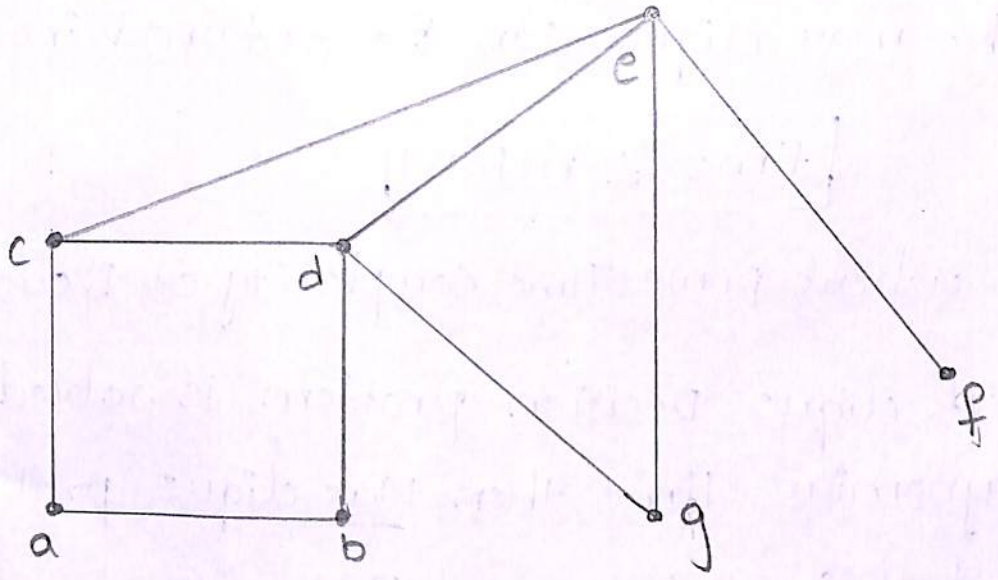
iv) NODE COVER PROBLEM:

The node cover problem is to find node cover of minimum size in a given graph.

→ The word "Node cover" means each node covers it's incident edges.

→ Thus by "Node cover" we expect to choose the set of vertices which cover all the edges in a graph

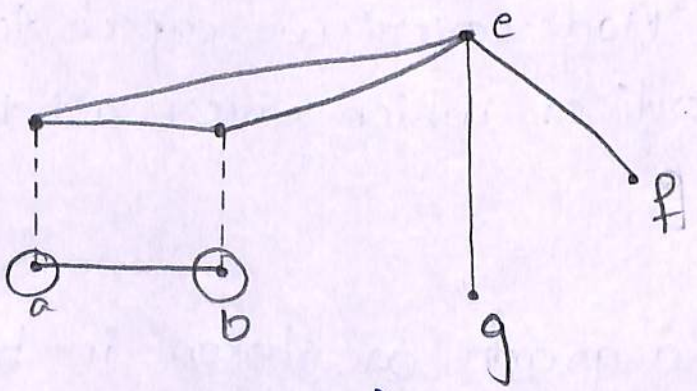
ex: Consider a graph as shown in below.



Sol:

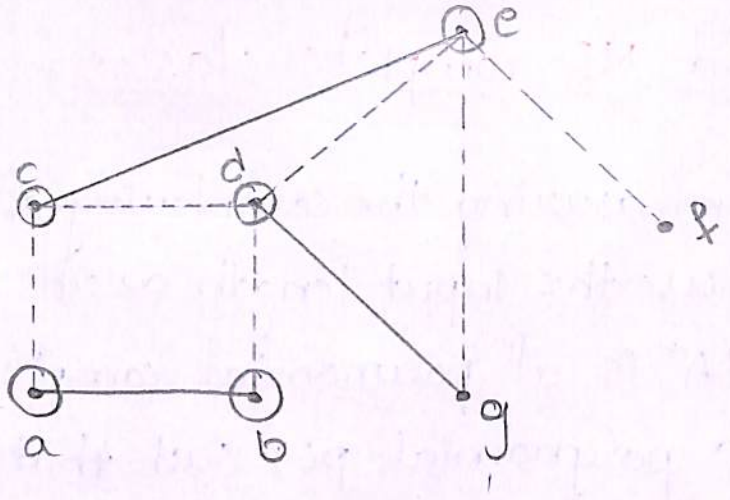
Now we will select some arbitary edge and delete all the incident edges. repeat this process until all the edges wich are incident i.e incident edges get deleted.

Step 1: select an edge a-b and delete all the Incident edges to a and i.e edge (a-c, b-d)



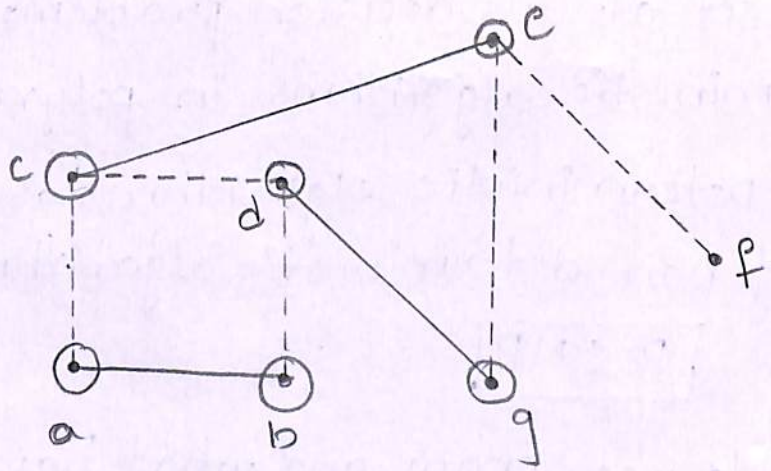
$w = \{a, b\}$

Step 2: Now select an edge c-e and delete all the incident edges.



$w(a, b, c, e)$

Step 3: Select an edge d-g and All the incident edges are already deleted.



$w = \{a, b, c, e\ d, g\}$

∴ Thus we obtain node cover is

a, b, c, e, d, g.

NP-hard and NP-complete classes :-

To measuring the complexity of an algorithm, we use the input length as the parameter An algorithm "A" is of polynomial complexity if there exists a polynomial p() such that the computing time of A is O(P(n)) for every input of size "n"

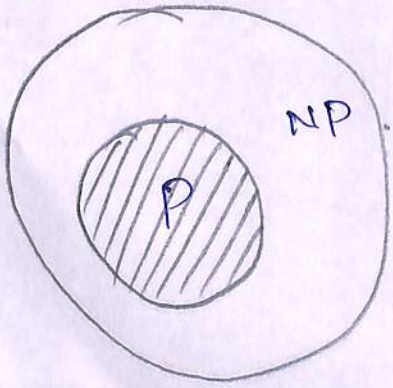
→ P is the set of all decision problem solvable by Deterministic Algorithms in polynomial time.

→ NP is the set of all decision problems solvable by NON-deterministic algorithms in polynomial time.

→ since the Deterministic Algorithms are just a special case of non-deterministic algorithm, so we conclude that $P \subseteq NP$

→ what we do not know, and what has become Perhaps (maybe), the most famous unsolved problem in computer Science, is whether $P = NP$ or $P \neq NP$

→ Is it possible that for all the problems in NP



∴ All 'p' problems are solve in NP but not viceversa

fig:- relation blw P & NP.

(i) Satisfiability is in P if and only if P=NP :-

Now ready to define the NP-hard and NP-complete classes of problems. First we define the notion of reducibility.

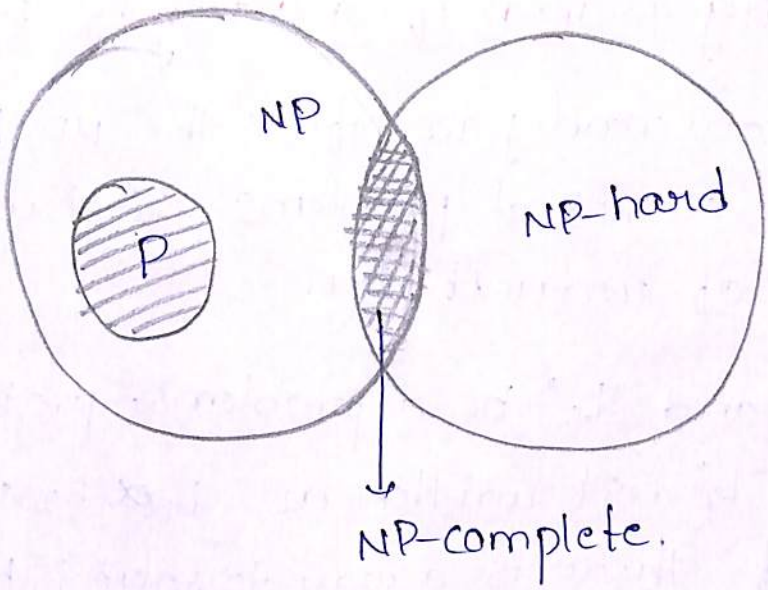
→ Let "L_1" and "L_2" be a problems. problem "L_1" reduces to "L_2". It written as $L_1 \propto L_2$.
if and only if there is a way to solve L_1 by a deterministic polynomial time Algorithm using deterministic algorithm that solves "L_2" in polynomial time.

This implies that, if we have a polynomial time Algorithm to L_2, then we can solve L_1 in polynomial time. one can readily verify that "$\propto$" is a transitive relation i.e if $L_1 \propto L_2$ and $L_2 \propto L_3$ then $L_1 \propto L_3$.

→ A problem L is NP-hard if and only if satisfiability reduces to L i.e satisfiability $\propto$ L

→ A problem L is NP-complete if and only if L is NP-hard and $L \in NP$

→ The relationship among P, NP, NP-complete and NP-hard problems as show below.



→ From the above relation -

* All P problems can solved in 'NP' and All NP Problems can not solve in 'P'

* NP-complete problem nothing but NP problems and NP-hard problem. i.e.,

$$\boxed{\text{NP-complete} = \text{NP} + \text{NP-hard}}$$

* only a decision problems can be solved in NP-complete

* However, An optimization problem may be solved in NP-hard

* optimization problems cannot be NP-complete where as Decision problem can be NP-complete

→ There also exist NP-hard Decision problems that are not NP-complete.

→ Two problems L_1 & L_2 are said to be Polynomially equivalent if and only if

$$L_1 \propto L_2 \text{ and } L_2 \propto L_1$$

→ There some NP-hard problems that are not NP complete for example "Halting problem"

2M **Halting problem:**

The halting problem states that it is possible to determine whether an algorithm will ever halt (or) enter in a loop on certain input?

5M **5) COOK'S THEOREM:**

cookes theorem states that if P=NP then satisfiability lies in P. It is known that satisfiability is in NP. Hence P=NP then satisfiability is in P and also vice versa i.e. if satisfiability is in P, then P=NP.

To prove this $Q(A, I)$ has to formulated from polynomial time NON-Deterministic Algorithm A and input I. when the length of input 'I' is n and the time complexity of Algorithm 'A' $= P(n)$ then the length of Q.

$Q \Rightarrow O(P^3(n) \log n) = O(P^4(n))$

The Deterministic algorithm determines Q and whether state Q is satisfiable or not.

if $O(q(m))$ is time required to formulate length 'm' is satisfiable, then the complexity of the algorithm 'O' is

$$\boxed{O(P^3(n) \log n + q(P^3(n) \log n))}$$

if p is satisfiable then the complexity of polynomial time Function will to $O(r(n))$.

Hence, if satisfiability is found P, then a deterministic 'D' in p can be obtained for each of the NON-deterministic algorithm A is in NP.

thus. If satisfiability is in p then $\boxed{P = NP}$

Important Questions:

1. Explain the classes of P and NP?
2. write about Tractable and Intractable problems.
3. Explain NP-complete problems? [10M]
4. Explain the P, NP, NP-hard, NP-complete classes. Give relationship between them. [10M]
5. Explain about cook's theorem [2M]
6. clique problem [2M]
7. Node cover problem [2M]