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

intractable = unlikely to be solved efficiently

P = class of all <u>search</u> problems that are solvable in polynomial time

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

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

Roughly, problem where we can efficiently verify solutions.

So P=NP or P≠NP addresses whether or not:

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

Formally what is a search problem?

Search problem:

Given instance I (for example graph G)
we are asked to find a solution if
one exists
and if no solution exists we output NO.

Further to be a search problem
if we are given a solution S
we can verify (check) in
time polynomial in $|I|$
that S is a solution to I.

In other words, when there is a solution,
given such a solution we can check efficiently it
is a solution.
If there are NO solutions then we don't
need to do anything.

Efficiently means $Poly(|I|)$

Thus we need that the solution $|S| \leq poly(|I|)$.

& we need to show an algorithm A that takes I & S as input and in polynomial-time verifies that S is a solution to I.

Examples of search problems:

k-coloring:

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

output: Assign each vertex a color in $\{1,\dots,k\}$ so that adjacent vertices get different colors.

and NO if no such k-coloring exists for G.

Given G & a k-coloring in $O(|V|+|E|)$ time we can verify that it is a valid coloring.

Hence, coloring $\in NP$.

SAT:

input: $\underbrace{\text{Boolean formula } f \text{ in CNF}}$ with n variables & m clauses

output: Satisfying assignment if one exists
NO otherwise

What's this mean?

Variables $x_1, x_2, \ldots, x_n$ which take values TRUE or FALSE

literals $x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots, x_n, \overline{x_n}$

$\wedge$ = and
$\vee$ = or

clause is the OR of some literals
example: $(x_2 \vee \overline{x_4} \vee x_1)$
So either $x_2 = T$ or $x_4 = F$ or $x_1 = T$

f in CNF is the AND of m clauses:
example: $(\overline{x_2}) \wedge (x_2 \vee \overline{x_4} \vee x_1) \wedge (x_3 \vee x_2 \vee \overline{x_1})$

SAT $\in$ NP

Why? Given f and an assignment, in $O(n)$ time per clause we can verify that each clause is satisfied & hence in $O(nm)$ total time we can verify that the assignment satisfies f. $\boxtimes$

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.

Given instance $\{w_1, \ldots, w_n, v_1, \ldots, v_n, B\}$
& solution S,
We can verify in $O(n)$ time that the total weight is $\leq B$
but how do we verify that it has maximum value?
(need to do it in $\text{poly}(n, \log B)$)

Not clear how to do it.

This is an optimization problem.

Look at search version:

<u>input:</u> $\underbrace{w_1, \dots, w_n, v_1, \dots, v_n, B}_{\text{as before}}$ & goal g.

<u>output:</u> subset S with

total weight $\le B$ $\left(\sum_{i \in S} w_i \le B\right)$

& total value $\ge g$ $\left(\sum_{i \in S} v_i \ge g\right)$

& NO if no such S exists.

Given S, in $O(n)$ time can check that it has total weight $\le B$ & total value $\ge g$.

$\Rightarrow$ Knapsack-search $\in$ NP.

<u>Note:</u> if we can solve the search version in poly-time then we can solve the optimization version in poly-time by doing binary search for max $g \in \{1, \dots, V\}$ which has a solution where $V = \sum_{i=1}^{n} v_i$

MST:

input: $G=(V,E)$ with positive edge lengths
output: tree T with min weight

MST is a search problem.
Hence, MST $\in$ NP.

Why?

Given G & T we can run Kruskal's or Prim's to verify that T has min weight & then run BFS or DFS to verify that T is a tree.

NP stands for nondeterministic polynomial time.
= problems that can be solved in poly-time on a nondeterministic 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 can verify solutions efficiently then we can construct solutions efficiently

(eg, ~~is~~ verifying a proof for a theorem is as hard as constructing a proof)

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

What are these problems?

NP-complete problems: hardest problems in NP.

Colorings is NP-complete.

This means:

a) Colorings $\in NP$

b) if we can solve colorings 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 colorings.

How to show (b)?

Reductions:

Problems A & B (example A = MST, B = Colorings)

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

means we can reduce A to B.

$A \to B$ means that:

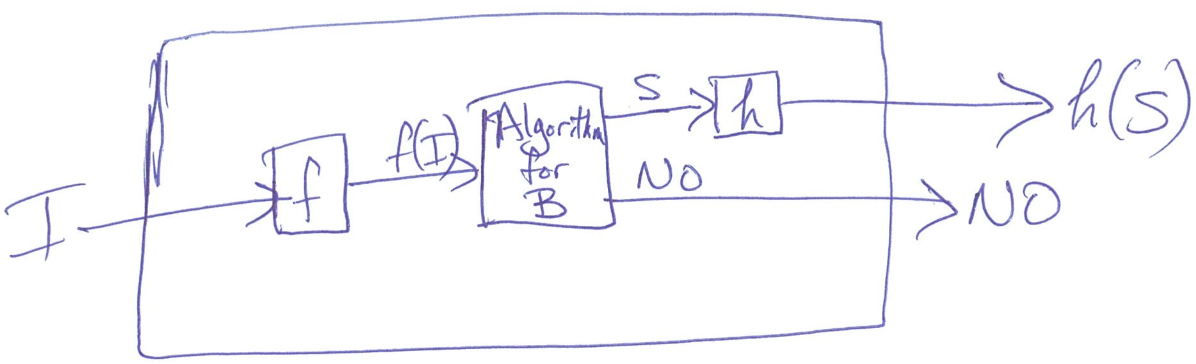
if we can solve B in poly-time then we can use that algorithm to solve A in poly-time.

So we suppose there is an efficient algorithm for B and we use that algorithm as a black-box to solve A.

In other words:

take input I for A (that we want to solve)

1) create input f(I) for B
2) Run black box algorithm for B
3) Given solution S for f(I) we convert it to get solution h(S) for I
& given NO for f(I) we output NO for I.



We 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: Colorings is NP-complete
we need to show:
a) Colorings $\in NP$
b) for all $A \in NP$,
$A \rightarrow$ Colorings

How to do (b) for all $A \in NP$?

Suppose we know SAT is NP-complete.
Hence we know that for every $A \in NP$,

$$A \to SAT.$$

If we show $SAT \to$ Colorings
then $A \to SAT \to$ Colorings

So $A \to$ Colorings.

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

a) Colorings $\in NP$

b) for a known