NP HARD & NP COMPLETE PROBLEM:

Polynomial Time:

- Linear Search : n
- Binary Search : $\log n$
- Insertion Sort : n^2
- Merge Sort : $n \log n$
- Matrix Multiplication : n^3.

Exponential Time:

- 0/1 Knapsack : 2^n
- TSP : 2^n.
- Sum of subsets : 2^n
- Graph Colouring : 2^n
- Hamiltonian Cycle : 2^n.

15/06/2023

UNIT-5

NP-Hard and NP Complete Problem:

Polynomial time (P) (Deterministic)
- → MST, Optimal Merge patterns
- → Searching
- → Sorting
- → Matrix Chain Multiplication
- → Huffmann Coding
- → Single source Shortest path.

Non-Polynomial Time (NP)
- → 0/1 Knapsack.
- → N-Queens
- → TSP
- → Hamiltonian Cycle.
- → Graph Colouring.

↓ complexity is in exponential form.

* Algorithm NSearch (A, n, key)

```
{ j = choice();          → O(1)
  if (key == A[j])
  { write(j)
    success(); }         → O(1)

  write(0);
  failure(); }           → O(1)
```

O(1) constant time algorithm

→ choice(); → returns a position which will probably have a key element from the given array.

* choice()
  success()
  failure()
} Non-deterministic functions.

→ NP Hard & NP complete provide a set of rules.

* In order to solve the exponential time algorithms in polynomial time complexity

→ Reducibility is a property which deals with if we are having one instance of one problem; if we can convert it to other problem and if we are having a solution; which satisfies all the problem.

→ Satisfiability Problem: (SAT)

Conjunctive Normal Form:

$(x_1 \wedge \bar{x}_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge \bar{x}_3)$.

↳ We need to check 2^n combinations in order to tell for which formula it is true

→ It can be defined as whenever conjunctive normal form of n boolean variables are given; find all possible combinations of $x_1, x_2 \ldots, x_n$ for which the formula is true.

* Algorithm Eval(E, n)

```
{ for i:=1 to n do
    xi = choice(T, F)
    if (E(x1, x2 ...., xn)) then
        success();
    else
        failure();
}
```

→ Algorithm NSort(A,n)

```
{ for i:=1 to n do B[i]:=0;
  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
```

```
    if B[i] > B[i+1]
        then Failure();
write (B[1:n]);
Success();
}
```

* ~~Satisfiability~~

→ NP hard & NP complete are the frameworks designed to find the algorithms for exponential time complexity problems.

→ When we are unable to write deterministic (polynomial) type algorithms we write non-deterministic algorithms.

→ Deterministic algorithms
  ↳ we know the significance of each and every step of the algorithm.

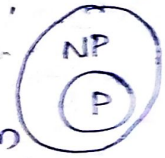
→ Non-Deterministic algorithms:
  ↳ we use non-deterministic functions such as choice(), success(); failure() whose description we are not known presently.

~~P → set of deterministic~~

P → set of deterministic algorithms which take polynomial time.

NP → set of non-deterministic algorithms which take polynomial time.

$P \subseteq NP$



* We require a base problem which helps in solving the exponential time problems which is given Satisfiability Problem.

* Satisfiability Problem: (CNF) — 2^n

Propositional Calculus formula using Boolean variables (using ∧ & ∨).

$x_i = \{x_1, x_2, x_3\}$

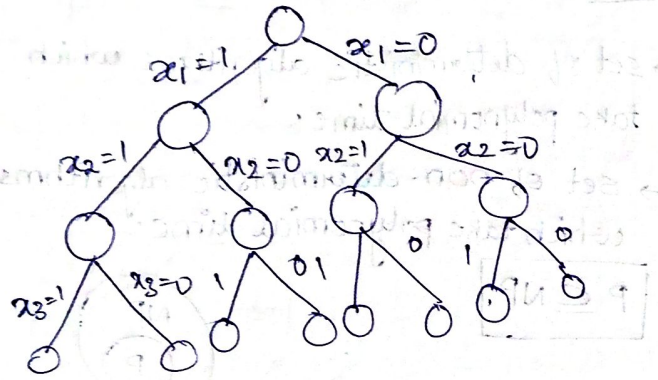
$CNF = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$

clauses C_1 clauses C_2

→ Satisfiability problem helps in finding out the values of x_i such that the CNF problem is satisfied.

if there n variables $x_1, x_2 \cdots x_n$

$\Rightarrow 2^n$ combinations & 2^n time.



→ We have to show that if satisfiability problem is solved in polynomial time then all the exponential time problems can be solved in polynomial time.

Satisfiability → NP hard;

0/1 KS, TSP, SS, GC, HC → Hard problem.

→ To show the relation blw Hard problems & Satisfiability problem we use the procedure of Reductions.

*Reduction:

→ We take an example of satisfiability problem and convert it to 0/1 Knapsack or any of the Hard problem.

Let $\propto$

I_1 0/1 Knapsack I_2

If any algorithm satisfies 0/1 KS or Let problem; it shall solve the other one too.

→ Conversion of the algorithm generated shall take polynomial time.

∴ Satisfiability is NP hard hence any problem that can be related also shall be known as NP hard.

→ Reduction property satisfies transitivity hence if.

Sat $\propto L_1 \Leftrightarrow L_1 \propto L_2 \Rightarrow$

Sat $\propto L_2$

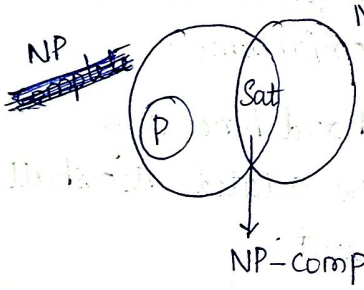
Then problems L_1 & L_2 become NP hard.

→ For the satisfiability problem are having a non-deterministi algorithm.

→ If a problem has a non-deterministic algorithm; it called NP hard & NP complete problem.

Sat problem: NP hard & NP complete:

If a problem L is formed from Sat problem then L is called NP hard and if we write a Non-deterministic algorithm then it is called NP complete.



P=NP ⇒ It is existing but we don't know.

↓

Cook's Theorem: Sat problem is in P iff P is NP.

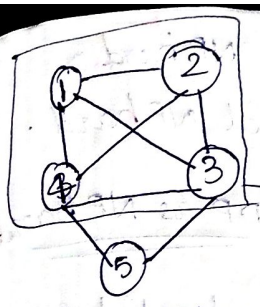
* CLIQUE DECISION PROBLEM:

↳ NP hard graph problem.

Complete graph: for every vertex there is an edge connecting to other vertices.

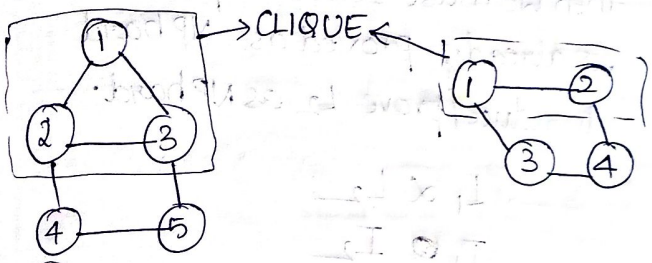
n → vertices

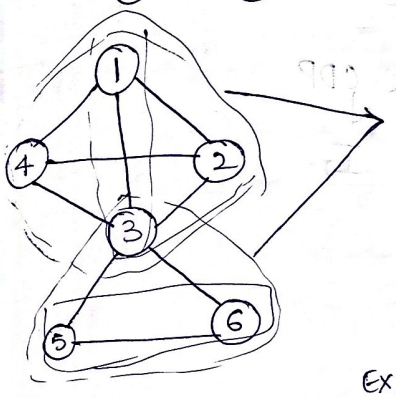
→ $|V| = n$, $|E| = \frac{n(n-1)}{2}$



* Subgraph of a graph which is → CLIQUE.

* Subgraph of any graph which is a complete graph is called clique.





CLIQUE → 4 size, 3 size, 2 size

Ex: find if a graph has clique of size k.

→ Decision Problem: Whose answer is Yes/No (T/F)

→ Optimization Problem: Whose question deals with max/min.

Ex: find the max clique size.

If a decision problem is NP hard then optimization is called NP hard.

→ Procedure to prove CDP as NP hard.

If a problem given is denoted as L_2

then we must select a problem L_1 which is already proved as NP hard to reduce/prove L_2 as NP hard.

$L_1 \propto L_2$

$I_1 \quad I_2$

Satisfiability $\propto$ CDP

↓ I_1 ↓ I_2

we must take CNF formula to reduce to CDP.

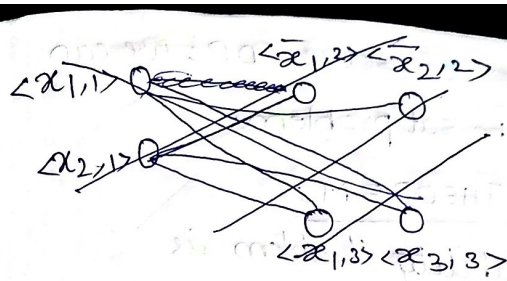
x_1, x_2, x_3

$F = (x_1 \vee x_2) \wedge (\bar{x_1} \vee \bar{x_2}) \wedge (x_1 \vee x_3)$

$C_1 \quad C_2 \quad C_3$

$F = \bigwedge_{i=1}^{k} C_i$ ∵ k = 3

we must have a clique of size 3

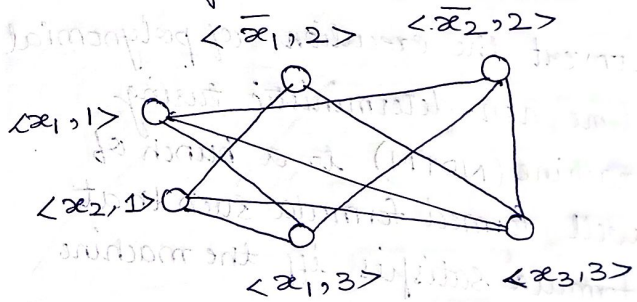


$V = \{ \langle a, i \rangle \mid a \in C_i \}$

$E = \{ \langle a, i \rangle \langle b, j \rangle \mid i \neq j \text{ and } b \neq \bar{a} \}$

→ We must not connect the vertices of similar clauses.

→ We must not connect the vertex with its negation.



→ clique of size of 2 & 3

x_1, x_2, x_3

0 1 1

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

$= 1 \wedge 1 \wedge 1 = 1$

2 HARD

nomial

Ling
Bina
Ins
Mer
Ma
ntio
0/1
T
S
Gr
th

⇒ If we can solve CDP; we can solve the sat problem.

* COOK'S THEOREM:

→ Satisfiability Problem is NP complete.

↓

SAT is a propositional logic which gives TRUE as O/P for any assignment.

3CNF → Conjunctive Normal form
→ every clause has 3 literals.

① Convert the execution of polynomial time non-deterministic turing machine (NDTM) to a bunch of well formed formula such that formula satisfies iff the machine accepts input.

② Show the sum of the lengths of a formula is polynomial in the size of the problem.

* TSP as NP-hard:

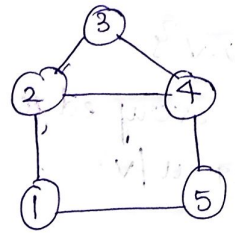
Corresponding decision problem to TSP is to determine whether the complete directed graph $G=(V,E)$ with an edge cost $c(u,v)$ has a tour cost atmost M.

→ TSP decision problem:

Hamiltonian $\propto$ TSP.

① Consider a graph $G=(V,E)$ be a instance of the Hamiltonian cycle and we need to construct an instance of TSP.

$G=$

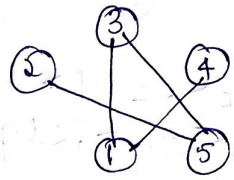


$1 \longrightarrow 5 \longrightarrow 4 \longrightarrow 3 \longrightarrow 2 \longrightarrow 1$

② To form a TSP construct a complete graph G' which is V, E'

i.e. $G'=(V,E')$

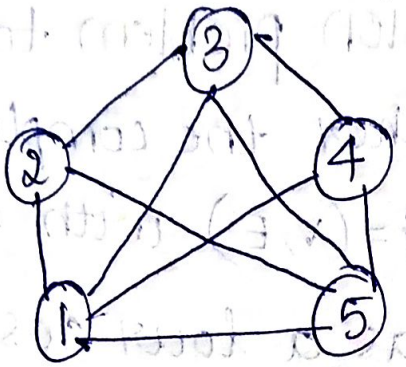
G'



(Consider the edges which are not in G that gives G')

Now complete graph is $G + G'$

$G'' =$



Approx_Vertex_cover(G)

$C = \emptyset$

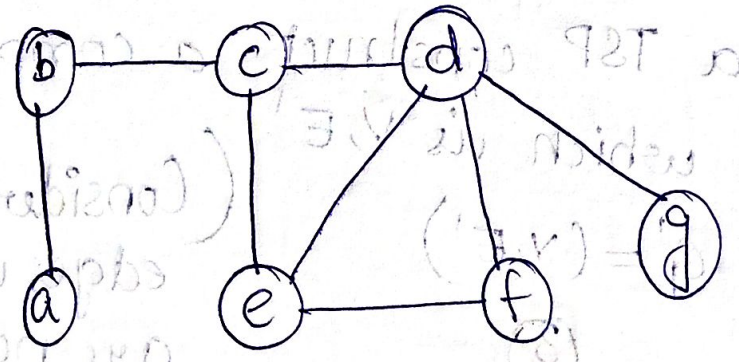
$E' = E$ while $E' \neq \emptyset$

Let $\langle u, v \rangle$ be an arbitrary edge from E'

$C = C \cup \{u, v\}$

remove v from E every edge incident on either u / v.

return C



cover: $\{\langle b, c \rangle, \langle d, f \rangle\}$

cover: $\{\langle b, c \rangle, \langle e, f \rangle, \langle d, g \rangle\}$