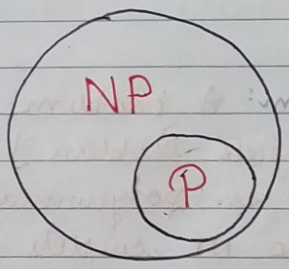
NP Hard and NP Complete

Non Deterministic Algo:

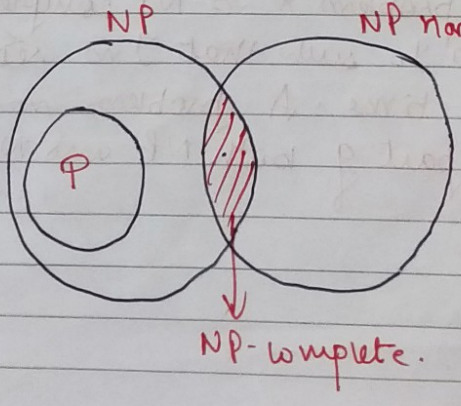
```
Algo N_Search(A, n, key)
{
    j = choice();          ---- O(1)
    if (key = A[j])
    {
        write(j);
        success();         ---- O(1)
    }
    write(0);
    failure();             ---- O(1)
}
```

O(1)

P & NP:



NP & NP-Hard & NP complete Relationship.



The class P: ⇒ The set of all problems that can be solved by polynomial time algorithms.
Eg: decision of searching, shortest path.

Note: To prove a decision problem is not in P, we need to prove it is not possible to develop a polynomial time algorithm to solve it.

The class NP: All decision problems that can be solved by non deterministic polynomial algorithms is included in class NP.

Note: All P problems are in class NP.
If we can replace the non-deterministic guessing of Stage 1 with deterministic algorithm for decision problem.

NP hard Problem: A problem X is NP-Hard if there is a NP complete problem Y, such that Y is reducible to X in polynomial time. NP-hard problems are as hard as NP complete problem.
NP hard problems need not be in NP class.

NP Complete: A problem X is NP complete if there is an NP problem Y, such that Y is reducible to X in polynomial time. A problem is NP complete if it is a part of both NP and NP-hard.

Deterministic Algorithm: A det. algorithm is the one that produces the same output for a problem instance each time the algorithm is run.

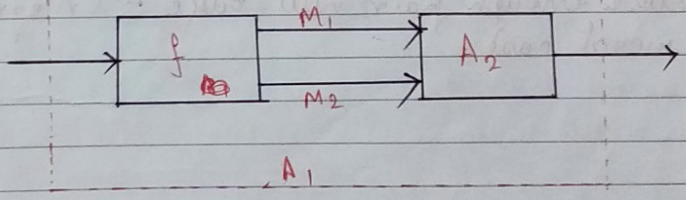
Non Deterministic algorithm: It is a two-way procedure that takes as its input an instance I of a decision problem and does the following:

Stage 1: Non Deterministic Guessing stage

Stage 2: Deterministic Guessing stage

Problem Reduction: A problem P_1 is said to be reducible to another problem P_2 in polynomial time (denoted $P_1 \leq_p P_2$) if every instance of P_1 can be reduced to an instance of P_2 in polynomial time.

Example:



A_1: An algorithm that squares a matrix

A_2: An algorithm that multiplies two matrices.

1. Closest pair problem:

i/p: A set S of n numbers, threshold t.

o/p: A pair s_i, s_j such that $|s_i - s_j| \leq t$.

Eg: $S = \{70, 24, 61, 11, 35, 42, 53, 19\}$.

Sorted Array S:

11, 19, 24, 35, 42, 53, 61, 70

8 5 11 7 11 8 9

$\min_{1 \leq i \leq n} |s_i - s_{i+1}| = 5$

⇒ Closest pair of point : (19, 24)

↳ Complexity of Sorting: $O(n \log n)$

↳ To find closest pair: $O(n \log n + n)$

* This reduction tells the fact there is an $\Omega(n \log n)$ lower bound on sorting doesn't prove the close enough pair will take $\Omega(n \log n)$ time in worst case.

2. Longest Increasing Subsequence:

T = Sort(S)

$C_{ins} = C_{del} = 1$

$C_{sub} = \infty$

Return $(|S| - \text{EditDistance}(S, T, C_{ins}, C_{del}, C_{sub})/2)$

Example: S = {7, 2, 1, 3, 4, 8, 5, 9}

T = Sort(S) = {1, 2, 3, 4, 5, 7, 8, 9}

optimal Alignment:

	7	2	1	3	4	8	5			9
1		2		3	4		5	7	8	9

operations: del(7)
ins(1)
del(1)
del(8)
ins(7)
ins(8)

Penalty = 6

Edit distance will return $\Rightarrow \frac{8-6}{2} = 5$.

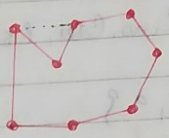
LIS: 2 3 4 5 9 of length 5.

↳ This reduction takes O(n log n) time.

↳ Edit dist. is O(|S|·|T|).

↳ Here our reduction gives us a simple but not optimal polynomial time algorithm.

3. Convex Hull: A polygon is convex if the straight line segment drawn b/w any two points inside the polygon P must lie completely within the polygon.



→ not a convex hull.

Sorting problem can be reduced to Convex hull.

↳ we can translate each number to a point by mapping x to (x, x^2).

↳ It means each integer is mapped to a point on the parabola $y = x^2$

Sort (S): ↳ for each $i \in S$ create point (i, i^2)

↳ Call subroutine convex-hull on this point set,

↳ Creating and reading of the points takes $O(n)$.

↳ The sorting has lower bound of $\Omega(n \log n)$.

If we could compute convex hull in better than $n \log n$, this reduction implies that we could sort faster than $\Omega(n \log n)$ which violates lower bound.

1. Hamiltonian cycle and TSP:

i/p: An unweighted graph G.

o/p: Does there exist a simple tour that visits each vertex of G without repetition.

* Hamiltonian cycle has some obvious similarity to the travelling salesman problem. Both problems seek a tour that visits each vertex exactly once.

HC Problem: Given a Graph $G=(V,E)$ with no edge weights determine whether there exists a HC in G.

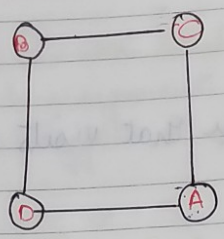
Assumption: Let there be a deterministic polynomial time algorithm for the decision version of the TSP problem.

Reduction Step: Transform $G=(V,E)$ to a weighted graph $G^* = (V, E^*)$ where $E^* = \frac{V(V-1)}{2}$ as follows

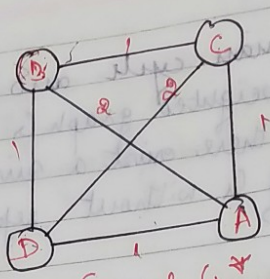
↳ Every pair in (v, v) there exist an edge in G, create an edge (v, v) of weight 1 in G^*

↳ Every pair (v, v) where NO edge exists in G, create an edge (v, v) of weight 2.

* If G has a HC, then we should be able to find a minimum weight tour of total weight equal to the number of vertices (V) in G^*
↳ weight of each edge in minimum tour will be 1.



Graph G



Graph G*

i) A - C - B - D - A.

The weight of each edge in the above minimum weight tour is 1 leading to a total weight of 4, the number of vertices

All edges of weight 1) in this minimum weight tour also exist in G.

Hence G ~~is~~ has a HC.

⇒ TSP is hard, atleast as hard as the Hamiltonian cycle

5. Independent set and Vertex Cover.

Proof: Ind. set $\leq_p$ Vertex cover.

Problem: Vertex cover.

Input: A Graph $G = (V, E)$ and integer $K \leq |V|$

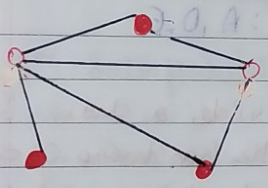
Output: Is there any subset S of at most K vertices such that every $e \in E$ contains atleast one vertex from S?

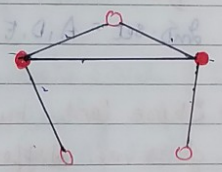
Problem: Independent set.

Input: A graph G and integer $K \leq |V|$

Output: Does there exist an independent set of K vertices in G.

Ind. set $\leq_p$ Vertex cover





$(G, K) \xrightarrow{f} (G, |V| - K)$

Vertex Cover (G, K)

$G' = G$

$K' = |V| - K$

Return the answer to Independent Set (G', K')

This simple reduction shows that hardness of vertex cover implies that Ind. set also must be hard. It is easy to reverse the roles of two problems in particular reduction.

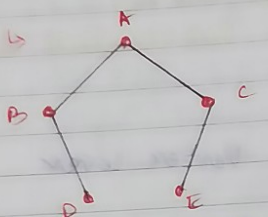
6. The Clique and Independent set.

In case of ind. set we looked for no edges b/w two vertices of S. This contrasts with clique, where we insist that there is always an edge b/w two vertices.

Independent set (G, K)

Construct a graph $G' = (V', E')$ where $V' = V$ and $\forall (i,j)$ not in E add (i,j) in E'.

Return the answer to clique (G', K)



Ind. set = A, D, E

Clique : A, D, E

⇒ These last two reductions provide a chain between three different problems. The hardness of clique is implied by the hardness of Independent set which is implied by hardness of vertex cover.

$\lfloor -2 \rfloor \quad \lfloor 3^{\vee} \rfloor$
$\Rightarrow \lfloor -3 \rfloor = 3$

7. Satisfiability: To analyze the hardness of all problems using reductions, we must start with a single problem that is absolutely, certifiably unbreakably hard. The mother of all NP complete problem is named Satisfiability.

Problem: Satisfiability

Input: A set X of Boolean variables and a set C of clauses over X.

Output: Does there exist a satisfying truth assignment for C over X - i.e a way to set the variables $x_1, x_2, x_3 \cdots x_n$ true or false so that each ~~question~~ clause contains at least one true literal/term.

Statement: Given a set of clauses $C_1, C_2, \cdots C_k$ over a set of variables $X = \{x_1, x_2 \cdots x_n\}$ does there exist a satisfying truth assignment.

3-Satisfiability: There is a special case of SAT that will turn out to be equivalently difficult and is somewhat easier to think about, this is the case in which all clauses contains exactly 3 term/literals over a set of Boolean variables X.

8. Reducing 3-SAT to Independent Set.

3-SAT $\leq_p$ Independent Set

which means 3-SAT reduces to Ind. set Problem in Independent time.

Given a 3SAT Boolean Equation we try to obtain the Independent set in polynomial time

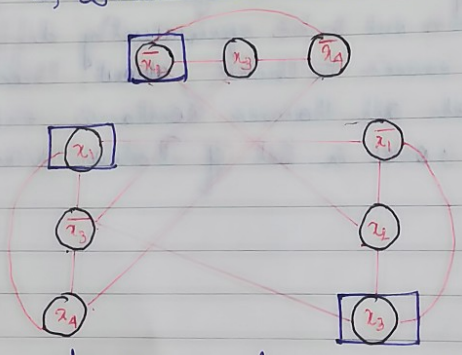
Proof: ~~3-SAT~~ let p be a ~~proposition~~ Boolean expression in 3-CNF we want to express ϕ as a graph so that if the graph has a Independent set then ϕ is satisfiable

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

Steps: i) Create a vertex for each literal

ii) Connect each literal to other two literals in the same clause

iii) Connect each literal x_i to $\bar{x}_i$



Let n be the number of variables in ϕ and m be the number of Boolean clauses. Note that ϕ can be satisfied if the graph has an Independent set of size m. The graph can be constructed in polynomial time so the reduction takes polynomial time.

9. Transitivity of Reductions:

↳ $\leq_P$ reduction is a transitive relation.

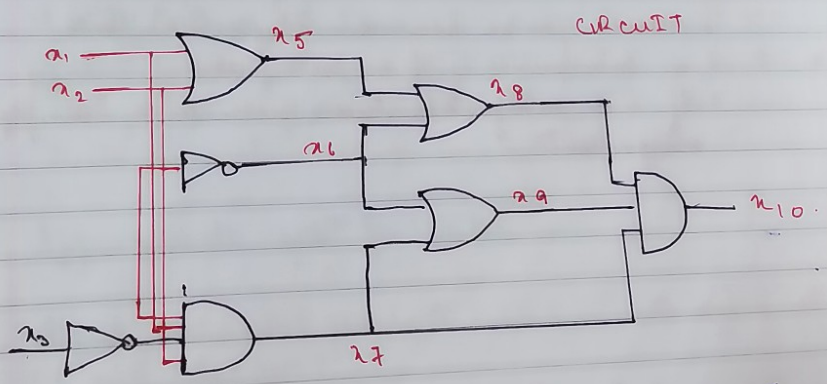
↳ If $X \leq_P Y$ and $Y \leq_P Z$ then $X \leq_P Z$

Proof: 3-SAT $\leq_P$ INDEPENDENT SET $\leq_P$ VERTEX COVER

10. Circuit SAT: Circuit satisfiability problem is the ~~circuit th~~ problem that can be reduced ~~in~~ in polynomial time to an instance of formula satisfiability

↳ Intuitive Induction:

- Look at the gate that produces the circuit output
- Inductively express each of gate's inputs as formulas
- Formula for the circuit is then obtained by writing an expression that applies the gate's function to its input formulas.



$x_{10} = (x_7 \wedge x_8 \wedge x_9) = (x_1 \wedge x_2 \wedge x_4) \wedge (x_6 \wedge x_3) \wedge (x_6 \wedge x_7)$

$= (x_1 \wedge x_2 \wedge \bar{x}_3) \wedge (\bar{x}_1 \vee (x_1 \vee x_2)) \wedge (x_4 \vee (x_1 \vee x_2)) \wedge$

$$\phi = x_{10} \wedge (x_4 \leftrightarrow \overline{x_3}) \wedge (x_3 \leftrightarrow (x_1 \vee x_2)) \wedge (x_6 \leftrightarrow \overline{x_0}) \wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9))$$

Correct Reduction: For every wire x_i in C, given a variable x_i in the formula.

→ Every gate can be expressed as
$x_0 \leftrightarrow (x_{i1} \circ x_{i2} \circ \cdots \circ x_{ij})$

→ The final formula ϕ is the AND of the circuit output variable & conjunction of all clauses describing the operation of each gate.

Conclusion:

→ Reduction can be done in polynomial time.

→ C is satisfiable if ϕ is satisfiable.

- If C is satisfiable then there is a satisfying assignment.

Approximation Algorithm

Q. What is approximation Algorithm?

→ We can call an algorithm that is not obsessed with the exact optimal solution and settles for a near optimal solution an approximation algorithm.

Q. What is approximation ratio?

→ For an algo A for the problem P has an approx. ratio of $\rho(n)$ if for any input size n, the cost C of the solution produced by the algorithm A is within a factor of $\rho(n)$ of the cost C^* of an optimal solution.

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$$

Vertex Cover Problem:

Goal: Goal is to find a vertex cover of minimum size in a given undirected graph. We call such a vertex cover an optimal vertex cover.

Approximate sol^n for vertex cover:

Input: Undirected graph $G = (V, E)$ with vertex set V and Edge set E.

Output: A vertex cover $C \subseteq V$. G which is not necessarily optimal.

Algorithm:

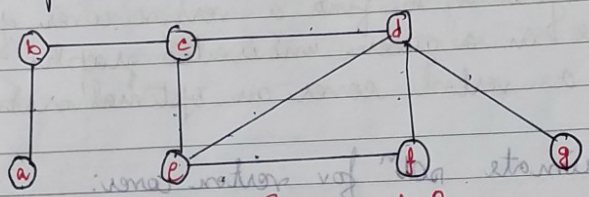
APPX-VERTEX-COVER (G)

1. $C = \phi$
2. $E' = G.E$
3. while $E' \neq 0$
4. let (u, v) be an arbitary edge of E'
5. $C = C \cup \{u, v\}$
6. remove from E' every edge incident on either u or v
7. return C.

→ This algorithm returns a vertex cover whose size is guaranteed to be no more than twice the size of an optimal vertex cover.

Time complexity: $O(V+E)$ using adjacency list.

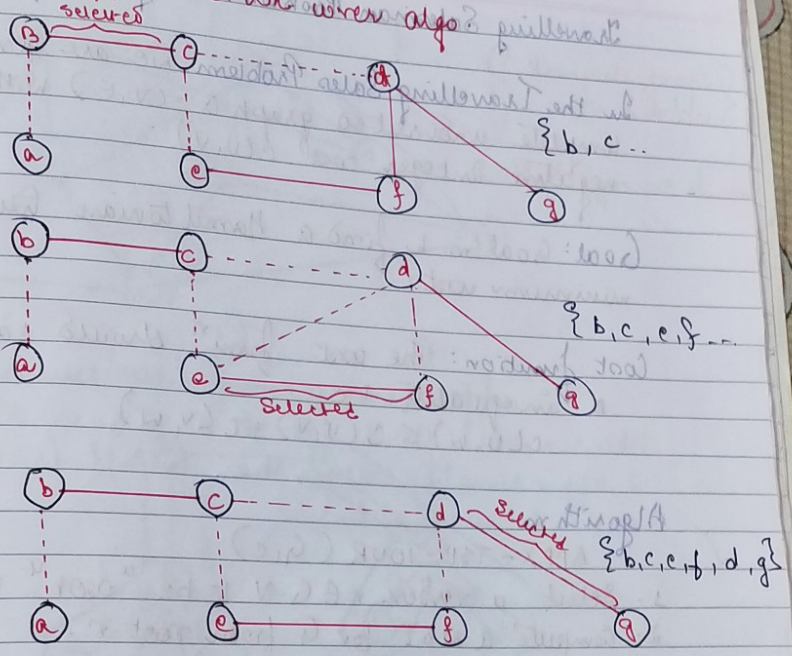
Example.



optimal soln: $\{b, c, d\}$

{Appx}

A/c to Appx Vertex cover algo:



Theorem: It is a polynomial time 2- appx algorithm.

Proof: Ranga Slide (Pg 9)

Travelling Salesman Problem using Triangle Inequality

In the Travelling Sales Problem we are given a complete undirected graph $G = (V, E)$ with non-negative integer cost $c(u, v)$.

Goal: Goal is to find a Hamiltonian tour with minimum cost.

Cost function: The cost fun^n should satisfy the triangle inequality i.e

$$c(u, w) \leq c(u, v) + c(v, w)$$

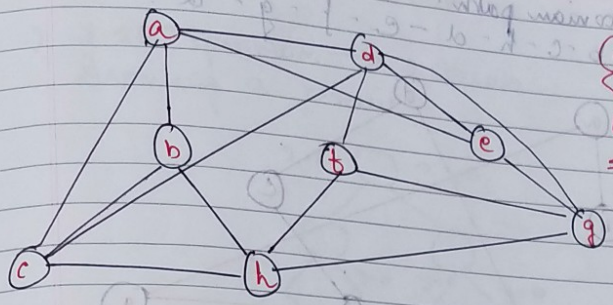
Algorithm:

APPX-TSP-TOUR (G, c)

1. Select a vertex $r \in G.V$ to be "root" vertex.
2. Compute a MST for G from root r using MST-PRIM (G, c, r).
3. Let H be a list of vertices, ordered according to when they are visited first in preorder tree walk of T.
4. Return Hamiltonian cycle H

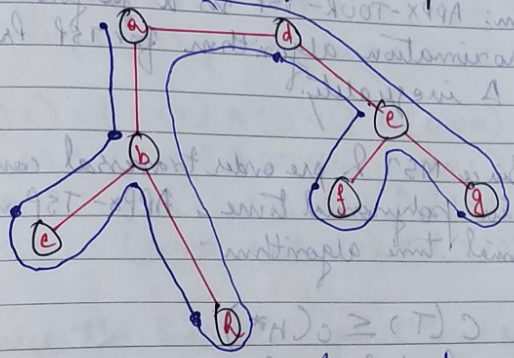
Time Complexity: $\Theta(|V|^2)$

Eg: Tsp a tour.



{ Complete graph with 8 vertices
Edges not shown

→ Construct MST from given Graph



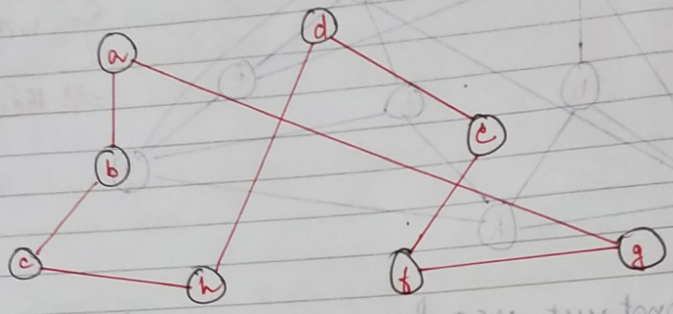
→ Showing pre order Traversal in above MST

a-b-c-b-h-b-d-e-f-e-g-g-dc-d-a.

→ Hamiltonian path: Delete vertex which are visited more than ~~once~~ once.

a-b-c-h-d-e-f-g-a.

Hamiltonian path:
a - b - c - h - d - e - f - g - a



Theorem: APPX-TOUR-TSP is a polynomial time 2-approximation algorithm for TSP problem with Δ inequality.

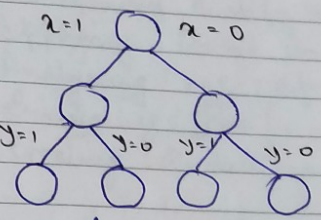
Proof: Since MST & pre order traversal can be found in polynomial time, APPX-TSP is a polynomial time algorithm.

$$c(T) \leq c(H^*)$$

Descriptive Proof Pg (16).

Prove that 2SAT Problem is a P Problem:

:) 2 SAT problem ⟶ 2 literals.
Drawing a circuit:



Suppose we have a boolean function:
$f(x,y) : (x \vee y) \wedge (\bar{x} \vee y) \wedge (x \vee \bar{y})$
Checking by traversing circuit using DFS:

$f(1,1) = (1 \vee 1) \wedge (0 \vee 1) \wedge (0 \vee 0) = 0$
$f(1,0) = (1 \vee 0) \wedge (0 \vee 0) \wedge (0 \vee 1) = 0$
$f(0,1) = (0 \vee 1) \wedge (1 \vee 1) \wedge (1 \vee 0) = 1$
$f(0,0) = (0 \vee 0) \wedge (1 \vee 0) \wedge (1 \vee 0) = 0$

Hence the boolean function is satisfiable.
Since traversal was done using DFS which takes polynomial time so 2SAT is also a polynomial class problem.