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:

$$\max (c^T x)$$

subject to

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

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

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

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

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.
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.

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

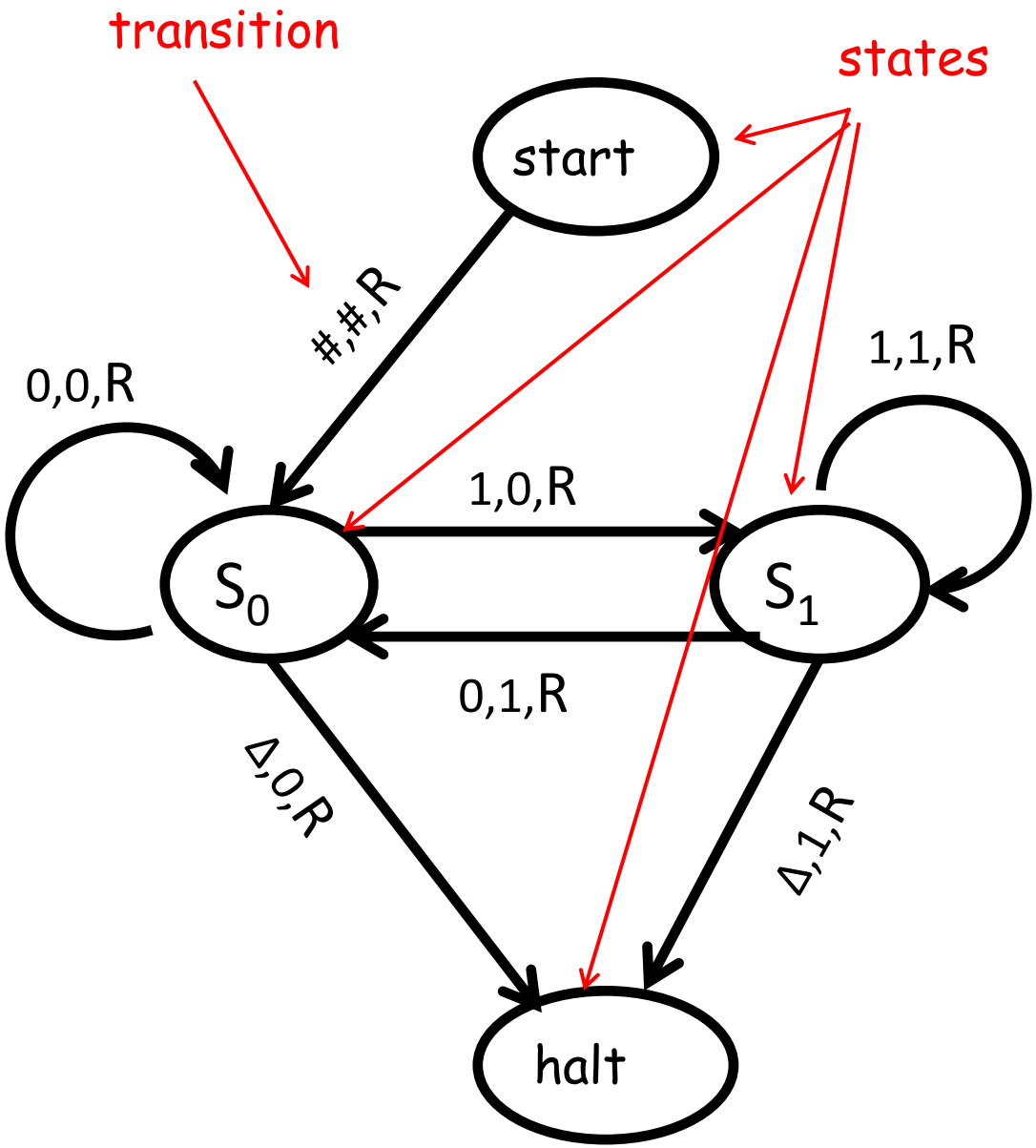
The machine that takes a binary string and appends 0 to the left side of the string.

Input: #10010Δ
Output: #**0**10010Δ

- leftmost char

Δ – rightmost char

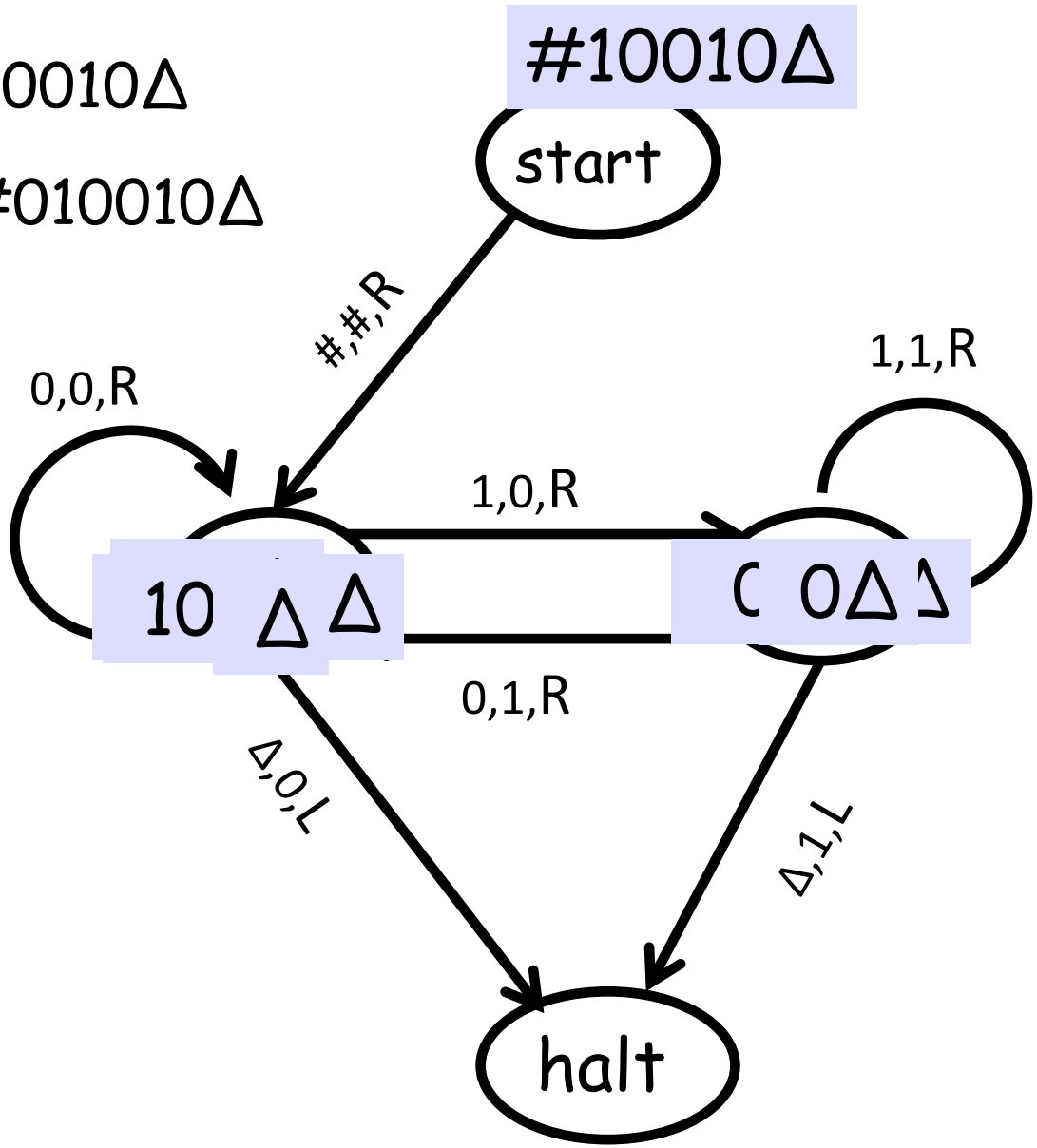
Transition on each edge
read,write,move (L or R)



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.

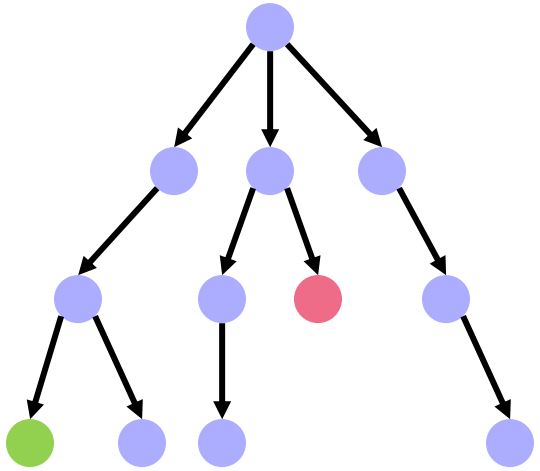
Why is the Halting Problem so important?
Because a lot of really practical problems are the halting problem in disguise.

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

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

$Y \leq_p X$

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

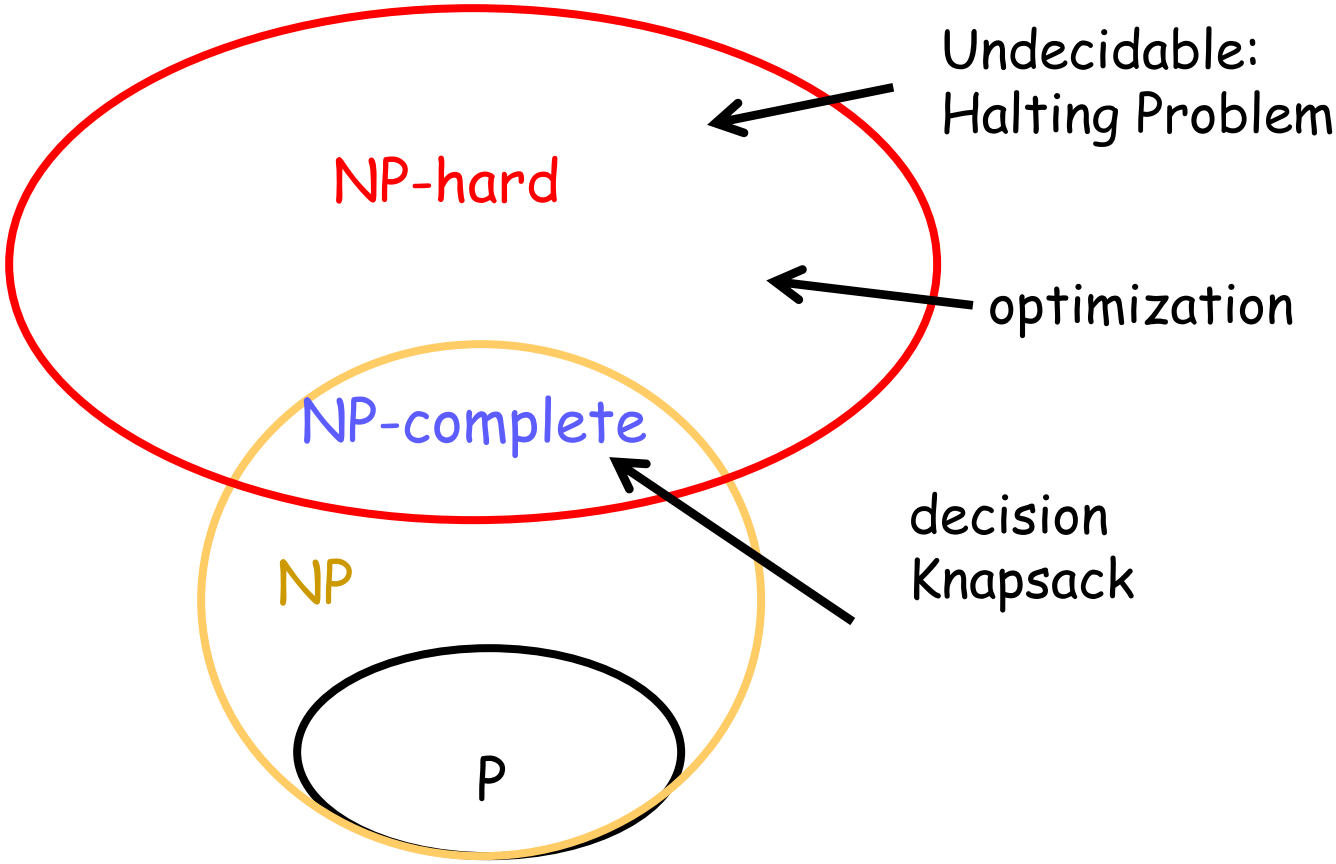
X is *NP-Hard*, if $\forall Y \in NP$ and $Y \leq_p X$.

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.

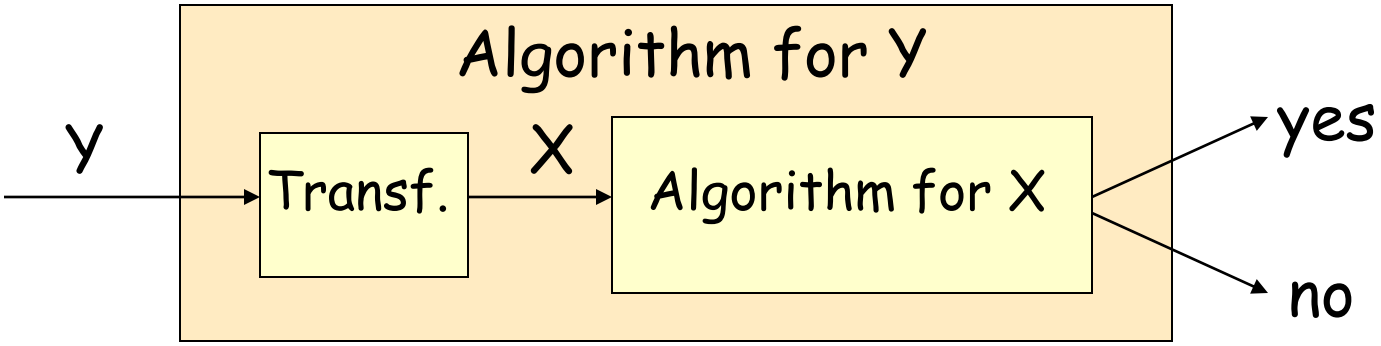
NP-Completeness Proof Method

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:

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

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.

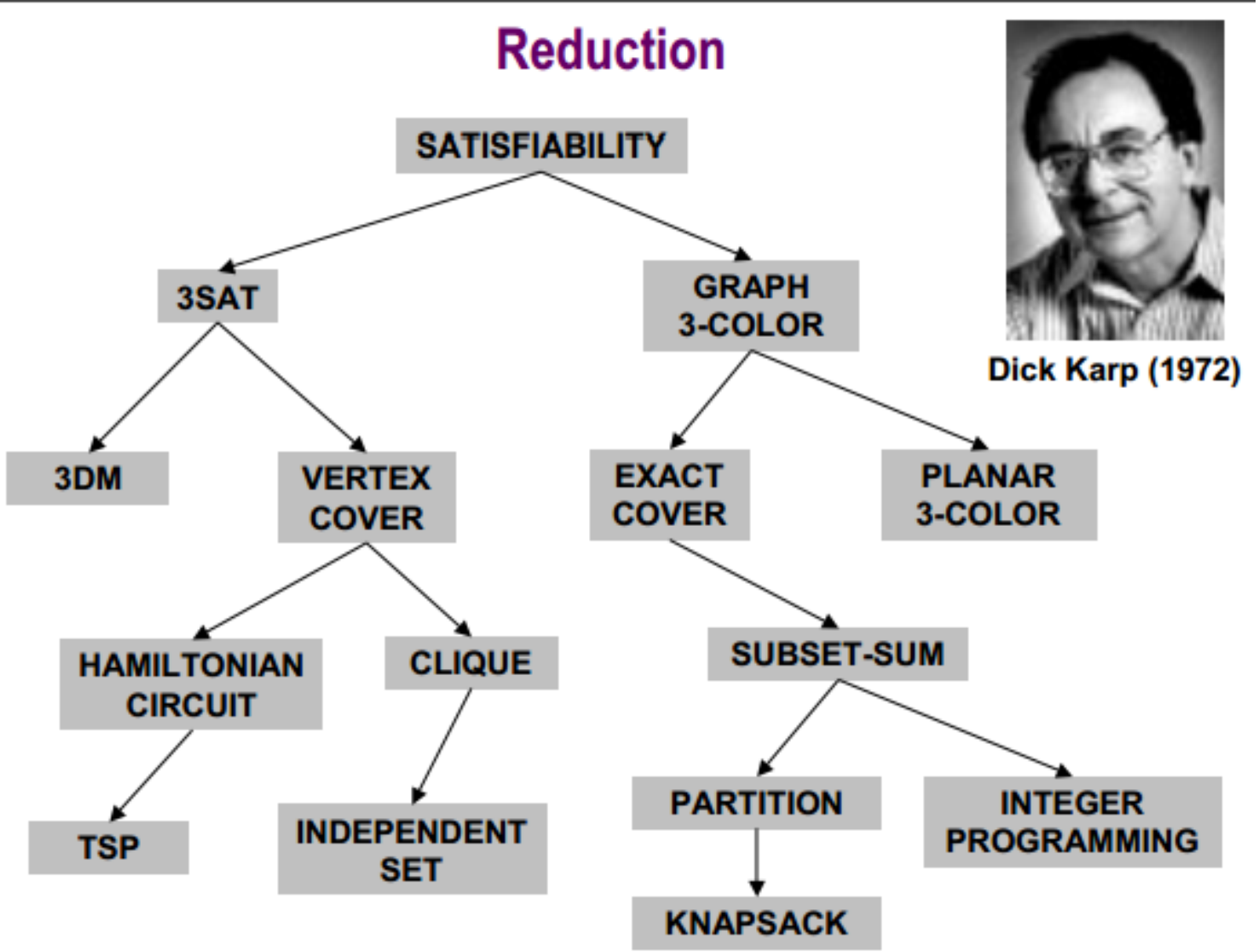
Cook-Levin Theorem (1971)

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.



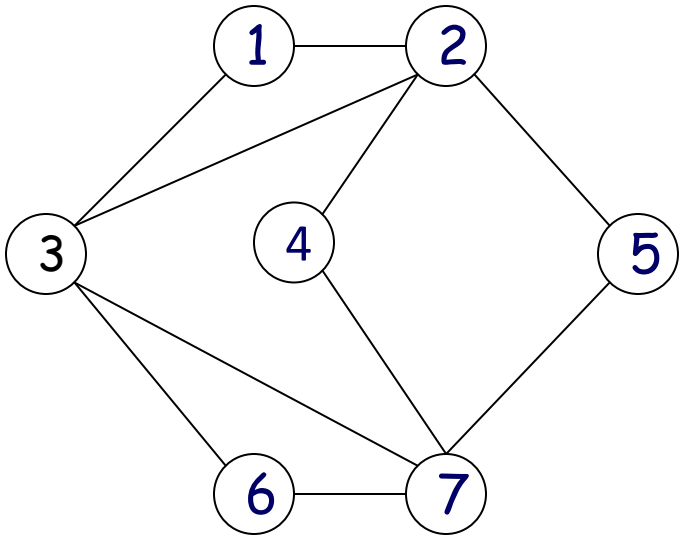
Dick Karp (1972)

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.



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.

Decision Version (IndSet):

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

Optimization vs. Decision

Optimization vs. Decision Problems

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

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 $\forall\, Y \in NP$

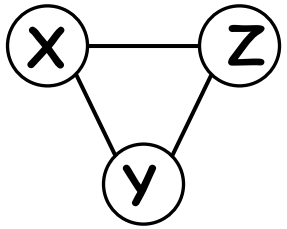
Reduce from 3-SAT.

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:



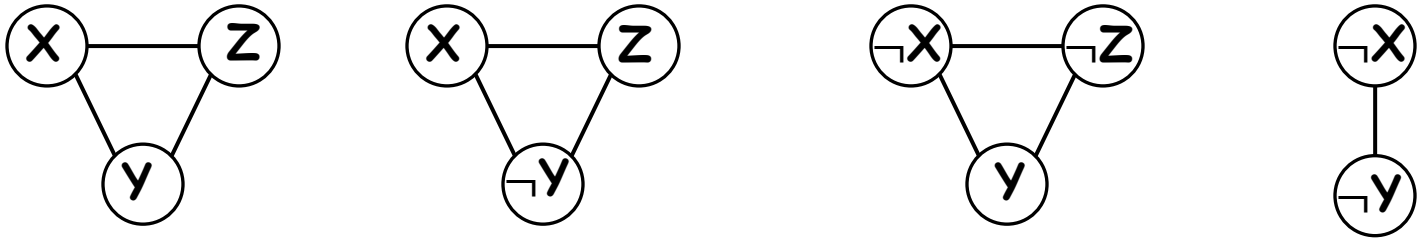
Next, we need to connect gadgets.

As an example, consider the following instance:

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

3SAT $\leq_p$ IndSet

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

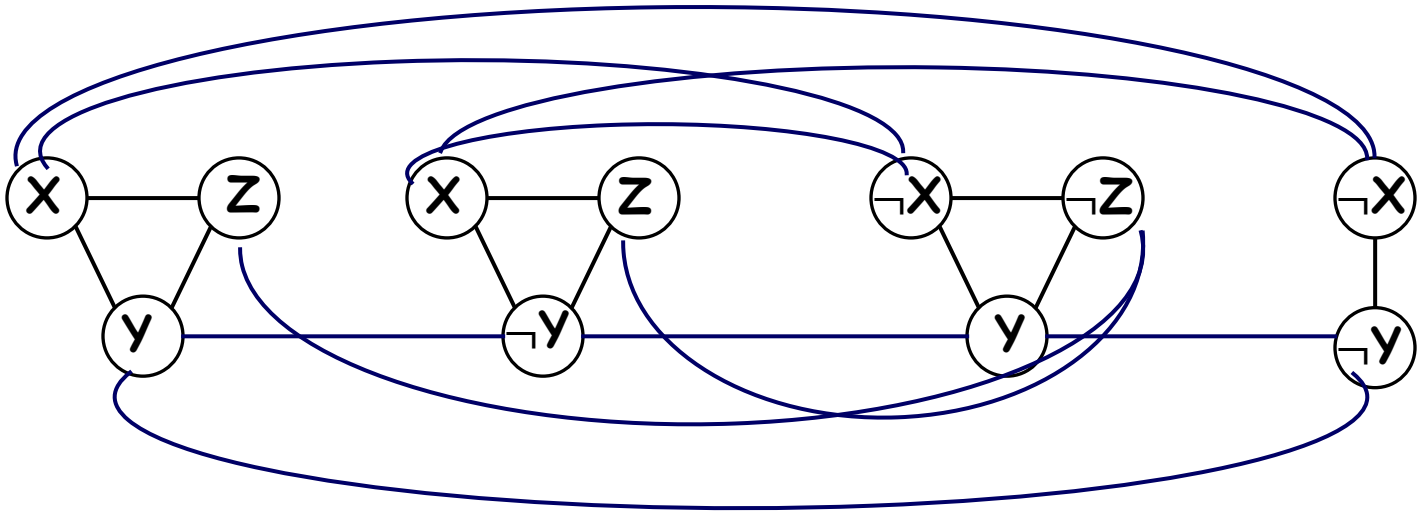


How do we connect gadgets?

<u>Claim:</u>

3SAT $\leq_p$ IndSet

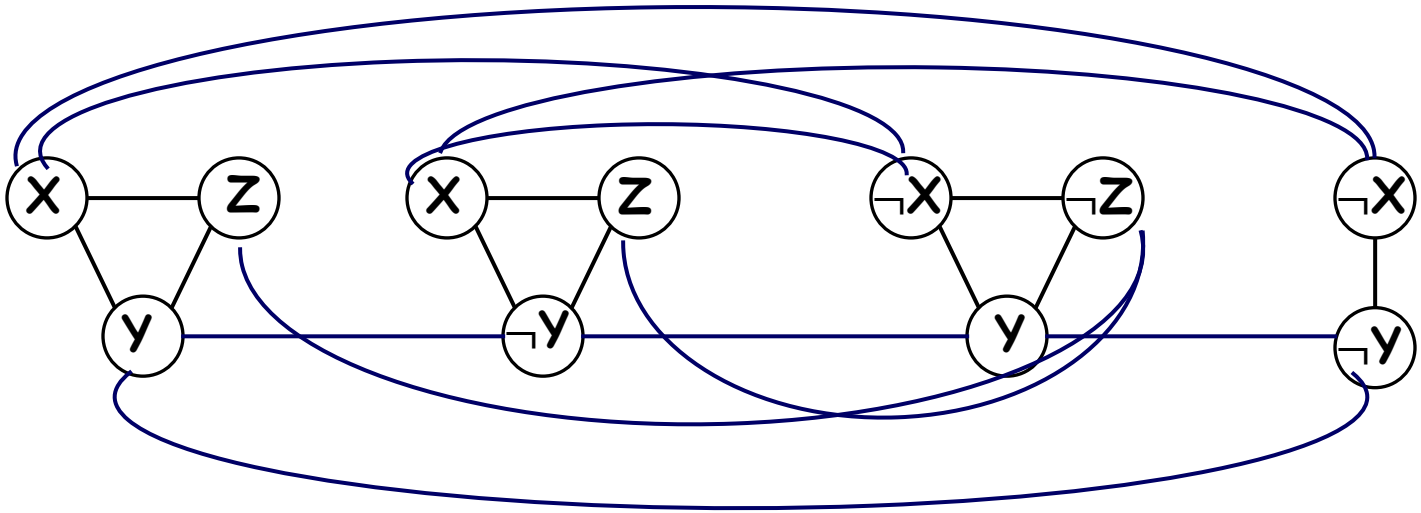
$(X \vee Y \vee Z) \wedge (X \vee \neg Y \vee Z) \wedge (\neg X \vee Y \vee \neg Z) \wedge (\neg X \vee \neg Y)$



Proof. $\Longrightarrow$)

3SAT $\leq_p$ IndSet

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



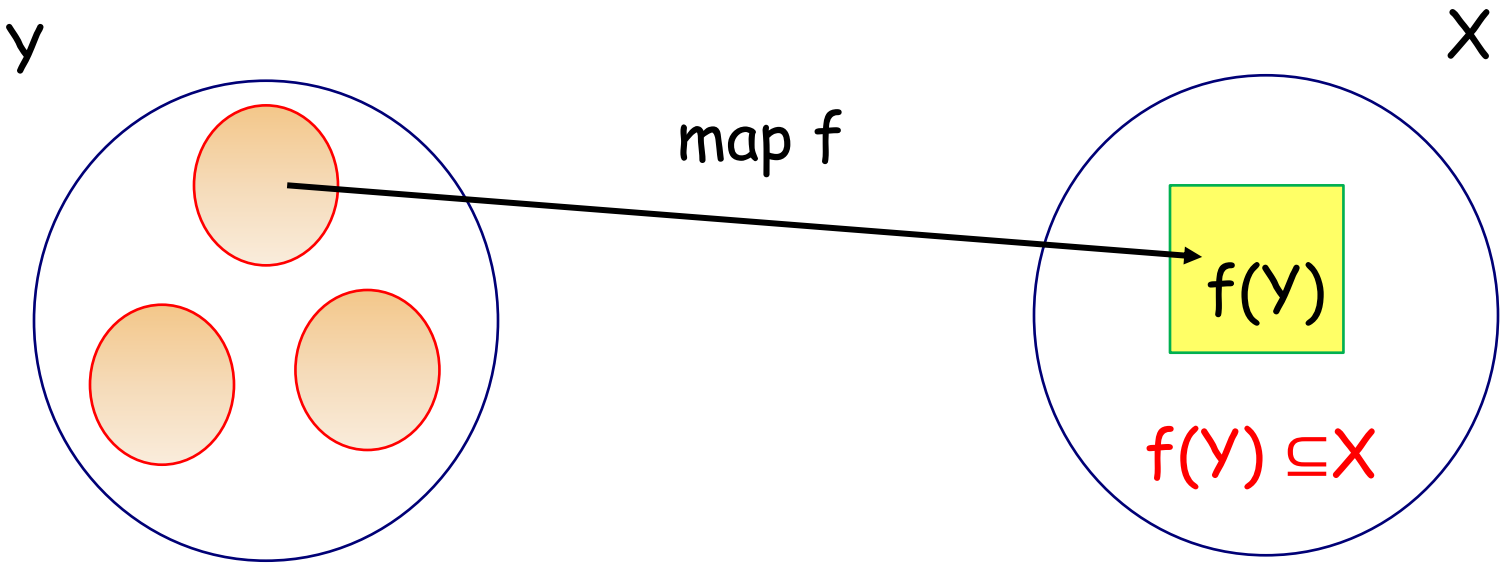
Proof. $\Leftarrow$)

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.



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 $Y \leq_p X$

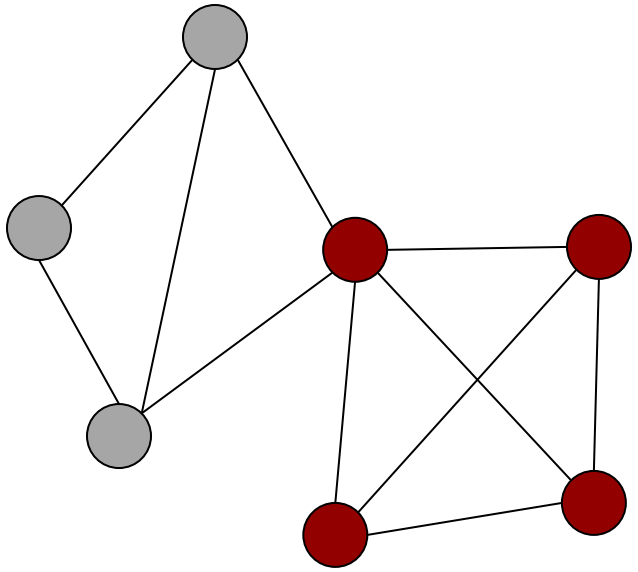
1. Describe a polynomial-time algorithm to transform an arbitrary instance of Y into a special instance of X.
2. Prove that if an instance of Y is True, then an instance of X is True.
3. Prove that if an instance of X is True, then an instance of Y is True. (This part causes the most trouble.)

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.

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

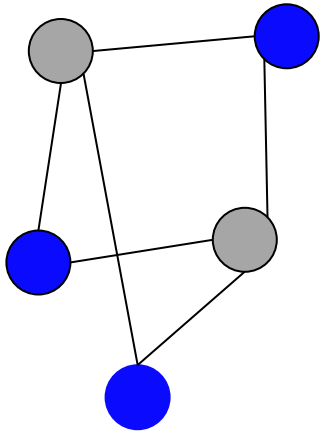


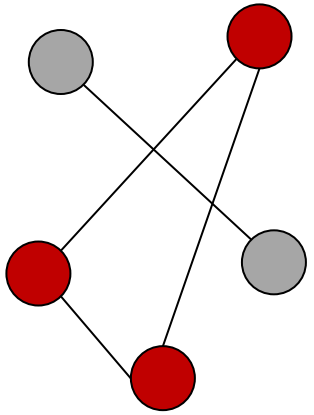
MaxClique $\leq_p$ MaxIndSet

Any graph G=(V, E) has a complement graph G'=(V, E'), where edge (u, v)$\in$E' if and only if (u, v)$\notin$E. In other words, G$\cup$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.