$P$, $NP$, and $NP$-complete

**Definition**

$P = \{ \text{decision problems solvable in poly. time} \}$

We will study what is/is not in this class.

Careful of

- machine model
- bit complexity
- input size
- # bits

Recall from last day:

$A \leq_p B$ for problems $A, B$ "$A$ reduces to $B$"

means: we can use a poly. time algorithm for $B$

to make a poly. time algorithm for $A$. 
There is a large class of decision problems not known to be in \( P \) but all equivalent in the sense that \( A \leq_p B \) for all \( A, B \) in the class. (recall definition of \( \leq_p \)) i.e. poly. time alg. for one yields poly. time alg. for all.

A few problems in the class:
- Hamiltonian path / cycle.
- TSP - given edge weighted graph, number \( k \), is there a TSP tour of weight \( \leq k \)?
- IND. SET - given graph, number \( k \), is there an ind. set of size \( \geq k \)?

Common feature: if the answer is YES there is some succinct info. to verify it, "certificate"(in particular, the TSP tour, the ind. set)
Contrast this with NO answer.
A verification alg. takes input + certificate and checks it.

Definition Alg. A is a verification alg. for problem the decision problem \( X \) if
- A takes two inputs \( x, y \) and outputs YES or NO
- for every input \( x \) for problem \( X \), \( x \) is a YES for \( X \) iff there exists a \( y \) “certificate” s.t. \( A(x, y) \) outputs YES.

Furthermore, A is a polynomial time verification alg. if
- A runs in poly. time
- there is a polynomial bound on the size of the certificate, i.e.,
  \[ \forall x, \text{ } x \text{ is a YES input for } X \text{ iff } \exists y \text{ with } \text{size}(y) \leq (\text{size}(x))^k \text{, const.} \text{ s.t. } A(x, y) \text{ outputs YES} \]

\[ \text{NP = } \exists \text{ decision problems that can be verified in polynomial time?} \]
- i.e. have poly. time verification algorithms “non-deterministic polynomial time”

Example: Subset Sum \( \in \text{NP} \)
- Given numbers \( w_1, \ldots, w_n \) and \( W \)
- is there a subset \( S \subseteq \{ 1, \ldots, n \} \) s.t. \( \sum_{i \in S} w_i = W \)
Certificate is $S$

Verification alg: check that $\sum w_i = W \in S$

Poly. time

? Is there a poly. time verification alg. for NO answers?

What could you give to verify that no subset has sum $W$?

Open.

Example TSP (decision version) $\in$ NP

Given graph $G$, weights on edges, number $k$,
does $G$ have a TSP tour of length $\leq k$?

certificate: the tour, i.e. permutation of vertices

Poly. time verification alg:

- check it's a permutation
- check that edges exist
- check that $\sum$ edge weights in tour $\leq k$

$\text{coNP} = \{ \text{decision problems where the NO instances can be verified in poly. time} \}$

e.g. Primes: given number $n$, is it prime?

Primes $\in$ coNP

Easy: to verify $n$ is not prime, show natural numbers $a, b \geq 2$ s.t. $a \cdot b = n$

In fact Primes $\in$ NP 2002
OPEN QUESTIONS

1. $P = \ ?$ $NP$
   \text{worth } \$1 \text{ million}

2. $NP = \ ?$ $coNP$

3. $P = \ ?$ $NP \cap coNP$

Properties

- $P \leq NP$, $P \leq coNP$

- any problem in $NP$ can be solved in time $O(2^{nk})$ by trying all certificates one by one

\[ \text{https://en.wikipedia.org/wiki/P\_versus\_NP\_problem} \]
Definition: A decision problem $X$ is NP-complete if
1. $X \in NP$
2. For every $Y \in NP$, $Y \leq_p X$

i.e., $X$ is [one of] the hardest problems in NP.

Two important implications of $X$ being NP-complete
1. If $X \in P$, then $P = NP$
2. If $X$ cannot be solved in poly-time, then no NP-complete problem can be solved in poly-time
3. If $X \in coNP$, then $NP = coNP$ (this needs proof)

The first NP-completeness proof is difficult

Subsequent NP-completeness proofs are easier because $\leq_p$ is transitive.

So to prove $Z$ is NP-complete, we just need to prove $X \leq_p Z$ where $X$ is a known NP-complete problem.
Summary: To prove $Z$ is NP-complete
1. prove $Z \in \text{NP}$
2. prove $X \leq_P Z$ for known NP-complete problem $X$

Our first NP-complete problem: Circuit Satisfiability
[proof later - also definition]

2nd NP-complete problem: Satisfiability
[proof later, but will define the problem now]

Satisfiability
Input: a Boolean formula made of Boolean variables, $\land$ "and", $\lor$ "or", $\neg$ "not"

E.g. $\neg(x_1 \land x_2) \lor (x_3 \land (x_5 \lor \neg x_4))$

Question: Is there an assignment of True/False to the variables to make the formula True?

Ex. Satisfiability $\in \text{NP}$.  
Sat is NP-complete, even the special case of "CNF" - Conjunctive Normal Form

E.g. $(x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor x_4) \land (x_3 \lor x_4 \lor \neg x_5)$

clause is $\lor$ of literals

formula is $\land$ of clauses
In fact it's still NP-complete when all clauses have 3 literals — 3-SAT
but 2-SAT is in P

3-SAT
Input: A Boolean formula that is an Λ of clauses, each clause an Λ of 3 literals, each literal a variable or negation of variable.
Question: Is there an assignment of True/False to variables that makes the formula True.

[[3-SAT is NP-complete [pf. later]].

**Ind. Set**
Input: Graph G=(V,E), number k
Q: Does G have an independent set of size ≥ k
i.e. a set S ⊆ V s.t.
there is no edge (u,v) with u,v ∈ S

**Thm. Ind. Set is NP-complete**

**Pf.** 1. Ind. Set ∈ NP — we saw this already
2. 3-SAT ≤p Ind. Set.
Suppose we have a (black box) poly-time alg. for Ind. Set.
Give a poly-time alg. for 3-SAT
Input: 3-SAT formula F
clauses C₁, ..., Cₘ, variables x₁, ..., xₙ
\[ C_i = (l_{i1} \lor l_{i2} \lor l_{i3}) \]

Create a graph \( G \) on vertices \( l_{ij} \) \( i=1...m \), \( j=1,2,3 \)

- join literals in a clause \( C_i \) (ind. set will "pick" one)

- join literals that are negation of each other e.g. \((x_1 \lor \neg x_2 \lor x_3) \land (x_1 \lor x_2 \lor \neg x_3)\)

\( G \) has poly. size — 3m vertices

**Claim**: \( G \) has an ind. set of size \( \geq m \) if \( F \) is satisfiable

Thus our 3-SAT alg. is

- construct \( G \)
- run Ind. Set alg. on \( G, m \)
- output the YES/NO answer.

This alg. runs in poly. time.

**Proof of Claim**

\[ \iff \text{Suppose } F \text{ is satisfiable} \]

Pick one vertex from each \( \Delta \) corresponding to a True literal. Gives ind. set of size \( m \).
Suppose $G$ has independent set $S$ of size $m$. $S$ can only have one vertex from each $\Delta$. $S$ cannot use $x$ and $\bar{x}$.

Thus we can set all literals in $S$ True and this satisfies the formula. (If a variable isn’t set by $S$ (i.e. neither $y$ nor $\bar{y}$ in $S$), then can set it arbitrarily.

Ex. Carry out this construction on an example