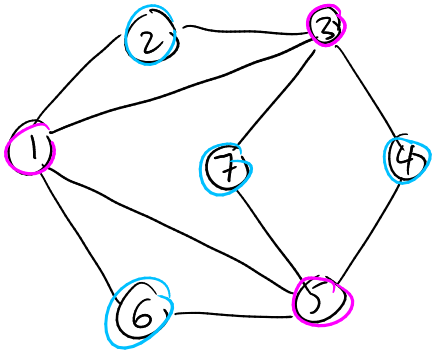
P vs NP!

Problems where we don't know better solution than brute force.

(1) Independent set.

Def'n Given an undirected graph $G = (V, E)$, an independent set is a subset of the vertices $S \subseteq V$ such that no two vertices in S are adjacent.

eg.



Independent sets

$\{2, 4, 6, 7\}$

$\{1, 7, 4\}$

$\{1\}$

$\emptyset$

Interesting Q's:

- How many independent sets?
- Largest independent set? ←

Brute force to find largest ind. set?

- List all subsets — $O(2^V)$
- Check whether each is independent $O(V^2)$? $O(E)$?
- Remember biggest

$O(V^2 2^V)$ — don't know fundamentally better alg. than this.

"Decision problem" = problem w/ Y/N answer.

IND-SET: Given G, number k, is there an independent set in G of size $\geq k$?

② SAT (Satisfiability)

Defn A term is either a variable or the negation of a variable.

eg. x_2 $\quad \neg x_3$ $\quad$ etc.

$(\overline{x_3})$

Defn A clause is one or more terms combined with OR.

eg $x_1 \vee x_3 \vee \overline{x_5}$

x_2

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \overline{x_7} \vee x_8$

Defn A truth assignment is an assignment of T or F to each variable.

eg. $\{x_1 \mapsto T,\ x_2 \mapsto F,\ x_3 \mapsto T\}$.

A truth assignment satisfies a clause if it makes it true.

eg. $\{x_1 \mapsto T,\ x_2 \mapsto F,\ x_3 \mapsto F,\ x_4 \mapsto F\}$

Satisfies $x_1 \vee \overline{x_3} \vee x_4$

Defn A truth assignment satisfies a collection of clauses $C_1, C_2, \ldots, C_k$ if it satisfies all of them. ie. $C_1 \wedge C_2 \wedge \ldots \wedge C_k$.

eg. $(x_1 \vee \overline{x_2})$, $(\overline{x_1} \vee \overline{x_3})$, $(x_2 \vee \overline{x_3})$

is satisfied by $\{x_1 \mapsto T,\ x_2 \mapsto T,\ x_3 \mapsto F\}$.

x_1, $(x_3 \vee \overline{x_1})$, $(\overline{x_3} \vee \overline{x_1})$

— not satisfiable.

Q: Given a collection of clauses, are they satisfiable? (SAT)

Brute force? Try all truth assignments — $\Omega(2^n)$

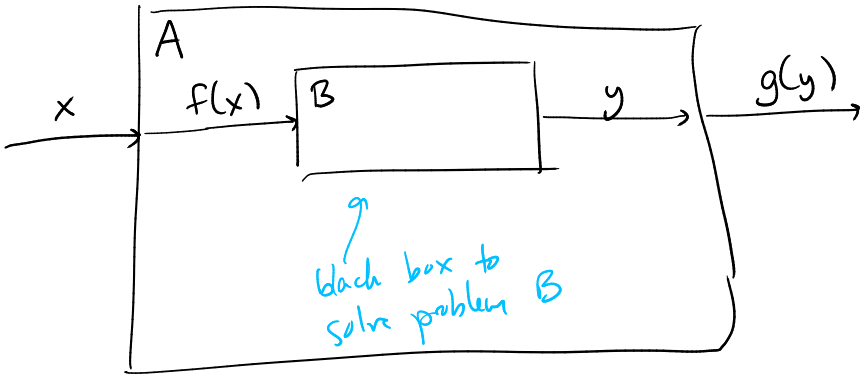
(n = # of variables)

3-SAT: Given a collection of clauses, each of which contains exactly 3 terms, are they satisfiable?

Turns out to be equivalently difficult as SAT.

How do we know which problems are harder than other problems?

Answer: reducibility.



If black box for B can be used to solve A (and f and g don't take too long), we say A is reducible to B and write

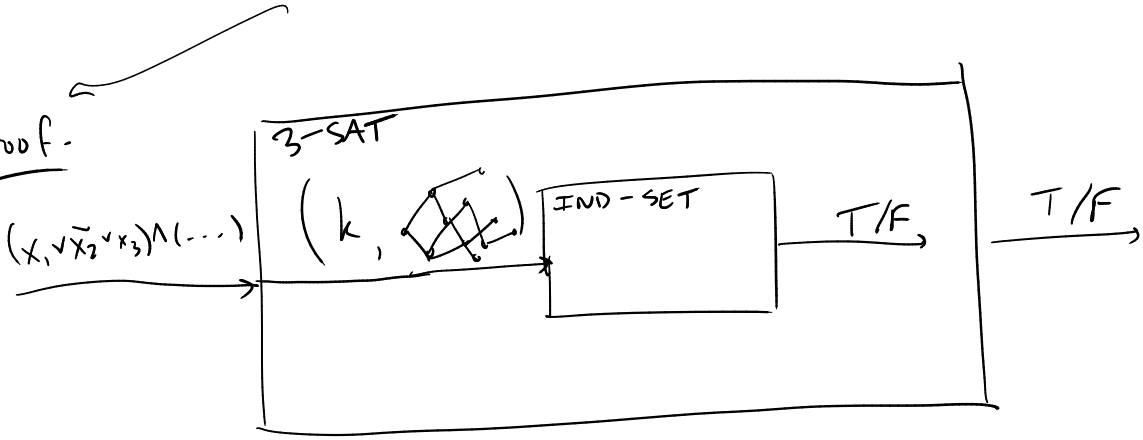
$$A \leq B$$

eg SAT $\leq$ 3-SAT. 3-SAT $\leq$ SAT

Bipartite matching $\leq$ Max flow

claim: 3-SAT $\leq$ IND-SET.

Proof.

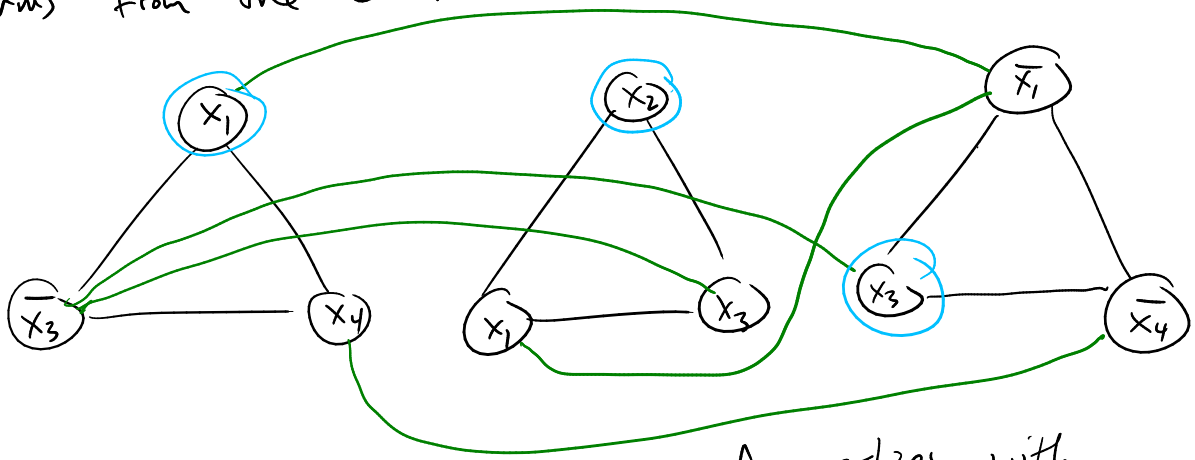


Given a collection of n clauses, eg. $(x_1 \vee \bar{x}_3 \vee x_4)$, $(x_2 \vee x_1 \vee x_3)$, $(\bar{x}_1 \vee x_3 \vee \bar{x}_4)$.

We must construct a graph G and pick a number k such that G has an independent set of size k iff the clauses are satisfiable.

Start w/ n triangles, (one per clause) & label the vertices of each triangle w/ terms from one clause

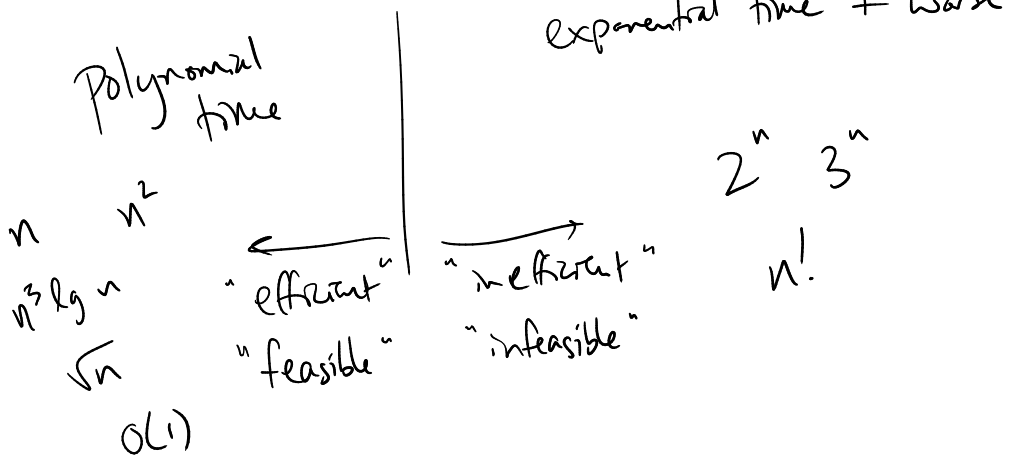
eg.



Add an edge connecting every pair of vertices with x_i and $\bar{x}_i$.

This graph has an independent set of size n iff the clauses are satisfiable.

Typically:



Defn If $A \leq B$ and the translation functions f (and g) take polynomial time to compute, then we say A is polynomial-time reducible to B, and write

$$A \leq_p B.$$

Thm If $A \leq_p B$ and B is solvable in polynomial time, then so is A.

Cor. If $A \leq_p B$ and A is not solvable in poly. time, then neither is B.

- From now on we will identify the size of some input as the number of bits needed to write it.
- A decision problem D will be identified with the set of binary strings corresponding to inputs for which the answer should be true.
- An algorithm A solves D if for all binary strings s,

$$A(s) = T \text{ iff } s \in D.$$

- If there is a polynomial $p(n)$ such that $A(s)$ always terminates in at most $p(|s|)$ steps, then we say A is a polynomial-time algorithm.

Defn P is the set of decision problems D which are solvable in poly. time, i.e. for which $\exists$ a poly-time algorithm which solves D.

Some problems may be easier to verify than to solve.

Defn An algorithm B is a **poly-time certifier** for a decision problem D if:

- B is a poly-time algorithm taking inputs s and t
- There is a polynomial $p(n)$, such that for all bit strings s, we have $s \in D$ iff there is some bit string t (called a certificate) with $|t| \leq p(|s|)$ and $B(s,t) = T$.

Defn NP is the set of all decision problems with a poly-time certifier.