P & NP - Completeness

P : polynomial Time Solvable problems

Def: A problem is polynomial-time solvable if There is an algorithm that correctly Solves in $O(n^k)$, for some constant k.

Where n = input size

The Class P:

The set of polynomial-time problems.

Examples:

class P { Sorting algorithms already seen
~~Single Short~~
Finding minimum spanning tree

etc.

But Does the following belong to the class P?

① Cycle-free shortest path in graph with negative costs

② knapsack
[running time = $O(nW)$
but input size $\propto \log(W)$.]

$W = 2^b$
$O(n2^b)$.

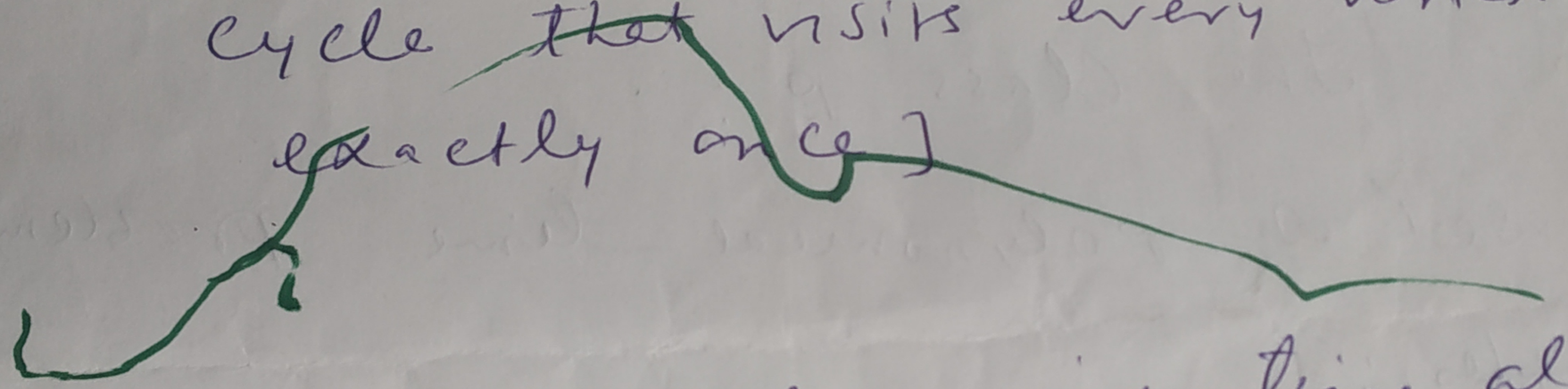
NP - Completeness

→ rough litmus test for Computational tractability

Traveling Salesman problem (TSP)

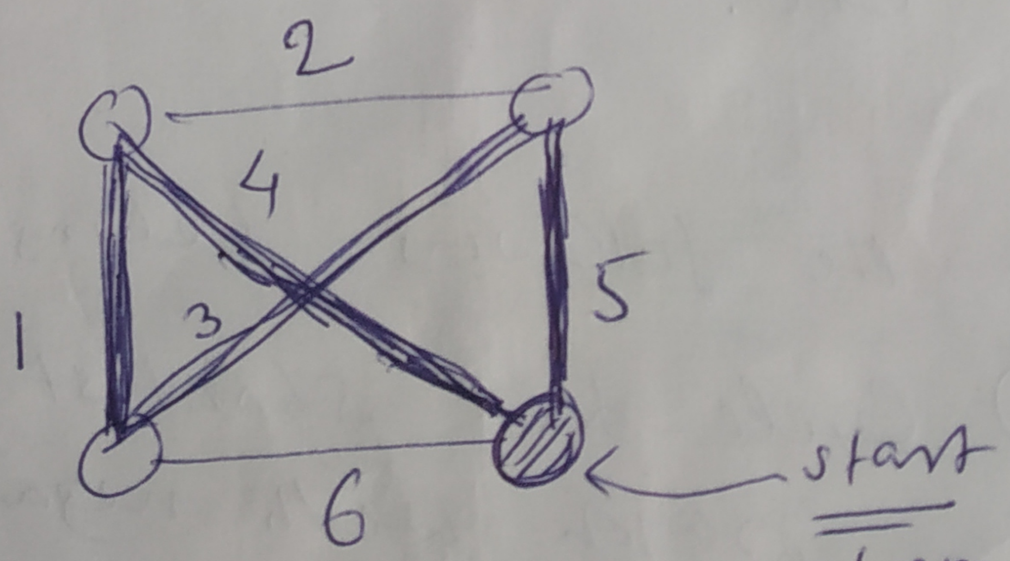
Input: Complete Undirected graph with nonnegative edge costs.

Output: a min-cost tour [i.e., a cycle ~~that~~ visits every vertex exactly once]



There is no polynomial time algorithm for TSP. problem.

Ex.



~~Optimal~~ Optimal = 13

Reductions & Completeness

Reduction

Def (informal): problem Π_1 reduces to problem Π_2 if:

Given a polynomial time subroutine for Π_2, can use it to solve Π_1 in polynomial time.

Ex. Detecting a cycle reduces to DFS
Computing median reduces to Sorting.

- if Π_1 reduces to Π_2
  - if Π_1 is not in P, then neither is Π_2, that is, Π_2 is at least as hard as Π_1.

Completeness

Def: Let C = a set of problems

The problem Π is C-Complete

if (i) $\Pi \in C$ (ii) everything in C reduces to Π.

$\Rightarrow$ Π is the hardest problem in all of C

Halting problem

Given a program and an input for it, will it ~~[illegible]~~ eventually halt?

Fact: (Turing '36) no algorithm, however slow, solves the halting problem.

The Class NP

↳ N stands for nondeterministic

Def:

A problem is in NP if

(1) Solutions always have length polynomial in the input size

(2) Purported solutions can be verified in polynomial time.

Examples –

Is there a TSP tour with length ≤ 1000?

NP – Complete

Def:

A problem Π is NP-Complete if

– (I) $\Pi \in NP$

(II) Π is NP-hard

What is NP-hard?

NP-hard

Cook's theorem: if there exists a polynomial time algorithm for SAT (~~Satisfiable~~ Satisfiability problem), then there is one for all problems in NP.

NP-hard

A problem is called NP-hard if it has the above property, that is, if it has a polynomial time algorithm then ~~Exist~~ is one for every problem in NP.

What is SAT (Satisfiability problem)?

To know about SAT, we need to know CNF (Conjunctive normal form).

CNF:
A boolean formula ϕ in CNF is an AND of clauses

$$C_1 \wedge C_2 \wedge C_3 \cdots \wedge C_k$$

($C_1, C_2, C_3, \ldots, C_k$: clause)

formula evaluates to T iff every clause evaluates to T.

boolean formula is in CNF is said to be satisfiable if there exists an assignment to the variables such that the formula evaluates to T.

[* **Clause**: OR of literals.

literal: $x, \bar{x}$

Ex. $(x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \ldots x_t)$ is a clause.]

A clause becomes false if all literals are false otherwise true.

SAT problem

I: boolean formula f in CNF

Q: is f satisfiable?

SAT $\in$ NP (purported Solution)

① A proof is an assignment to the variables.

② The purported Solutions can be verified in polynomial time.

SAT is NP-hard problem.

because

If there exists a polynomial time algorithm for SAT then there is one for all problems in NP.

NP-Completeness

A Problem Π is NP-Complete if

(1) $\Pi \in NP$

(2) Π is NP-hard.

(~~Over~~ Π is called NP-hard if every problem in NP is polynomial time reducible to Π)

(1) This implies that (1) a problem (NP) is in NP may not be NP-hard

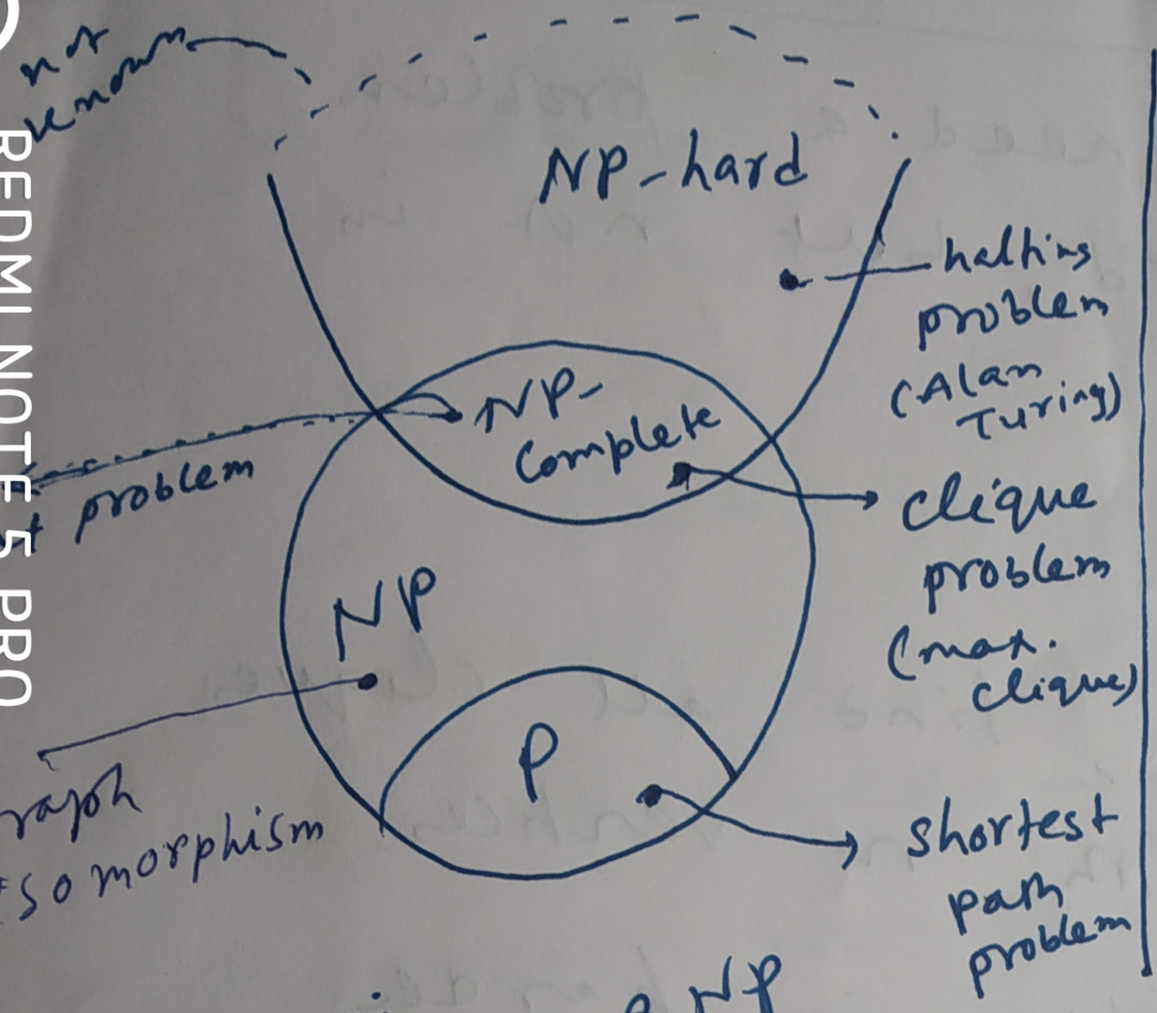
or

(2) a problem which is NP-hard may not be in NP

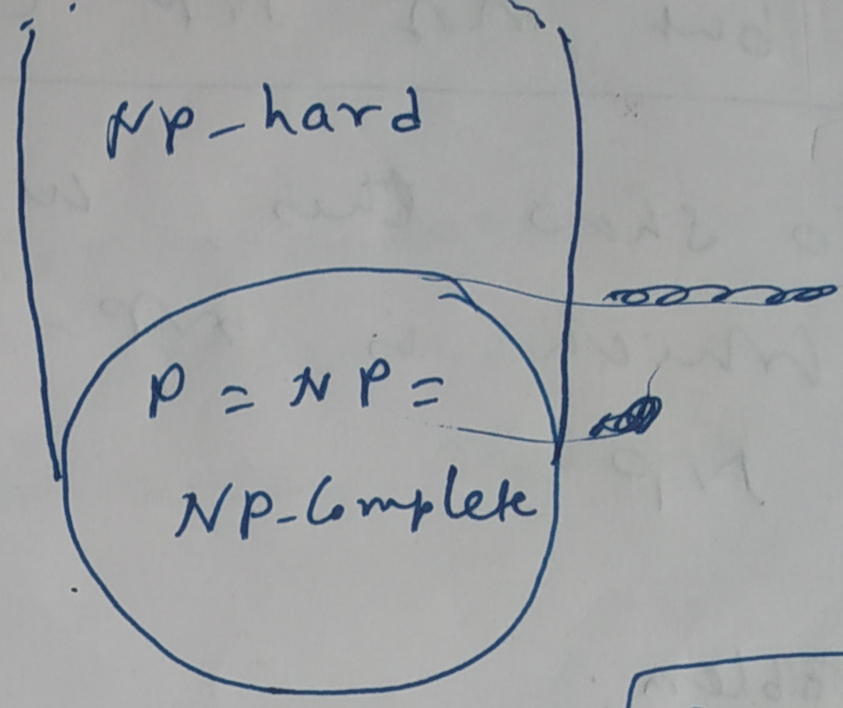
(~~i.e.~~ if we can find a problem in P such that the problem

(There may exist a NP-hard problem which is in P. if it is, NP = P).

Venn diagram for ~~NP~~ P, NP, NP-hard & NP-Complete



Assumption: $P \neq NP$



Assumption: $P = NP$

Graph Isomorphism is in NP but neither in P nor NP-complete

NP-hard problems (informally):

a class of problems at least as hard as the hardest problems in NP.

(For example: halting problem is even harder than any known NP-complete prob)

NP-intermediate problem:

problems that are in NP, but ~~not~~ neither in P nor NP-Complete. Ladner's theorem shows that such problem exists. such problem: graph is

Can a problem be in NP but not NP-complete?

To show this, we need a problem which is NP-hard, but not in NP

problem:
Given some n, find all cliques in all graphs with n vertices.

clearly this problem is harder than the general clique problem (i.e: find the largest clique given a graph of n vertices & m edges).

since we can solve this problem, we can solve the previous clique problem

Answer to this problem is actually all subsets of n vertices which form cliques.

But also note that to verify that we have the correct answer, we have to check that we have all subsets which form cliques.

So, this problem is NP-hard, but not in NP.

Graph Isomorphism

Isomorphism:

a one-to-one correspondance between two sets that preserves binary relationships between elements of the sets.

For example, a set of natural numbers can be mapped on to the set of even numbers by multiplying each natural number by 2.

Graph isomorphism problem

To determine whether two finite graphs are isomorphic.

Def:

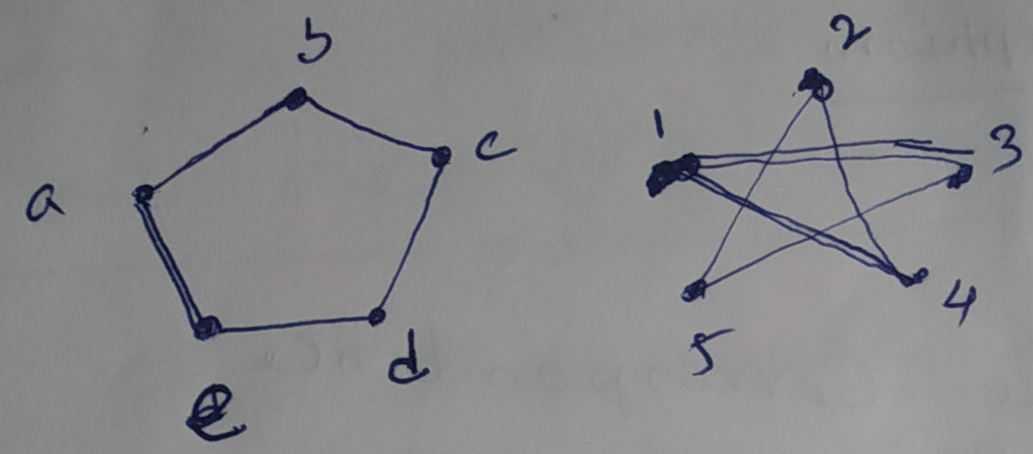
Two graphs $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ is Isomorphic

Bijection $\phi : V_1 \longrightarrow V_2$

$(u, v) \in E_1 \Leftrightarrow (\phi(u), \phi(v)) \in E_2$

(Informally, whether two graphs are ~~Same~~ structurally similar or not).

Ex.



Degree of each vertex $\in G_1$ is 2

Degree of each vertex $\in G_2$ is 2

$\phi(a) = 1$

$\phi(b) = 3$
$\phi(e) = 4$
$\phi(c) = 5$
$\phi(d) = 2$
} Since a is connected to b & e
(Since b is connected to c)
(d is connected to e and c
2 is connected to 4 and 5)

bijection or mapping.

General:

Graph Isomorphism problem is in NP because it can be verified in polynomial time.

Proof is difficult.

This is not a NP-Complete problem.

& it is not in P.

This is problem which is known NP-intermediate problem.

(Researches are working on this problem).

Halting problem

(unsolvable)

Halting problem is not NP- Complete but np-hard

There exists no algorithm to solve this problem.

So, it is not in NP

So, it is not NP- Complete.

Show that Halting problem is NP- hard.

To show it, we will show that Satisfiability problem is a halting (NP- hard) problem.

Let us Construct an algorithm A whose input is boolean formula X Suppose X has n variables Algorithm A tries out all possible 2^n possible truth assignments and verifies if X is satisfiable

- if X is satisfied, A stops (halts)
- if X is nA satisfiable, A enters an infinite loop.
- Hence A halts on input iff X is satisfiable.

- if we have a polynomial time algorithm for the halting problem, Then we could Solve the satisfiability problem in polynomial time using A and X as input to the algorithm for the halting problem

Thus the halting problem is NP-hard problem which is not in NP.

So, it is not NP-Complete.

even it is harder than ~~known~~ SAT ~~NP-hard~~, So it is NP-hard.

It is not in NP because input cannot be verified in poly-time.

Note: Why halting problem is not in NP?

Ans: all problems in NP are decidable but the halting problem is undecidable.