NP and Computational Intractability

* Till now, we have developed efficient algorithms for a wide range of problems.

* However, we haven't yet discussed about how to quantify / characterize range of problems that can't be solved efficiently.

* As earlier, we settled for the definition of efficiency as - polynomial time algorithms.

The advantage is that, it helps in proving that 'certain problems cannot be solved' in polytime time (efficient) algorithm.

* Some initial progress was made in proving that certain extremely hard problems cannot be solved by efficient algorithms.

* We donot know of polynomial time algorithms for some problems and we cannot prove that no polynomial time algorithm exists.

* A large class of problems in this "grey area" have been characterized and it has been proved that -

" A polynomial time algo. for any one of them would imply existence of a polynomial time algo. for all of them."

These are NP-complete problems.

d Discovering that a problem is NP-complete - we stop ~~for~~ searching for an efficient algorithm.

Polynomial Time Reduction

We compare the relative difficulty of different problem -

"Problem Y is at least as hard as problem X."

This brings the notion of reduction -

"We will show that a particular problem Y is at least as hard as problem X by arguing that - if we had a "black box" capable of solving Y, then we could also solve X."

Here, we add the assumption that "Y" can be solved in polynomial time. We can now ask following question -

"Can some instances of problem 'X' be solved using a polynomial no. of standard computational steps plus a polynomial no. of calls to a black box that solves Y"?

If the answer to this question is Yes then we write

$X \leq_p Y$ and read as -:

X is polynomial-time reducible to Y. or
Y is at least as hard as X.

⇒ Suppose $X \leq_p Y$ ⇒ If Y can be solved in polynomial time, then X can be solved in polynomial time.

⇒ Suppose $X \leq_p Y$ ⇒ If X cannot be solved in polynomial time then Y cannot be solved in polynomial time.

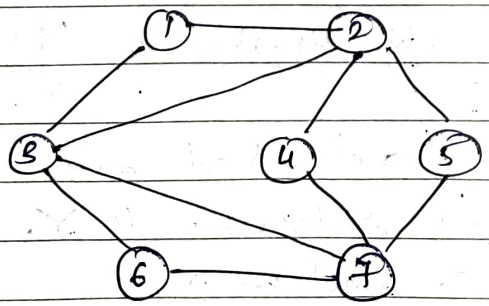
If we know a problem 'X' to be hard then 'Y' must be hard or else it could be used to solve X.

Independent Set

In Graph $G=(V,E)$, we say that a set of nodes $S \subseteq V$ is independent if no two nodes in S are joined by an edge.

It is easy to find smallest independent set in graph (ex. set of node 1 forms an I.S)

However, it is difficult to find I.S. of maximum size.



Smallest I.S. of size 1
$S=\{1\}\ \{2\}\ \{3\}\ \ldots\ \{7\}$.

I.S. of size 2
$S=\{1,4\}\ \{1,5\}\ \{1,7\}\ \ldots$

I.S of size 3.
$S=\{3,4,5\}$.

Max. size $S=\{1,4,5,6\}$.

We frame the ~~pro~~ decision problem as below -

"Given a ~~set~~ graph 'G' and a number 'k', does G contain an I.S. of size at least 'k'?"

Equivalence bet^n optimization & decision problem.

Optimization	Decision
(find max. size of an I.S.).	(whether I.S. of size 'k' is at least present).

Given algo 'A' for solving opt. problem.
we can easily use 'A' to solve decision problem, we can compare 'k' with max. size I.S.

Similarly,
Given algo 'B' to solve decision problem we can easily use 'B' to solve opt. problem
Here, we can find max 'k' for which algo 'B' gives "yes" as an o/p.

Vertex Cover

Given a graph $G=(V,E)$, we say that a set of nodes $S \subseteq V$ is a vertex cover (V.C) if every edge $e \in E$ has at least one end in S.

Here, vertices do the covering and edges are covered

Ex: $\{1,2,3,4,5,6,7\}$ - V.C - of max. size
but it is difficult to obtain V.C of smallest size
Ex: $\{2,3,7\}$

Informal Ex :- Problem Y :- How to prepare tea.
Algo :- Pour water, heat, take tea leaves etc...

Problem X : How to prepare tea if there is already water in boiler.
Algo :- Remove water from boiler and follow algo. for Y.

We reduced problem X to problem Y.

Formal Ex :- Problem Y :- Compute square of number.
(Efficient Algo is available)
Problem X :- Compute multiplication of two integer,

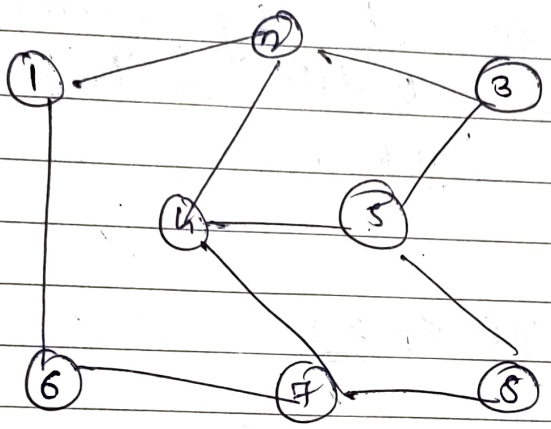
We use - $ab = ((a+b)^2 - (a-b)^2)/4$.

Thus, we have reduced multiplication (problem X) into squaring (problem Y).

We have taken an instance of X :- ab
transformed it quickly into some instance of Y :-
$(a+b, a-b)$
Solve these instance of Y by squaring algo.
finally apply another fast manipulation on result (~~add~~ subtraction (addition, division) To get "ab".

① I.S. $\leq_p$ V.C.

Consider following graph.



Prove :-
S is an I.S. if and only if V-S is a V.C.

Let 's' is I.S.
for edge $e = (u, v)$ $\Rightarrow$ Both u, v cannot be S.
So one of them must be in V-S.
It shows that every edge has at least one end in V-S so V-S is V-C.

Conversely suppose V-S is V.C.
Let u, v be two nodes in S.
If they are joined by $e = (u, v)$, Both u, v will not be in V-S contradicting our assumption that V-S is vertex cover.
It follows that no two nodes in S are joined by edge e. So, 'S' is I.S.

Ex:- I.S. of size 4 = {1, 4, 8, 3} {6, 4, 8, 3},
V.C of size 4 = {2, 8, 5, 6, 7} {1, 2, 5, 7}.

Proof for I-S ≤p V-C

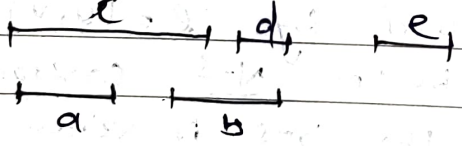
If we have a black box to solve V-C. then we can decide whether G has an I.S. of size 'k' by asking the black box whether G has an V.C. of size 'n-k'.

Proof for VC ≤p I-S

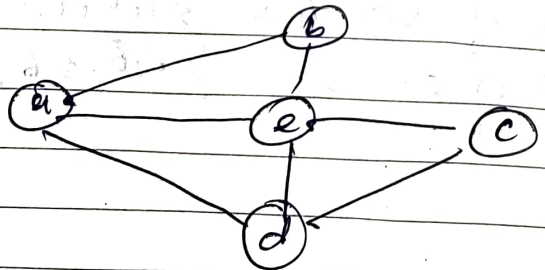
If we have an black box to solve I.S. then we can ~~as~~ decide whether G has an V.C. of size 'k' by asking the black box whether G has an I.S. of size 'n-k'.

Ex ② Interval scheduling ≤p clique.

Consider following instance of Int-scheduling



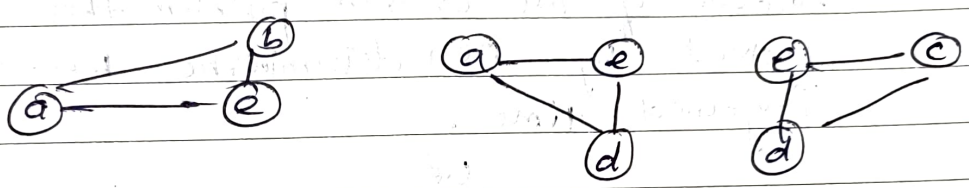
Create a graph with classes as nodes and add edges if there is no conflict in between them.



Clique ⇒ In Graph $G=(V,E)$ a subset $S \subseteq V$. is a clique, if there is an edge between every pair of vertices in 'S'.

Decision problem – Given a graph $G=(V,E)$ and an integer 'k', is there a clique of size at least k ?

How we find the clique of max. size in G.



These are nothing but solutions to Interval Scheduling problem.

Thus solving Clique problem gives us solution to I. Sch. problem.

Hence Int Sch $\leq_p$ Clique.

3 SAT (Satisfiability Problem)

Clause is defined as-

$t_1 \vee t_2 \vee t_3 \cdots \vee t_l$

$t_i \in \{ x_1, x_2, x_3, \cdots, \bar{x_1}, \bar{x_2}, \bar{x_3} \cdots \}$ — Length 'l' if contains 'l' terms.

Let we have 'k' such clauses -

$C_1, C_2, C_3, \cdots, C_k$

An assignment satisfies a collection of clauses $C_1 - C_k$ if it causes all C_i to evaluate to 1.

In other words -

$C_1 \wedge C_2 \wedge C_3 \wedge \cdots \wedge C_k$ evaluates to 1.

Ex: $(x_1 \vee \bar{x_2})(\bar{x_1} \vee \bar{x_3})(x_2 \vee \bar{x_3})$ — 2 CNF.

Decision Problem -

Given a set of clauses $C_1 - C_k$ over a set of variables $(x_1 - x_n)$, does there exist a satisfying truth assignment?.

If all clauses contains exactly 3-terms, we call this problem 3-satisfiability or 3-SAT.

3-SAT $\leq_p$ Clique

Let there as 'k' clauses -

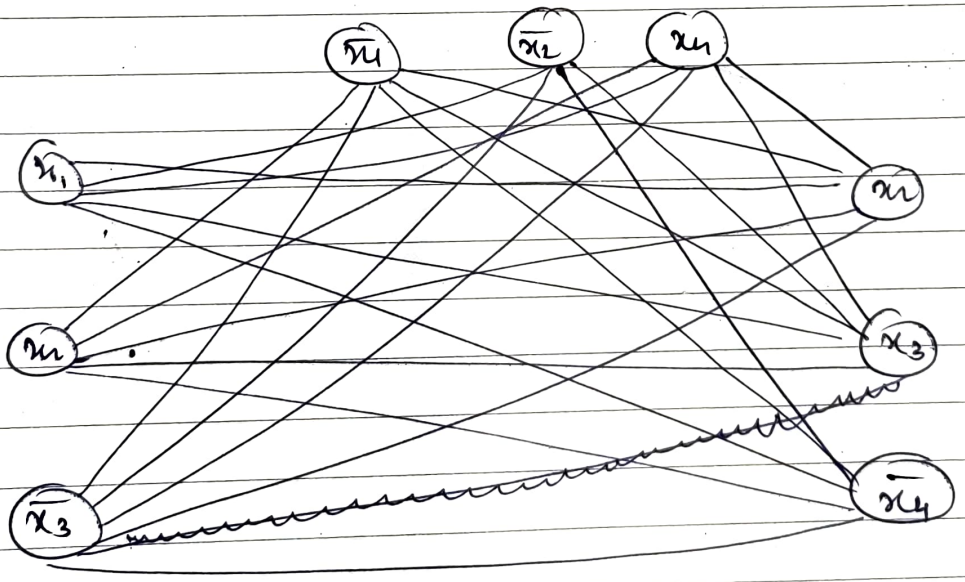
① Create 'G' of 'k' clusters with 3 nodes in each.

② Each node is labeled with literal in clause

③ Put edge between all pairs of nodes in different clusters except pair of form $(x_i, \bar{x}_i)$

④ No edge is put in b/t nodes of same cluster

$\phi = (x_1 + x_2 + \bar{x}_3)(\bar{x}_1 + \bar{x}_2 + x_4)(x_2 + x_3 + \bar{x}_4)$



G has a clique of size 'k' if ϕ is satisfiable

x_2 x_4 x_3

Make all of them to be true

Hence, solving clique solves 3-SAT.
3-SAT $\leq_p$ Clique

P, NP, NP-Complete, NP-Hard

* **P-class →**
P stands for polynomial Time.
Collection of all decision problem that can be solved by deterministic machine in polynomial time.

* **NP- Class**
NP stands for Non-Deterministic Polynomial time.
Collection of all decision problem that can be solved by non-deterministic machine in polynomial time.

OR

Decision problem that can be verified (not solve) in polynomial time.

" Class of decision problems which admit an algorithm that verifies a given solution in polynomial time".

Eg:- 100 employees needs to be paired in 200 rooms. However, some of them are not compatible. Now create pairing of employees.

* **NP- Complete**
A problem 'X' is said to be in NP-C.

If -
(1) It is NP. ($X \in NP$)
(2) It is as hard as any other problem in NP.
$Y \leq_p X$ where Y is another problem known to be NP-Complete

NP-Hard

NP-H problems are a class of problems that are at least as hard as NP-C problems, but they may not be in NP.