Analysis of Algorithms

V. Adamchik CSCI 570 Spring 2020

Lecture 12 University of Southern California

NP-Completeness

Reading: chapter 9

Outline

The P vs. NP problem

Polynomial reduction — exAm

23 Problems of Hilbert

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

Hilbert's 10th problem:

Given a multivariate polynomial with integer coeffs, e.g. $4x^2y^3 - 2x^4z^5 + x^8$, *devise an effectively calculable procedure* for determining whether it has an integer root.

Described a model of computation, known today as the Turing Machine.

Alan Turing
(1936, age 22)

A problem P is *decidable* (or *computable*) 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

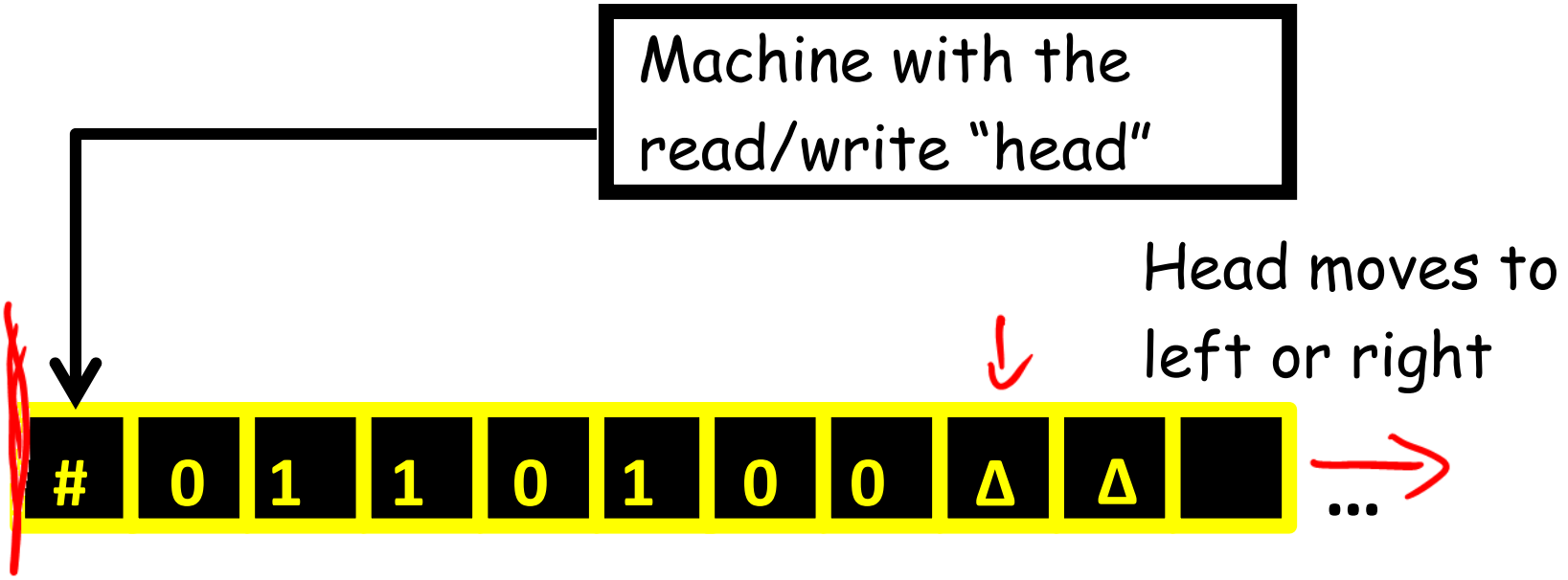
At most finitely many kinds of symbols

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



input (the "tape"), indefinitely extensible to the right.

input consists of symbols from an alphabet Σ

and Δ represent the start and end of the input

The machine never overwrites the leftmost symbol, but can overwrite the rightmost symbol.

When halt(accept/reject) state is reached, the machine halts. It might also never halt, in which case we say it loops.

High Level Example of a Turing Machine

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

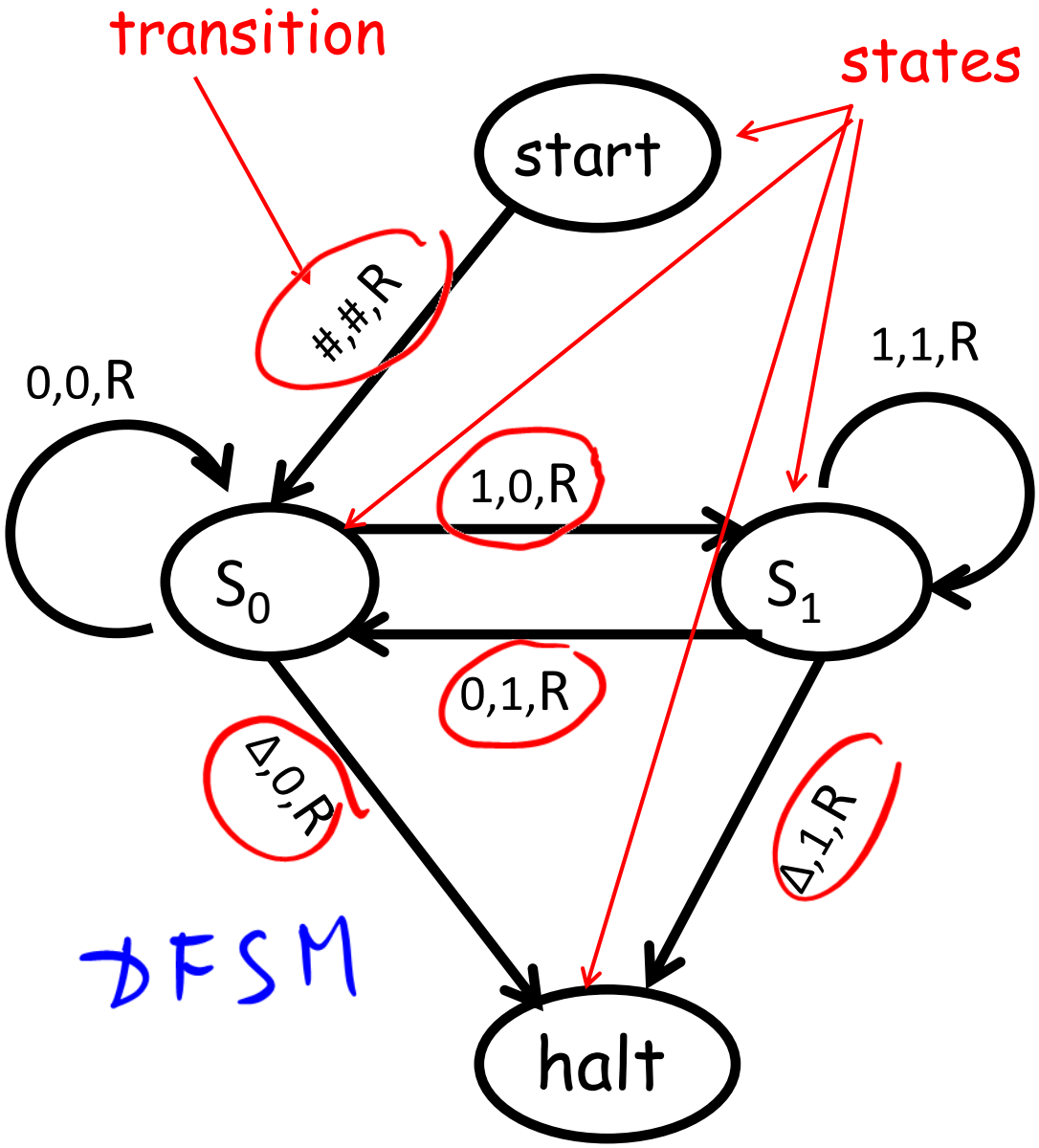
Input: #10010Δ

Output: #010010Δ

- leftmost char

Δ - rightmost char

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



Runtime Complexity

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.

Algorithms

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

We say that P has an algorithm.

A problem P is *undecidable* if there is no algorithm that solves it.

$\mathbb{R} \leftrightarrow \mathbb{N}$?

The intuition is based on the fact that there are as many problems as there real numbers, and only as many programs as there are integers, so there are not enough programs to solve all the problems.

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.

chess

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.

A fundamental complexity class PSPACE is a class of decision problems that can be solved using a reasonable amount of memory (regardless time).

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.

Hypercomputation

In 1995 Hava Siegelman proposed
Artificial Recurrent Neural Networks (ARNN).
She claims that ARNNs can "compute"
non-computable functions.

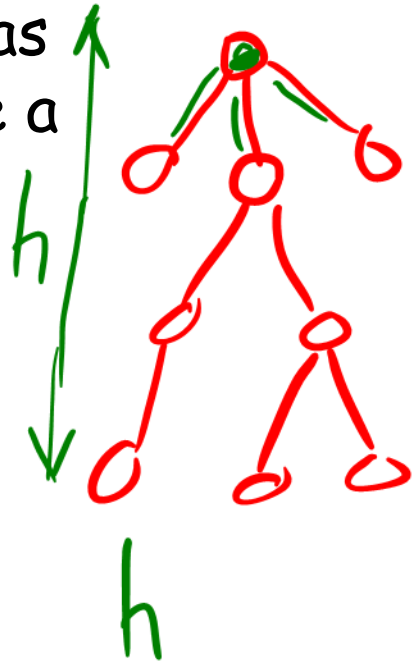
In 2006 Martin Davis criticized the idea in a paper
"The Myth of Hypercomputation."

Some people were disagree with Martin saying
that it's a myth that hypercomputation is a myth.

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.

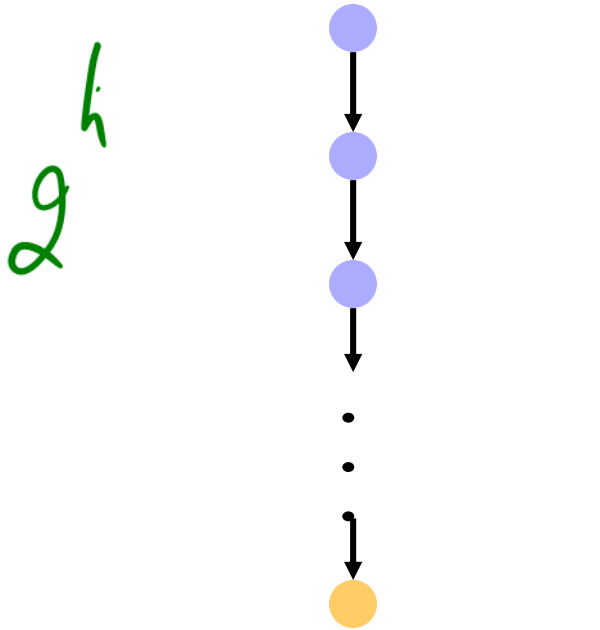
Nondeterministic TM defined in the same way as deterministic, except that a computation is like a tree, where at any state, it's allowed to have a number of choices.



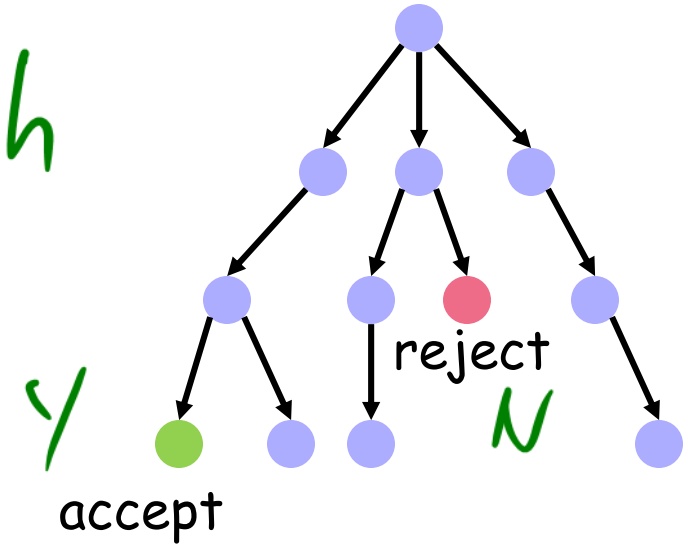
The big advantage: it is able to try out many possible computations in parallel and to accept its input if any one of these computations accepts it.

Computations

deterministic computation



nondeterministic computation



accepts if <u>some</u> branch reaches an accepting configuration

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.

NP does NOT stand for "non-polynomial".

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

!!

NP-problems like FIND a needle in a haystack: hard to find but always easy to verify.

P versus NP

It has been proven that Nondeterministic TM can be simulated by Deterministic TM.

But how fast we can do that simulation?

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

Dana Scott and Michael Rabin have introduced the concept of nondeterminism (Turing award in 1976)
Without the notion of nondeterminism,
there would be no P=NP

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.

"I am a liar"

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 undecidable.

textbook

Polynomial Reduction

Denoted by

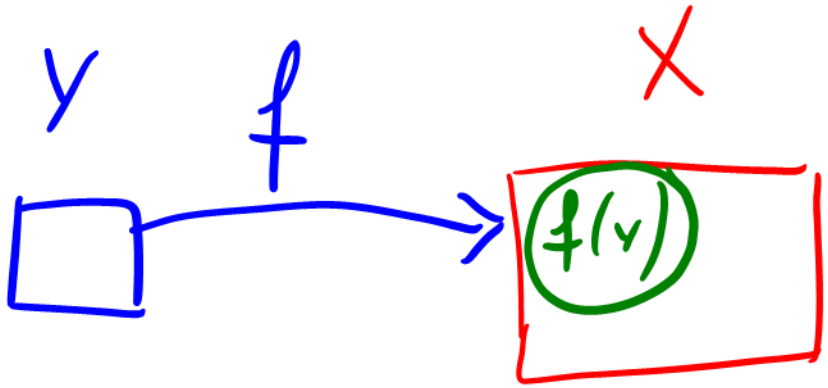
$$Y \leq_p X$$

Problem Y is "easier" than X.

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.



$f(y) \notin X$

$Y \leq_p X$

$X \in P \Rightarrow Y \in P$

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

$Y \leq_p X \leq_p Z \leq_p S$

If we can solve X, we can solve Y.

The *contrapositive* of the above statement:

If we cannot solve Y, we cannot solve X.

We use this to prove NP completeness: knowing that Y is hard, we prove that X is harder.

I know that Y is hard, using $Y \leq_p X$ that X is also hard.

Two ways of using $Y \leq_p X$

1) X is easy $X \in P \Rightarrow Y \in P$

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

Lect. 1-11

Lect. 12

2) Y is hard $Y \in NP$

Then X is at least as hard as Y

from Y to X

P and NP

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

theoretical

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

practical

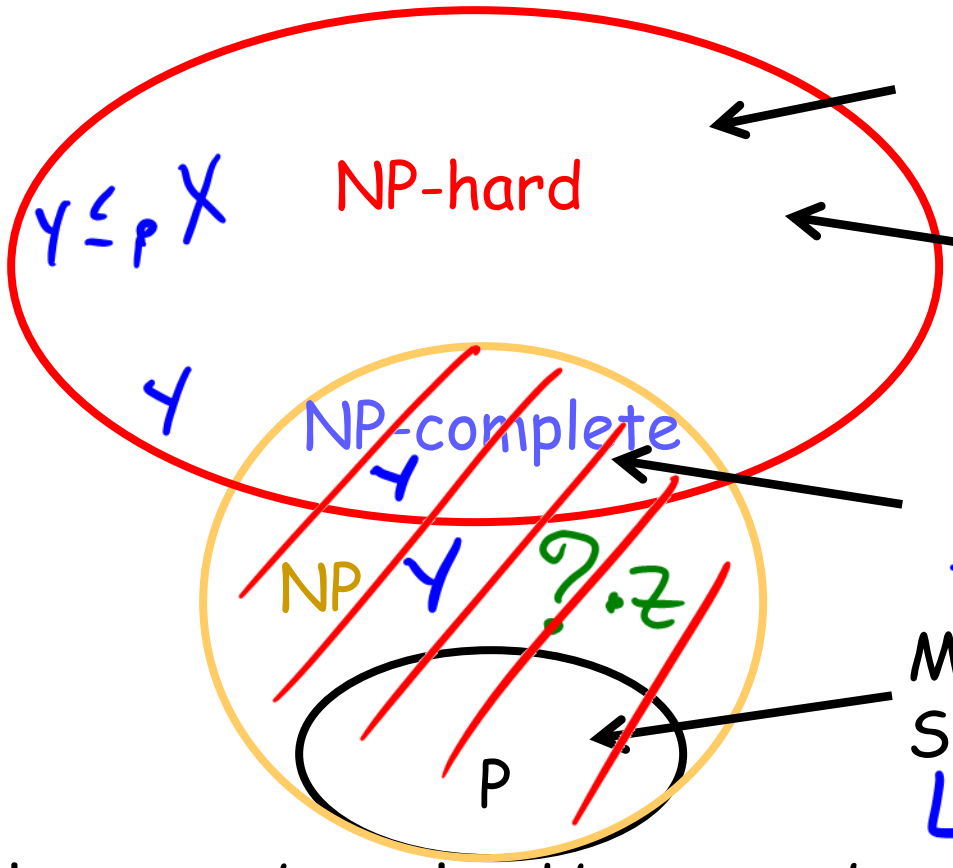
NP = set of problems for which solution can be verified in polynomial time by a deterministic TM.

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)



Halting Problem

Chess, TSP, optim.

Tetris, Knapsack, SAT, Vertex Cover

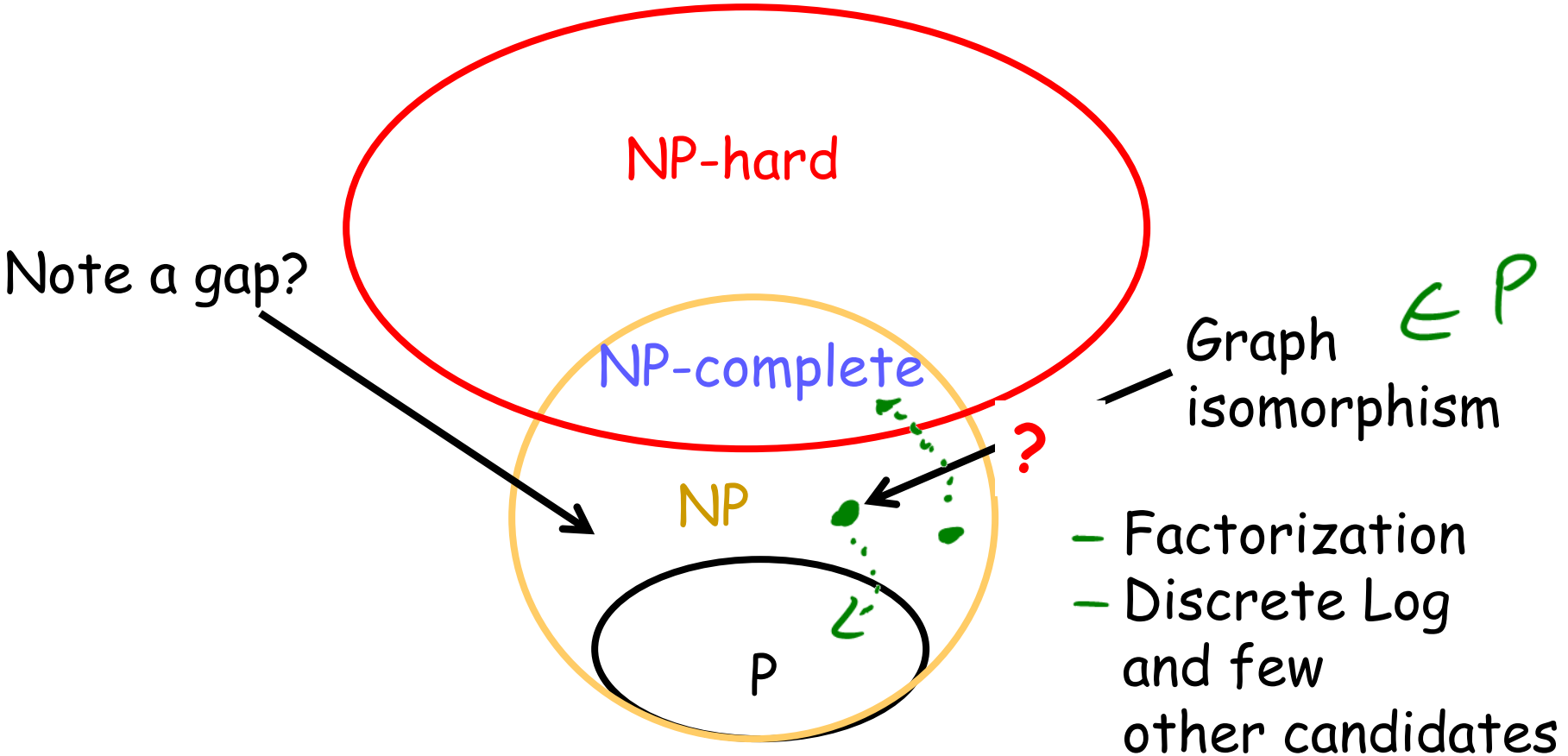
Max-Flow
Sorting
LP

P ⊂ NP

NPC problems can be solved by a *nondeterministic* TM in polynomial time.

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

NP-Intermediate

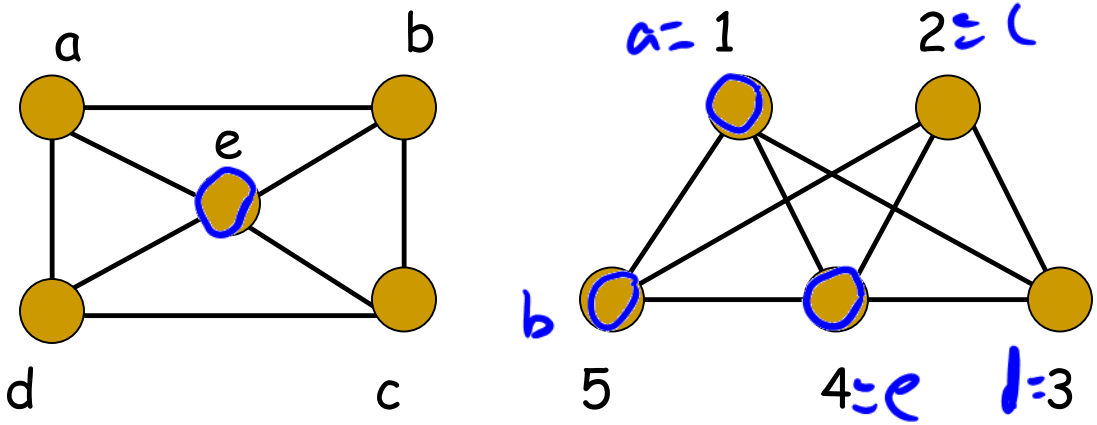


It is proven that the gap is not empty.

But we do not know any natural problems in there.

Graph Isomorphism

Two graphs $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ are isomorphic if there is a bijective function $f: V_1 \rightarrow V_2$ such that $(v,w) \in E_1 \Leftrightarrow \{f(v),f(w)\} \in E_2$



The two graphs look differently but are structurally the 'same', up to the renaming of the vertices.

Problem is in NP, but

- No NP-completeness proof is known
- No polynomial time algorithm is known

Boolean Satisfiability Problem (SAT)

A propositional logic formula is built from variables, operators AND (conjunction, ∧), OR (disjunction, ∨), NOT (negation, ¬), and parentheses:

clause and clause and

$$(X_1 \vee \neg X_3) \wedge (X_1 \vee \neg X_2 \vee X_4 \vee X_5) \wedge \ldots$$

or

= True

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

2^4

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.

Cook-Levin Theorem (1971)

Theorem. CNF SAT is NP-complete. $Y \leq_P X$

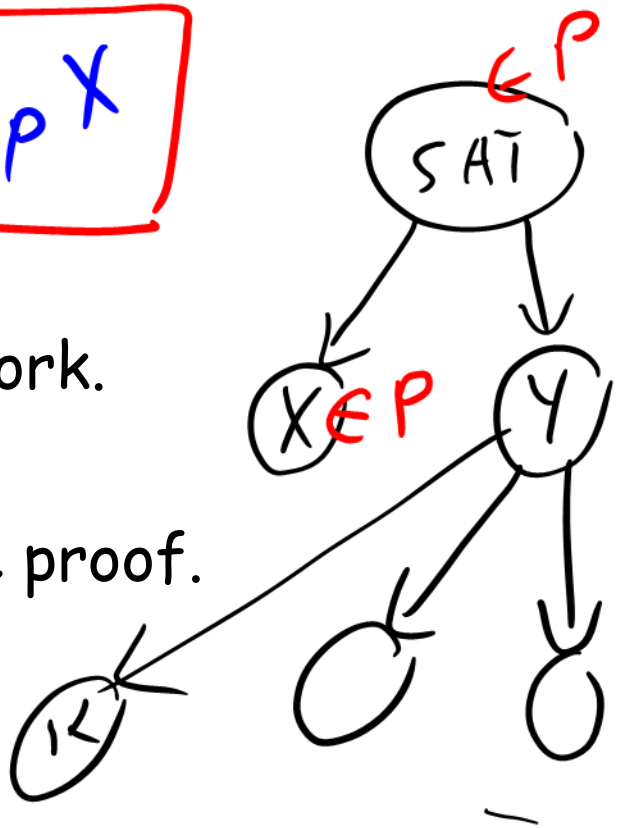
$SAT \leq_P X$

No proof...

Cook received a Turing Award for his work.

You are not responsible for knowing the proof.

$P = NP$



Disjunctive Normal Form

A propositional logic formula that is a disjunction of conjunctions of literals:

$$(X_1 \wedge \neg X_3 \wedge X_4) \vee (\neg X_1 \wedge X_2 \wedge \neg X_5) \vee \ldots$$

Such a formula is satisfiable if and only if at least one of its clauses is satisfiable.

And a conjunction is satisfiable if and only if it does not contain both X and $\neg X$ for some variable X.

This can be checked in linear time.

Independent Set

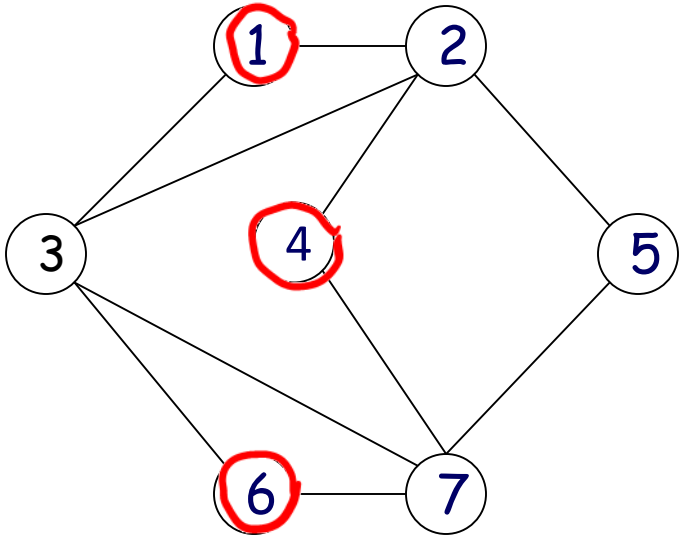
Given a graph, we say that a subset of vertices is "independent" if no two of them are joined by an edge.

$IS = \{1\}$

$\{1, 4\}$

$\{1, 4, 6\}$

$\max |IS| = 3$



Independent Set

Optimization Version:

NPH

Given a graph, find the largest independent set.

Decision Version:

NPC

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

k = 5

max = 7

Optimization vs. Decision

k = 2, k = 4, k = 8 → k = 6

k = 7

Optimization vs. Decision

NPH NPC

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

$NPC = NPH \cap NP$

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 at least k?

1) NP
2) NP-hard

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 at least k?

Is it in NP-hard?

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

$Y = SAT$

Reduce from 3-SAT.

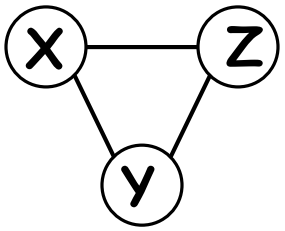
$(\overset{\leq 3}{\ldots}) \wedge (\overset{\leq 3}{\ldots}) \wedge$

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:



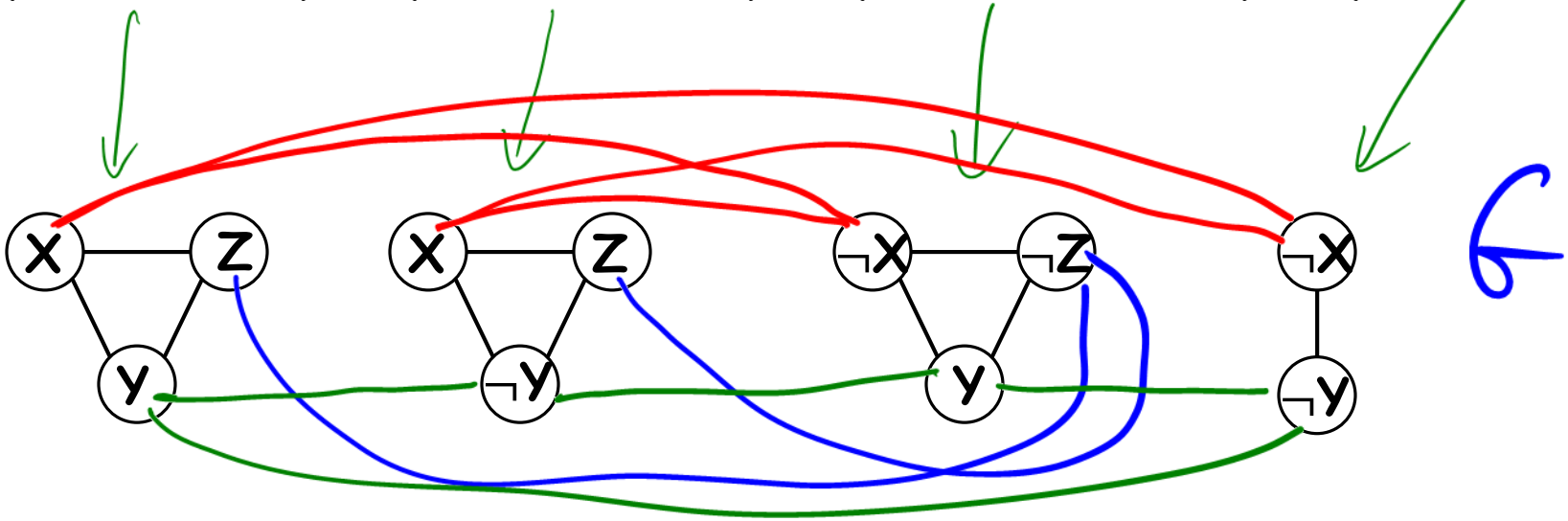
Next, we need to connect gadgets.

As an example, consider the following instance:

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

3SAT $\leq_p$ IndSet

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



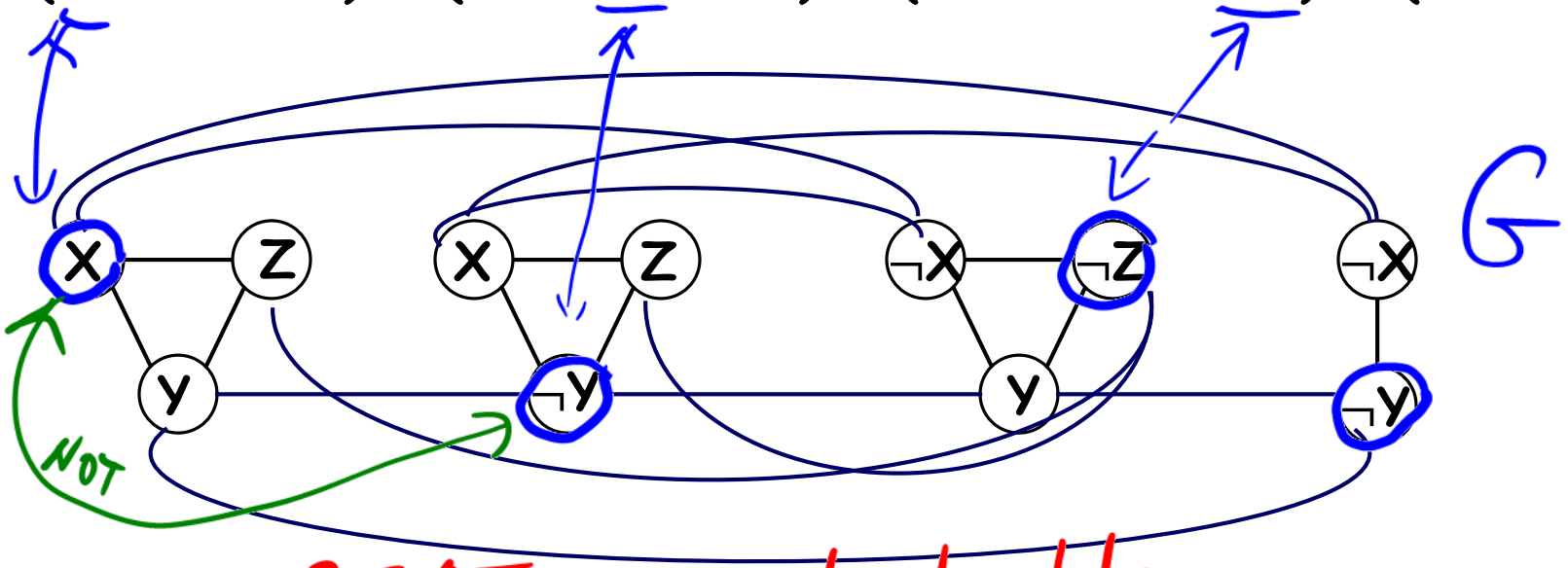
How do we connect gadgets?

Claim: 3SAT with k clauses is satisfiable iff G has an IS of size k.

$\Rightarrow$ $\Leftarrow$

$3SAT \leq_p$ IndSet

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



G

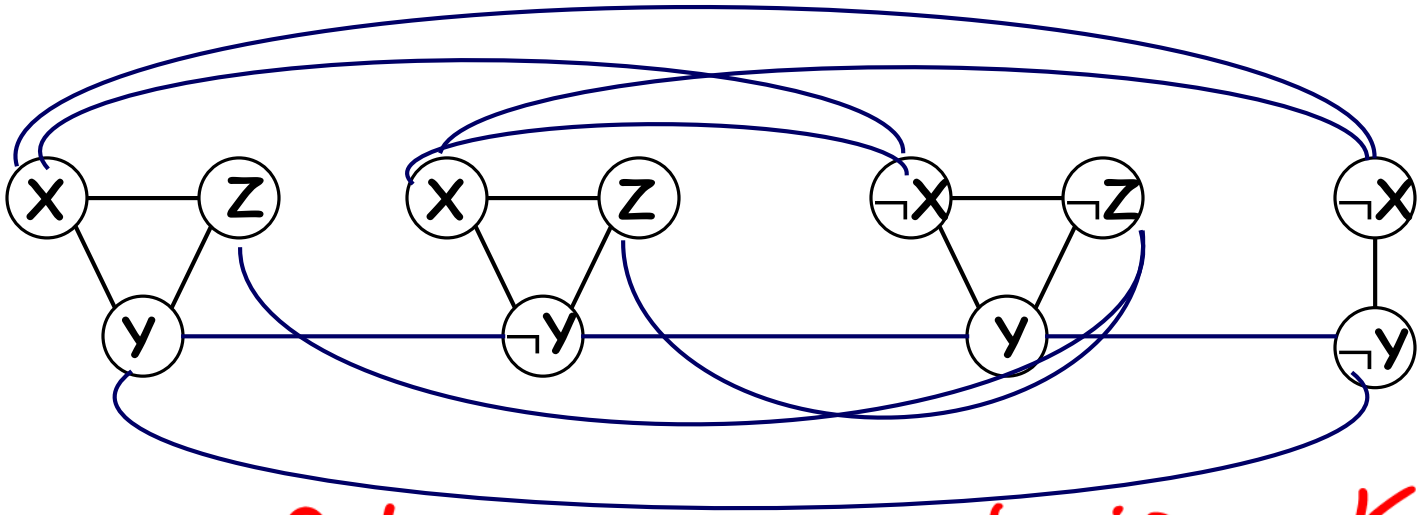
Proof. $\Rightarrow$) 3SAT is satisfiable.

$x = \neg y = \neg z = True$

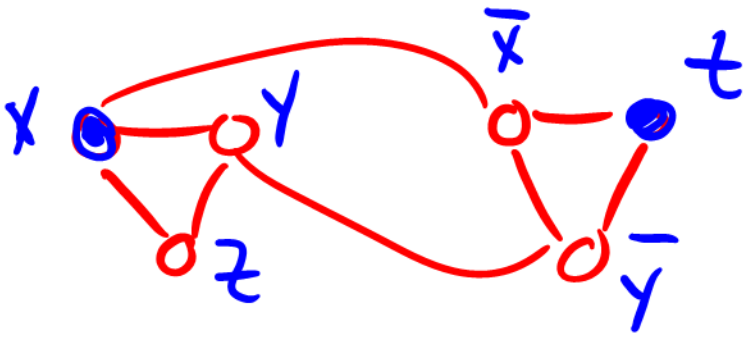
True variables make an IS

$3SAT \leq_p IndSet$

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



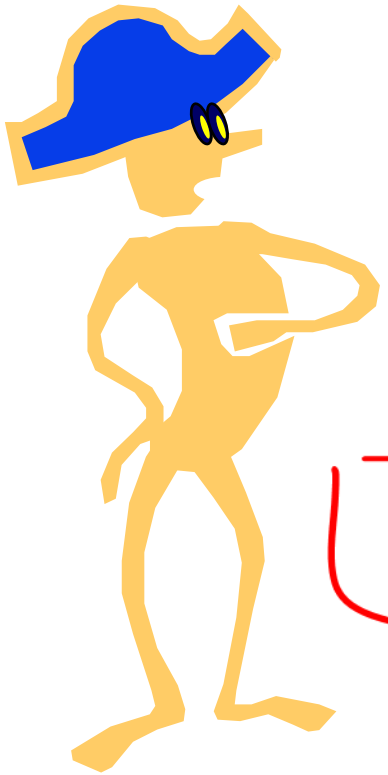
Proof. $\Leftarrow$) f has an IS of size K



$(x \vee y \vee z) \wedge (\bar{x} \vee t \vee \bar{y})$

x = True

t = True

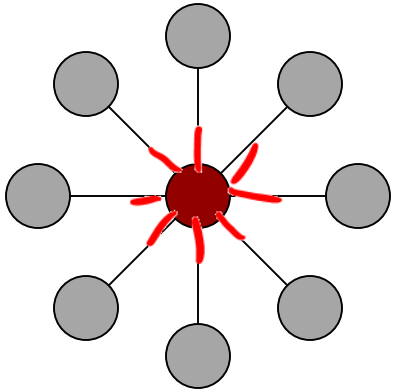


Vertex Cover (VC)

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

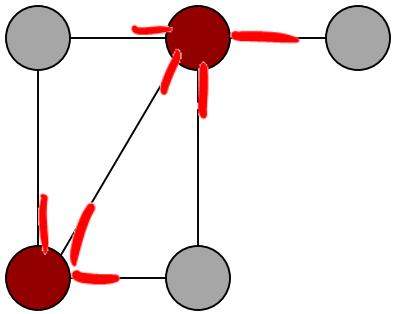
$VC = V - IS$

IS



Decision Problem.

Given G and k, does G contain a VC of size at most k?



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

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

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. $\Leftarrow$) Given VC

Pick $\forall x, y$ such that $x \notin VC$, $y \notin VC$

edge (x, y) does not exist, by contradiction

It follows,

$x, y \in \overline{IS}$

Vertex Cover in NP-Complete

Ind. Set $\leq_p$ Vertex Cover

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.*

$IS \leq_p VC$

$VC \leq_p IS$

$Y \leq_p X \not\Longrightarrow X \leq_p Y$!!

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.

SAT $\in$ NPC

2-SAT $\in$ P

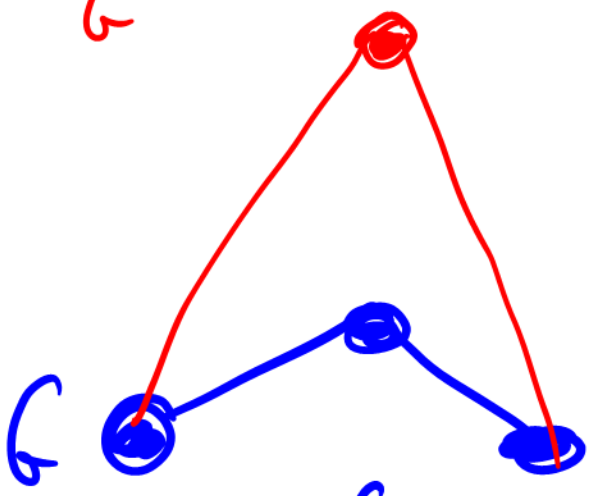
Prove: VC $\leq_p$ VC-even

1. VC-even $\in$ NP

2. VC-even $\in$ NPH

Reduction: convert any f to f' with all even degrees

VC $\leq_p$ VC-even



A graph has an even number of odd degree vertices.

Claim. G has a VC of size k iff G' has a VC of size k+1.

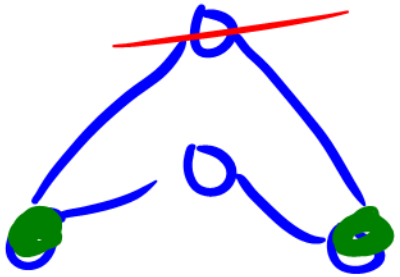
$\Rightarrow$ in G, $|VC| = k$

$VC(G') = VC(G)$ + red vertex $= k+1$.

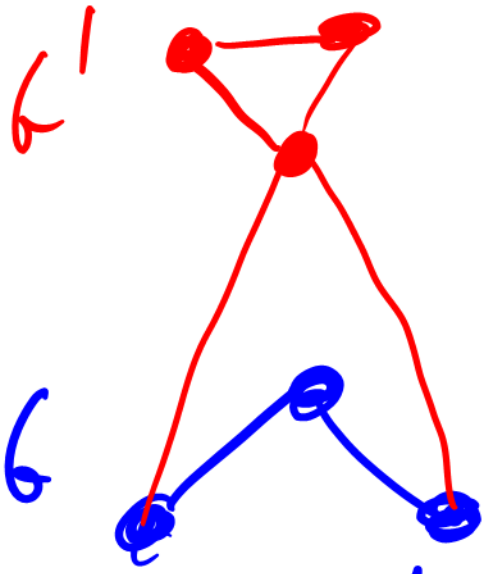
$\Leftarrow$ in G', $|VC| = k+1$

remove a vertex

$VC(G) = k+1$



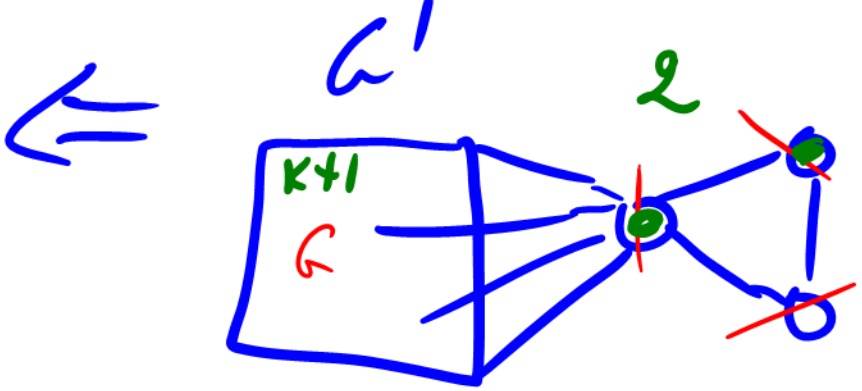
VC $\leq_p$ VC-even



Claim. $VC(G) = k$ iff $VC(G') = k+2$ ($k+3$)? No

$\Rightarrow$ by construction

$\Leftarrow$

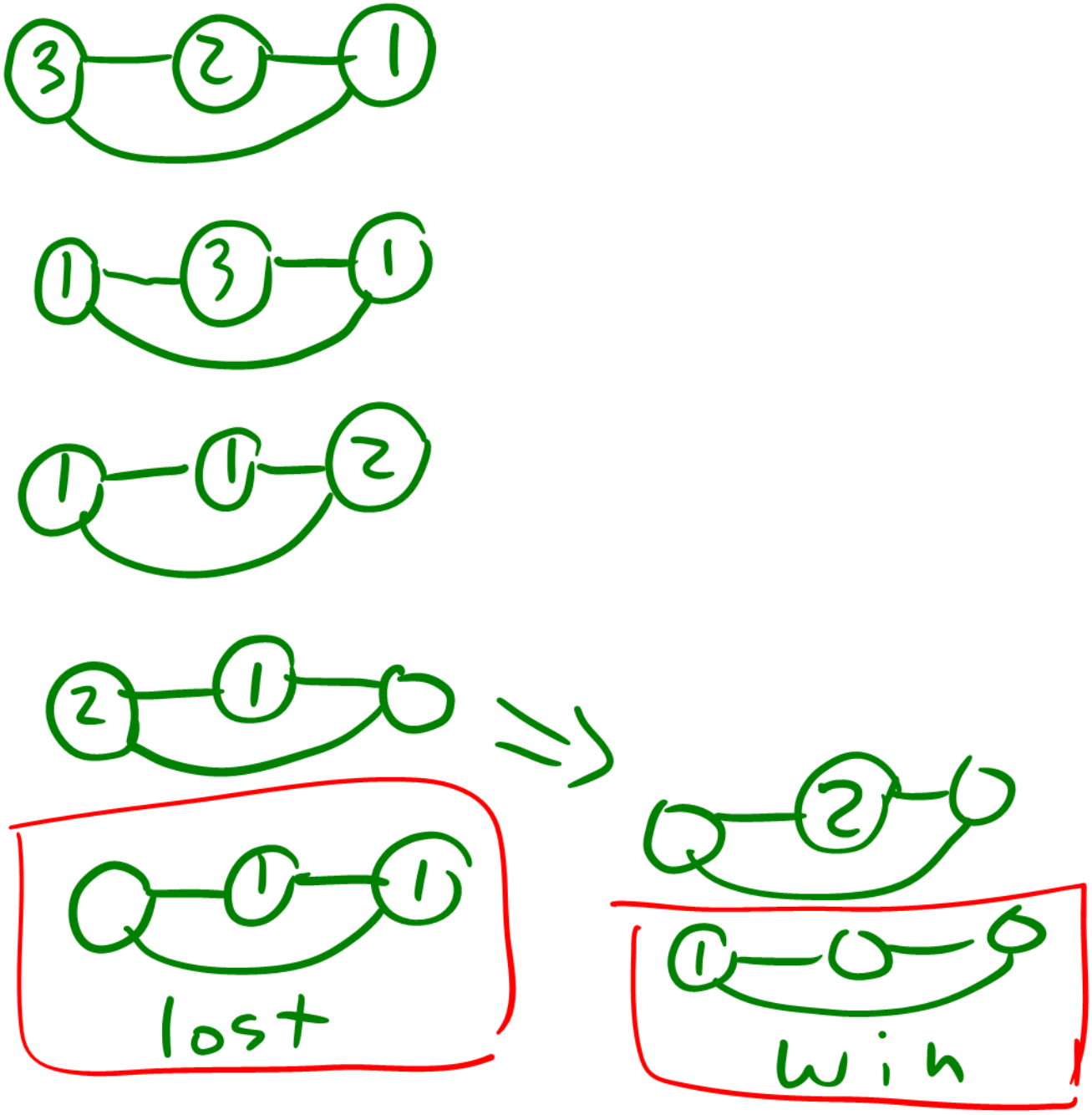


$VC(G') = k+2$, Remove Δ

Discussion Problem 2

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.

1 PebblesPr $\in$ NP

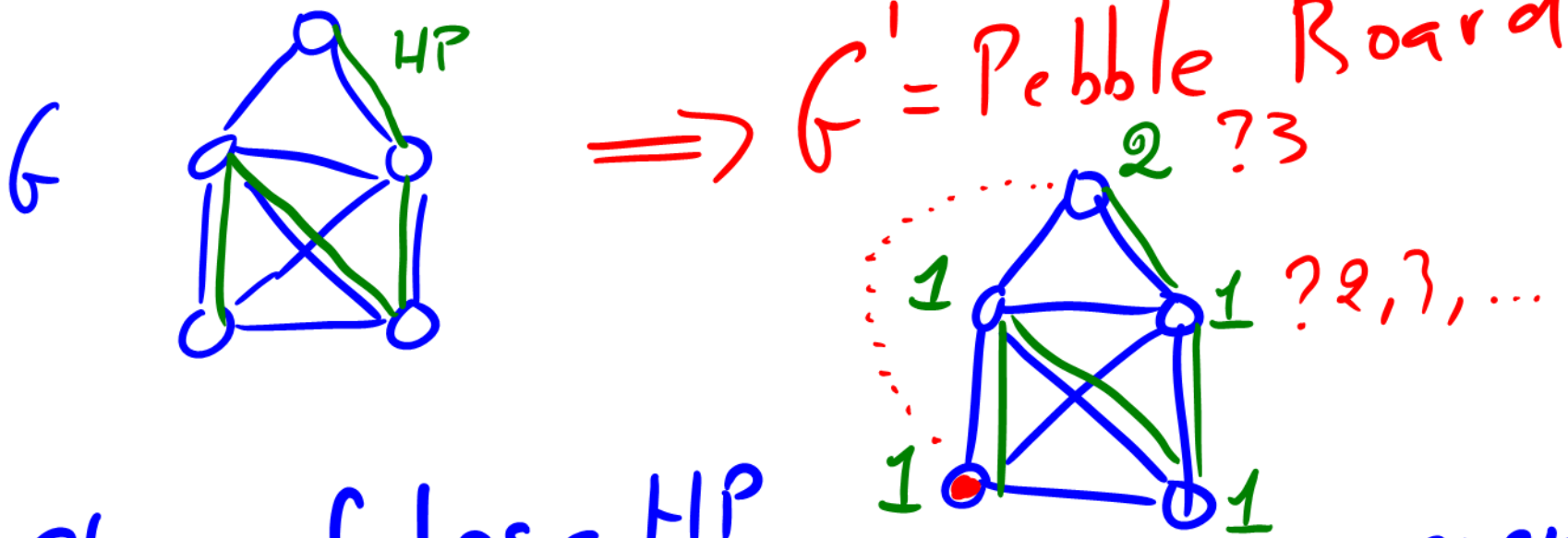

3 2 1
1 3 1
1 1 2
2 1
1 1
lost
2
1
win

2). PebblesPr $\in$ NP-hard

Hamiltonian Path (HP)

$HP \leq_p$ PebblesPr

Construction:



Claim. G has a HP iff G' has a winning sequence.

Proof.

$\Rightarrow$ f has a HP.
in f' follow a HP

$\Leftarrow$ f' has a wit. sequence
HP is a sequence!

2	1	1	1	1	
0	2	1	1	1	
0	0	2	1	1	
0	0	0	2	1	
0	0	0	0	2	
0	0	0	1	0	

f'