Module-4 NP-complete

Problem

polynomial

A problem is said to be polynomial if there exists an algorithm that solves the problem in time

$T(n) = O(n^c)$

where c is constant

- sorting $O(n \log n)$
- searching $O(n)$
- All pair shortest path (n^3)
- minimum cost spanning tree $O(E \log E) = O(E^2)$

(P class of problem)

- efficient algorithm

exponential

A problem is said to be exponential if no polynomial time algorithm can be developed for it. An algorithm solves the problem in time

$T(n) = O(2^n)$

Ex: Travelling sales

- Hamiltonian problem
- Graph colouring

(NP class (problem))

What are P, NP, NP-Hard and NP-complete problem:

P → P is a set of problem that can be solved in polynomial time using deterministic algorithm

Class P Problem

If a problem can be solved in polynomial time on a regular computer that problem is known as class P problem.

- As P class problem verified in polynomial time also, hence it is also a subset NP class problem

$x = 6$

$x = 2b$

$\Rightarrow b = 6/2 = 3$

$x = 2 \times 3 = 6$

- P class problem solve and verified in polynomial time.

-> easy to solve and easy to verify.

-> It takes polynomial time to solve and verify

-> $P \subseteq NP$ (as it is verified in polynomial time)

NP : NP is a set of ~~polynomial~~ problem that can be solved in polynomial time using nondeterministic algorithm

* P is a subset of NP

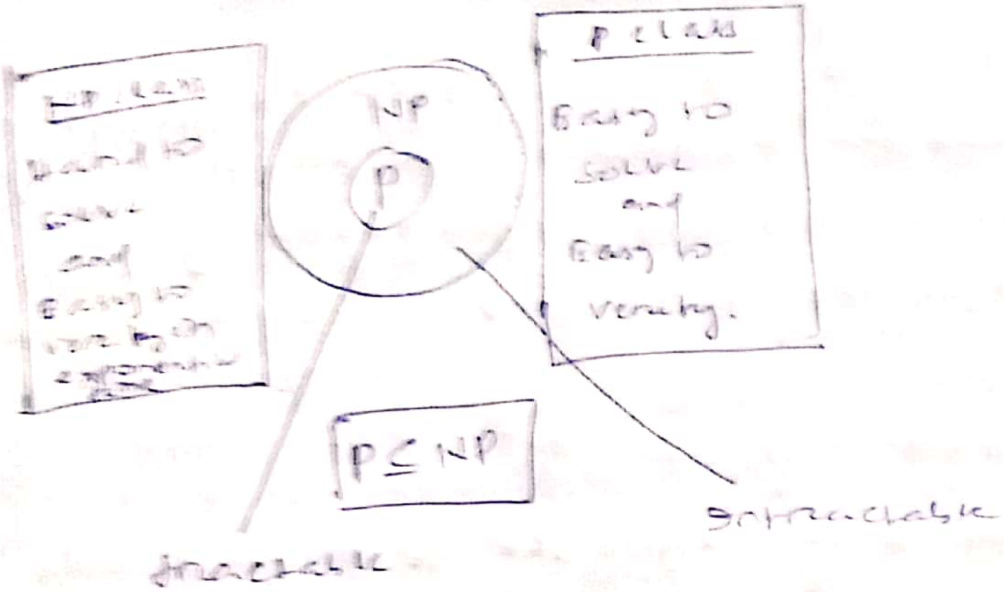
* Any problem that can be solved by deterministic machine in polynomial time can also be solved by nondeterministic machine in polynomial time

- NP problem can be solved. by a polynomial time using nondeterministic algorithm that always make a right guess among the given set of choices.
- NP class problem it is verified in polynomial time
  - ↳ Hard to solve but easy to verify.
  - → It take exponential time to solve.
- NP problem can be solved by a polynomial time using nondeterministic algorithm a magical algorithm that always make a right guess among the given set of choices.
- NP class problem it is solved in exponential time but solution verification easy but once a solution is provided it must be easy for correctness.

problem
- solution time — polynomial time
  - ↳ exponential time
- verification time — polynomial time — NP

→ The NP class problem is verified in polynomial time.

→ The P class problem not only it is solved on polynomial time but it ~~also~~ is verified also polynomial time.



Is P = NP

If you can prove P = NP then information security or online security is vulnerable to attack.

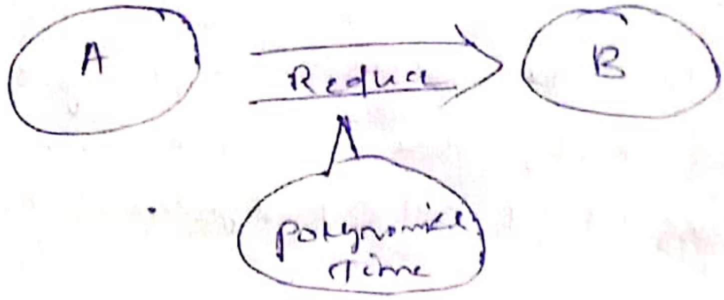
Everything become more efficient such as transportation, scheduling, understanding DNA etc.

→ All the security algorithm under NP.

→ If you prove P ≠ NP then you can prove that there are some problem that can never be solved.

→

Reduction



Let A and B are two problem then problem A reduces to problem B iff there is a way to solve A by determinist algorithm that solve B in polynomial time.

↳ Reduce A to B

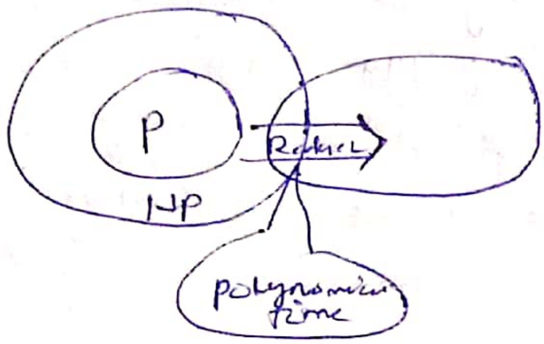
→ If A is reducible to B we denote it by $A \propto B$

Properties.

1. If A is reducible to B and B is in P then A in P
2. A is not in P implies B is not in P.

NP-Hard problem

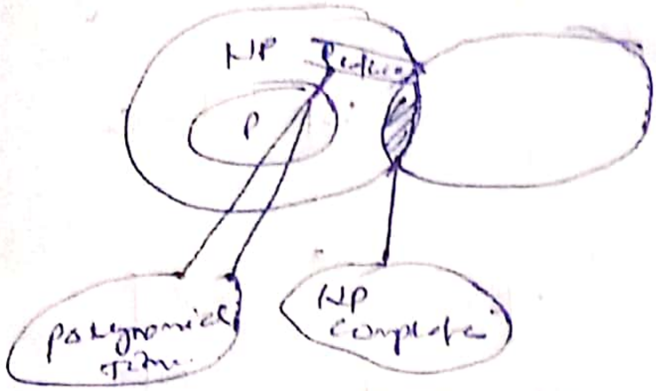
A problem is NP-Hard if every problem in NP can be polynomial reduced to it.



⇒ All the problem NP is reduced to another set 2 ~~to~~ reduction will take polynomial time

NP-complete

A problem is NP-complete when it is in NP and it is NP-Hard.



The class of problem NP-complete problem is the intersection of NP and NP-Hard class.

* NP-complete problem are decision problem
* NP-Hard problem are optimization problem
* If problem A is decision problem and problem B is optimization problem then it is possible A $\propto$ B.

Ex: knapsack Decision problem can be converted to knapsack optimization problem

→ Decision problem can be converted to optimization problem

* All the NP-complete problem are NP-Hard
* But all the NP-Hard problem are not NP-complete.

Computation complexity problem

- P-class
- NP-class
  - NP-Hard
  - NP-complete

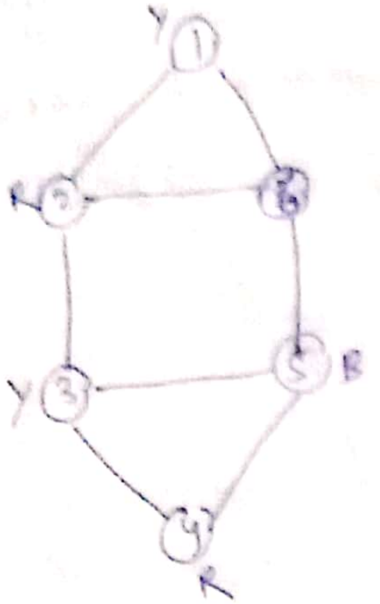
Cromatic number Decision problem

That is also called graph colouring. App^n of backtracking is graph colouring.

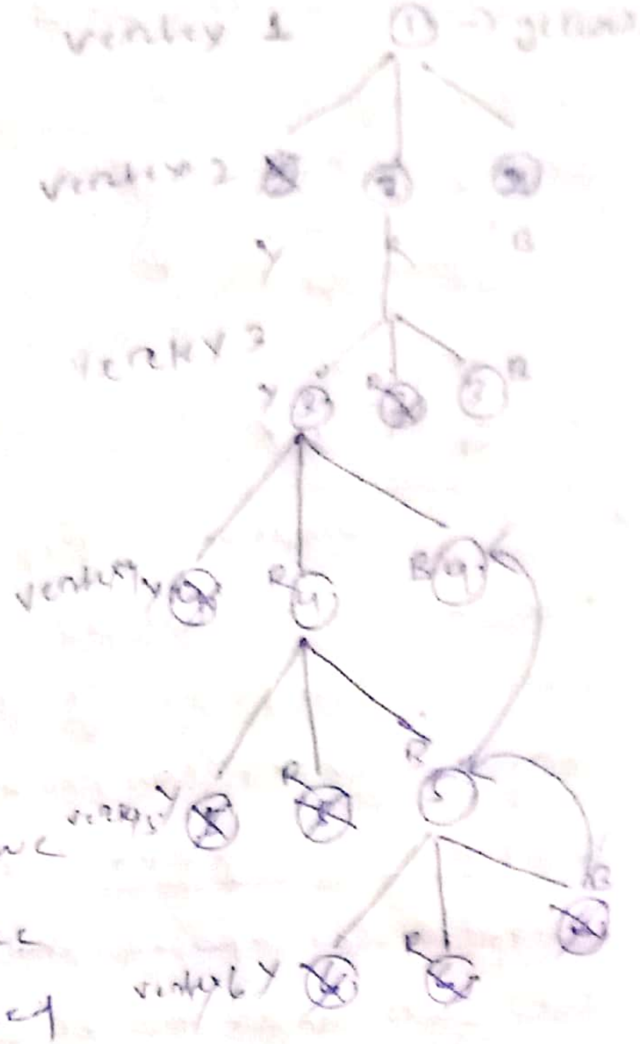
-> Graph colouring is a problem of colouring each vertex in graph in such a way that no two adjacent vertices have same colour.

Graph colouring

- m colouring Decision problem
  - ↓ Whether m colours can be used to color a graph
- m-colouring Optimization problem
  - ↓ What is the smallest no. of colour that can be used to colour a graph
  - [Cromatic Number]

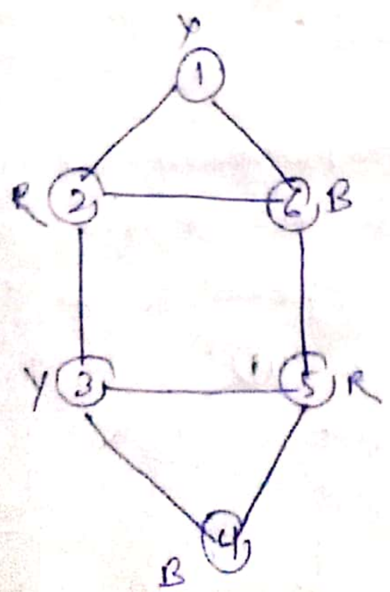


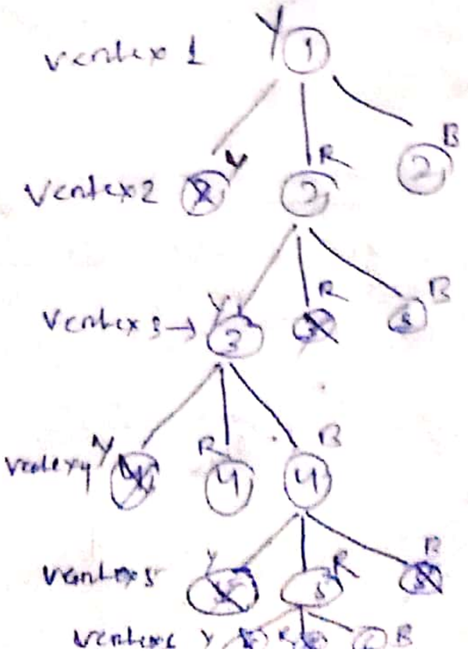
Yellow color	Red color	Blue color



* vertex 4 now we can update Blue colour from Red colour using backtracking

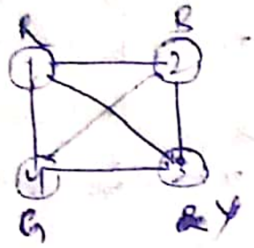
* vertex 5 now we can update red colour using backtracking





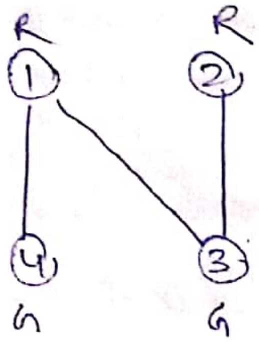
Cromatic no. of some special graph

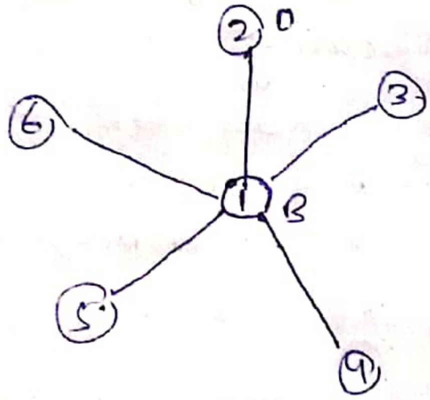
complete graph: In a undirect graph if every vertex is having edge to every other vertex.



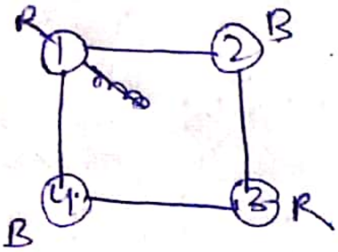
* Every complete graph with n vertices required n colors. so the cromatic no. of complete graph is same as no. of vertices in the graph.

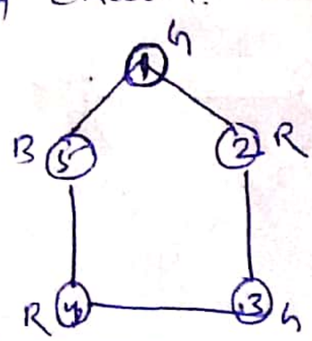
Bi-partiate graph: It is a set of graph vertices decomposed into two disjoint sets such that no two graph vertices within the same set are adjacent.





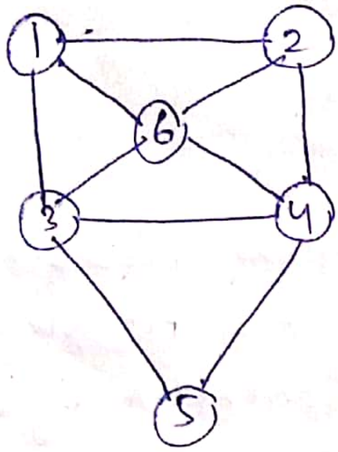
cycle graph: It is graph that consists of a single cycle of some no. of vertices connected in a closed chain.





Clique problem:

It is a complete subgraph of a graph.



1-2 is a complete subgraph. It is a clique of size 2.

1-2-3-4 is a complete subgraph. It is a clique of size 4.

1-2-3 is a complete subgraph. It is a clique of size 3.

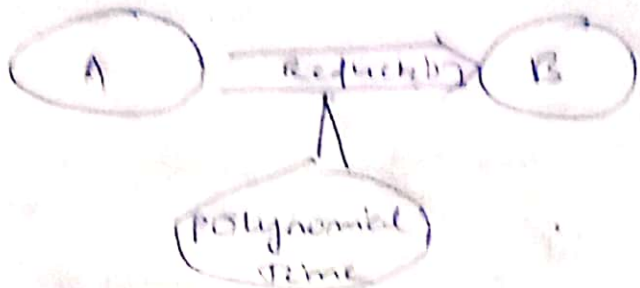
3-4-5 is a complete subgraph. It is a clique of size 2.

A clique problem

- Decision problem
  - It has truth ~~value~~ value True or False (Is there a clique of size 2, 3, 4)
- Optimization problem
  - What is the max/min clique size of a graph (What is the max/min clique size of a graph)

Scanned with CamScanner

Reduction:



Let A and B are two problems then problem A reduces to problem B iff there is a way to solve A by deterministic algorithm that solve B in polynomial time.

Properties:

1. If A is reducible to B in P then A in P.

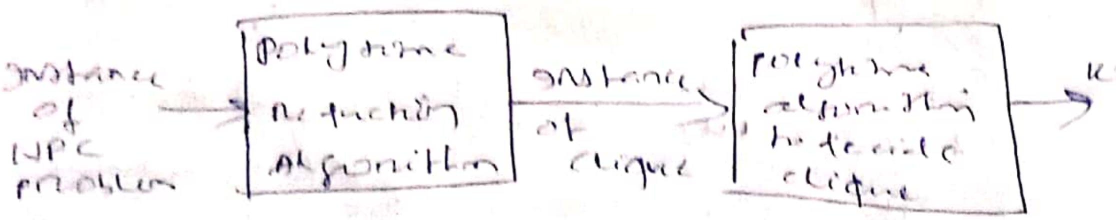
2. A is not in P implies B is not in P.

Clique is NP complete:

To prove this we need two conditions

① Clique is $\in$ NP

② A NPC problem (3CNF-SAT) can be reduced to clique decision problem



$\phi = C_1 \wedge C_2 \ldots C_k$

True $(l_1 \vee l_2 \vee l_3) \wedge (l_1^2 \vee l_2^2 \vee l_3^2) \ldots$

Scanned with CamScanner

$\phi = (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$ $k = 2$

Graph

① ~~v_i^r $v_j^s \rightarrow r \neq s$~~

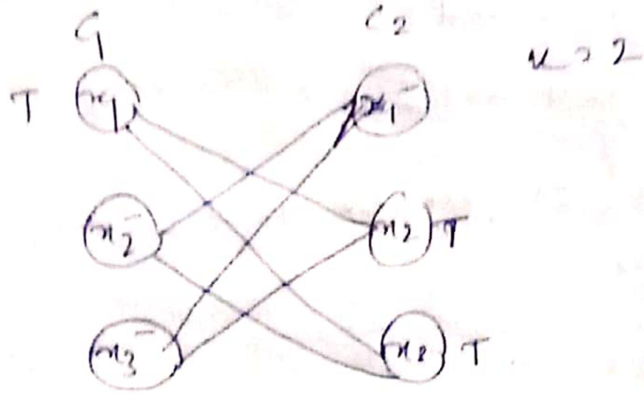
② $v_i^r = v_j^s$

Graph

① v_i^r $v_j^s \rightarrow r \neq s$ ($c(s,r)$)

② $v_i^r \neq v_j^s \rightarrow x_1 \neq \bar{x}_1$

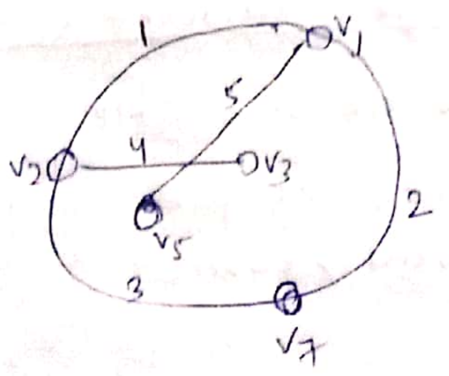
(consistency)



∴ ~~co~~ clique is NPC □

vertex cover is NP-complete

What is vertex cover problem



• vertex cover of undirected graph $G(V,E)$ is subset $V' \subseteq V$ such that

$\forall (u,v) \in E$

then $u \in V'$
or
$v \in V'$
or
both u and $v \in V'$

→ minimum vertex that cover all the edges in G.

$V = \{v_1, v_2, v_3, v_4, v_5\}$

v_1 & v_2 cover all the edges

v_1 cover 1, 5 and 2

v_2 cover 1, 4 and 3

v_1 and v_2 are the vertex that cover all the edges

$V' = \{v_1, v_2\}$

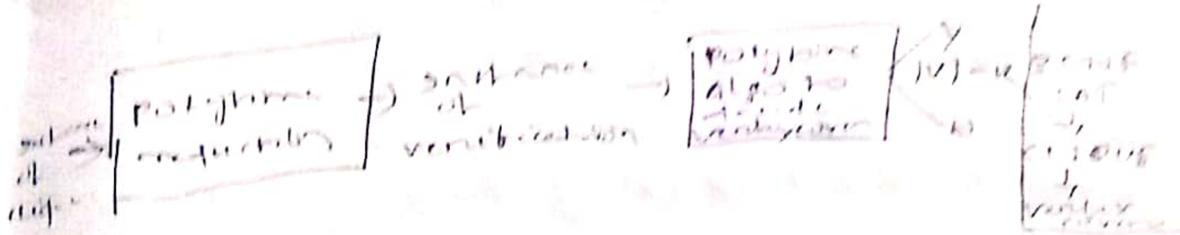
where V' is the subset of V.

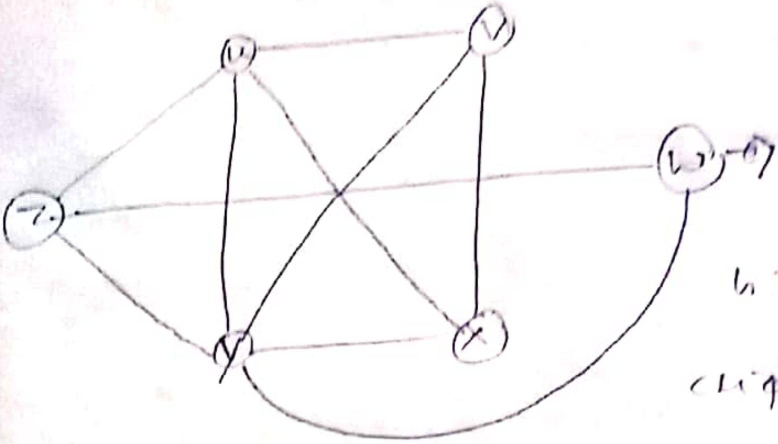
$v_2 \to v_3$ (edge)
$v_2 \to v_1$ peml

<u>Prove vertex cover problem is np.</u>

① vertex cover problem ∈ NP.

② A NPC problem clique can be reduced to vertex cover

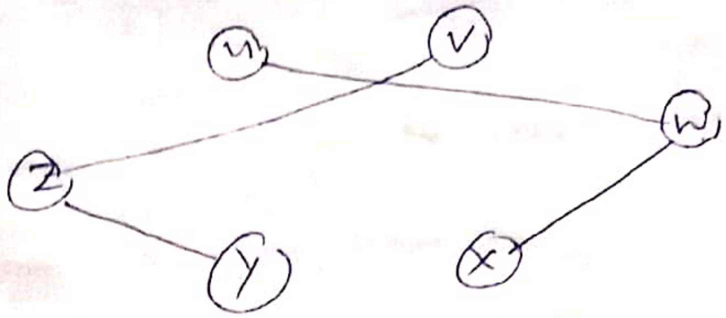




$G = \langle V, E \rangle$

clique $V' = \{u, v, x, y\}$

⇓



$G = \langle V, E \rangle$ $\bar{G}$ produced by reduction

~~clique =~~ edge

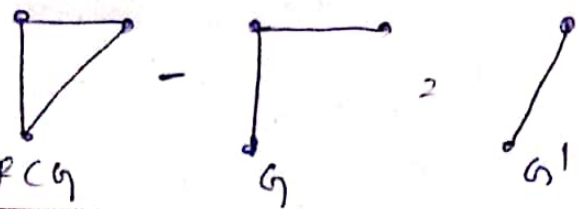
that has vertex cover

$= V - V' = \{w, z\}$

$\Rightarrow$ clique $\leq_p$ vertexcover

$[\ \because G' = RCG - G\ ]$

Ex:



$\therefore$ vertex cover = 2

$\{w, z\}$ include all edges

conversely we can say

$G' - V' \subseteq V \rightarrow |V'| = |V| - K$

$\forall u, v \in \bar{E} \rightarrow u \in V'$ and $v \in V'$