UNIT - V

NP-Hard and NP- Complete problems

- Basic Concepts
- Non- deterministic algorithms
- NP- Hard and NP- complete classes
- NP - Hard problems
- Cook's theorem.

Basic Concepts

P and NP Class problems

- There are two groups in which a problem can be classified. The first group consists of the problems that can be solved in polynomial time.

  Ex: searching of an element from list $O(\log n)$, sorting of elements $O(\log n)$

- The second group consists of problems that can be solved in non-deterministic polynomial time.

  Ex: knapsack problem $O(2^{n/2})$, Travelling salesperson problem $O(n^2 2^n)$

- Any problem for which answer / solution 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 optimization problem is called optimization algorithm.

  Ex: Finding minimum cost spanning tree using kruskal's algorithm.

Difference between P and NP class problems

P class problems	NP class problems
1. An algorithm in which for given input, the definite output gets generated is called polynomial time algorithm i.e., P class.	1. An algorithm is called NP class when for given input there are more than one paths that the algorithm can follow. Due to which one cannot determine which path is to be followed after particular stage.
2. All the P class problems are basically determinisbe.	2. All the NP class problems are basically non deterministic.
3. Every problem which is a P class is also in NP class.	3. Every problem which is in NP is not the P class problem.
4. P class problems can be solved efficiently	4. NP class problems cannot be solved efficiently as efficiently as P class problems.
5. Examples: Binary Search, Bubble Sort.	5. Examples: Knapsack problem, Trarelling salesperson problem.

Non-deterministic Algorithm

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

Non-deterministic Algorithm - The algorithm in which every operation may not have unique result, rather there can be specified set of possibilities for every operation. Such an algorithm is called non-deterministic algorithm.

→ The non-deterministic algorithm is a two stage algorithm.

i) Non-deterministic (Guessing) stage -

Generate an arbitrary string that can be thought 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 returns yes if the candidate solution represents actual solution.

- The following algorithm gives three functions

i) Choose

ii) Fail

iii) Success.

Algorithm Non_Det ( )

// A [1:n] is a set of elements

// we have to determine the index i of A at which element x is located.

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

Choose :- Arbitrarily chosen one of the element from given input set.

Fail : Indicates the unsuccessful completion

Success : Indicate successful completion.

– The non-deterministic algorithm complexity is $O(1)$. when A is not ordered then the deterministic search algorithm has a complexity $\Omega(n)$.

Ex:-

1. Nondeterministic search

```
j := choice (1,n);
if (A[j] = x) then
{
    write (j);
    Success ();
}
write (0);
Failure ();
```

nondeterministic complexity - $O(1)$

- deterministic search algorithm complexity $\Omega(n)$.

2. Nondeterministic sorting

```
Algorithm NSort (A, n)
// sort n positive integers
{
    for i := 1 to n do B[i] := 0;    // Initialize B[]
    for i := 1 to n do
    {   j := choice (1,n);
        if B[j] ≠ 0 then Failure ();
        B[j] := A[i];
    }
    for i := 1 to n-1 do    // verify error
        if B[i] > B[i+1] then Failure ();
    write (B[1:n]);
    Success ();
}
```

- complexity $O(n)$ - Non deterministic algorithm
- complexity $\Omega(n \log n)$ - deterministic algorithm

NP - Hard and NP - complete classes

NP - Hard

NP - hardness (Non-deterministic polynomial - time hard) in computational complexity theory, is a class of problems that are informally, 'at least as hard as the hardest problems in NP'.

- A problem H is NP-hard when every problem L in NP can be reduced in polynomial time to H.

Defi:-

A decision problem H is NP - hard when for every problem L in NP, there is a polynomial - time reduction from L to H.

NP - complete

In computational complexity theory, a decision problem is NP-complete when it is both in NP and NP - hard.

- The set of NP complete problems is often denoted by NP-C (or) NPC.

Defi:-

A decision problem C is NP- Complete if

i, C is in NP and

ii, Every problem in NP is reducible to C in polynomial time.

- C can be shown to be in NP by demonstrating that a candidate solution to C can be verified in polynomial time.

Non deterministic knapsack algorithm

```
Algorithm NDKP (p, w, n, m, r, x)
{
    W := 0;
    P := 0;
    for i:=1 to n do        // guessing stage
    {
        x[i] := Choice (0,1);
        W := W + x[i] * w[i];
        P := P + x[i] * p[i];
    }
    if ((W > m) or (P < r)) then    // verification stage
        Failure ();         // checking whether this assignment is
                            feasible and whether the resulting
    else Success();         profit is atleast r.
}
```

- A successful termination is possible iff the answer to the decision problem is yes.

- The time complexity is O(n).

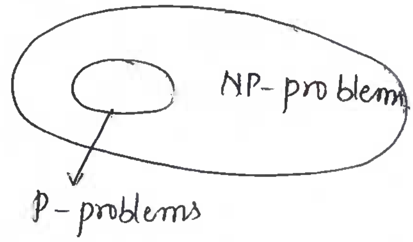
Ex:- Non deterministic

Properties of NP-Complete and NP-Hard problems

P denotes the class of all deterministic polynomial-language problems and NP denotes the class of all non-deterministic polynomial language problems.

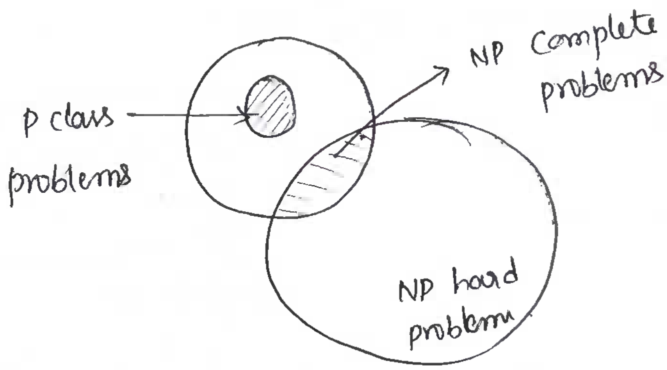
$P \subseteq NP$

but $P = NP$?



- Problems which are known to lie in P are often called as tractable. problems which lie outside of P are often termed as intractable.
- Thus, the question of whether $P = NP$ or $P \neq NP$ is the same as that of asking whether there exist problems in NP which are intractable or not.
- In 1971, S.A. Cook proved that a particular NP problem known as SAT (satisfiability of sets of Boolean clauses) has the property that if it is solvable in polynomial time, so, are all NP problems. This is called a NP-Complete problem.
- Let A and B are two problem then problem A reduces to B if and only if this is a way to solve A by deterministic polynomial time algorithm using a deterministic algorithm that solves B in polynomial time.

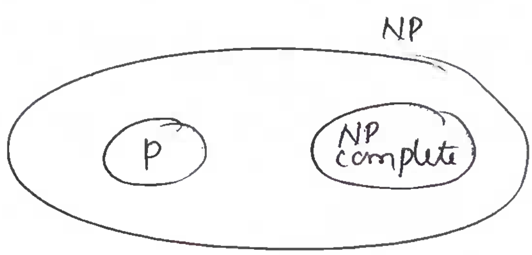
- A reduces to B can be denoted as $A \propto B$. In other words, we can say that if there exists any polynomial time algorithm which solves B then we can solve A in polynomial time. We can also state that if $A \propto B$ and $B \propto C$ then $A \propto C$.



Relationship between P, NP, NP-Complete and NP-hard problems.

- A NP problem such that if it is in P, then NP = P. If no a problem (not necessarily NP) p has this same property then it is called NP-hard.
- Thus the class of NP-complete problem is the intersection of the NP and NP-hard classes.

- The decision problems are NP-complete but optimization problems are NP-hard. However if problem A is a decision problem and B is optimization problem then it is possible that $A \propto B$.



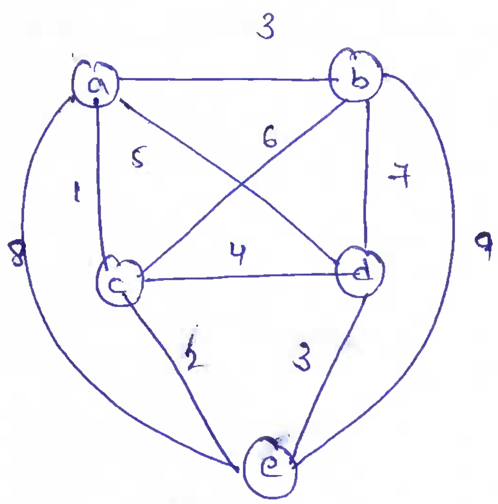
- For instance the knapsack decision problem can be knapsack optimization problem.

Example of NP class problem

Travelling sales person problem

This problem is stated as given a set of cities and cost to travel between each pair of cities, determine whether there is a path that visits every city once and returns to the first city. such that the cost travelled is less.

Ex:-



Tour path -

a - b - d - e - c - a

Tour cost = 16

- This problem is NP problem as there may exist some path with shortest distance between the cities.

- If we get the solution by applying certain algorithm then travelling sales man problem is NP complete problem.

- If we get no solution at all by applying an algorithm, then the above problem belongs to NP hard class.

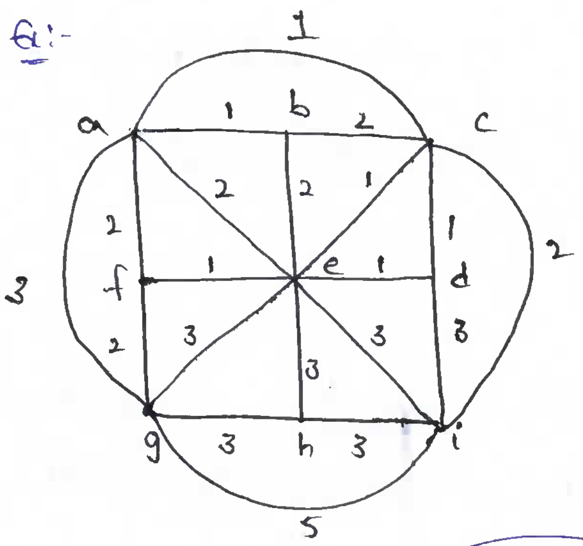
Ex for P class problem

1. Kruskal's algorithm

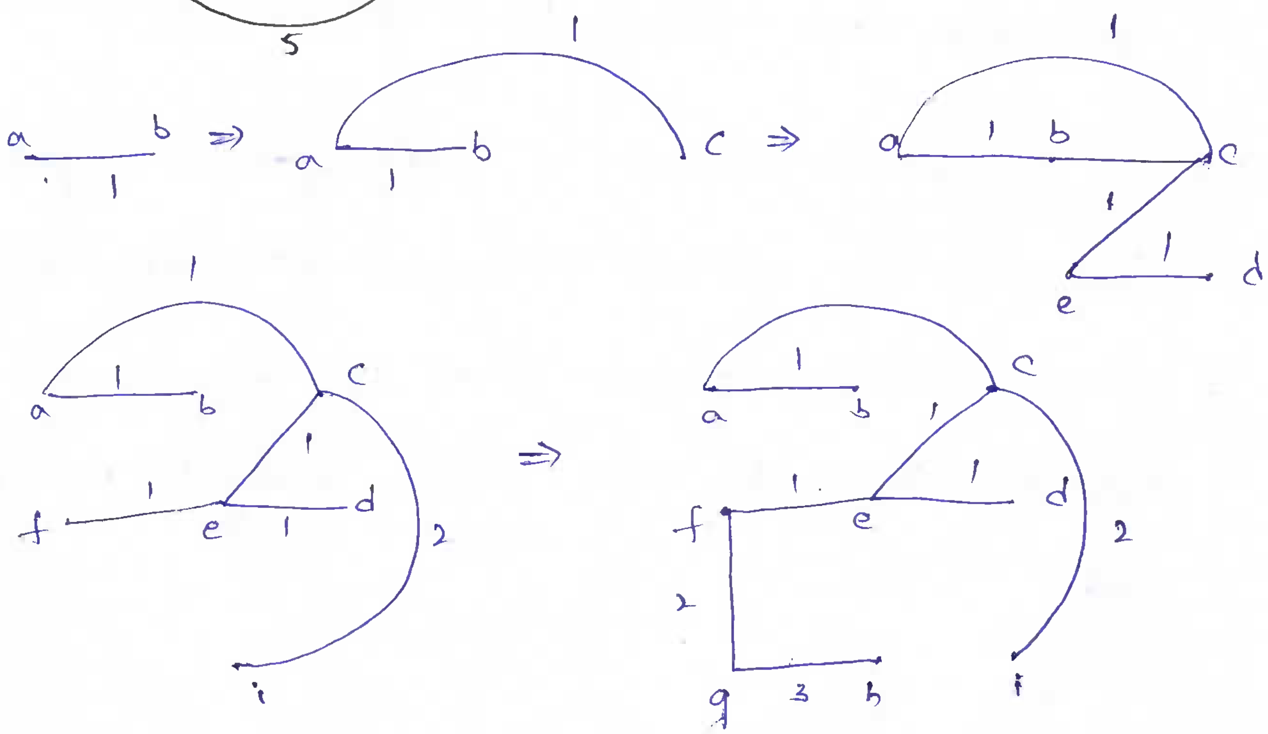
In this, the minimum weight is obtained without forming circuit.

- Each time, the edge of minimum weight has to be selected from the graph.
- It is not necessary in this algorithm to have edges of minimum weights to be adjacent.

Ex:-



Using kruskal's algorithm, we will find minimum weight.



min. weight = 12

- 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.

Note:

The NP class problems are the decision problems that can be solved by non-deterministic polynomial algorithms.

Tractable problems:

The problems that can be solved in polynomial time are called tractable problems.

Ex:- searching an element from a list,
sorting the list,
performing matrix multiplication

Intractable problems:

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

Ex:- Tower of Hanoi,
8 queen's problem.

Definition of P:

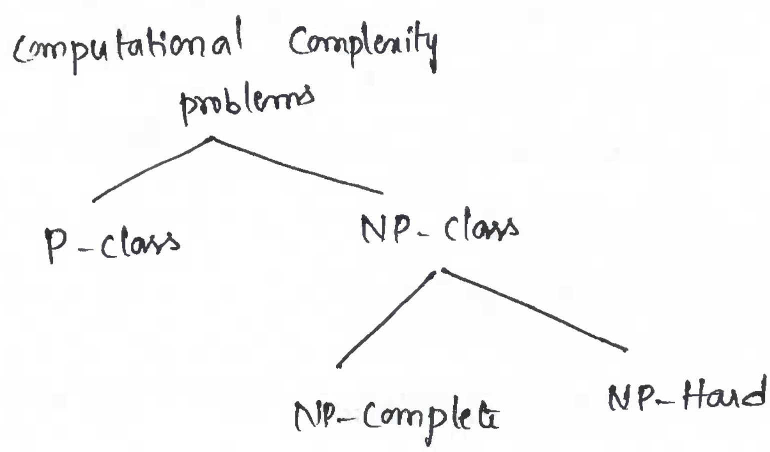
problems that can be solved in polynomial time. (P stands for polynomial)

Ex:- Searching of key element, sorting of elements, All pair shortest path.

Definition of NP:

problems that can be solved in non-deterministic polynomial time. (NP stands for Non-deterministic polynomial)

Ex:- Travelling salesperson problem, Graph coloring problem, knapsack problem, Hamiltonian problems.



- A problem P is called NP-complete if
  i, it belongs to class NP
  ii, Every problem in NP can also be solved in polynomial time.

- There are some NP-hard problem that are not NP-complete. for ex, halting problem. "The halting problem states that 'Is it possible to determine whether an algorithm will ever halt or enter in a loop on certain input?'"

- Two problem P and Q are said to be polynomially equivalent if and only if $P \propto Q$ and $Q \propto P$.

Proving NP problems

path problem (PP)

Given a graph $G = (V, E)$ to find out whether there is a path from u to v, $u, v \in V$ which is of length l, ~~sho~~ such that $l \leq n$ for some n.

- This problem answers in a yes or a no and is hence a decision problem.
- Moreover, since there exists an algorithm that accomplishes the above task in polynomial time, the path problem is a problem that belongs to the P class.

Shortest path problem (SPP)

Given a graph $G = (V, E)$ and the corresponding matrix, the problem is to find the shortest path from u to v, $v \in V$.

- This problem is an optimization problem, since it requires us to find a path that minimizes the distance between the given vertices.
- However, there are many algorithms which accomplish the above task in polynomial time.
- Hence, the problem belongs be' to the class P.

Longest path problem (LPP)

Given a graph $G = (V, E)$ and the corresponding matrix, the problem is to find the longest path from u to v, $v \in V$.

- This problem is an optimization problem, since it requires us to find a path that maximizes the distance between the given vertices.
- However, the problem is an NP-complete problem, as against the shortest path problem.

Subset Sum problem

Given a set S and a number k, the problem is to find out the subset of the given set having sum of its elements equal to k.

- For example, if the set S is {1, 2, 3, 4, 5} and the value of desired sum is 6, then the possible subsets that have the sum of its elements as 6 are {1,5} & {2,4}. The problem seem simple.
- However, if a set has n elements and the desired sum is say m, then the brute-force approach would require the crafting of all possible 2^n subsets, calculating their sum and finding out which subset gives the desired sum.
- The number of subsets of a set having n elements is 2^n, the problem is therefore an exponential one.
- However, the above problem requires enlisting of all the subsets of a given set, finding the sum of all of them and then checking whether the sum of elements of that subset is same as the given number or not.
- The problem, though not unsolvable, has exponential time complexity.
- There is no algorithm for the problem that runs in polynomial time.

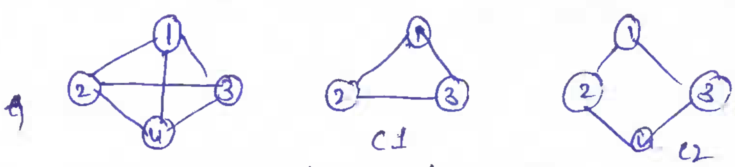
- However, if the solution is known, it is easy to verify the solution in polynomial time. Therefore, the problem can be categorized as an NP-complete problem.

0-1 <u>Integer programming</u>

Given a set of equations of the form $A \times X = B$, where A and B are integer vectors, then there exists a vector X in $\{0,1\}$, which satisfies the above equation.

- The problem does not have a polynomial time algorithm.
- However, if the answer is given, it would be easy to find whether it is correct or not.
- Therefore, the problem can be categorized as an NP-complete problem.

<u>Clique problem</u>



A clique is a complete sub-graph of a given graph. The problem is to find whether a clique of a given number of nodes exists, in the given graph or not.

- Suppose there are n nodes in a graph. There would be 2^n subsets of the set of nodes.
- Each graph would be checked (if it is a complete graph).
- The procedure is of exponential complexity.

- However, if the solution to the above problem is found given, then it would be easy to see whether the solution is correct or not.
- The problem is therefore an NP-Complete problem.

Maximum Clique problem:

- A clique is a complete sub-graph of a given graph. The problem is to find the biggest clique of a given number.
- The problem is not just NP-complete. Until we have the cost of all paths, it would not be possible to find which is the maximum clique.
- The problem, therefore can be categorized as NP-hard problem.

NP Complete problems

To prove whether particular problem is NP complete (or) not, we use polynomial time reducibility. i.e.,

$A \xrightarrow{poly} B$ & $B \xrightarrow{poly} C$ then $A \xrightarrow{poly} C$

- the reduction is an important task in NP completeness proofs.

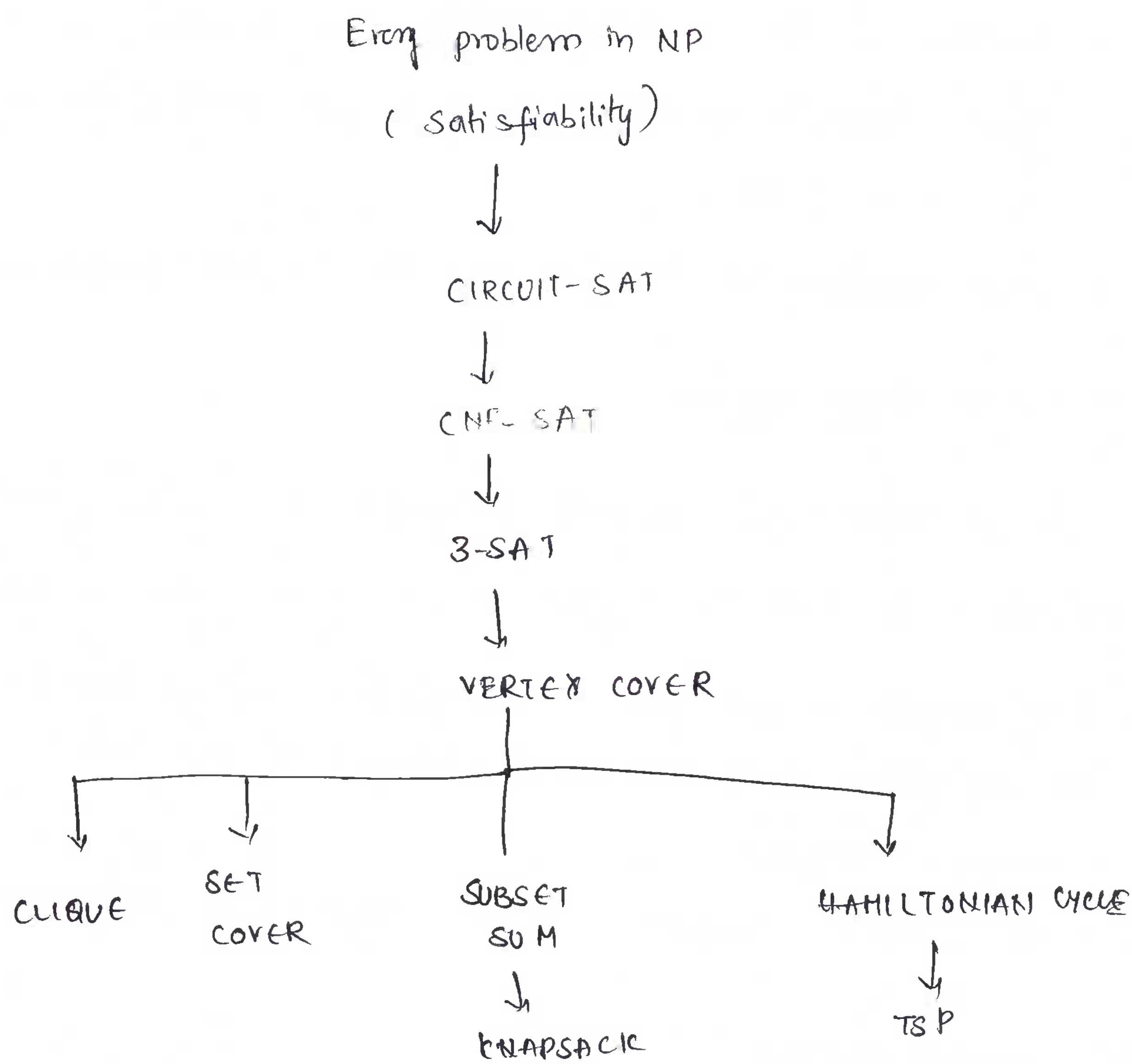


fig: Reduction in NP completeness.

- Types of reductions are

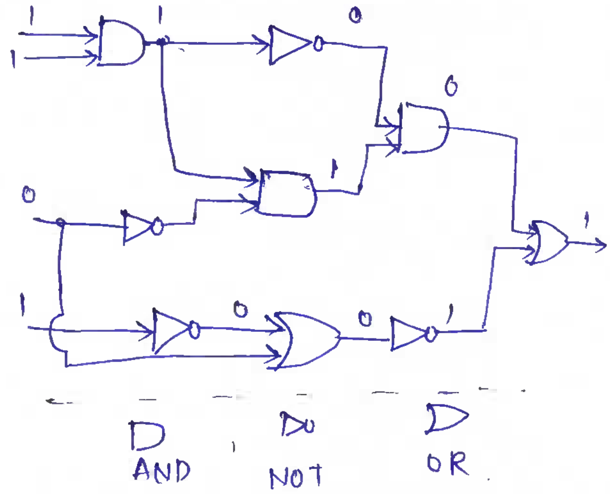
1. Local replacement: In this reduction, $A \to B$ by dividing input to A in the form of components and then these components can be converted to components of B.

2. Component design: In this reduction, $A \to B$ by building special component for input B that enforce properties required by A.

CIRCUIT-SAT:

This is a problem in which a boolean circuit is taken as input with single output node. And then finds whether there is an assignment of values to the circuit's input so that we get its output value as 1. This assignment of values is called satisfying assignment.

Theorem: CIRCUIT-SAT is in NP.



proof:

- Let us construct a non-deterministic algorithm which works in polynomial time.
- Then we should have choose() method which can guess the values of input node as well as output, then the algorithm says no, if we get output of Boolean circuit as 0.
- Similarly the algorithm says yes if we get output of Boolean circuit as 1.
- Now from the output of algorithm, we can guess the inputs to logic gates in the circuit.
- If the algorithm has output yes then we can say that boolean circuit has satisfying input values.
- Thus, CIRCUIT-SAT is in NP.

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 → and ↔ (if and only if).

- A Boolean formula is in Conjuctive Normal Form (CNF) if it is formed as collection of subexpressions. These subexpressions are called clauses.

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

- This formula evaluates to 1 if b, c, d are 1.

- The CNF-SAT is a problem which takes boolean formula in 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 of S.

- If all the clauses evaluate S to 1 then S is satisfied.

- Thus CNF-SAT is in NP-complete.

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 means each literal ORed to form a clause and each clause is ANDed to form Boolean formula S.

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

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 of S.
- If the S is evaluated as 1 then S is satisfied. Thus we can prove that 3-SAT is in NP complete.

0/1 knapsack problem - It can be proved as NP complete by reduction from sum of subset problem.

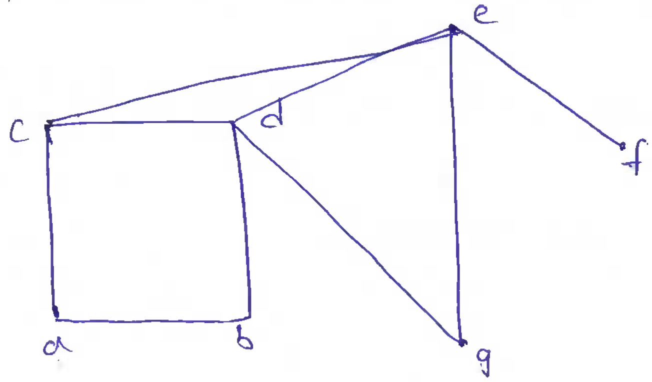
Hamiltonian cycle - It can be proved as NP-Complete, by reduction from vertex cover.

Travelling salesperson problem - It can be proved as NP-Complete by reduction from hamiltonian cycle.

Node cover problem

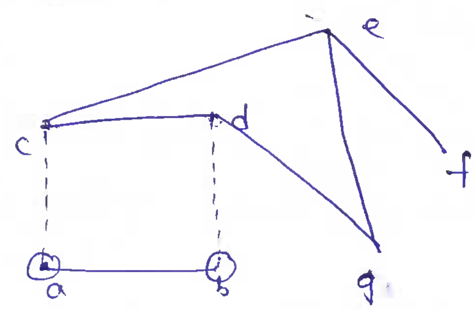
This problem is to find node cover of minimum size in a given graph. The word node cover means each node covers its incident edges. Thus by node cover, we expect to choose the set of vertices which cover all the edges in a graph.

G :-



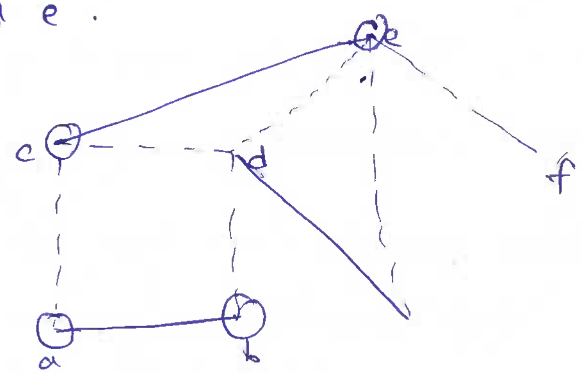
Now we will select some arbitrary edge and delete all the incident edges. Repeat this process until all the incident edges get deleted.

Step 1 : select an edge a-b and delete all the incident edges to node a & b.



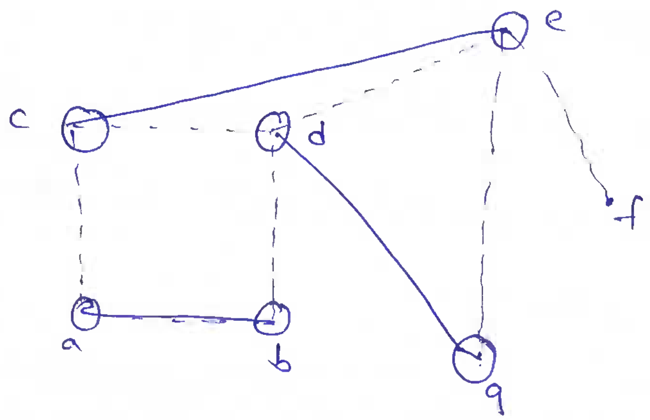
$w = \{a, b\}$

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



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

step 3: select an edge d-g. All the incident edges are already deleted.



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

$\therefore$ node cover is $\{a, b, c, d, e, g\}$

Cook's Theorem

In computational complexity theory, the cook-levin theorem also known as cook's theorem, states that the Boolean satisfiability problem is NP-complete. i.e., any problem in NP can be reduced in polynomial time by a deterministic turing machine to the problem of determining whether a boolean formula is satisfiable. (SAT).

proof:

Any instance of Boolean satisfiability problem is a boolean expression in which boolean variables are combined using boolean operators.

- An expression is satisfiable if its value results to be true on some assignments of boolean variables.

- The boolean satisfiability problem is in NP. This is because a non-deterministic algorithm can guess an assignment of truth values of variables.
- this algorithm can also determine the value of expression for corresponding assignment and can accept if entire expression is true.

Clauses	Meaning	Time
T_{ij0}	Cell i of input tape contains symbol j.	$O(P(n))$
Q_{s0}	Initial State	$O(1)$
H_{00}	Initial position of tape head	$O(1)$
$T_{jik} = T_{ij(k+1)} \vee H_{ik}$	Tape remains unchanged unless written	$O(P(n^2))$
$Q_{qk} \rightarrow \neg Q_{qk}$	one state at a time	$O(P(n))$
$H_{ik} \rightarrow \neg H_{ik}$	one read/write head position at a time	$O(P(n^2))$
$T_{ijk} \rightarrow \neg T_{ijk}$	one symbol per tape cell at a time	$O(P(n^2))$
$(H_{ik} \wedge Q_{qk} \wedge T_{ijk}) \rightarrow (H_{(i+j)(k+1)} \wedge Q_{q(k+1)} \wedge T_{ij(k+1)})$	possible transactions	$O(P(n^2))$
Disjunction of all clauses Q_F	Moving to accept State	$O(1)$.

The algorithm is composed of
- Input tape where in tape is divided in finite number of cells.
* ~~Input~~ The read/write head which reads each symbol from tape
* Each cell contain only one symbol at a time
* Computation is performed in number of states.
* The algorithm terminates when it reaches to accept state

The conjunction clauses for Boolean expressions are given in Above table.

Note that H denotes head, Q denotes states and T denotes tape.

If B is satisfiable, then there is an accepting state in the algorithm.

Thus the proof shown that boolean satisfiability problem can be solved in polynomial time. Hence all problems in NP could be solved in polynomial time.

- Hence the complexity class NP could be equal to P

NP Hard Graph problem

The strategy we adopt to show that a problem L_2 is NP-hard is :

1. pick a problem L_1 already known to be NP-hard.
2. Show how to obtain (in polynomial deterministic time) an an instance I' of L_2 from any instance I of L_1 such that from the solution of I', we can determine (in polynomial deterministic time) the solution to instance I of L_1.
3. Conclude from step (2) that $L_1 \alpha L_2$.
4. Conclude from steps (1) & (3) and the transitivity of α that L_2 is NP-hard.

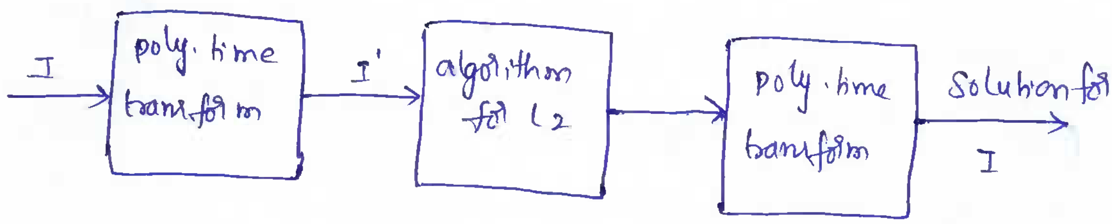


fig: Reduction of L_1 to L_2

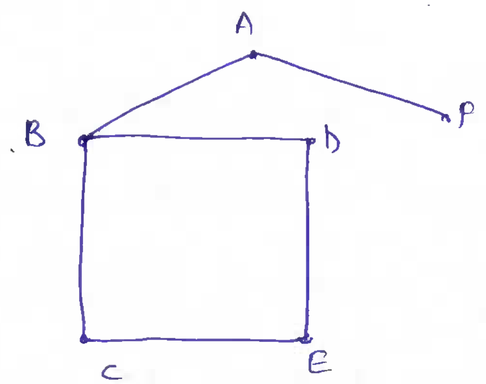
Clique Decision problem (CDP):

clique is a subgraph which is a complete graph.

- The clique decision problem (CDP) are
  1. finding maximum clique
  2. finding maximum weight clique in a weighted graph.

3. Solving the decision problem of testing whether a graph contains a clique larger than given size.

- All these problem are <u>NP-complete</u> problems



B, C, D E form a clique.

- To prove that CDP is NP hard, we will use the strategy i.e.,

i) pick up a known problem L_1 which is NP hard.
we will consider L_1 = CNF problem.

ii) we will select an instance I' from L_2 (i.e, from CDP) which can be obtained from and instance I of problem L_1 (i.e., CNF) in polynomial deterministic time, as a solution to instance I of L_1.

iii) That is $L_1 \propto L_2$ (CNF $\propto$ CDP)

iv) from this we conclude that CDP is NP hard.

Theorem: CNF satisfiability $\propto$ CDP (or) CDP is NP-complete.

proof:

Let F be a formula for CNF which is satisfiable.

$F = C_1 \wedge C_2 \wedge \cdots C_k$.

where c is a clause.

Every clause in CNF is denoted by a_i where $1 \le i \le n$.

If length of F is F and is obtained in time $O(m)$ then we can obtain polynomial time algorithm in CDP.

- let us design a graph $G = (V, E)$ with set of vertices

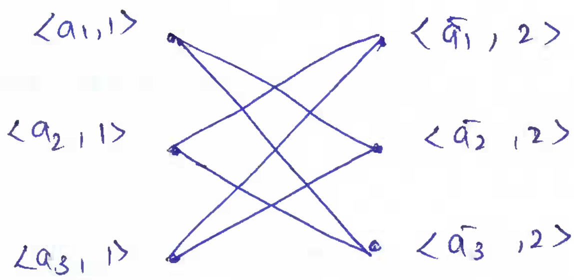
$$V = \{ \langle \sigma, i \rangle / \sigma \text{ is a literal in } c_i \}$$ and set of edges

$$E = \{ (\langle \sigma, i \rangle, \langle \delta, j \rangle) / i \neq j \text{ and } \sigma \neq \bar{\delta} \}.$$

for ex,

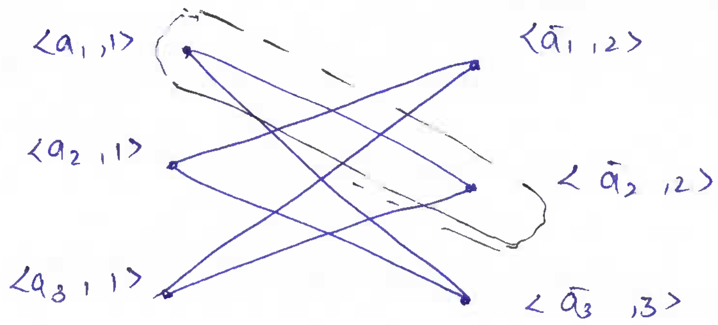
$$F = \underbrace{(a_1 \vee a_2 \vee a_3)}_{c_1} \wedge \underbrace{(\bar{a}_1 \vee \bar{a}_2 \vee \bar{a}_3)}_{c_2}$$

The graph G can be drawn as follows



The formula F is satisfiable if and only if G has a clique of size $> k$.

- The F will be satisfiable only when atleast one literal σ in c_i is true.
- The above graph has six clique of size two. F is satisfiable as well.



- A clique of size 2.

∴ As CNF Satisfiablity is NP complete. ∴ CDP is also NP complete.

Node cover Decision problem (NCDP):

Theorem: The clique decision problem $\propto$ the node cover decision problem.

proof:

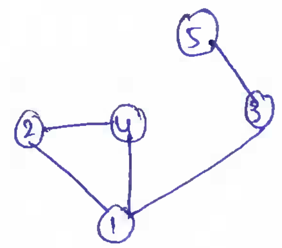
Let $G=(V,E)$ for finding the clique decision problem k be the given instance.

- we will construct complement graph G'.
- The graph G has clique of size k and then only the graph G' will have node cover of size $n-k$.

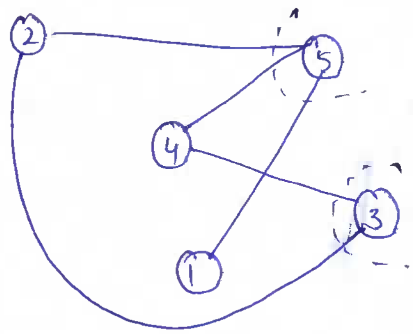
$$G' = \{V, E' \mid E' = \{a,b\}\} \text{ where } a,b \in V \text{ but } \{a,b\} \notin E.$$

for ex,

step1: Let, G be the graph.



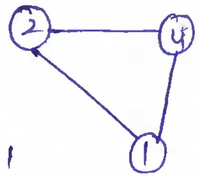
Step 2 : Let G' be the complement of graph G,



Step 3 : Then node cover for G' is = {3, 5}

Note that total number of vertices for G = G' = 5

Step 4 : The clique for graph G for k = 3 will be



Step 5 : from step 3 and step 4, n−k = node cover of G'.

i.e., 5−3 = 2 i.e., Node cover of G'.

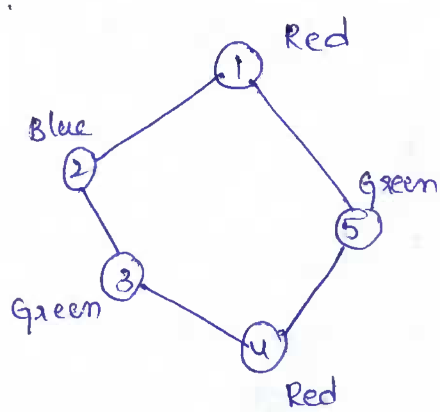
The Chromatic Number Decision problem (CNDP)

Chromatic number

Let G = (V, E) is a graph. Let f : V → {1, 2, .. m} is defined for all the vertices of the graph. This is a function used for coloring the graph. The coloring should be such that no two adjacent vertices have the same color.

for ex:-

The function f returns m = 3 for coloring this graph. The number 3 is called as chromatic number.



Determining whether the graph G can be colored by m colors is called chromatic number decision problem.

Theorem: The satisfiability with at the most 3 literals per clause $\propto$ chromatic Number Decision problem (CNDP)

Proof:

we have to prove if the formula F for CNF (with 3 literals) is satisfiable then graph G is n+1 colorable.

Let F be a CNF formula with 3 literals per clause $C_1, C_2, \ldots C_m$ be some clauses.

Let x_i be the variable in F where $1 \le i \le n$.

LHS - formula F is satisfiable

1. Let $f(y_i) = i$
2. If x_i = True then $f(x_i) = i$, $f(\bar{x}_i) = n+1$ else $f(x_i) = n+1$, $f(\bar{x}_i) = i$

~~3. If x_i = True then f'~~

3. If x_i is in C_j and $\bar{x}_i$ = True then $f(C_j) = f(x_i)$
   If $\bar{x}_i$ is in C_j and $\bar{x}_i$ = True then $f(C_j) = f(\bar{x}_i)$

RHS - i.e., G is n+1 colorable

1. y_i is assigned with color i
2. $f(x_i) \neq f(\bar{x}_i)$. That means either $f(x_i) = i$ and $f(\bar{x}_i) = n+1$ or $f(x_i) = n+1$ and $f(\bar{x}_i) = i$.

3. If $f(c_j) = i = f(x_i)$, then assign $x_i = T$. If $f(c_j) = i = f(\bar{x}_i)$

then assign $\bar{x}_i = T$.

4. For atleast one x_i, the x_i and $\bar{x}_i$ are not in c_j.

The $f(c_j) \neq n+1$.

5. If $f(c_j) = i = f(x_i)$ then $(c_j, x_i) \neq E$. If x_i is in c_j,

then c_j is true.

If $f(c_j) = i = f(\bar{x}_i)$ then $(c_j, \bar{x}) \neq E$.

If $\bar{x}_i$ is in c_j then c_j is true.

- This proves that if F is satisfiable then G is n+1 colorable.

∴ As the 3 SAT problem is NP hard CNPD is also

NP hard.

=