Analysis of Algorithms

V. Adamchik

Lecture 11

Spring 2023

CSCI 570

University of Southern California

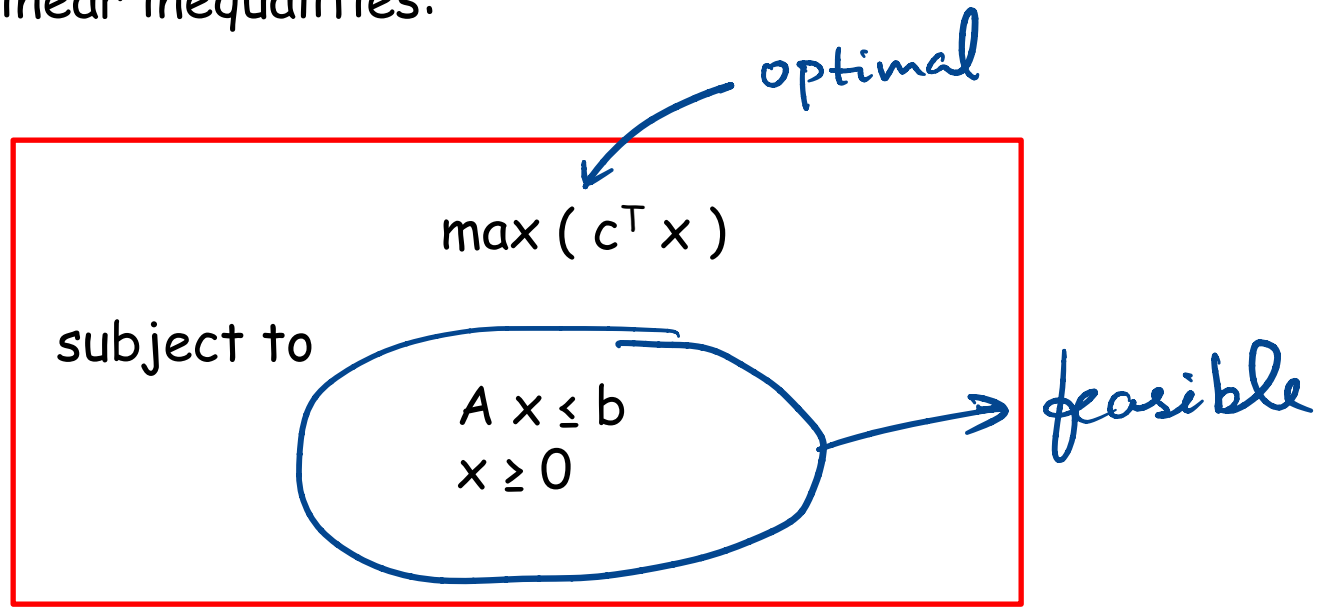
Linear Programming

NP-Completeness

Reading: chapter 8 and 9

Linear Programming

We say that a maximization linear program with n variables is in standard form if for every variable x_k we have the inequality $x_k \geq 0$ and other m linear inequalities:



The LP can be solved in polynomial time for real variables x_k.
In case of integer variables, we do not have a polynomial solver.

Dual LP

To every linear program there is a dual linear program

Similar to reduction but not really reduction.

Duality

Definition. The dual of the standard (primal) maximum problem

$$\max_x c^T x$$
$$Ax \leq b \text{ and } x \geq 0$$

is defined to be the standard minimum problem

$$\min_y b^T y$$
$$A^T y \geq c \text{ and } y \geq 0$$

Weak Duality

$\max (c^T x)$

$A x \le b$
$x \ge 0$

primal linear program

$\min (b^T y)$

$A^T y \ge c$
$y \ge 0$

dual linear program

Weak Duality. The optimum of the dual is an upper bound to the optimum of the primal.

$$\text{opt(primal)} \le \text{opt(dual)}$$

Theorem (The weak duality).
Let P and D be primal and dual LP correspondingly.
If x is a feasible solution for P and y is a feasible solution for D, then $c^T x \le b^T y$

Strong Duality

$\max (c^T x)$

$A x \leq b$
$x \geq 0$

$\min (b^T y)$

$A^T y \geq c$
$y \geq 0$

Theorem (The strong duality).
Let P and D be primal and dual LP correspondingly.
If P and D are feasible, then

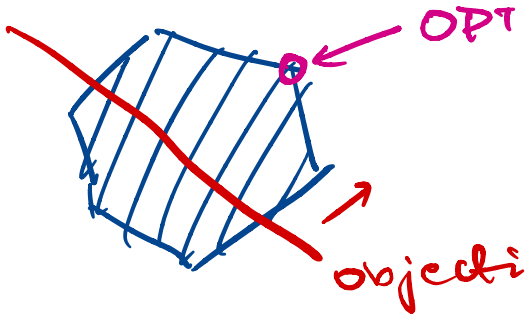
$$c^T x = b^T y$$

$$\in \mathbb{R}$$

Review Questions

6. (T/F) Every LP has an optimal solution.
7. (T/F) If an LP has an optimal solution it occurs at an extreme point.
8. (T/F) If an LP is feasible and bounded, then it must have an optimal solution.
9. (T/F) An LP allows strict inequalities in the constraints.
10. (T/F) An LP for which the feasible region is unbounded has the finite optimal solution.
11. (T/F) The weak duality theorem does not always hold for an integer linear program. [If the question asks for strong duality then it might be false because here we have integers & not real nos]
12. (T/F) An LP must be infeasible if its dual problem is unbounded.
13. (T/F) Both the primal and the dual can be infeasible.
14. (T/F) There is no duality gap in linear programming.

↳ Since the problem is linear we take that primal = dual

8. 

Since our polygon is bounded we will have an optimal sol^n at the extreme. If not bounded no guarantee

10. $\max(x) \quad x \geq 0$
↳ counter example

11. $c^T x \leq b^T y$

12 $c^T x \leq b^T y$?
↳ $-\infty$ → because it is $\min(b^T y)$
↓
$c^T x \leq -\infty \Rightarrow$ this doesn't make any sense hence TRUE

13. TRUE – ?
by Example – In the textbook
(It is found in the end of his annotated notes)

NP-Completeness

23 Problems of Hilbert

In 1900 Hilbert presented a list of 23 challenging (unsolved) problems in math

#1 The Continuum Hypothesis	impossible, 1963
#8 The Riemann Hypothesis	unproved yet
#10 On solving a Diophantine equations	impossible, 1970
#18 The Kepler Conjecture	proved,1998

Hilbert's 10th problem

Given a multivariate polynomial with integer coeffs, e.g. $4x^2y^3 - 2x^4z^5 + x^8$, "*devise a process according to which it can be determined in a finite number of operations*" whether it has an integer root.

Mathematicians: we should try to formalize what counts as a '*process*' and an '*operation*'.

Hilbert's 10th problem

In 1928 Hilbert rephrased it as follows:

Given a statement in first-order logic,
devise an "*effectively calculable procedure*"
for determining if it's provable.

Mathematicians: we should try to formalize what counts as an '*calculable procedure*' (aka algorithm) and an '*efficient calculable procedure*'.

Hilbert's conjecture

Hilbert conjectured that any mathematical proposition can be decided (proved true or false) by mechanistic logical methods.

In 1931 it was unexpectedly disproven by Gödel.

Gödel showed that for any formal theory, there will always be undecidable propositions.

Mathematicians started to look for practical techniques (computation) for proving undecidability.

Gödel (1934):
Discusses some ideas for definitions of what functions/languages are "computable", but isn't confident what's a good definition.

Church (1936):
Invents lambda calculus, claims it should be the definition of "computable".

Gödel and Post (1936):
Argue that Church's definition isn't justified.

Meanwhile… a certain British grad. student in Princeton, unaware of all these debates…

In 1935 Alan Turing described a model of computation, known today as the Turing Machine (TM).

Alan Turing
(1936, age 22)

A problem P is *computable* (or *decidable*) if it can be solved by a Turing machine that halts on every input.

We say that P has an algorithm.

Turing Machines were adopted in the 1960's, years after his death.

Turing's Inspiration

Human writes symbols on paper

The paper is a sequence of squares

No upper bound on the number of squares

At most finitely many kinds of symbols

Human observes one square at a time

Human has only finitely many mental states

Human can change symbols and change

    focus to a neighboring square, but only

    based on its state and the symbol it observes

Human acts deterministically

High Level Example of a Turing Machine

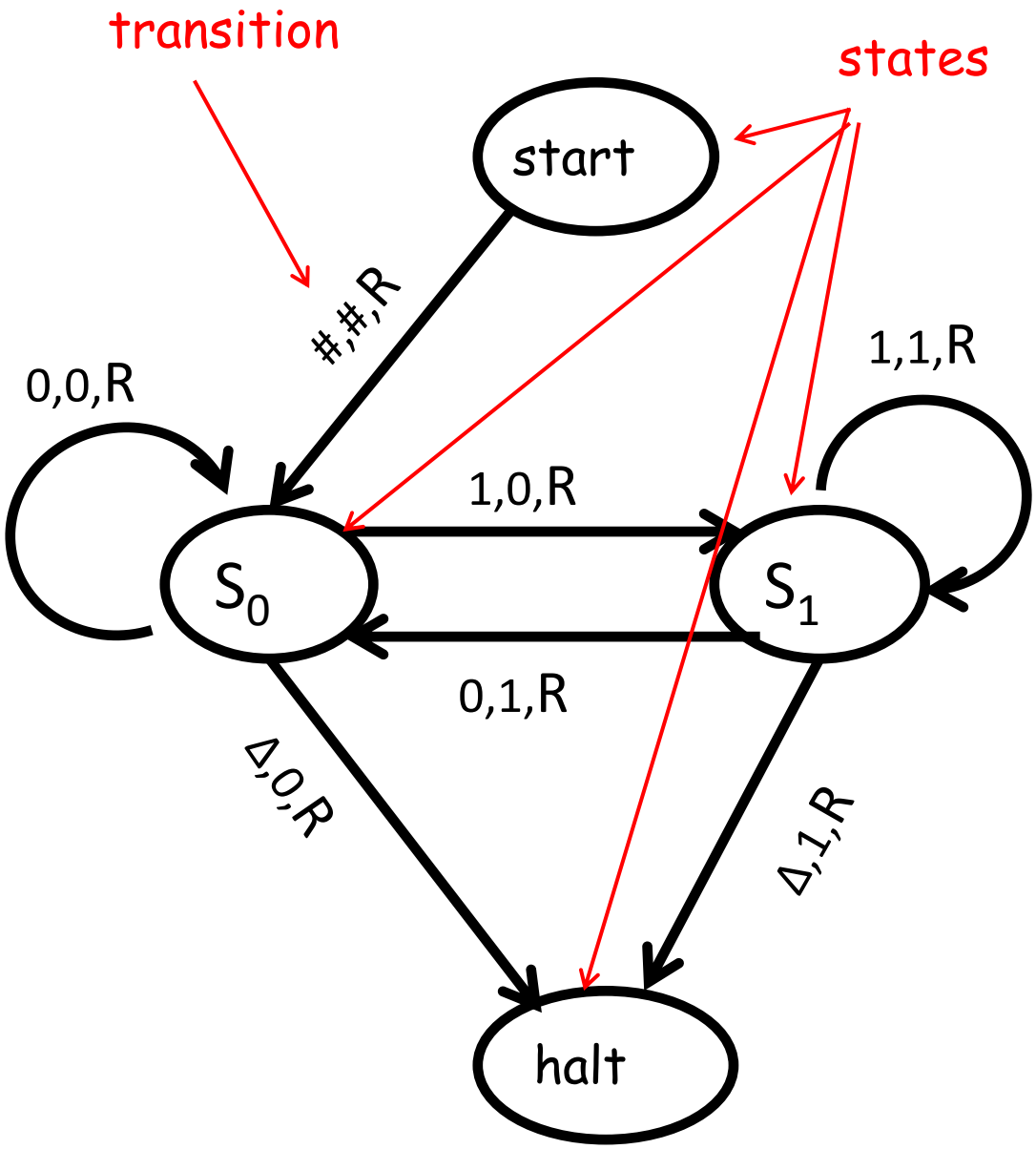
The machine that takes a binary string and appends 0 to the left side of the string.

Input: #10010Δ
Output: #**0**10010Δ

- leftmost char

Δ – rightmost char

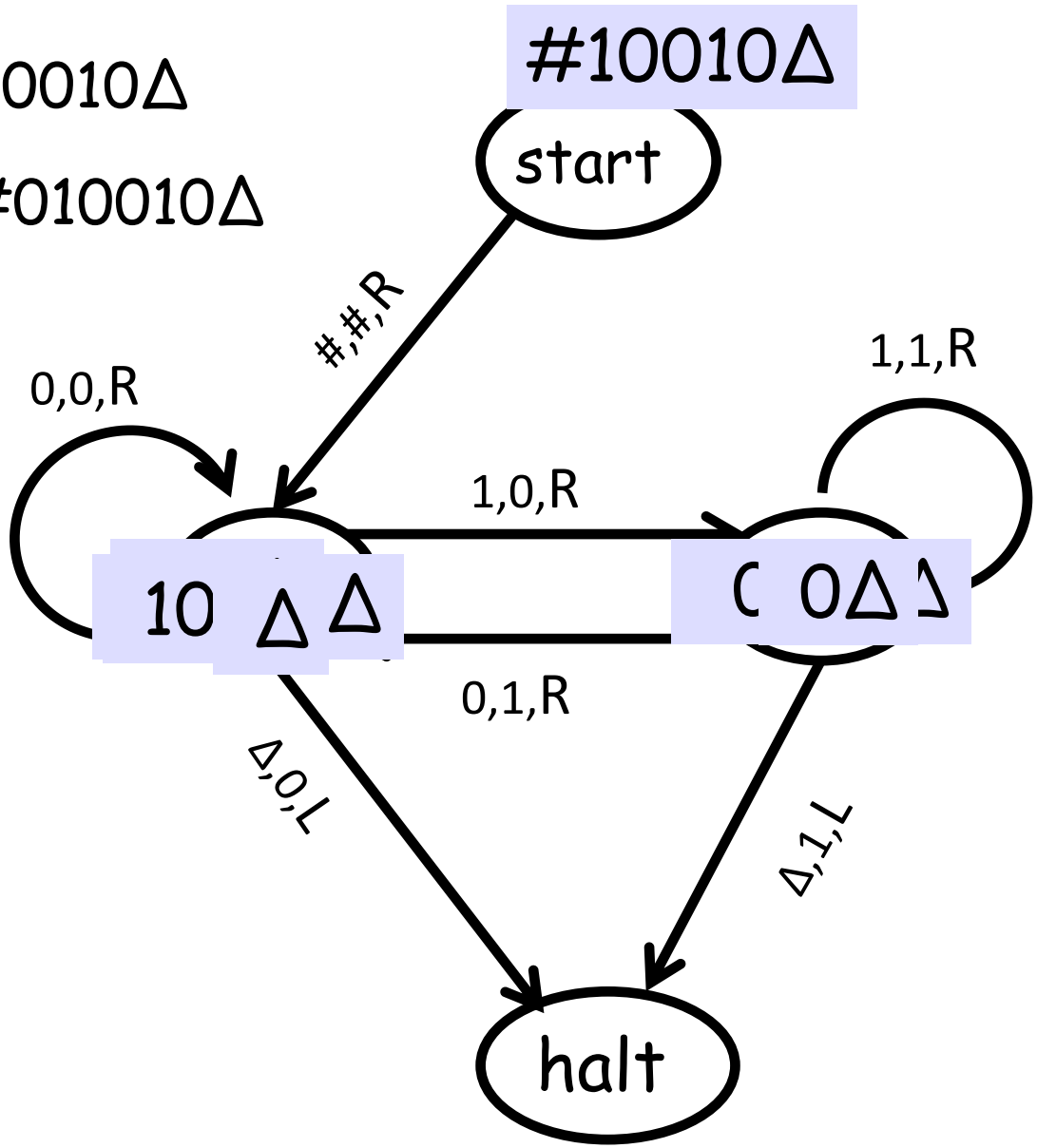
Transition on each edge
read,write,move (L or R)



Deterministic Turing Machine

Input: #10010Δ

Output: #010010Δ



The Church-Turing Thesis

"*Any natural / reasonable notion of computation can be simulated by a TM.*"

This is not a theorem.

Is it… …*an observation?*
…*a definition?*
…*a hypothesis?*
…*a law of nature?*
…*a philosophical statement?*

Everyone believes it.

No counterexample yet.

Super-Turing computation

In 1995 Prof. Hava Siegelmann proposed
Artificial Recurrent Neural Networks (ARNN).

She proved mathematically that ARNNs have computational powers that extend the TM.

She claims that ARNNs can "compute" Turing non-computable functions.

As of today, the statement is not proven nor disproven.

Runtime Complexity

A problem P is *decidable* if it can be solved by a Turing machine that always halts.

Let M be a Turing Machine that halts on all inputs.

Assume we compute the running time purely as a function of the length of the input string.

Definition: The running complexity is the function $f : N \to N$ such that f(n) is the maximum number of steps that M uses on any input of length n.

Complexity Classes

A fundamental complexity class P (or PTIME) is a class of decision problems that can be solved by a deterministic Turing machine in polynomial time.

A fundamental complexity class EXPTIME is a class of decision problems that can be solved by a deterministic Turing machine in $O(2^{p(n)})$ time, where p(n) is a polynomial.

Undecidable Problems

Undecidable means that there is no computer program that always gives the correct answer: it may give the wrong answer or run forever without giving any answer.

The halting problem is the problem of deciding whether a given Turing machine halts when presented with a given input.

Turing's Theorem: The Halting Problem is not decidable.

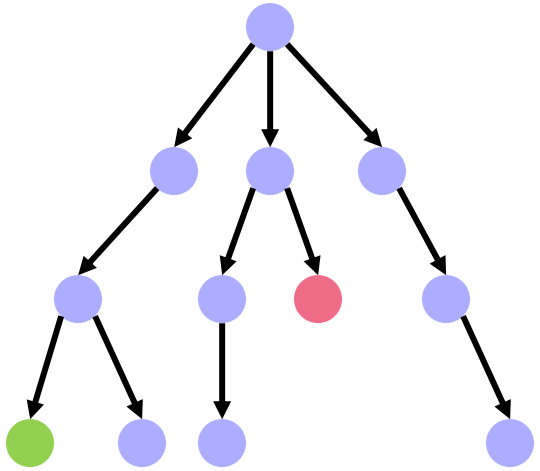
Why is the Halting Problem so important?
Because a lot of really practical problems are the halting problem in disguise.

Nondeterministic Turing Machine

The deterministic Turing machine means that there is only one valid computation starting from any given input. A computation path is like a linked list.

Nondeterministic Turing machine (NDTM) defined in the same way as deterministic, except that a computation is like a tree, where at any state, it's allowed to have a number of choices.

One way to visualize NDTM is that it makes an exact copy of itself for each available transition, and each machine continues the computation.



Complexity Class: NP

A fundamental complexity class NP is a class of decision problems that can be solved by a nondeterministic Turing machine in polynomial time.

Equivalently, the NP decision problem has a certificate that can be checked by a polynomial time deterministic Turing machine.

We will be using the last definition for proving NP completeness.

P versus NP

It has been proven that Nondeterministic TM can be simulated by Deterministic TM. Rabin & Scott in 1959 shown that adding non-determinism does not result in more powerful machine.

But how fast we can do that simulation?

The famous $P \neq NP$ conjecture, would answer that we cannot hope to simulate nondeterministic Turing machines very fast (in polynomial time).

P and NP complexity classes

non deterministic

P = set of problems that can be solved in polynomial time by a deterministic TM.

NP = set of problems for which solution can be verified in polynomial time by a deterministic TM.

Polynomial Reduction: $Y \leq_p X$

To reduce a decision problem Y to a decision problem X (we write $Y \leq_p X$) we want a function f that maps Y to X such that:

1) f is a polynomial time computable

2) $\forall y \in Y$ (y is instance of Y) is YES if and only if $f(y) \in X$ is YES.

We use this to prove NP completeness:

knowing that Y is hard, we prove that X is at least as hard as Y.

$Y \leq_p X$

If we can solve X in polynomial time,
we can solve Y in polynomial time.

Examples:

Bipartite Matching $\leq_p$ Max-Flow

Circulation $\leq_p$ Max-Flow

known to be hard ← $Y \leq_p X$

If we can solve X, we can solve Y.

The *contrapositive* of the statement "if A, then B" is "if not B, then not A."

If we cannot solve Y, we cannot solve X.

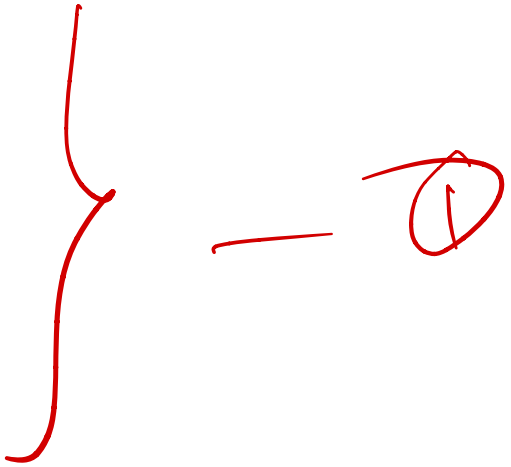
Knowing that Y is hard, we prove that X is harder.

In plain form: X is at least as hard as Y.

Two ways of using $Y \leq_p X$

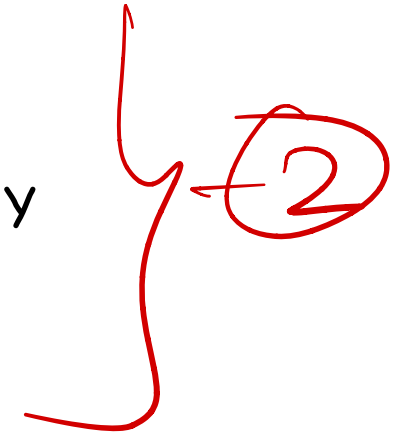
1) Knowing that X is easy

If we can solve X in polynomial time,
we can solve Y in polynomial time.



2) Knowing that Y is hard

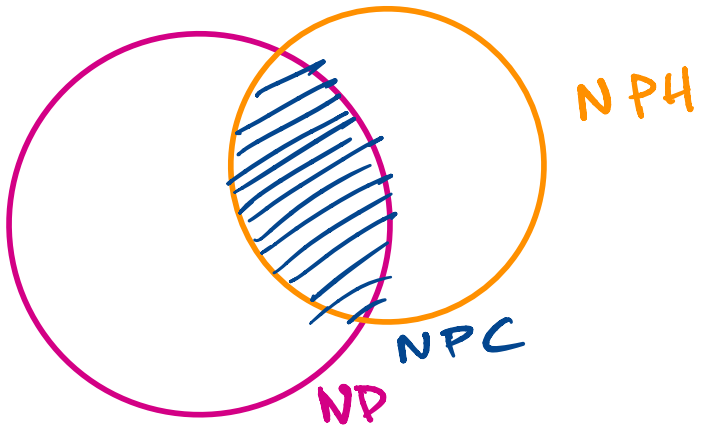
Then we can prove that X is at least as hard as Y



NP-Hard and NP-Complete

X is *NP-Hard*, if $\forall Y \in NP$ and $Y \leq_p X$.

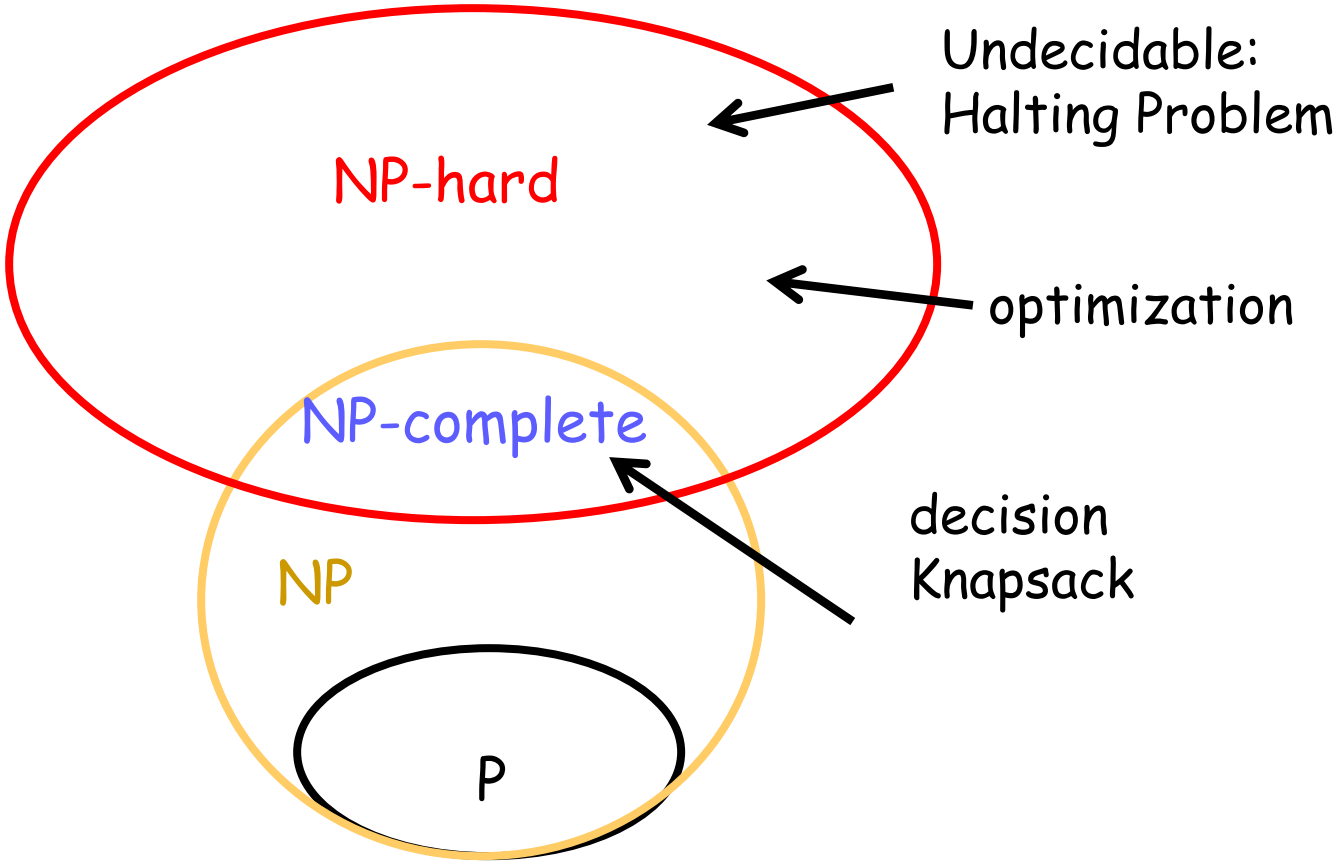
X is *NP-Complete*, if X is NP-Hard and $X \in NP$.



Venn Diagram (P ≠ NP)

NPH problems do not have to be in NP.

NPC problems are the most difficult NP problems.



It's not known if NPC problems can be solved by a *deterministic* TM in polynomial time.

Imagine P = NP → NPC = P
NPH = NPH

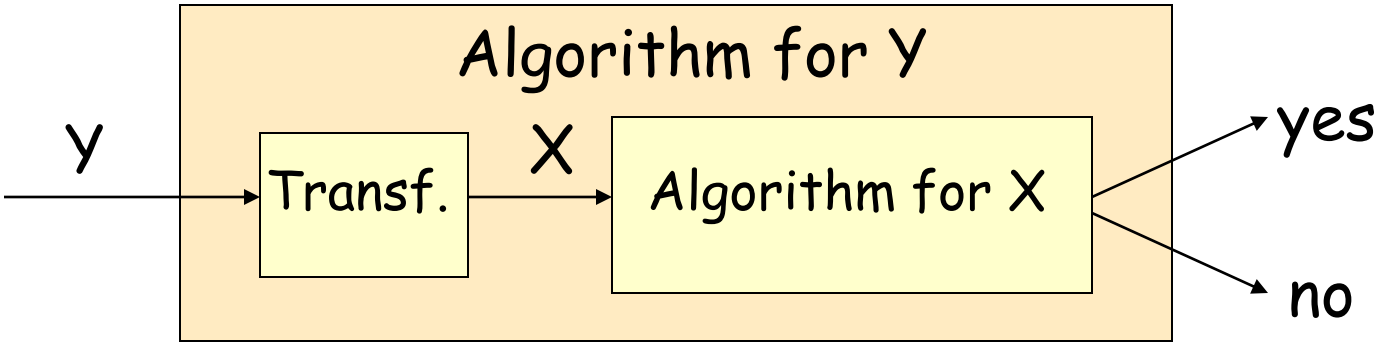
NP-Completeness Proof Method

To show that X is NP-Complete:

1) Show that X is in NP

2) Pick a problem Y, known to be an NP-Complete

3) Prove $Y \leq_p X$ (reduce Y to X)



Boolean Satisfiability Problem (SAT)

A propositional logic formula is built from variables, operators AND (conjunction, $\wedge$), OR (disjunction, $\vee$), NOT (negation, $\neg$), and parentheses:

$$(X_1 \vee \neg X_3) \wedge (X_1 \vee \neg X_2 \vee X_4 \vee X_5) \wedge \ldots \rightarrow \text{True}$$

A formula is said to be satisfiable if it can be made TRUE by assigning appropriate logical values (TRUE, FALSE) to its variables.

A formula is in conjunctive normal form (CNF) if it is a conjunction of clauses.
A literal is a variable or its negation.
A clause is a disjunction of literals.

what does each thing mean here?

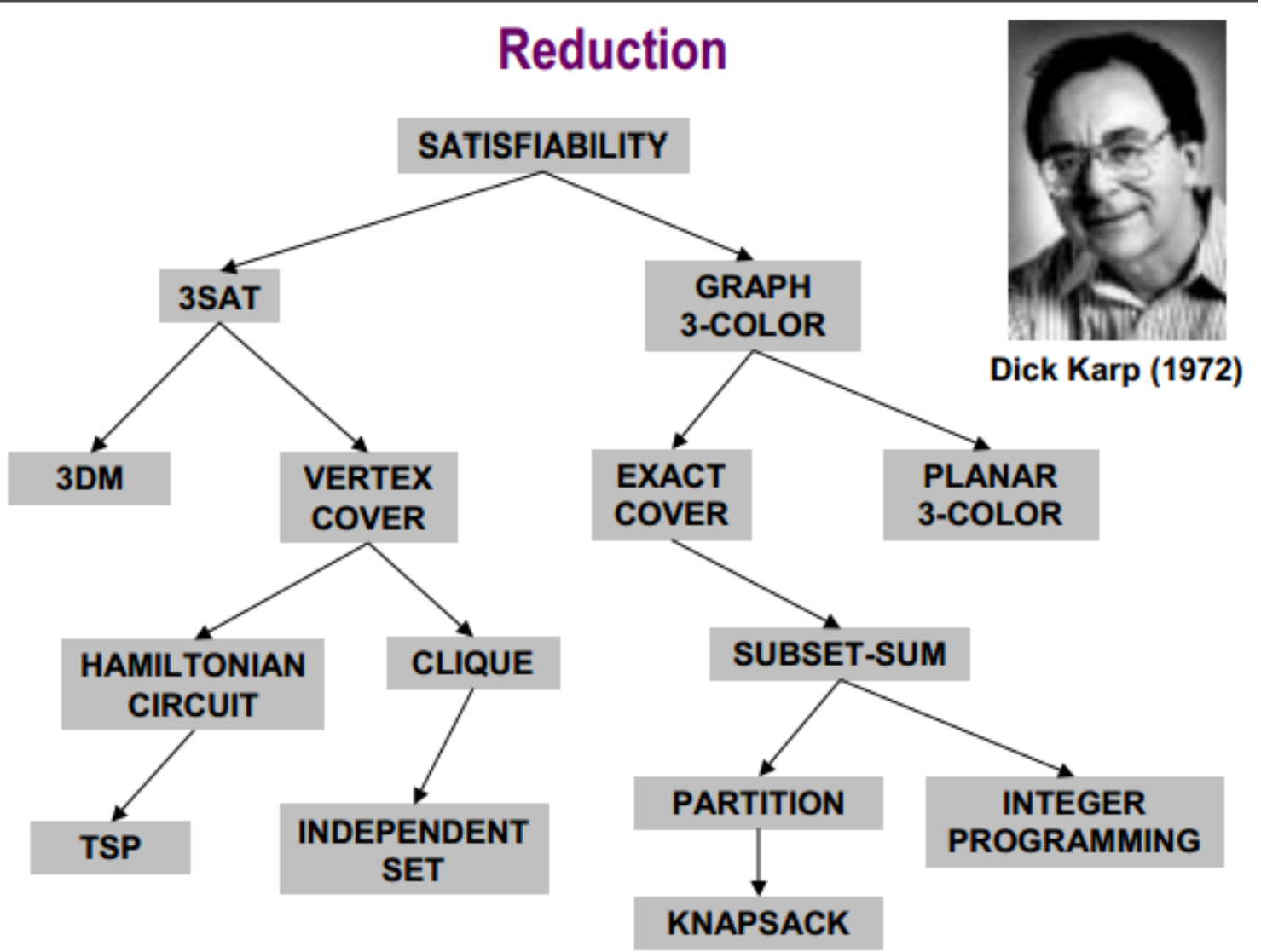
Cook-Levin Theorem (1971)

Theorem. CNF SAT is NP-complete.

The proof of this theorem is outside of the scope of this course.

Cook received a Turing Award for his work.

You are not responsible for knowing the proof.



Dick Karp (1972)

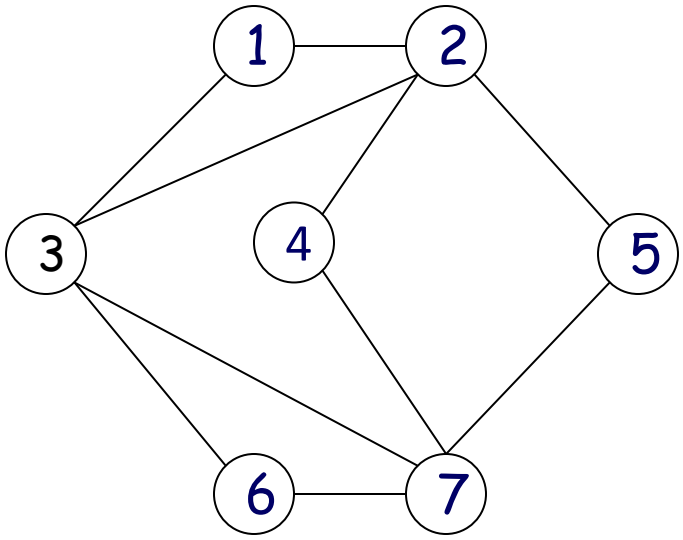
Karp introduced the now standard methodology for proving problems to be NP-Complete.

He received a Turing Award for his work (1985).

Independent Set

Given a graph, we say that a subset of vertices is "independent" if no two of them are joined by an edge.

IS = {1}
= {3, 4}
= {1, 4, 5}



The maximum independent set problem, MaxIndSet, asks for the size of the largest independent set in a given graph.

Independent Set

Optimization Version (MaxIndSet):
Given a graph, find the largest independent set. ⇒ NP-Hard

Decision Version (IndSet): ⇒ NP-Complete
Given a graph and a number k, does the graph contains an independent set of size k?

Optimization vs. Decision

Optimization vs. Decision Problems

OPT → decision

If one can solve an optimization problem (in polynomial time), then one can answer the decision version (in polynomial time)

decision → OPT

Conversely, by doing binary search on the bound b, one can transform a polynomial time answer to a decision version into a polynomial time algorithm for the corresponding optimization problem

In that sense, these are essentially equivalent.
However, they belong to two different complexity classes.

Independent Set is NP Complete

Given a graph and a number k, does the graph contains an independent set of size k?

Is it in NP?

We need to show we can verify a solution in polynomial time.

Given a set of vertices, we can easily count them and then verify that any two of them are not joined by an edge.

Independent Set is NP Complete

Given a graph and a number k, does the graph contains an independent set of size k?

Is it in NP-hard?

We need to pick Y such that $Y \leq_p$ IndSet for $\forall\, Y \in NP$

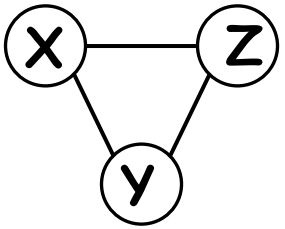
Reduce from 3-SAT.

3-SAT is SAT where each clause has at most 3 literals.

3SAT $\leq_p$ IndSet

We construct a graph G that will have an independent set of size k iff the 3-SAT instance with k clauses is satisfiable.

For each clause (X ∨ Y ∨ Z) we will be using a special gadget:



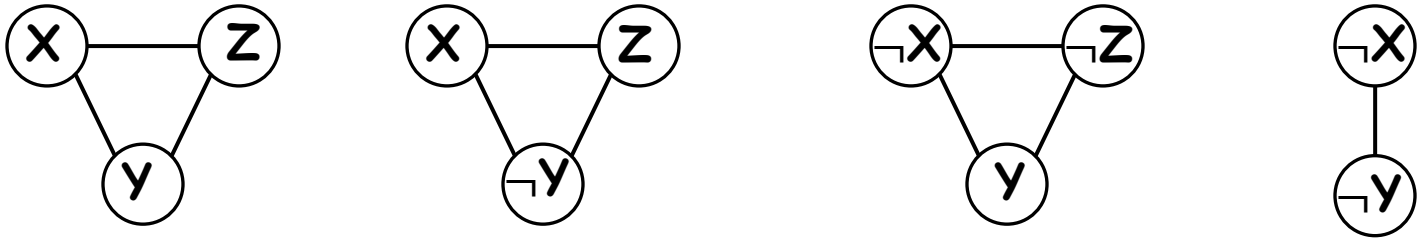
Next, we need to connect gadgets.

As an example, consider the following instance:

(X ∨ Y ∨ Z) ∧ (X ∨ ¬Y ∨ Z) ∧ (¬X ∨ Y ∨ ¬Z) ∧ (¬X ∨ ¬Y)

3SAT $\leq_p$ IndSet

$(X \vee Y \vee Z) \wedge (X \vee \neg Y \vee Z) \wedge (\neg X \vee Y \vee \neg Z) \wedge (\neg X \vee \neg Y)$

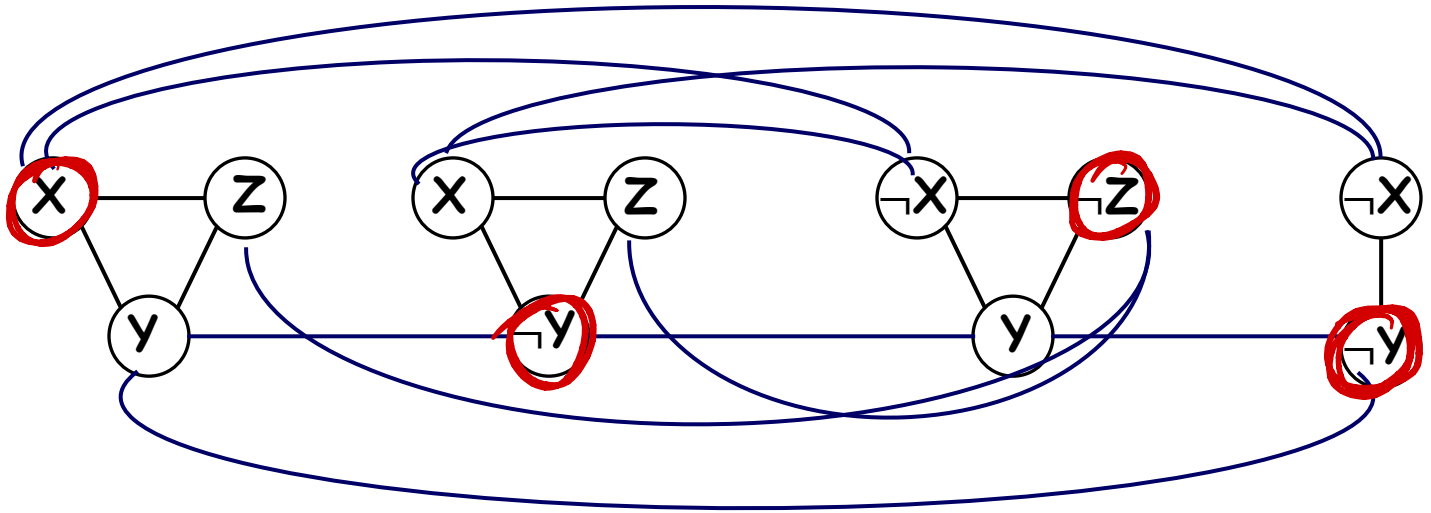


How do we connect gadgets?

<u>Claim:</u>

3SAT $\leq_p$ IndSet

$(X \vee Y \vee Z) \wedge (X \vee \neg Y \vee Z) \wedge (\neg X \vee Y \vee \neg Z) \wedge (\neg X \vee \neg Y)$



Proof. $\Rightarrow$) Given 3SAT satisfiable

For example, $X = T$

$\neg Y = T$

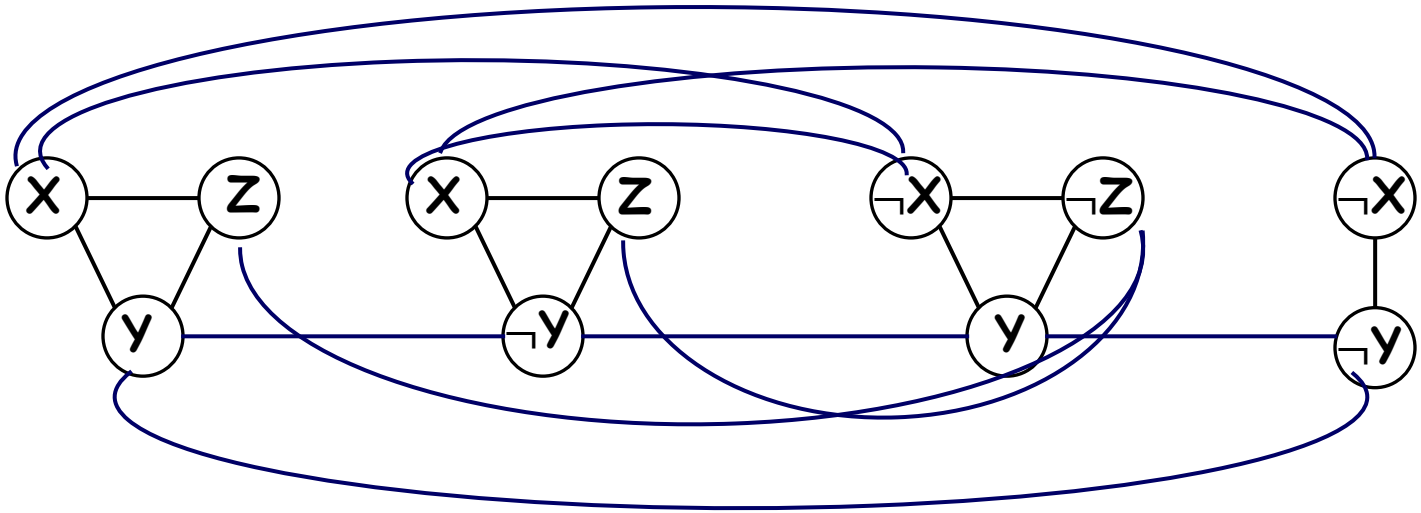
$\neg Z = T$

Goal: set on IS. true literals make an IS.

3SAT $\leq_p$ IndSet

$(X \vee Y \vee Z) \wedge (X \vee \neg Y \vee Z) \wedge (\neg X \vee Y \vee \neg Z) \wedge (\neg X \vee \neg Y) = T$

T T T T



Proof. $\Leftarrow$) Given a triangular graph G $|IS(G)| = k$

Goal: to get an assignment

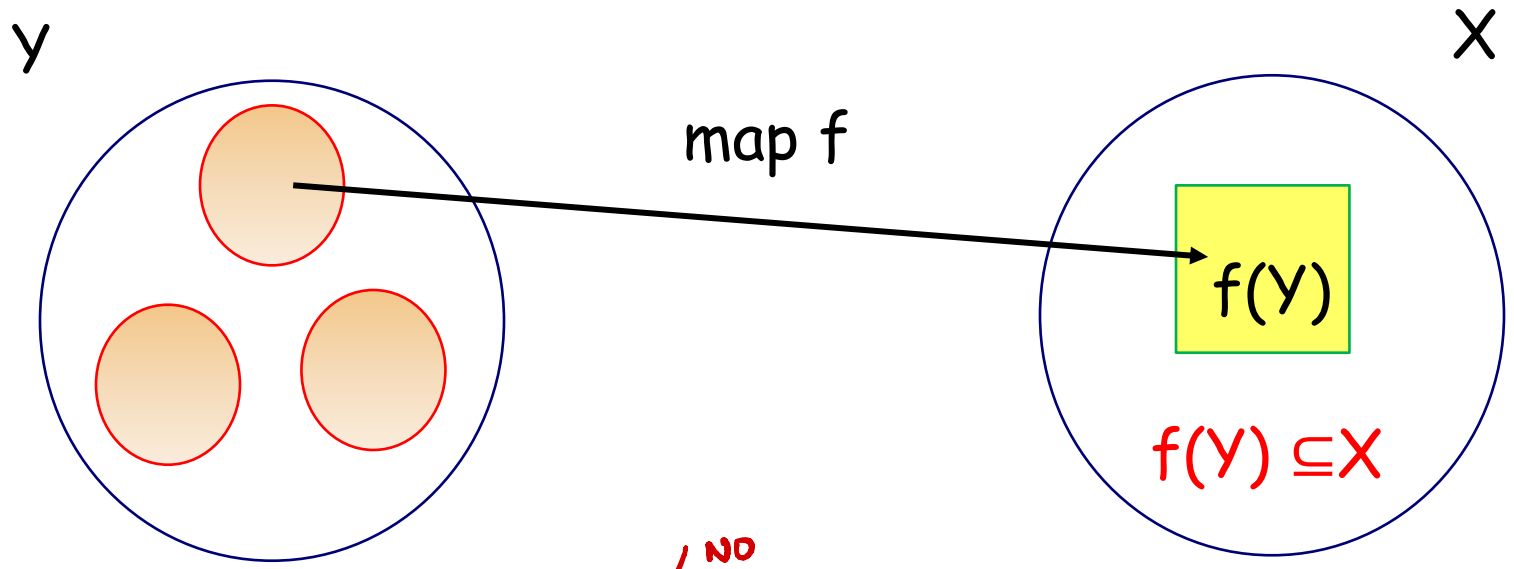
$X = T, \neg Z = T, \neg Y = T$

The confusing point is that the reduction $Y \leq_p X$ only "works one way", but the correctness proof needs to "work both ways".

The correctness proofs are not actually symmetric.

The proof needs to handle arbitrary instances of Y, but only needs to handle the special instances of X produced by the reduction.

This asymmetry is the key to understanding reductions.



$(Y \leq_p X) \not\rightarrow (X \leq_p Y)$ NO

3SAT $\leq_p$ IndSet

Reduction from 3SAT to IndSet consists of three parts:

• we transform an arbitrary CNF formula into a special graph G and a specific integer k, in polynomial time.

• we transform an arbitrary satisfying assignment for 3SAT into an independent set in G of size k.

• we transform an arbitrary independent set (in G) of size k into a satisfying assignment for 3SAT.

The General Pattern $Y \leq_p X$

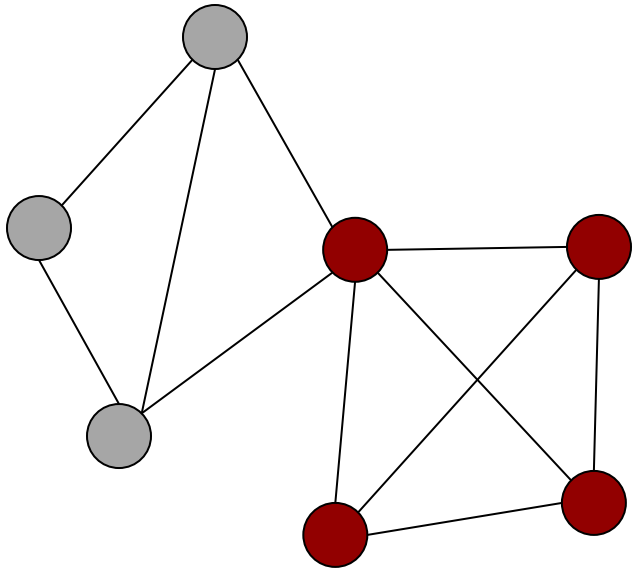
1. Describe a polynomial-time algorithm to transform an arbitrary instance of Y into a special instance of X.
2. Prove that if an instance of Y is True, then an instance of X is True.
3. Prove that if an instance of X is True, then an instance of Y is True. (This part causes the most trouble.)

Clique

A clique is another name for a complete graph, that is, a graph where every pair of vertices is connected by an edge.
The MaxClique problem asks for the number of vertices in its largest complete subgraph in a given graph

We can prove that MaxClique is NP-hard using the following easy reduction from IndSet.

$$\text{MaxClique} \leq_p \text{MaxIndSet}$$

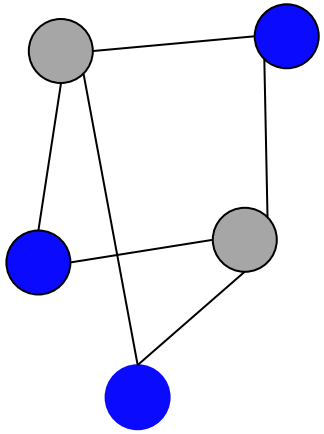


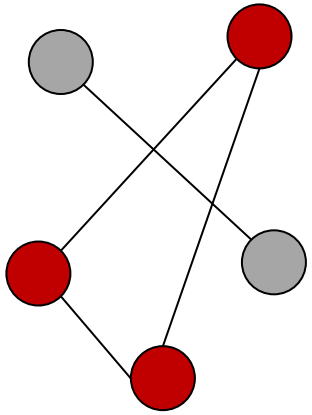
MaxClique $\leq_p$ MaxIndSet

Any graph G=(V, E) has a complement graph G'=(V, E'), where edge (u, v)∈E' if and only if (u, v)∉E. In other words, G∪G' is a complete graph.

Consider the largest Independent Set in G.

Consider the largest Clique in G'.





The largest independent in G has the same vertices (and thus the same size) as the largest clique in the complement of G.

Analysis of Algorithms

V. Adamchik
Lecture 12
Fall 2023

CSCI 570
University of Southern California

NP-Completeness - II

Reading: chapter 9

In 1936 Alan Turing described:

- A simple formal model of computation now known as Turing machines.
- A proof that TM can NOT solve the halting problem.
- A proof that NO Turing machine can determine whether a given proposition is provable from the axioms of first-order logic.
- Compelling arguments that a problem not computable by a Turing machine is not "computable" in the absolute (human) sense.
- A non-deterministic Turing machine: for each state it makes an arbitrary choice between a finite of possible transitions.

Non-Deterministic Turing Machine

- NDTM is a choice machine: for each state it makes an arbitrary choice between a finite (possibly zero) number of states.
- The computation of a NDTM is a tree of possible configuration paths.
- One way to visualize NDTM is that it makes an exact copy of itself for each available transition, and each machine continues the computation.
- Rabin & Scott in 1959 shown that adding non-determinism does not result in more powerful machine.
- For any NDTM, there is a DTM that accepts and rejects exactly the same strings as NDTM.
- P vs. NP is about whether we can simulate NDTM in polynomial time.

Complexity Classes

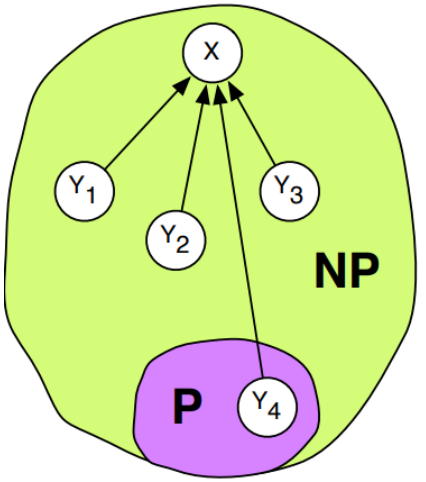
P = set of problems that can be solved in polynomial time by a DTM.

NP = set of problems that can be solved in polynomial time by a NDTM.

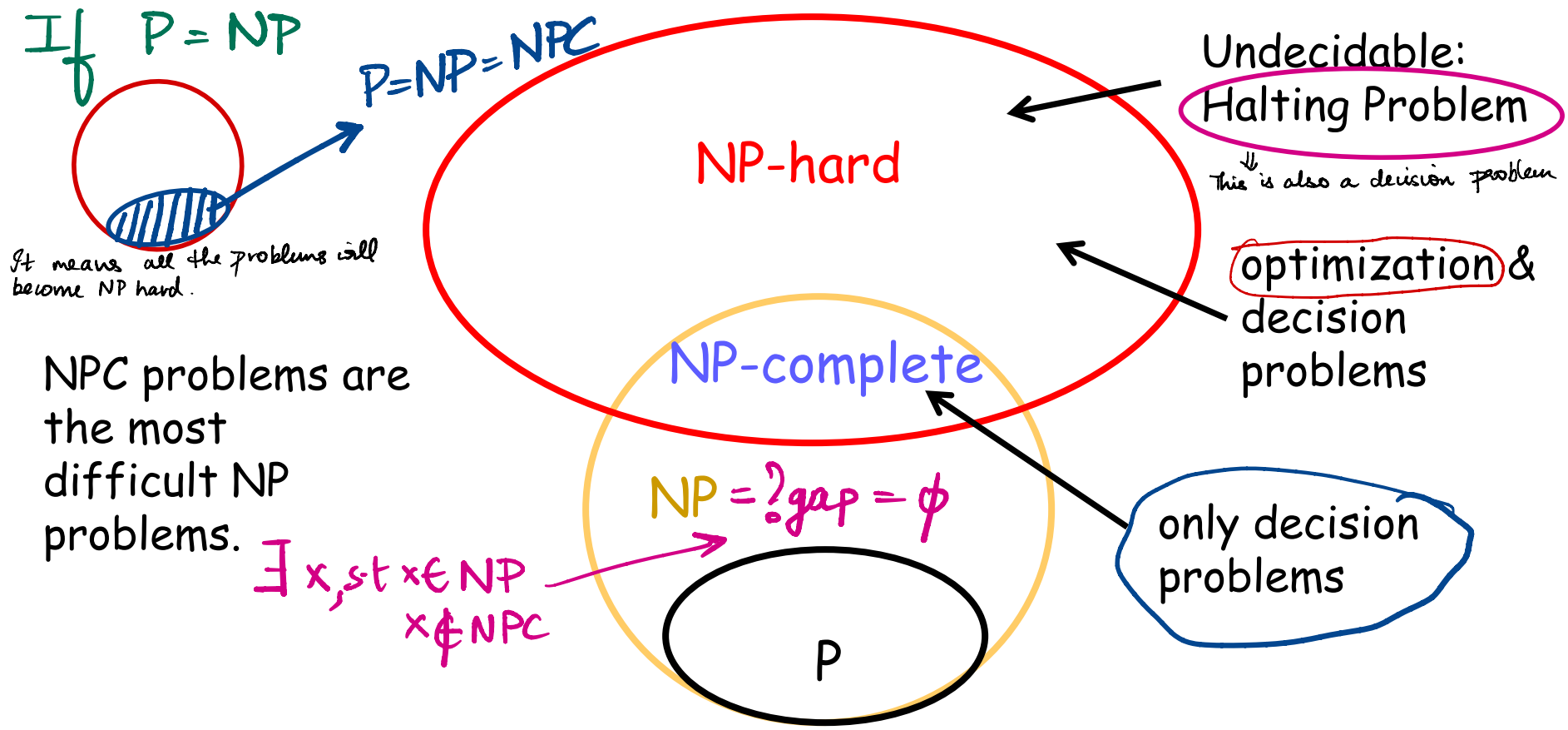
NP = set of problems for which solution can be verified in polynomial time by a deterministic TM.

X is *NP-Hard*, if $\forall Y \in NP$ and $Y \leq_p X$.

X is *NP-Complete*,
if X is NP-Hard and $X \in NP$.



Venn Diagram (P ≠ NP)



NPC problems are the most difficult NP problems.

It's not known if NPC problems can be solved by a *deterministic* TM in polynomial time.

NPC problems can be solved by a *non-deterministic* TM in polynomial time.

Victors Additional True / False Questions:

1. NP complete has all decision problems
   ⇒ TRUE!!
2. ALL decision problems belong to NP complete
   ⇒ FALSE!!!

NP-Complete Problems

Cook-Levin Theorem: CNF SAT is NP-complete.

Independent Set:

Given graph G and a number k, does G contain a set of at least k independent vertices?

Vertex Cover:

Given a graph G and a number k, does G contain a vertex cover of size at most k.

A Hamiltonian cycle:

Given a graph G, does G contain a cycle that visits each vertex exactly once.

NP-Completeness Proof Method

To show that X is NP-Complete:

1) Show that X is in NP
2) Pick a problem Y, known to be an NP-Complete
3) Prove $Y \leq_p X$ (reduce Y to X)

Step to follow in exam

In lecture 11 we have proved that Independent Set is NP-Complete by reduction from 3-SAT ($3SAT \leq_p IndSet$)

Reduction from 3SAT to IndSet consists of three parts:

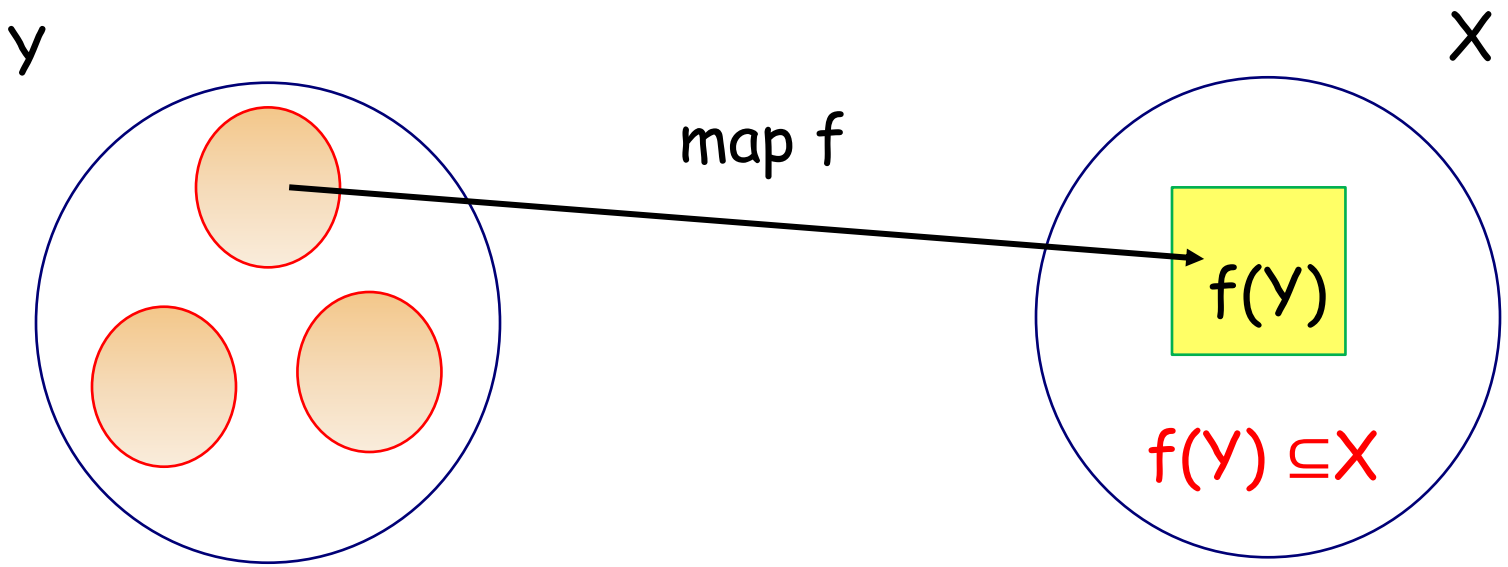
- we transform an arbitrary CNF formula into a special graph G and a specific integer k, in polynomial time.
- we transform an arbitrary satisfying assignment for 3SAT into an independent set in G of size k.
- we transform an arbitrary independent set (in G) of size k into a satisfying assignment for 3SAT.

The confusing point is that the reduction $Y \leq_p X$ only "works one way", but the correctness proof needs to "work both ways".

The correctness proofs are not actually symmetric.

The proof needs to handle arbitrary instances of Y, but only needs to handle the special instances of X produced by the reduction.

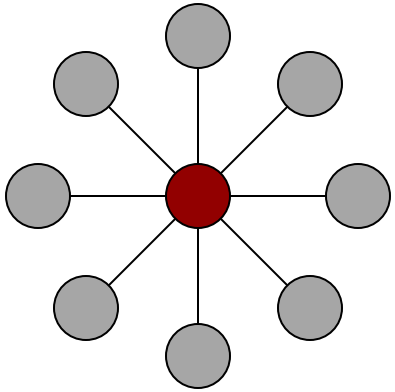
This asymmetry is the key to understanding reductions.



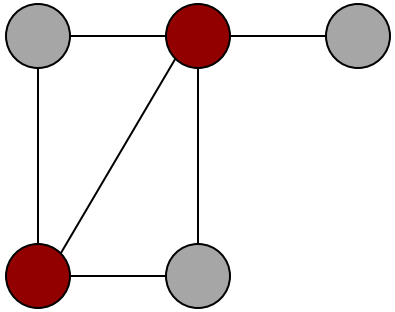
Vertex Cover

Given G=(V,E), find the smallest S⊂V s.t. every edge is incident to vertices in S.

VC ∪ IS = V



The minimum vertex cover problem, MinVC, asks for the size of the smallest vertex cover in a given graph.



Vertex Cover

Theorem: *for a graph G=(V,E), S is an independent set if and only if V-S is a vertex cover*

Proof. $\Rightarrow$) S is an IS

① $x \in IS$, then $y \notin IS$
then $y \in VC$

$\forall$ edge

② $y \in IS$, $\Rightarrow x \in IS \Rightarrow x \in VC$

③ $x, y \notin IS \Rightarrow x, y \in VC$

Vertex Cover

Theorem: *for a graph G=(V,E), S is a independent set if and only if V-S is a vertex cover*

Proof. ⇐) Given a VC

Pick $\forall x, y$ s.t. $x \notin VC$, $y \notin VC$

s.t. = such that

is it possible? ⇒ Ask yourself

edge (x,y) − ? ⇒ This means that there is no edge between x & y

It follows, $x, y \in IS$

⇓

This is because there is no edge between x & y

I THINK VICTOR MADE A MISTAKE ON HIS VIDEO

Min Vertex Cover in NP-Hard

MaxIndSet $\leq_p$ MinVC

By the previous theorem.

Vertex Cover in NP-Complete

Claim: *a graph G=(V,E) has an independent set of size at least k if and only if G has a vertex cover of size at most V-k.*

Ind. Set $\leq_p$ Vertex Cover

By the previous theorem.

Discussion Problem 1

Show that vertex cover remains NP-Complete even if the instances are restricted to graphs with only <u>even</u> degree vertices. Let us call this problem VC-even.

Prove: $VC \leq_p$ VC-even

VC-even: Given graph G with even degrees and number k

① VC-even $\in$ NP, prove it!! ⇒ Given a solution for the VC-even problem, the solution can be verified & validated in polynomial time. Hence, VC $\in$ NP

② VC-even $\in$ NP-Hard:

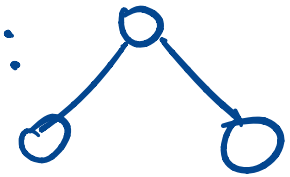
P.T.O

Given a graph with even degree, there are even no of vertices with odd degrees.

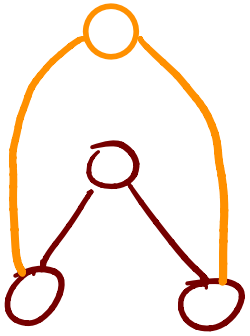
$VC \leq_p$ VC-even

Let graph G be the following:

G:



G':



Construct G' with all vertices having even degrees.

claim: $VC(G) = K$ iff $VC(G') = K+1$

Proof:

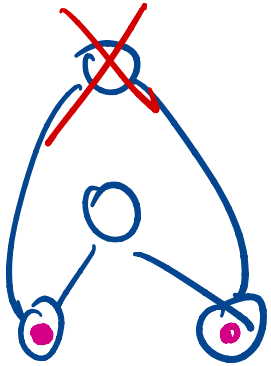
$\Rightarrow$ by construction, $VC(G') = VC(G) + 1$
$= K + 1$

$\Leftarrow$ Given $VC(G') = K+1$

remove a vertex!

P.T.O

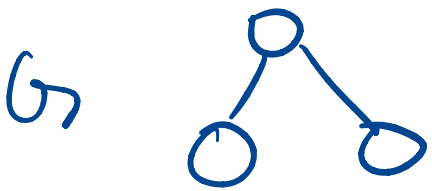
$VC \leq_p$ VC-even

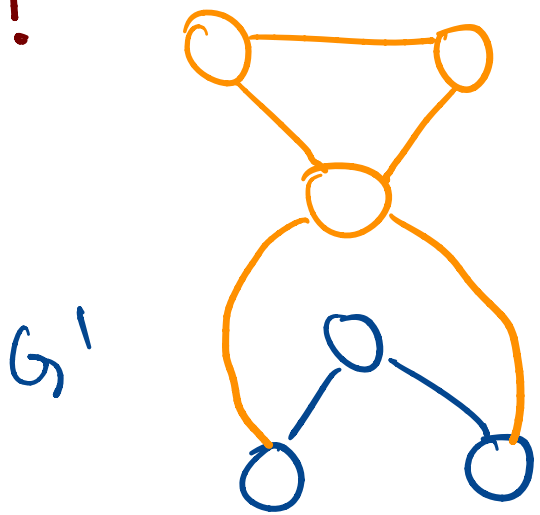


so, removing $\forall$ vertex does not mean that the VC changes.

Our construction is wrong!!

Neo construction





$VC \leq_p$ VC-even

at most

Claim: $VC(G) = k \iff VC(G') = k+2$

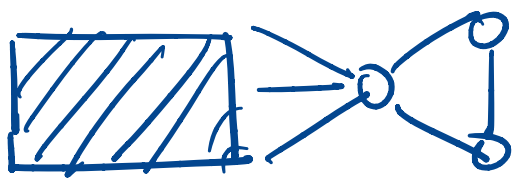
Proof

$\Rightarrow$ By construction

$\Leftarrow$ Given a $VC(G') = k+2$

Remove a triangle

G'



In such a case removing a triangle from G' will give you $VC(G) = k$

Hamiltonian Cycle Problem

A Hamiltonian cycle (HC) in a graph is a cycle that visits each vertex exactly once.

Problem Statement:
Given a directed or undirected graph $G = (V,E)$. Find if the graph contains a Hamiltonian cycle.

We can prove it that HC problem is NP-complete by reduction from SAT, but we won't.

Discussion Problem 2

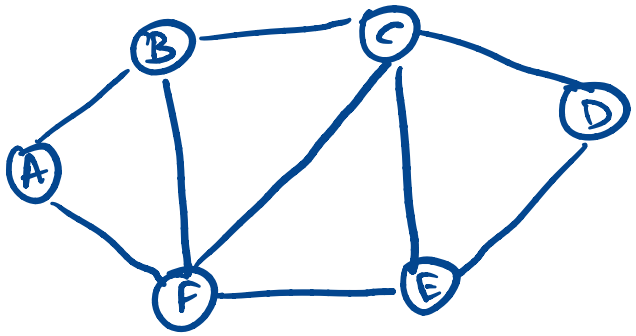
Assuming that finding a Hamiltonian Cycle (HC) in a graph is NP-complete, prove that finding a Hamiltonian Path is also NP-complete. HP is a path that visits each vertex exactly once and isn't required to return to its starting point.

① HP $\in$ NP, given a hamiltonian path we can verify its validity in polynomial time by traversing that path.

② HP $\in$ NPH $(HC \leq_p HP)$

a) construct a polynomial mapping
b) make a claim
c) prove the claim in both directions

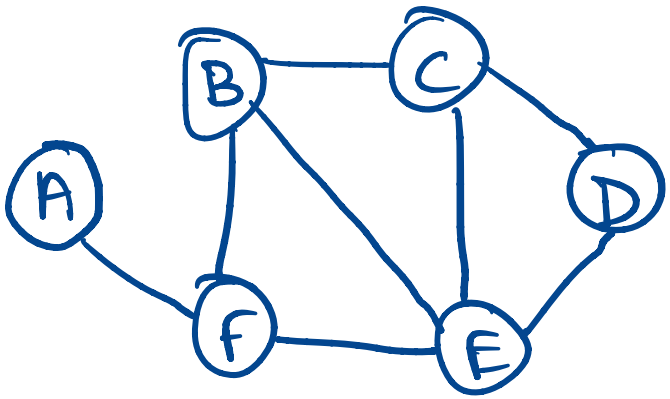
G:



$HC(G) = AFEDCBA$

Construct G'

G'



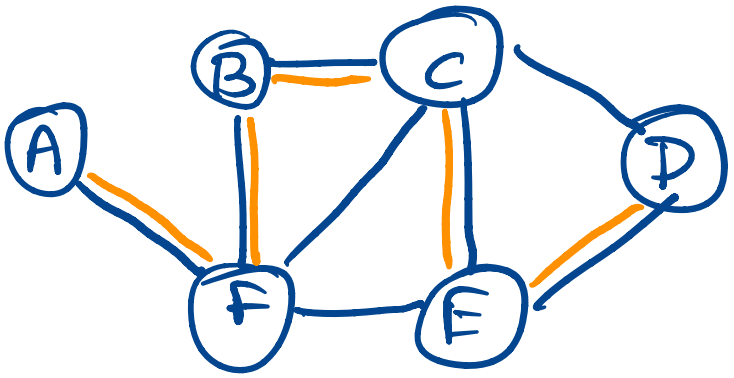
Claim : G has a HC $\iff$ G' has a HP

Proof:

$\Rightarrow$ By construction, if there is a given hamiltonian cycle then there should exist a hamiltonian path.

$\Leftarrow$ Given a hamiltonian path $HC(G')$

Goal: get a HC in G

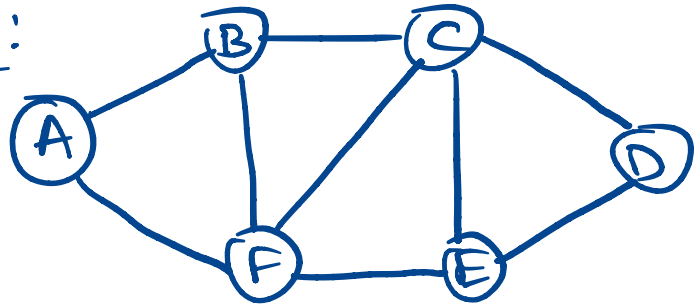


We cannot do anything to this to get a Hamiltonian cycle,

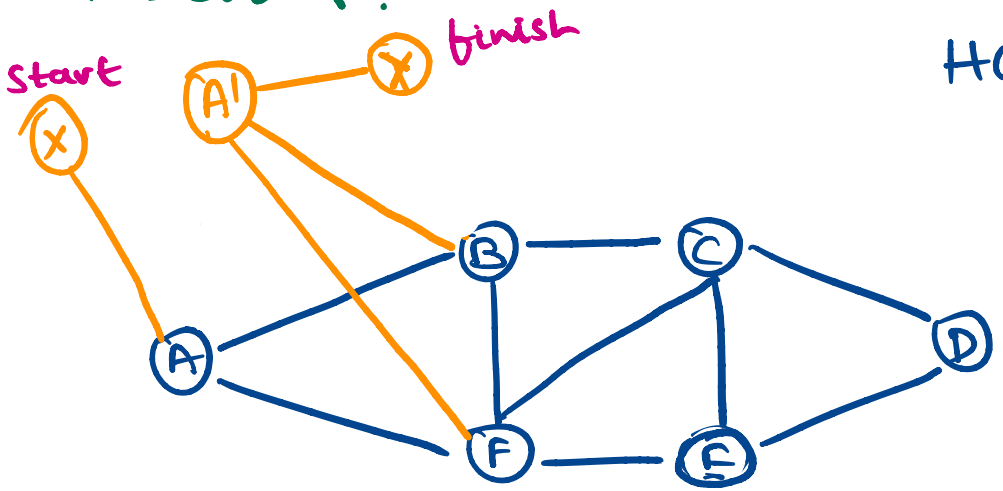
WRONG CONSTRUCTION!

P.T.O

G:



Construction :



HC(G) : A F E D C B A

HP(G') : X A F E D C B A' Y

vertex X : any vertex V

vertex A' : adjacent to V

vertex Y : adjacent to A'

P.T.O

Claim: G has a HC $\Leftrightarrow$ G' has a HP

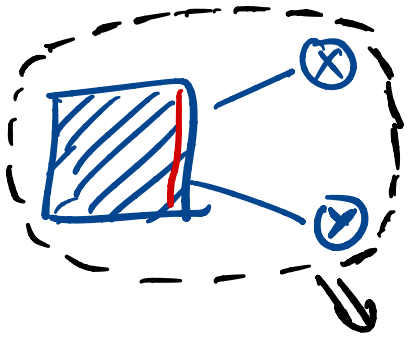
Proof

$\Rightarrow$ By construction, HP

ASK ANANNYA

$\Leftarrow$ Given a hamiltonian path HP(G')

Goal: HC(G)

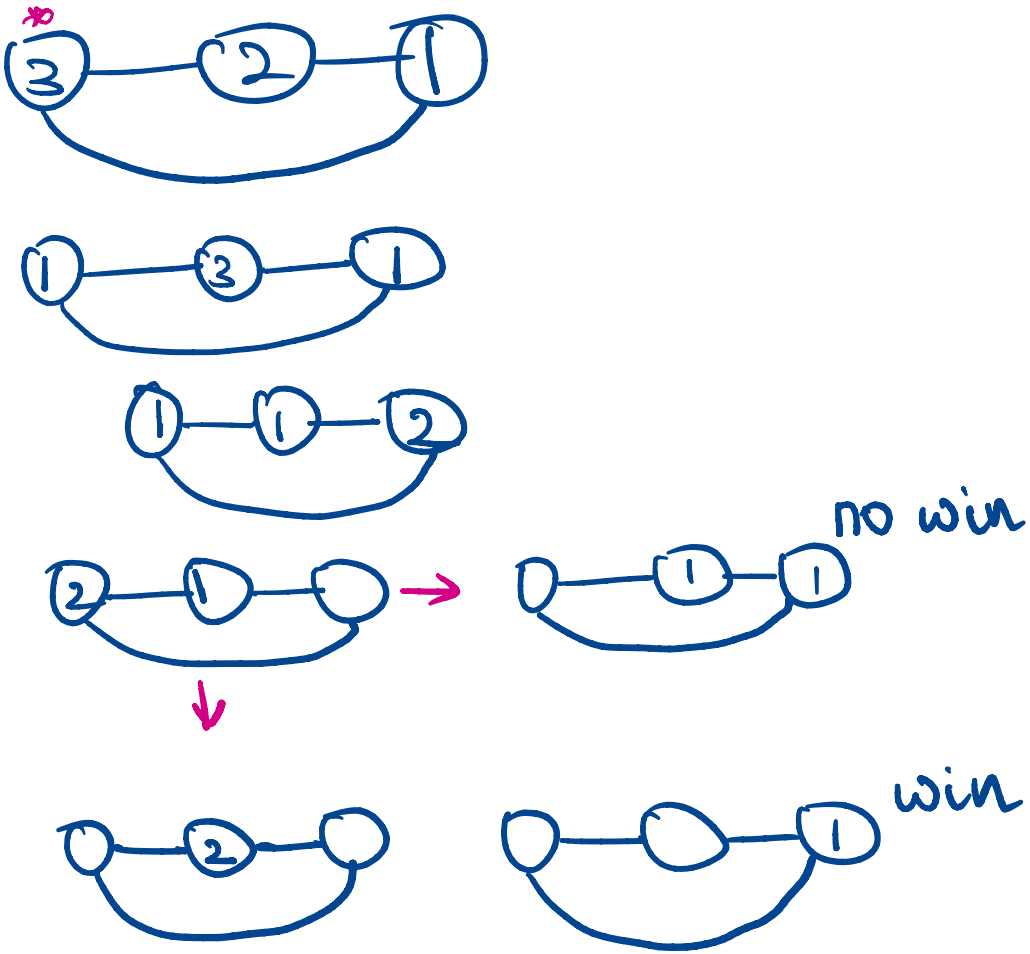


We need to have the A' vertex in the above node to guarantee that given a graph has a hamiltonian path, there exists a path thats creates an HC.

Discussion Problem 3

You are given an undirected graph G = (V, E) and for each vertex v, you are given a number p(v) that denotes the number of pebbles placed on v. We will now play a game where the following move is the only move allowed. You can pick a vertex u that contains at least two pebbles, and remove two pebbles from u and add one pebble to an adjacent vertex. The objective of the game is to perform a sequence of moves such that we are left with exactly one pebble in the whole graph. Show that the problem of deciding if we can reach the objective is NP-complete. Reduce from the Hamiltonian Path problem.

$HP \leq_p$ pebbles

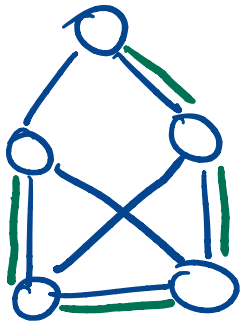


① Pebbles $\in NP$, Given the final result board it to polynomial time to check how many pebbles exist in the solution.

② $HP \leq_p$ Pebbles

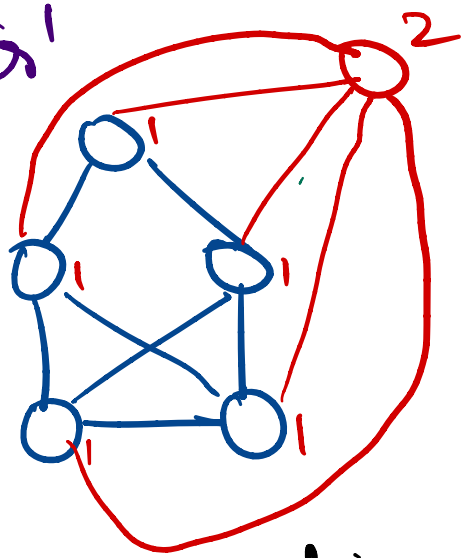
Construction

G



convert into pebbly game by assigning pebbles to each vertex →

G'



We have created this construction intentionally to prove our point. Constructing like this will force us to make a single move & no cycles.

Claim: G has a HP $\iff$ G' has a winning sequence

$\Rightarrow$ Given : G has a HP

Goal : Find a winning sequence

Example

2 1 1 1 1 1

0 2 1 1 1 1

0 0 2 1 1 1

0 0 0 2 1 1

0 0 0 0 2 1

0 0 0 0 0 2

0 0 0 0 1 0

follow the path

$\Leftarrow$ Given : G' has a winning sequence

Goal: Find HP in G.

We wont visit the same vertex twice, except the last move, hence this proves that there exists a HP. & Hence NP-Hard.

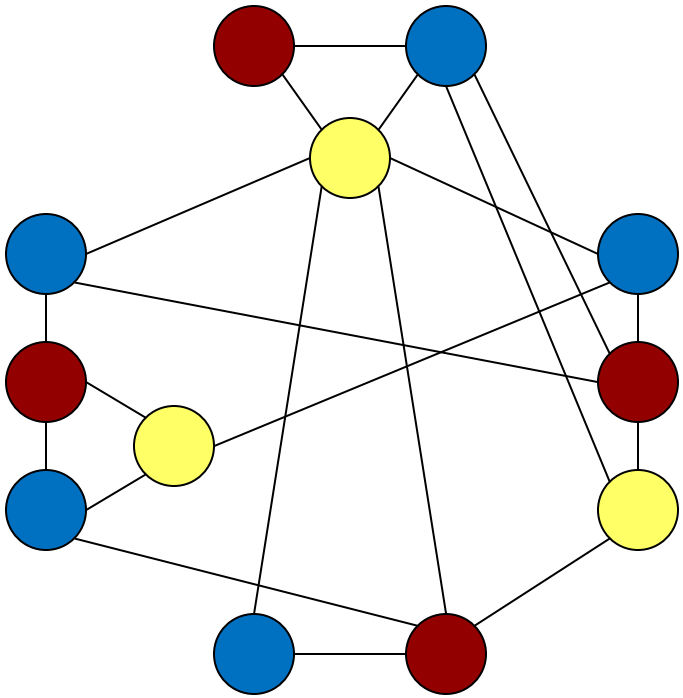
Graph Coloring

Given a graph, can you color the nodes with $\leq k$ colors such that the endpoints of every edge are colored differently?

① Planar, $K = 4$

② $K = 2$

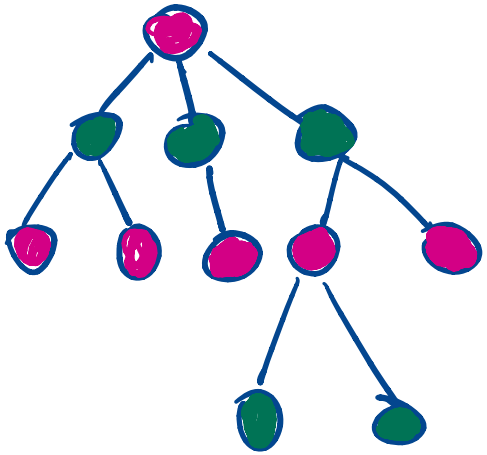
Theorem. (k>2)
k-Coloring is NP-complete.

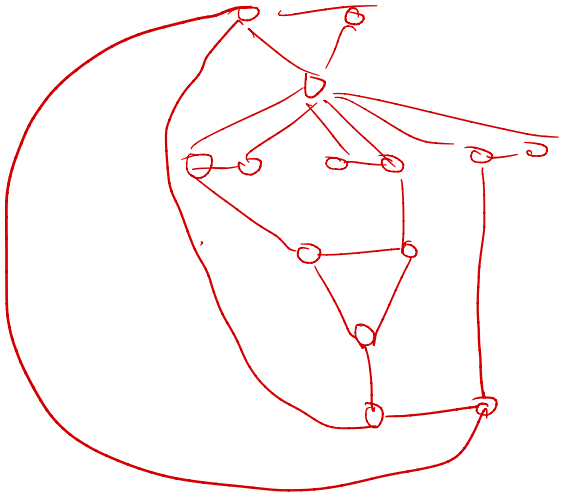


Graph Coloring: k = 2

How can we test if a graph has a 2-coloring?

BFS



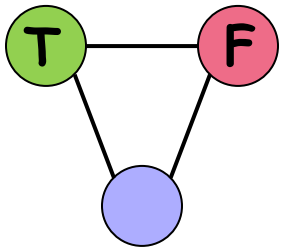


3-SAT $\leq_p$ 3-colorable

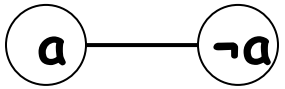
We construct a graph G that will be 3-colorable iff the 3-SAT instance is satisfiable.

Graph G consists of the following gadgets.

A truth gadget:

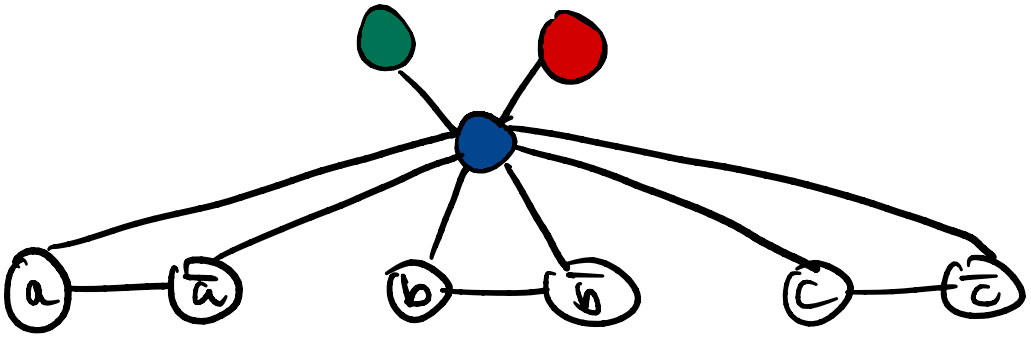


A gadget for each ~~variable~~ literal:



3-SAT $\leq_p$ 3-colorable

Combining those gadgets together (for three literals)

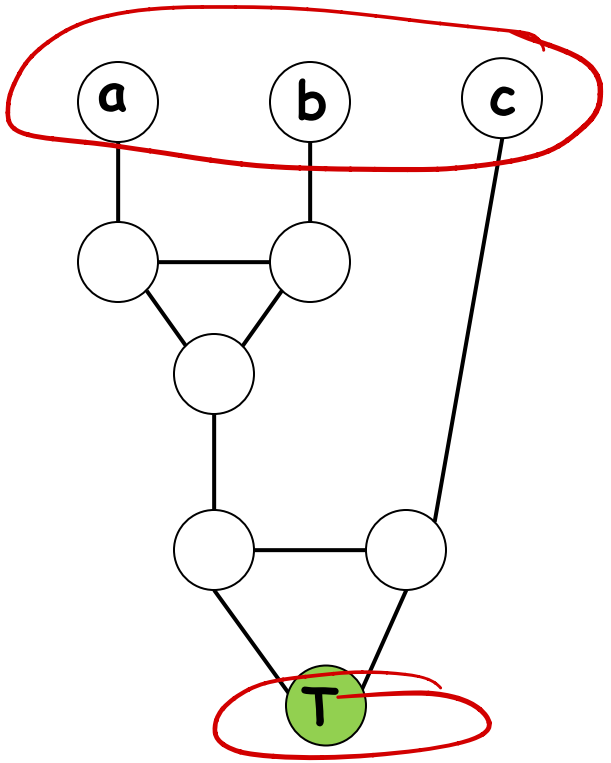


(a v b v c)

3-SAT $\leq_p$ 3-colorable

A special gadget for each clause !!

gadget for variable



Truth gadget

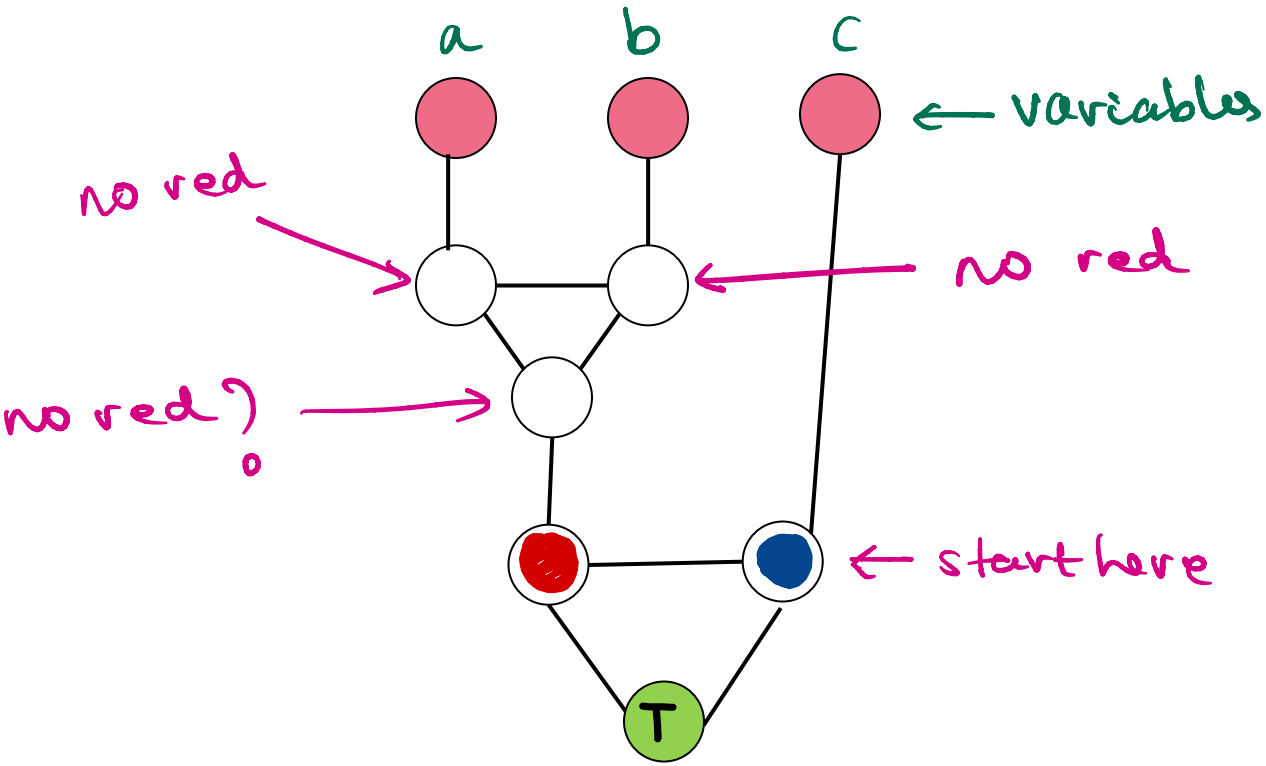
This gadget connects a truth gadget with variable gadgets.

we can color this graph with 3 colors $\Leftrightarrow$ one of the literals is true

If a=b=c =F, then we cannot color this graph with 3 colors

3-SAT $\leq_p$ 3-colorable

Suppose all a, b and c are all False (red).

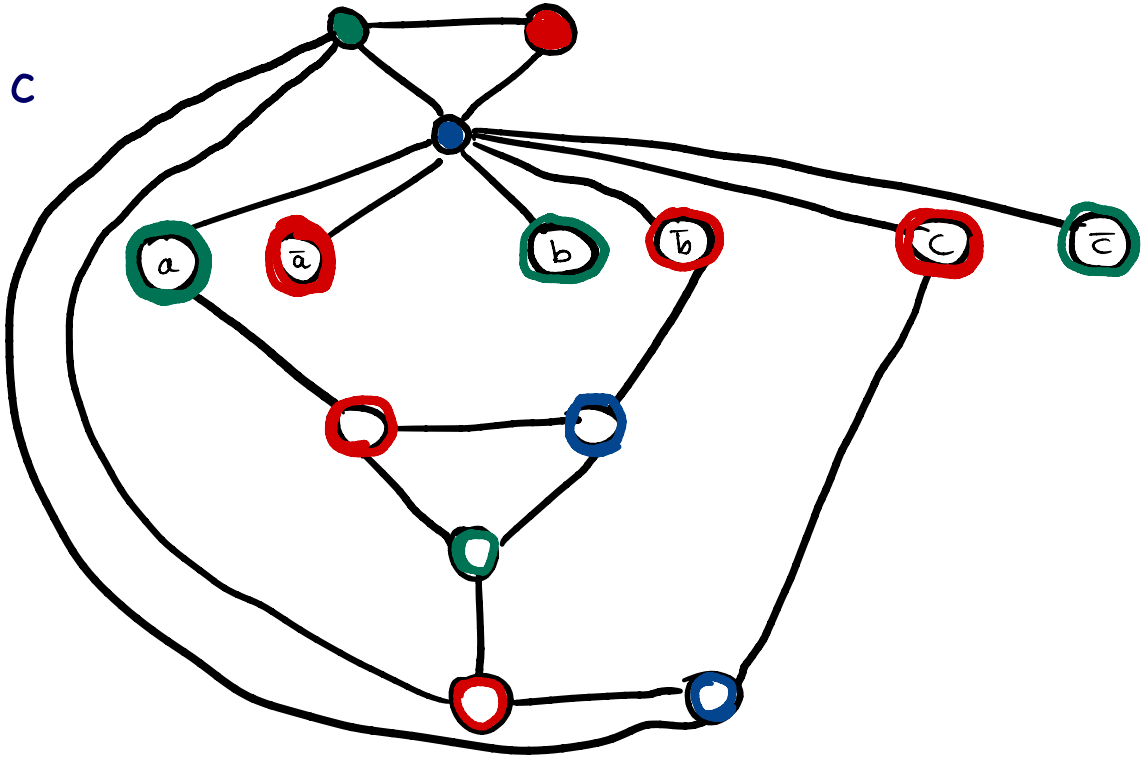


3-SAT $\leq_p$ 3-colorable

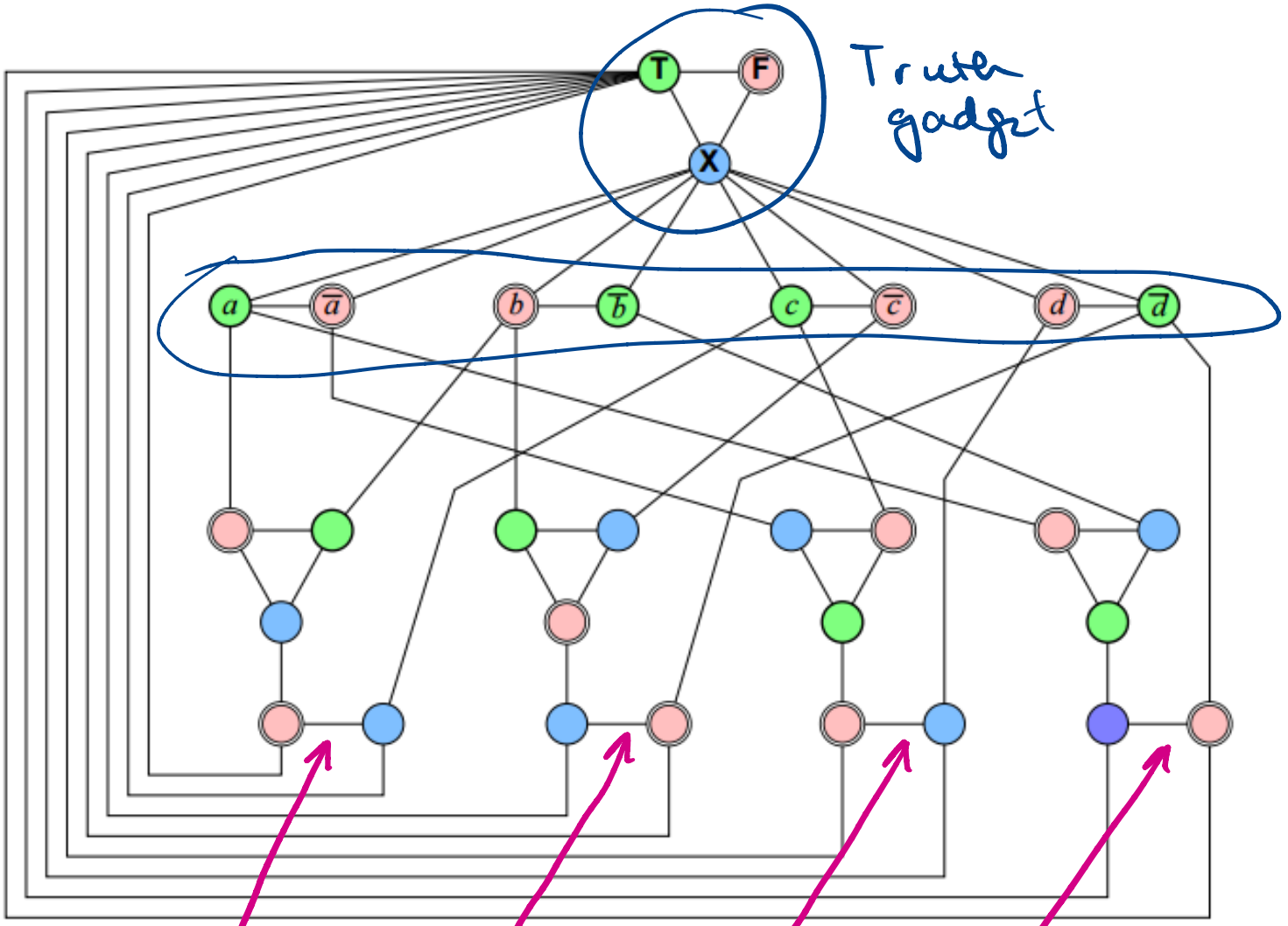
We have showed that if all the variables in a clause are false, the gadget cannot be 3-colored.

let $a = T$, $\bar{b} = c = F$

Example: a ∨ ¬b ∨ c



Example with four clauses



a=c=T

b=d=F

A 3-colorable graph derived from a satisfiable 3CNF formula.

$$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$$

3-SAT $\leq_p$ 3-colorable

Claim: *3-SAT instance is satisfiable if and only if G is 3-colorable.*

Proof: ⇒) By construction

T = Green, F = red

Truth gadget → color

Variable gadget → color

Clause gadget → coloring is forced

3-SAT $\leq_p$ 3-colorable

Claim: *3-SAT instance is satisfiable if and only if G is 3-colorable.*

Proof: $\Leftarrow$) Given a special graph which is 3 colorable

Goal: Find a truth assignment

Look at the variable gadget & assign red to False & green to True.

Sudoku: $n^2 \times n^2$

NP-?

NP-hard?

2			3		8		5	
		3		4	5	9	8	
		8			9	7	3	4
6		7		9				
9	8						1	7
				5		6		9
3	1	9	7			2		
	4	6	5	2		8		
	2		9		3			1

9 - colors

sudoku graph

vertex : 81

edges : $81 \cdot \frac{20}{2} = 810$

Sudoku Graph

Sudoku $\leq_p$ 9-color

did we prove
suduto $\in$ NP-hard

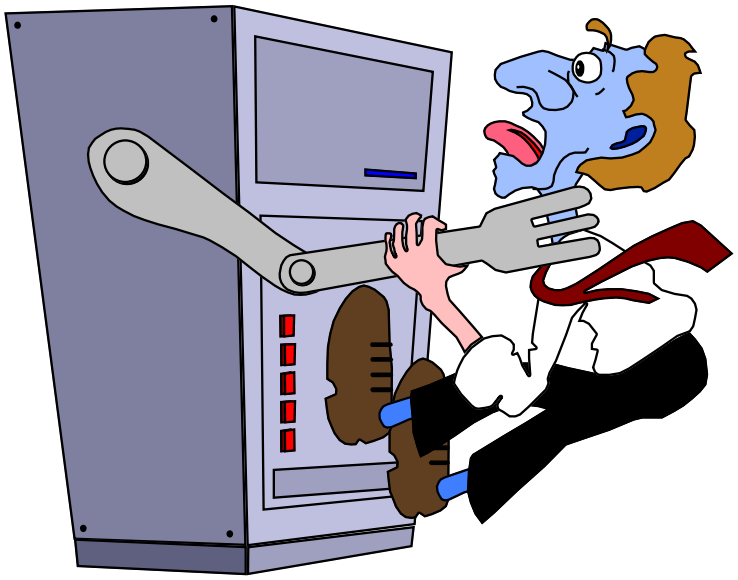
NO!!

2			3		8		5	
		3		4	5	9	8	
		8			9	7	3	4
6		7		9				
9	8						1	7
				5		6		9
3	1	9	7			2		
	4	6	5	2		8		
	2		9		3			1

Sudoku

Constructing a Sudoku graph, we have proved:

Don't be afraid of NP-hard problems.



Many reasonable instances (of practical interest) of problems in class NP can be solved!

The largest solved TSP an 85,900-vertex route calculated in 2006. The graph corresponds to the design of a customized computer chip created at Bell Laboratories, and the solution exhibits the shortest path for a laser to follow as it sculpts the chip.