COMP 3011
DESIGN AND ANALYSIS OF ALGORITHMS
FALL 2024

NP-Complete

LI Bo
Department of Computing
The Hong Kong Polytechnic University

REDUCTIONS

REMEMBER FIBONACCI?

- ***Leonardo Fibonacci*** (Italian mathematician)
- But today Fibonacci is most widely known for his famous sequence of numbers

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

- More formally, the Fibonacci numbers F_n are generated by the simple rule

$$F_n = \begin{cases} F_{n-1} + F_{n-2}, & n \geq 2 \\ 1, & n = 1 \\ 0, & n = 0 \end{cases}$$

- In fact, the Fibonacci numbers grow almost as fast as the powers of 2: for example, F_{30} is over a million, and F_{100} is already 21 digits long! In general, $F_n \approx 2^{0.694n}$.

A DIRECT ALGORITHM

$T(n) = T(n-1) + T(n-2) + 3$ for $n > 1$.

- By the recursive definition of F_n,

$T(n) \geq F_n \approx 2^{0.694n}$???

function $fib1(n)$

if $n = 0$: return 0 1 step

if $n = 1$: return 1 1 step

If $n \leq 1,\ T(n) \leq 2$

return $fib1(n-1) + fib1(n-2)$ $T(n-1) + T(n-2)$ +1 steps

- Whenever we have an algorithm, there are three questions we always ask:
  - 1. Is it correct? ✓
  - 2. How much time does it take, as a function of n?
  - 3. And can we do better? ✓

A POLYNOMIAL ALGORITHM

- Why not save the known results?

  function $fib2(n)$

  if $n = 0$ return 0 1 step

  create an array $f[0,1,\cdots,n]$ 1 step

  $f[0] = 0, f[1] = 1$ 2 steps

  for $i = 2, \cdots, n$: $n-1$ rounds

  $f[i] = f[i-1] + f[i-2]$ 1 step

  $n-1$ steps

  return $f[n]$ 1 step

- In total $T(n) = n + 4$ steps.

  polynomial steps!!!

Conclusion

- Fibonacci numbers can be computed efficiently.
- But we need to be very careful and work hard.

REMEMBER SATISFIABILITY?

Can we do better?

- ***Literal***. A Boolean variable or its negation x_i or $\overline{x_i}$
- ***Clause***. A disjunction of literals. $C_j = x_1 \vee \overline{x_2} \vee x_3$
- ***Conjunctive normal form (CNF)***. A propositional formula Φ that is a conjunction of clauses.

- ***SAT***. Given a CNF formula Φ, does it have a satisfying truth assignment?
- ***3-SAT***. SAT where each clause contains exactly 3 literals (and each literal corresponds to a different variable).

Exhaustive search: try all 2^n truth assignments.

$$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

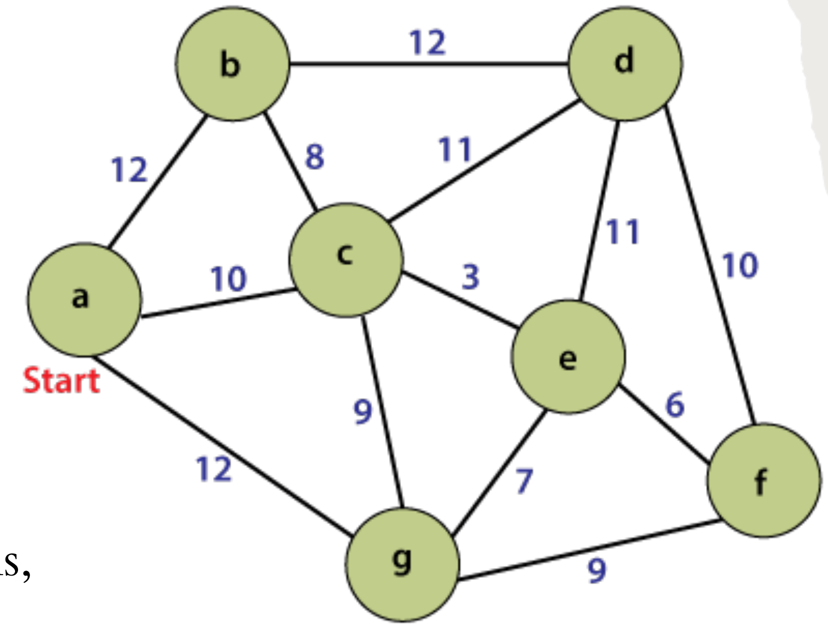
yes instance: x_1 = true, x_2 = true, x_3 = false, x_4 = false

Travelling Salesman Problem (TSP)

No known poly-time algorithm

We are given n cities $1, \cdots, n$, and a nonnegative integer distance d_{ij} between any two cities i and j (assume that the distances are symmetric, that is, $d_{ij} = d_{ji}$ for all i and j).

We are asked to find the shortest tour of the cities -- that is, the permutation π such that $\sum_{i=1}^{n} d_{\pi(i),\pi(i+1)}$ (where by $\pi(n+1)$ we mean $\pi(1)$) is as small as possible.



- We can solve this problem by enumerating all possible solutions, computing the cost of each, and picking the best.

- This would take time proportional to $n!$ (there are $\frac{1}{2}(n-1)!$ tours to be considered), which is not a polynomial bound.

Most outstanding and persistent failure.

ALGORITHM DESIGN PATTERNS

- Greedy.
- Divide and conquer.
- Dynamic programming.
- LP and Duality.
- Local search.
- Reductions.

REDUCTIONS

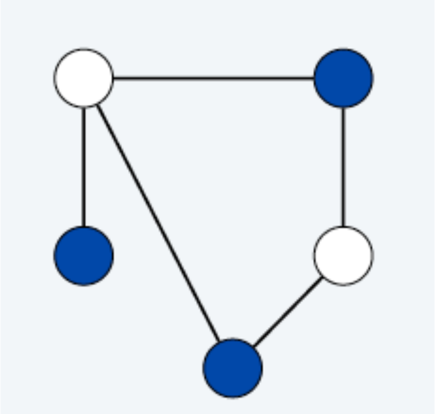
BIPARTITE GRAPHS

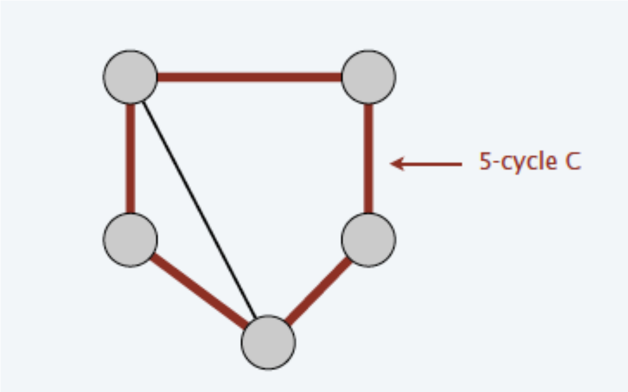
Lemma. Let G be a connected graph, and let $L_0, \dots, L_k$ be the layers produced by BFS starting at node s. Exactly one of the following holds.

i. No edge of G joins two nodes of the same layer, and G is bipartite.

ii. An edge of G joins two nodes of the same layer, and G contains an odd-length cycle (and hence is not bipartite).

Corollary. A graph G is bipartite if and only if it contains no odd-length cycle.





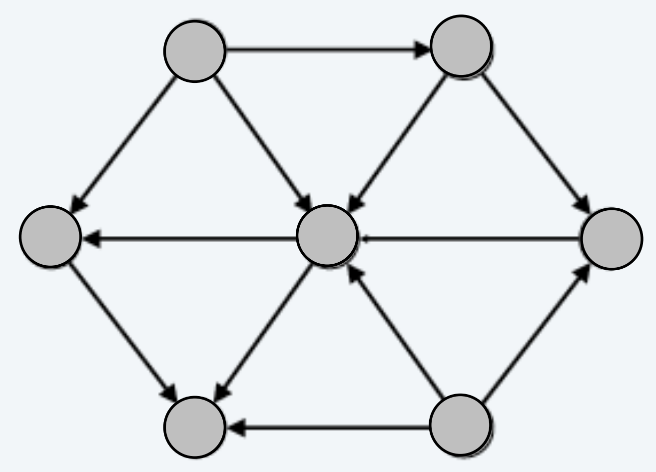
If the graph contains a cycle,
then no linear ordering is possible.

DIRECTED ACYCLIC GRAPHS

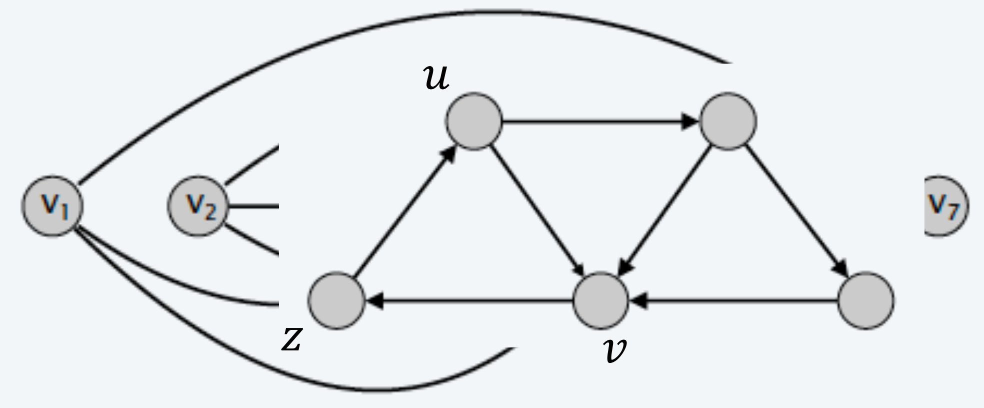
An ordering of the nodes so that all edges point "forward".

Definitions

- A directed acyclic graphs (DAG) is a directed graph that contains no directed cycles.
- A topological order of a directed graph $G = (V, E)$ is an ordering of its nodes as $v_1, v_2, \dots, v_n$ so that for every edge (v_i, v_j) we have $i < j$.



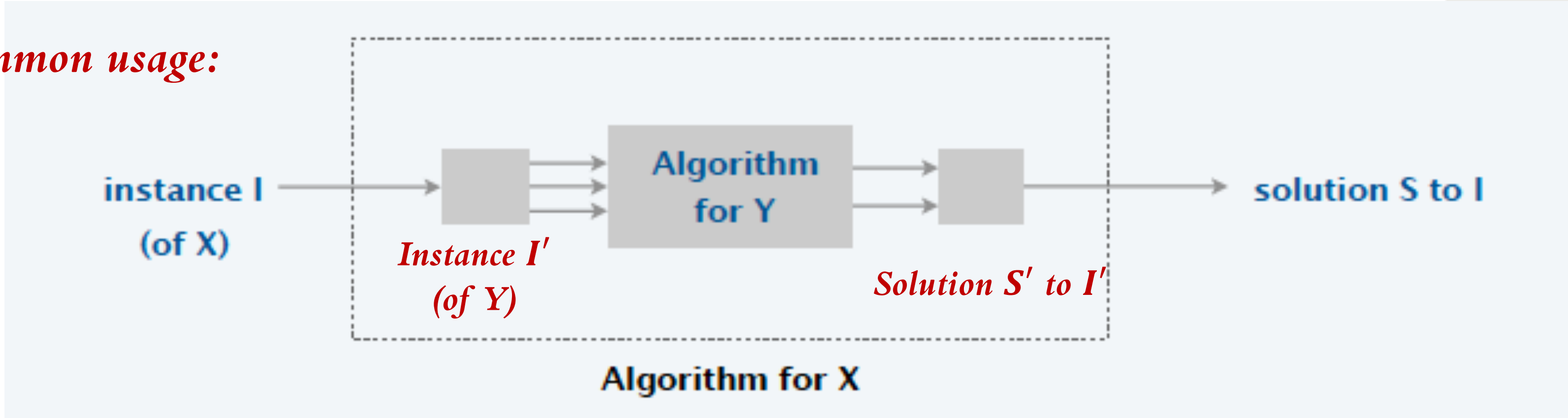
a DAG



a topological ordering

POLY-TIME REDUCTIONS

Common usage:



Desiderata. Suppose we could solve problem Y in polynomial time.

What else could we solve in polynomial time?

Reduction. Problem X polynomial-time reduces to problem Y ($X \leq_P Y$) if ***arbitrary instances of problem X*** can be solved using:

- Polynomial number of standard computational steps, and
- Polynomial number of calls to ***oracle*** that solves problem Y.

Each primitive operation takes a constant amount of time.

- Primitive Operations:
  - Arithmetic (such as add, subtract, multiply, divide, remainder, floor, ceiling),
  - Logic operations (and, or)
  - Read/write memory
  - Array indexing
  - Following a pointer
  - Data movement (load, store, copy)
  - Control (conditional and unconditional branch, subroutine call and return)

POLY-TIME REDUCTIONS

Design algorithms.

If $X \leq_P Y$ and Y can be solved in polynomial time, then X can be solved in polynomial time.

Establish intractability.

If $X \leq_P Y$ and X cannot be solved in polynomial time, then Y cannot be solved in polynomial time.

Establish equivalence.

If both $X \leq_P Y$ and $Y \leq_P X$, we use notation $X \equiv_P Y$. In this case, X can be solved in polynomial time if and only if Y can be.

INDEPENDENT-SET & VERTEX-COVER

INDEPENDENT-SET.

Given a graph $G = (V, E)$ and an integer k, is there a subset of k (or more) vertices such that no two are adjacent?

VERTEX-COVER.

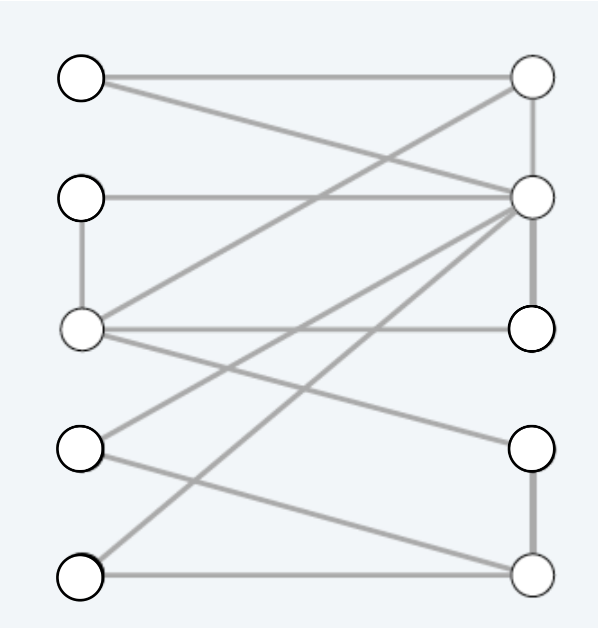
Given a graph $G = (V, E)$ and an integer k, is there a subset of k (or fewer) vertices such that each edge is incident to at least one vertex in the subset?

Q: Is there an independent set of size ≥ 6 ?

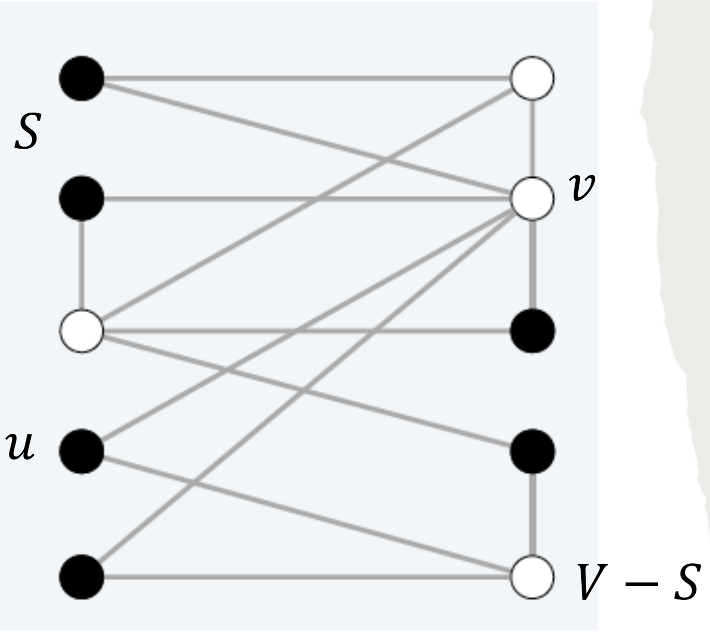
Q: Is there an independent set of size ≥ 7 ?

Q: Is there a vertex cover of size ≤ 4 ?

Q: Is there a vertex cover of size ≤ 3 ?



INDEPENDENT-SET & VERTEX-COVER



Theorem. INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.

Proof.

We show S is an independent set of size k if and only if $V - S$ is a vertex cover of size $n - k$.

$\Rightarrow$

- Let S be any independent set of size k.
- $V - S$ is of size $n - k$.
- Consider an arbitrary edge $(u, v) \in E$.
- S is independent
  $\Rightarrow$ either $u \notin S$, or $v \notin S$
  $\Rightarrow$ either $u \in V - S$, or v
- Thus, $V - S$ covers (u, v)

$\Leftarrow$

- Let $V - S$ be any vertex cover of size $n - k$.
- S is of size k.
- Consider an arbitrary edge $(u, v) \in E$.

or $v \in V - S$, or both.

$\notin S$, or both.

dent set.

INDEPENDENT-SET $\leq_P$ VERTEX-COVER

INDEPENDENT-SET $\geq_P$ VERTEX-COVER

Basic reduction strategies:

- Simple equivalence: INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.
- Special case to general case: VERTEX-COVER $\leq_P$ SET-COVER.
- Encoding with gadgets: 3-SAT $\leq_P$ INDEPENDENT-SET.

SET COVER AND VERTEX COVER

SET-COVER.

Given a set U of elements, a collection S of subsets of U, and an integer k, are there $\leq k$ of these subsets whose union is equal to U ?

VERTEX-COVER.

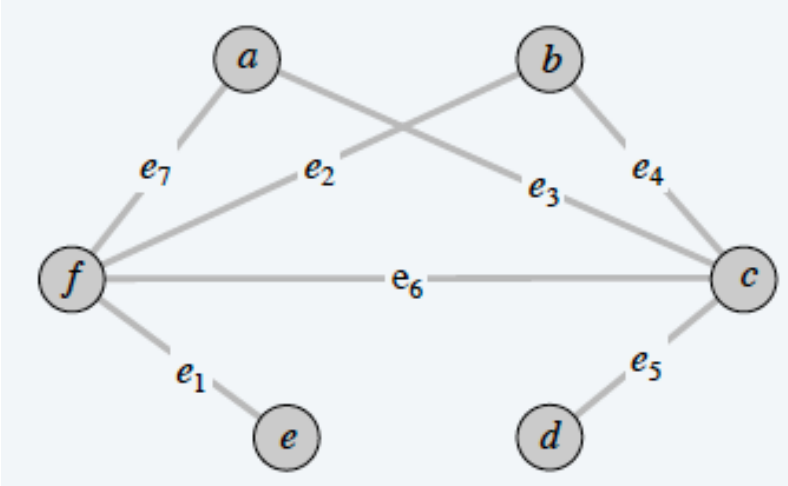
Given a graph $G = (V, E)$ and an integer k, is there a subset of k (or fewer) vertices such that each edge is incident to at least one vertex in the subset?

$U = \{ 1, 2, 3, 4, 5, 6, 7 \}$

$S_a = \{ 3, 7 \}$ $\quad$ $S_b = \{ 2, 4 \}$

$S_c = \{ 3, 4, 5, 6 \}$ $\quad$ $S_d = \{ 5 \}$

$S_e = \{ 1 \}$ $\quad$ $S_f = \{ 1, 2, 6, 7 \}$



SET COVER AND VERTEX COVER

Theorem. VERTEX-COVER $\leq_P$ SET-COVER.

Proof.

Given a VERTEX-COVER instance $G = (V, E)$ and k, we construct a SET-COVER instance (U, S, k) that has a set cover of size k iff G has a vertex cover of size k.

Construction

- $U = E$.
- $S_v = \{e \in E\colon\ e \text{ incident } to\ v\}$ for each $v \in V$.

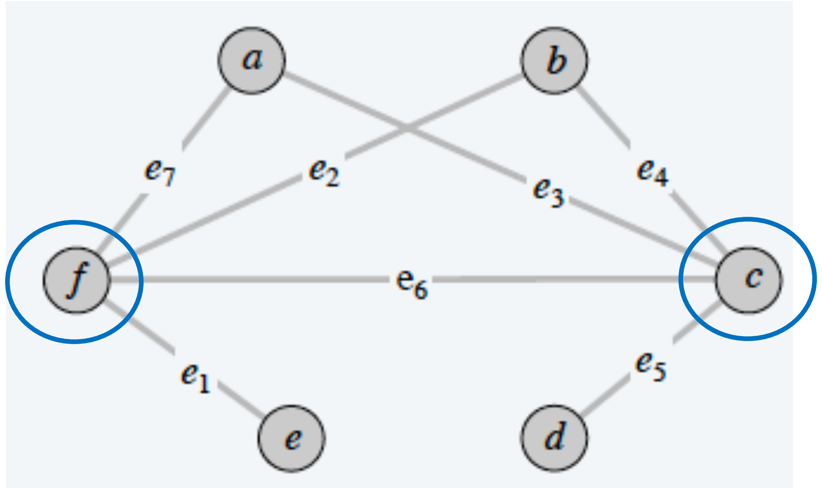
17

$$U = \{1, 2, 3, 4, 5, 6, 7\}$$

$S_a = \{3, 7\}$ $\qquad$ $S_b = \{2, 4\}$

$S_c = \{3, 4, 5, 6\}$ $\qquad$ $S_d = \{5\}$

$S_e = \{1\}$ $\qquad$ $S_f = \{1, 2, 6, 7\}$



SET COVER AND VERTEX COVER

Lemma. (U, S, k) contains a set cover of size k iff $G = (V, E)$ contains a vertex cover of size k.

$\Longrightarrow$

Let $Y \subseteq S$ be a set cover of size k in (U, S, k).
Then $X = \{v : S_v \in Y\}$ is a vertex cover of size k in G.

$\Longleftarrow$

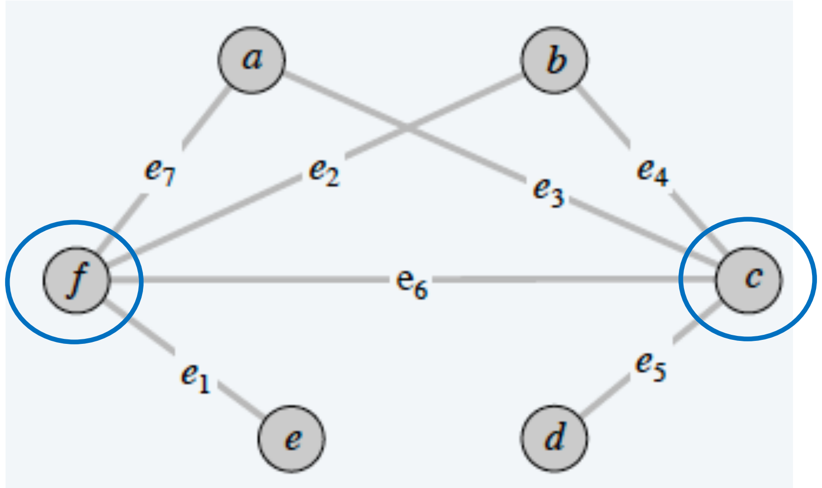
Let $X \subseteq V$ be a vertex cover of size k in G.
Then $Y = \{S_v : v \in X\}$ is a set cover of size k.

$U = \{1, 2, 3, 4, 5, 6, 7\}$

$S_a = \{3, 7\}$ $\quad$ $S_b = \{2, 4\}$

$S_c = \{3, 4, 5, 6\}$ $\quad$ $S_d = \{5\}$

$S_e = \{1\}$ $\quad$ $S_f = \{1, 2, 6, 7\}$



Basic reduction strategies:

- Simple equivalence: INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.
- Special case to general case: VERTEX-COVER $\leq_P$ SET-COVER.
- Encoding with gadgets: 3-SAT $\leq_P$ INDEPENDENT-SET.

Theorem. 3-SAT $\leq_P$ INDEPENDENT-SET.

Proof.

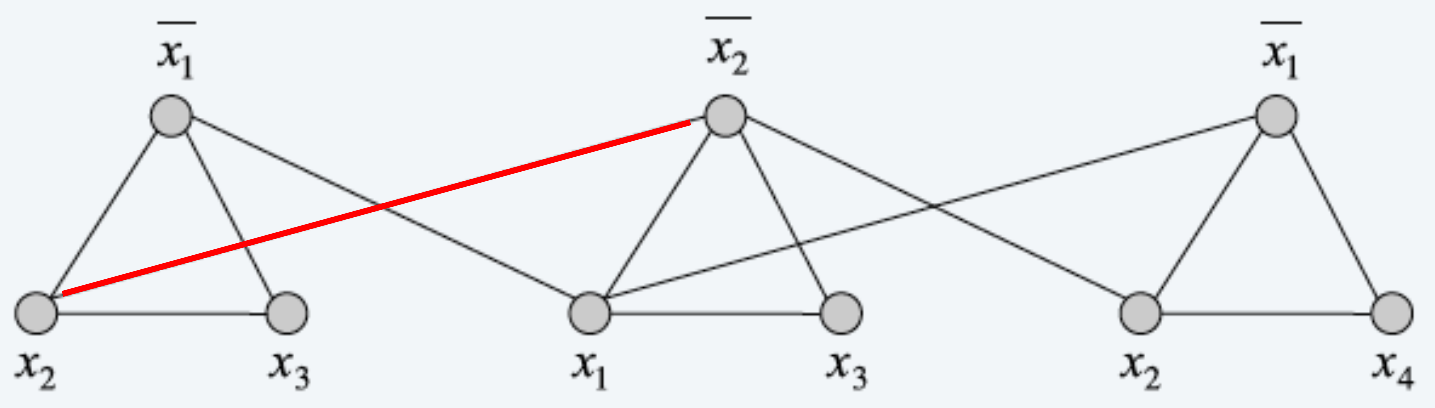
Given an instance Φ of 3-SAT, we construct an instance (G, k) of INDEPENDENT-SET that has an independent set of size $k = |\Phi|$ if and only if Φ is satisfiable.

At most one is in IS

At most one of x_i and $\overline{x_i}$ is in IS

Φ: $(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$

(G, k):



Construction

- G contains 3 nodes for each clause, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.

SAT⟹IS

- Consider any satisfying assignment for Φ.
- Select one true literal from each clause/triangle.
- This is an independent set of size $k = |\Phi|$.

SAT⟸IS

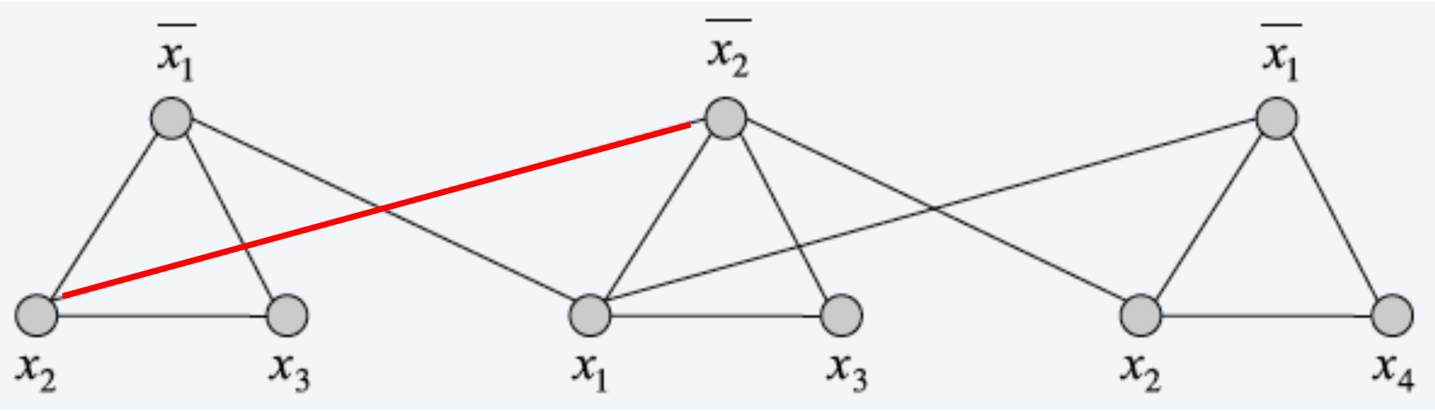
- Let S be independent set of size k.
- S must contain exactly one node in each triangle.
- Set these literals to *true* (and remaining literals consistently).
- All clauses in Φ are satisfied.

At most one is in IS

At most one of x_i and $\overline{x_i}$ is in IS

Φ: $(\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4)$

(G, k):



Construction

- G contains 3 nodes for each clause, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.

REVIEW

Basic reduction strategies:

- Simple equivalence: INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.
- Special case to general case: VERTEX-COVER $\leq_P$ SET-COVER.
- Encoding with gadgets: 3-SAT $\leq_P$ INDEPENDENT-SET.

Properties (Transitivity):

If $X \leq_P Y$ and $Y \leq_P Z$, then $X \leq_P Z$.

Proof idea: Compose the two algorithms.

Example: 3-SAT $\leq_P$ INDEPENDENT-SET $\leq_P$ VERTEX-COVER $\leq_P$ SET-COVER.

Decision problem. Does there exist a vertex cover of size $\leq k$?
Search problem. Find a vertex cover of size $\leq k$.
Optimization problem. Find a vertex cover of minimum size.

Three problems poly-time reduce to one another.

- VERTEX-COVER. Does there exist a vertex cover of size $\leq k$?
- FIND-VERTEX-COVER. Find a vertex cover of size $\leq k$.
- FIND-MIN-VERTEX-COVER. Find a vertex cover of minimum size.

Theorem.
VERTEX-COVER $\equiv_P$ FIND-VERTEX-COVER

$\leq_P$
Decision problem is a special case of search problem

$\geq_P$
To find a vertex cover of size $\leq k$:
- Determine if there exists a vertex cover of size $\leq k$.
- Find v s.t. $G - \{v\}$ has a cover of size $\leq k - 1$. (any vertex in a vertex cover of size $\leq k$ satisfies).
- Include v in the vertex cover.
- Find a vertex cover of size $\leq k - 1$ in $G - \{v\}$.

Theorem.
FIND-VERTEX-COVER $\equiv_P$ FIND-MIN-VERTEX-COVER

$\leq_P$
Search problem is a special case of optimization problem.

$\geq_P$
To find vertex cover of minimum size:
- Binary search (or linear search) for size k^* of min vertex cover.
- Solve search problem for given k^*.

NP COMPLETENESS

Decision Problem

- Problem X is a set of strings.
- Instance s is one string.
- Algorithm A solves problem X: $A(s) = \begin{cases} yes, & if\ s \in X \\ no, & if\ s \notin X \end{cases}$

X contains all the "yes" instances

s is one instance

Definition.
Algorithm A runs in polynomial time if for every string s, $A(s)$ terminates in $\le p(|s|)$ "steps," where $p(\cdot)$ is some polynomial function.

length of s

Definition.
Algorithm $C(s,t)$ is a certifier for problem X if for every string s such that $s \in X$ iff there exists a string t (certificate) such that $C(s,t) = yes$.

Definition.
P = set of decision problems for which there exists a poly-time algorithm.

on a deterministic Turing machine

Definition. Nondeterministic polynomial time!!!
NP = set of decision problems for which there exists a poly-time certifier.

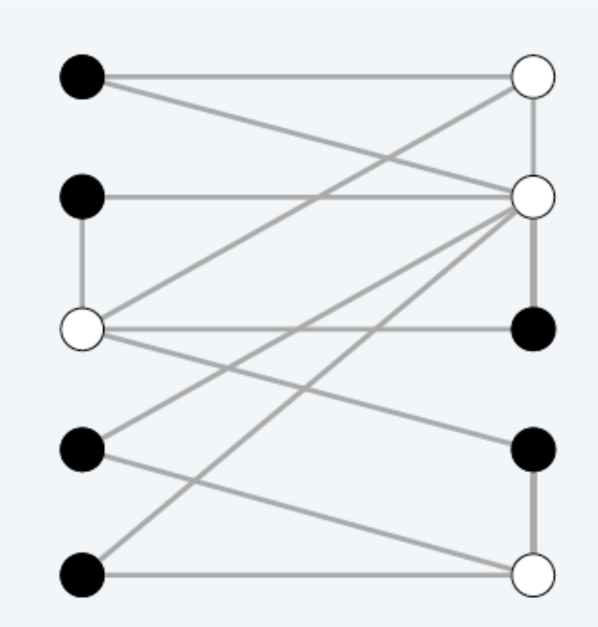
- $C(s,t)$ is a poly-time algorithm.
- Certificate t is of polynomial size: $t \le p(|s|)$ for some polynomial $p(\cdot)$.

Examples in NP:

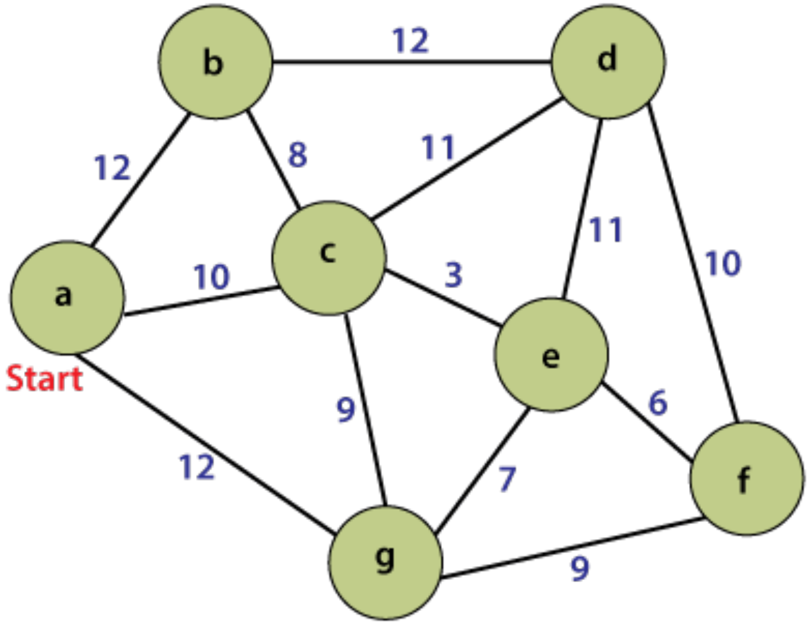
SAT & 3-SAT

instance s $\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$

certificate t $x_1 = true,\ x_2 = true,\ x_3 = false,\ x_4 = false$



Vertex Cover & Independent Set



TSP

- ***P:*** problems for which there exists a poly-time algorithm.
- ***NP:*** set of decision problems for which there exists a poly-time certifier.
- ***EXP:*** Decision problems for which there exists an exponential-time algorithm.

Proposition. $\boldsymbol{P \subseteq NP}$.

Consider any problem $X \in \boldsymbol{P}$.
- By definition, there exists a poly-time algorithm $A(s)$ that solves X.
- Certificate $t = \varepsilon$, certifier $C(s,t) = A(s)$.

Fact. $\boldsymbol{P \neq EXP}$

$\Rightarrow$ either $\boldsymbol{P \neq NP}$, or $\boldsymbol{NP \neq EXP}$, or both.

Proposition. $\boldsymbol{NP \subseteq EXP}$.

Consider any problem $X \in \boldsymbol{NP}$.
- By definition, there exists a poly-time certifier $C(s,t)$ for X, where certificate t satisfies $|t| \leq p(|s|)$ for some polynomial $p(\cdot)$.
- To solve instance s, run $C(s,t)$ on all strings t with $|t| \leq p(|s|)$.
- Return *yes* iff $C(s,t)$ returns *yes* for any of these potential certificates.

The main question: P vs. NP

Q. How to solve an instance of 3-SAT with n variables?
A. Exhaustive search: try all 2^n truth assignments.

Q. Can we do anything substantially cleverer?
Conjecture. $\underbrace{\text{No poly-time algorithm}}_{\textit{intractable}}$ for 3-SAT.

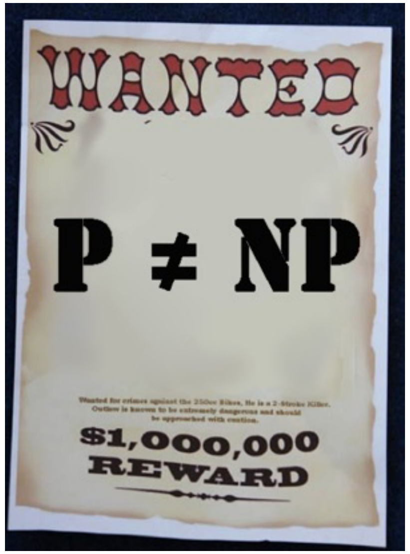
Consensus opinion

A set of problems in NP
NP-complete ⊆ NP

NP-complete
A problem $Y \in NP$ with the property that for every problem $X \in NP$, $X \leq_P Y$.

" I conjecture that there is no good algorithm for the traveling salesman problem. My reasons are the same as for any mathematical conjecture: (i) It is a legitimate mathematical possibility and (ii) I do not know."

— *Jack Edmonds 1966*



Millennium prize

NP-COMPLETE

NP-complete
A problem $Y \in NP$ with the property that for every problem $X \in NP$, $X \leq_P Y$.

Fundamental question.
Are there any "natural" NP-complete problems?

Proposition. If $Y \in NP$-complete, then $Y \in$ P iff P=NP.

Proof.

$\Leftarrow$

If P = NP, then $Y \in$ P because $Y \in$ NP.

$\Rightarrow$

Suppose $Y \in$ P.

- Consider any problem $X \in NP$. Since $X \leq_P Y$, we have $X \in P$.
- This implies $NP \subseteq P$.
- We already know $P \subseteq NP$. Thus P = NP.

Theorem. [Cook 1971, Levin 1973] SAT $\in$ NP-complete.

The Complexity of Theorem-Proving Procedures

Stephen A. Cook

University of Toronto

Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polyno-

certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but theorem 1 will give evidence that {tautologies} is a difficult set to recognize, since many apparently difficult problems can be reduced to determining tautologyhood. By reduced we mean,

ПРОБЛЕМЫ ПЕРЕДАЧИ ИНФОРМАЦИИ

Том IX　　1973　　Вып. 3

КРАТКИЕ СООБЩЕНИЯ

УДК 519.14

УНИВЕРСАЛЬНЫЕ ЗАДАЧИ ПЕРЕБОРА

Л. А. Левин

В статье рассматривается несколько известных массовых задач «переборного типа» и доказывается, что эти задачи можно решать лишь за такое время, за которое можно решать вообще любые задачи указанного типа.

После уточнения понятия алгоритма была доказана алгоритмическая неразрешимость ряда классических массовых проблем (например, проблем тождества элементов групп, гомеоморфности многообразий, разрешимости диофантовых уравнений

ESTABLISHING NP-COMPLETENESS

NP-complete
A problem $Y \in NP$ with the property that for every problem $X \in NP$, $X \leq_P Y$.

Remark.
Once we establish first "natural" NP-complete problem, others fall like dominoes.

Proposition.
If $X \in$ NP-complete, $Y \in$ NP, and $X \leq_P Y$, then $Y \in$ NP-complete.

Proof.
Consider any problem $W \in$ NP. Then, both $W \leq_P X$ and $X \leq_P Y$. By transitivity, $W \leq_P Y$.

Example: 3-SAT $\leq_P$ INDEPENDENT-SET $\leq_P$ VERTEX-COVER $\leq_P$ SET-COVER.

Recipe.
To prove that $Y \in$ NP-complete:
- Step 1. Show that $Y \in$ NP.
- Step 2. Choose an NP-complete problem X.
- Step 3. Prove that $X \leq_P Y$.

3-SAT

Theorem*. *3-SAT is NP-complete.

Recipe.

To prove that $Y \in$ NP-complete:

- Step 1. Show that $Y \in$ NP.
- Step 2. Choose an NP-complete problem X.
- Step 3. Prove that $X \leq_P Y$.

Step 1. 3-SAT is in NP: Given a truth assignment, we can check in poly time whether it satisfies all clauses.

Step 2. Choose SAT which is known to be NP-complete.

Step 3. Show SAT $\leq_p$3-SAT.

Any instance I for SAT $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ and variables $\mathrm{X} = \{x_1, x_2, \cdots, x_n\}$.

in poly time

Some instance I' for 3-SAT $\Phi' = C_1' \wedge C_2' \wedge \cdots \wedge C_{m'}'$ and variables $\mathrm{X}' = \{x_1', x_2', \cdots, x_{n'}'\}$.

Note that $|C_i'| = 3$ for all $i = 1,2,\cdots, m'$.

Step 3. Show SAT $\leq_p$3-SAT.

Any instance I for SAT $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ and variables $X = \{x_1, x_2, \cdots, x_n\}$.

Some instance I' for 3-SAT $\Phi' = C_1' \wedge C_2' \wedge \cdots \wedge C_{m'}'$ and variables $Y = \{y_1, y_2, \cdots, y_{n'}\}$.

Note that $|C_i'| = 3$ for all $i = 1,2, \cdots, m'$.

Claim: I is satisfiable if and only I' is satisfiable.

$C_j = (x_1 \vee x_2 \vee \cdots \vee x_k)$

C_j can be satisfied iff all clauses in C_j' can be satisfied!

Case 1. $k = 1$

$C_j = (x_1) \longrightarrow C_j' = (x_1 \vee y_j^1 \vee y_j^2) \wedge (x_1 \vee \overline{y_j^1} \vee y_j^2) \wedge (x_1 \vee y_j^1 \vee \overline{y_j^2}) \wedge (x_1 \vee \overline{y_j^1} \vee \overline{y_j^2})$

Add new variables: $Y_j = \{y_j^1, y_j^2\}$

Step 3. Show SAT $\leq_p$3-SAT.

Any instance I for SAT $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ and variables $X = \{x_1, x_2, \cdots, x_n\}$.

Some instance I' for 3-SAT $\Phi' = C'_1 \wedge C'_2 \wedge \cdots \wedge C'_{m'}$ and variables $Y = \{y_1, y_2, \cdots, y_{n'}\}$.

Note that $|C'_i| = 3$ for all $i = 1,2, \cdots, m'$.

Claim: I is satisfiable if and only I' is satisfiable.

$C_j = (x_1 \vee x_2 \vee \cdots \vee x_k)$

C_j can be satisfied iff all clauses in C'_j can be satisfied!

Case 2. $k = 2$

$C_j = (x_1 \vee x_2) \longrightarrow C'_j = (x_1 \vee x_2 \vee y_j^1) \wedge (x_1 \vee x_2 \vee \overline{y_j^1})$

Add new variables: $Y_j = \{y_j^1\}$

Step 3. Show SAT $\leq_p$3-SAT.

Any instance I for SAT $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ and variables $X = \{x_1, x_2, \cdots, x_n\}$.

Some instance I' for 3-SAT $\Phi' = C'_1 \wedge C'_2 \wedge \cdots \wedge C'_{m'}$ and variables $Y = \{y_1, y_2, \cdots, y_{n'}\}$.

Note that $|C'_i| = 3$ for all $i = 1,2, \cdots, m'$.

Claim: I is satisfiable if and only I' is satisfiable.

$C_j = (x_1 \vee x_2 \vee \cdots \vee x_k)$

C_j can be satisfied iff all clauses in C'_j can be satisfied!

Case 3. $k = 3$

$C_j = (x_1 \vee x_2 \vee x_3) \longrightarrow C'_j = (x_1 \vee x_2 \vee x_3)$

No need to add new variables.

Step 3. Show SAT $\leq_p$3-SAT.

Any instance I for SAT $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ and variables $X = \{x_1, x_2, \cdots, x_n\}$.

Some instance I' for 3-SAT $\Phi' = C_1' \wedge C_2' \wedge \cdots \wedge C_{m'}'$ and variables $Y = \{y_1, y_2, \cdots, y_{n'}\}$.

Note that $|C_i'| = 3$ for all $i = 1,2,\cdots, m'$.

Claim: I is satisfiable if and only I' is satisfiable.

$C_j = (x_1 \vee x_2 \vee \cdots \vee x_k)$

C_j can be satisfied iff all clauses in C_j' can be satisfied!

Case 4. $k = 4$

$C_j = (x_1 \vee x_2 \vee x_3 \vee x_4) \longrightarrow C_j' = (x_1 \vee x_2 \vee y_j^1) \wedge (\overline{y_j^1} \vee x_3 \vee x_4)$

Add new variables: $Y_j = \{y_j^1\}$

Step 3. Show SAT $\leq_p$3-SAT.

Any instance I for SAT $\Phi = C_1 \land C_2 \land \cdots \land C_m$ and variables $X = \{x_1, x_2, \cdots, x_n\}$.

Some instance I' for 3-SAT $\Phi' = C_1' \land C_2' \land \cdots \land C_{m'}'$ and variables $Y = \{y_1, y_2, \cdots, y_{n'}\}$.

Note that $|C_i'| = 3$ for all $i = 1,2, \cdots, m'$.

Claim: I is satisfiable if and only I' is satisfiable.

$$C_j = (x_1 \lor x_2 \lor \cdots \lor x_k)$$

C_j can be satisfied iff all clauses in C_j' can be satisfied!

Case 5. $k = 5$

$$C_j = (x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5) \longrightarrow C_j' = (x_1 \lor x_2 \lor y_j^1) \land (\overline{y_j^1} \lor x_3 \lor y_j^2) \land (\overline{y_j^2} \lor x_4 \lor x_5)$$

Add new variables: $Y_j = \{y_j^1, y_j^2\}$

Step 3. Show SAT $\leq_p$3-SAT.

Any instance I for SAT $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ and variables $X = \{x_1, x_2, \cdots, x_n\}$.

Some instance I' for 3-SAT $\Phi' = C_1' \wedge C_2' \wedge \cdots \wedge C_{m'}'$ and variables $Y = \{y_1, y_2, \cdots, y_{n'}\}$.

Note that $|C_i'| = 3$ for all $i = 1,2,\cdots,m'$.

C_j can be satisfied iff
all clauses in C_j' can be satisfied!

Claim: I is satisfiable if and only I' is satisfiable.

$C_j = (x_1 \vee x_2 \vee \cdots \vee x_k)$

Case 6. $k \geq 6$

$C_j = (x_1 \vee x_2 \vee \cdots \vee x_{k-1} \vee x_k)$

Add new variables: $Y_j = \{y_j^1, y_j^2, \cdots, y_j^{k-3}\}$

$$
\begin{aligned}
C_j' = {} & (x_1 \vee x_2 \vee y_j^1) \wedge (\overline{y_j^1} \vee x_3 \vee y_j^2) \\
& \wedge (\overline{y_j^2} \vee x_4 \vee y_j^3) \\
& \vdots \\
& \wedge (\overline{y_j^{k-3}} \vee x_{k-3} \vee y_j^{k-2}) \\
& \wedge (\overline{y_j^{k-2}} \vee x_{k-2} \vee y_j^{k-3}) \wedge (\overline{y_j^{k-3}} \vee x_{k-1} \vee x_k)
\end{aligned}
$$

Step 3. Show SAT $\le_p$3-SAT.

Any instance I for SAT $\Phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ and variables $X = \{x_1, x_2, \cdots, x_n\}$.

Some instance I' for 3-SAT $\Phi' = C_1' \wedge C_2' \wedge \cdots \wedge C_{m'}'$ and variables $Y = \{y_1, y_2, \cdots, y_{n'}\}$.

Note that $|C_i'| = 3$ for all $i = 1,2, \cdots, m'$.

Claim: The construction can be done in polynomial time.

- For each clause C_j in I, we add no more than n variables, and thus no more than mn new variables in total.
- For each clause C_j in I, we add no more than n clauses, and thus no more than mn clauses in total.

ESTABLISHING NP-COMPLETENESS

NP-complete
A problem $Y \in NP$ with the property that for every problem $X \in NP$, $X \leq_P Y$.

Remark.
Once we establish first "natural" NP-complete problem, others fall like dominoes.

Proposition.
If $X \in$ NP-complete, $Y \in$ NP, and $X \leq_P Y$, then $Y \in$ NP-complete.

Proof.
Consider any problem $W \in$ NP. Then, both $W \leq_P X$ and $X \leq_P Y$. By transitivity, $W \leq_P Y$.

Example: SAT $\leq_P$ 3-SAT $\leq_P$ INDEPENDENT-SET $\leq_P$ VERTEX-COVER $\leq_P$ SET-COVER.

Recipe.
To prove that $Y \in$ NP-complete:
- Step 1. Show that $Y \in$ NP.
- Step 2. Choose an NP-complete problem X.
- Step 3. Prove that $X \leq_P Y$.