Key questions:

- What's NP-completeness mean?
- What's $P = NP$ or $P \neq NP$ mean?
- How do we show that a problem is intractable?

  intractable = unlikely to be solved efficiently

Definitions:

P = class of all <u>search</u> problems that can be solved in polynomial time

NP = class of all <u>search</u> problems (regardless of time required to solve)

What's a <u>search</u> problem?

Roughly, a problem where we can efficiently verify solutions

Given a solution we can verify this is a solution in polynomial-time.

Formally, what is a search problem?

<u>Search problem:</u>

<u>has the following input/output form:</u>

Given instance I (e.g., graph G)

we are asked to find a solution if one exists and if no solution exists we output NO.

<u>additional requirement:</u>

if we are given a solution S for instance I, then we can verify (i.e., check) that S is a solution to I in time polynomial in $|I|$.

So $P = NP$ or $P \neq NP$ addresses whether or not:

Solving a problem (i.e., constructing a solution)	is as easy as	verifying a solution
P	$\stackrel{?}{=}$	NP

To show a problem P is a search problem, we need to show an algorithm A that takes as input (I, S) & in poly-time verifies that S is a solution to I.

<u>Examples of search problems:</u>

<u>k-coloring:</u>

<u>input:</u> undirected $G=(V,E)$ & integer $k>0$.

<u>output:</u> Assign each vertex to a color in $\{1,\dots,k\}$ so that adjacent vertices get different colors, and output NO if no such k-coloring exists for G.

Given G & a k-coloring σ (so $\sigma(v)$ is the color for vertex v) then in $O(|V|+|E|)$ time we can verify that σ is a valid coloring.

Hence, <u>coloring</u> $\in NP$.

SAT:

input: Boolean formula f in CNF with n variables & m clauses

output: Satisfying assignment if one exists
NO otherwise.

SAT ∈ NP:

Given f and assignment σ,
in $O(n)$ time/clause can verify that the clause is satisfied & hence in $O(nm)$ total time we can verify that σ satisfies f.

Knapsack:

input: n objects with integer weights $w_1, \ldots, w_n$ & integer values $v_1, \ldots, v_n$
Capacity B

output: subset S of objects with total weight $\leq B$ & maximum total value.

Is Knapsack $\in$ NP?

Given instance $\{w_1, \ldots, w_n; v_1, \ldots, v_n; B\}$ & solution S, in $O(n)$ time can check total weight is $\leq B$, but how do we verify that S has maximum value?

(need to do it in time poly$(n, \log B)$ but only approach is to run Knapsack to find an optimal solution S' which takes time poly(n, B)).

This is the optimization version, not search.

Look at search version

Knapsack-search:

input: $\overbrace{w_1, \ldots, w_n; v_1, \ldots, v_n; B}^{\text{as before}}$ & $\overbrace{\text{goal } g}^{\text{new aspect}}$

output: subset S with total weight $\leq B$ & total value $\geq g$. & NO if no such S exists.

Knapsack-search $\in$ NP:

Given instance & solution S, in poly-time ~~$O(n)$~~ poly$(n, \log B)$ can check that it has total weight $\leq B$ & total value $\geq g$.

<u>Note:</u> if we can solve the search version in poly-time then we can solve the optimization version in polytime by binary search over $g \in \{1, \ldots, V\}$.

So knapsack $\rightarrow$ knapsack-search.

<u>MST:</u>

<u>input:</u> $G=(V,E)$ with $w(e)>0$ for $e \in E$.

<u>output:</u> tree T with min weight.

<u>MST $\in$ NP:</u>

Given G & T, run Kruskal's or Prim's alg. to find in poly(n) time a MST T'. Check that $w(T)=w(T')$ & then run BFS/DFS to check that T is a tree.

NP stands for nondeterministic polynomial time
= problems that can be solved in poly-time on a non-deterministic machine

allowed to guess at each step (there exists a path to the accepting state)

NP = all search problems.
P = search problems that can be solved in poly-time
Hence, $P \subset NP$.

if $P = NP$:
means that if we can verify solutions efficiently then we can construct solutions efficiently.
(e.g. verifying a proof is as hard as constructing a proof)

if $P \neq NP$:
means there are some search problems that can't be solved in poly-time.

if $P \neq NP$: what search problems can't be solved in poly-time?

NP-complete problems = hardest problems in NP
= most difficult NP problem to solve.

SAT is NP-complete

This means:

a) $SAT \in NP$

b) if we can solve SAT in poly-time then we can solve every problem in NP in poly-time.

Thus if $P \neq NP$ then there is no poly-time algorithm for SAT.

How to show (b)?

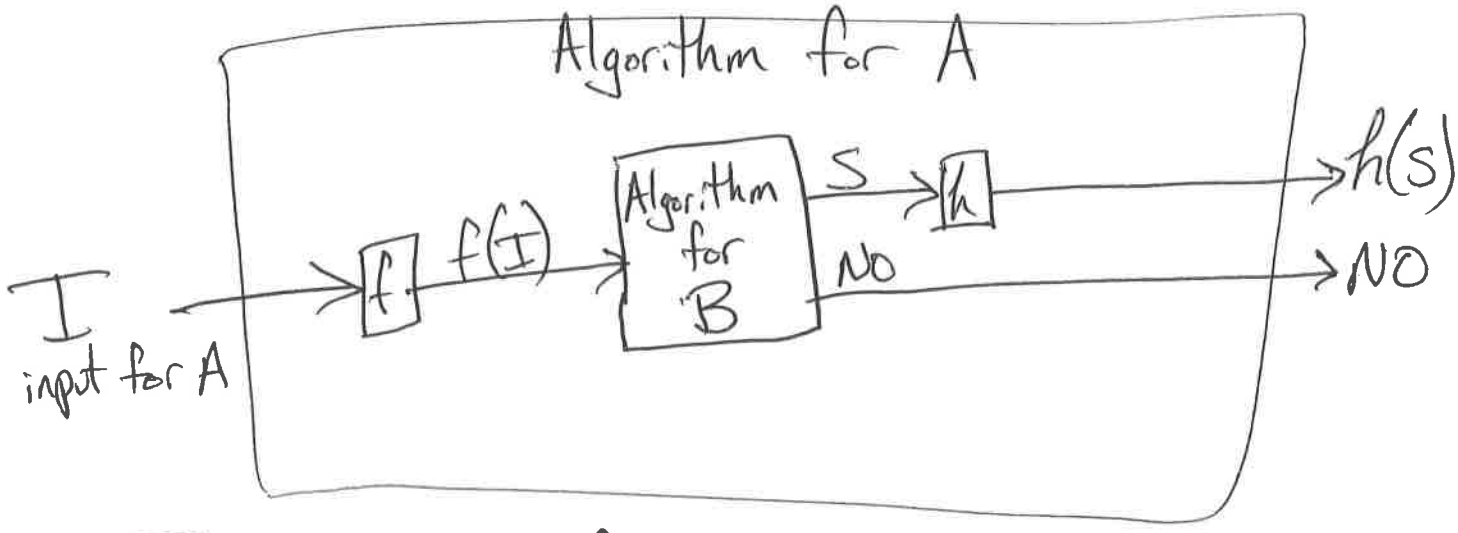
Reductions:

Problems A & B (example: A=MST, B=SAT
or A=2SAT, B=SCC)

$A \to B$ or $A \leq B$

means: A reduces to B

if we can solve B in poly-time then we can use that alg. for B as a black-box to solve A in poly-time.



Take input I for A:

1) define input f(I) for B
2) run black-box alg. for B on f(I)
3) Given solution S for f(I) produce solution h(S) for I
& given NO for f(I) then output NO for I.

To reduce $A \to B$,

need to define f & h

& prove that

if S is a solution to $f(I)$

then $h(S)$ is a solution to I

& if NO solution for $f(I)$

then NO solution for I.

To show:

- SAT is NP-complete

need to show:

a) $SAT \in NP$

b) for all $A \in NP$,

$A \to SAT$.

How to do (b)?

Suppose we know SAT is NP-complete (somehow?)

So for every $A \in NP$, $A \to SAT$.

Suppose we want to show:

Colorings is NP-complete.

if we show:

$SAT \to$ Colorings

then $A \to SAT \to$ Colorings

Hence, to show Colorings is NP-complete, we need to show:

a) Colorings $\in NP$

& b) for a known NP-complete problem B, $B \to$ Colorings.