Analysis of Algorithms

V. Adamchik

CSCI 570

Lecture 11

University of Southern California

NP-Completeness

Reading: chapter 9

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

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

Alan Turing (1936, age 22)

We say that P has an algorithm.

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

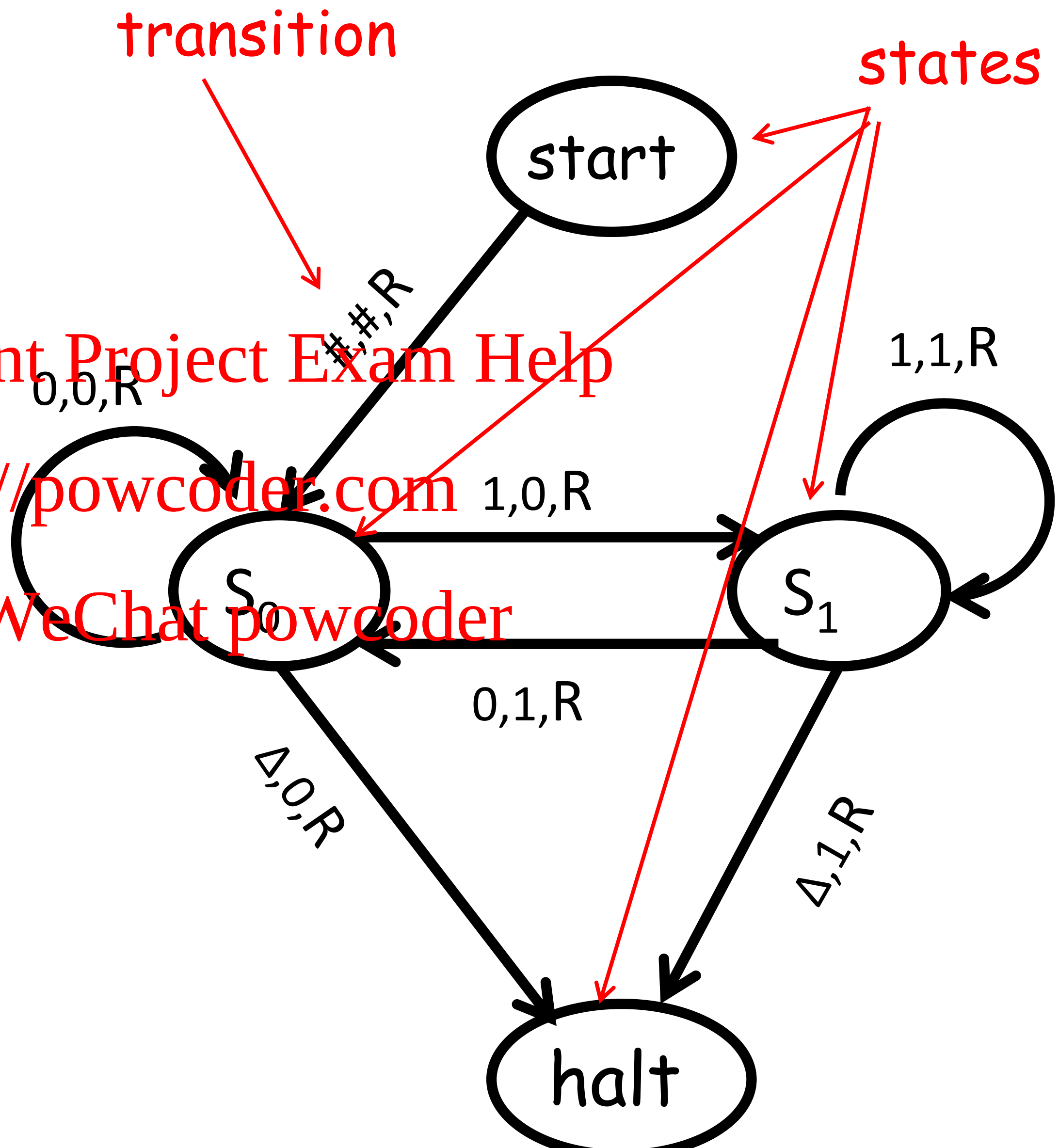
High Level Example of a Turing Machine

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)



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.

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.

Super-Turing computation

In 1995 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

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.

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.

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 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

$Y \le_p X$

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) X is easy

If we can solve X in polynomial time,

we can solve Y in polynomial time.

2) Y is hard

Then X is at least as hard as Y

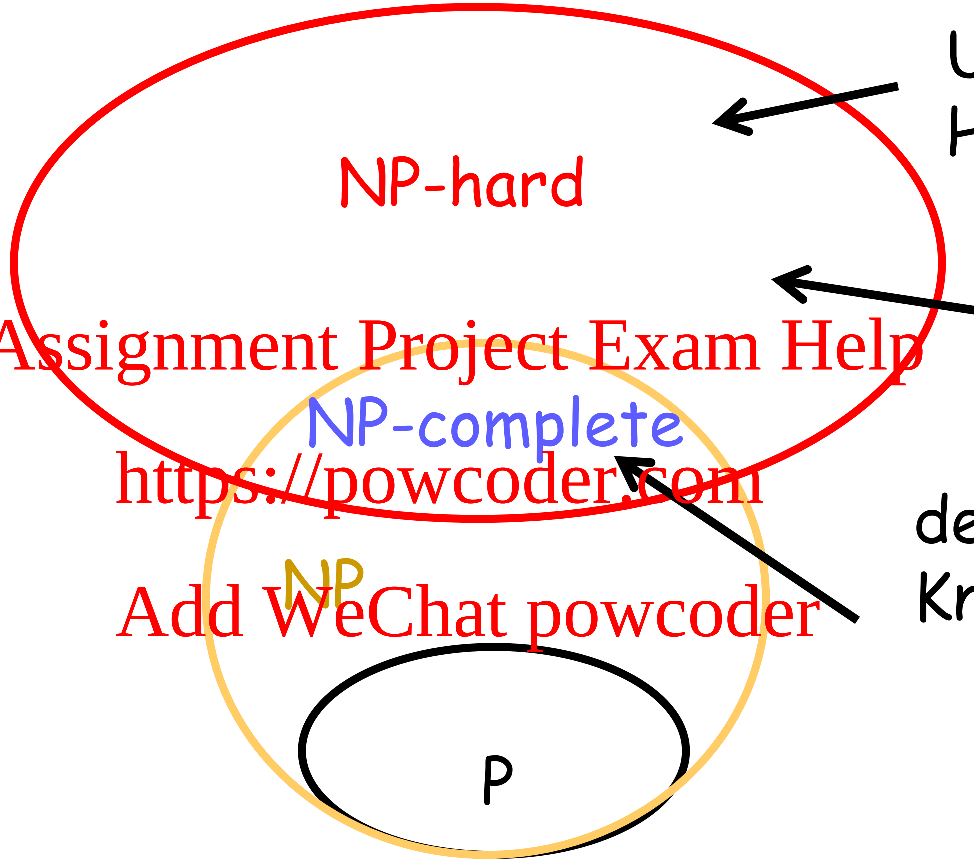
NP-Hard and NP-Complete

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.

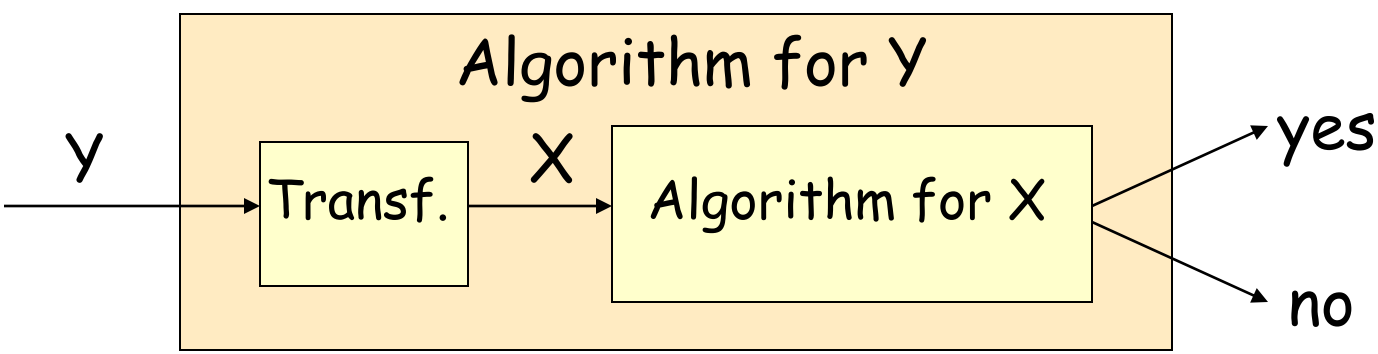
NP-Completeness Proof Method

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)

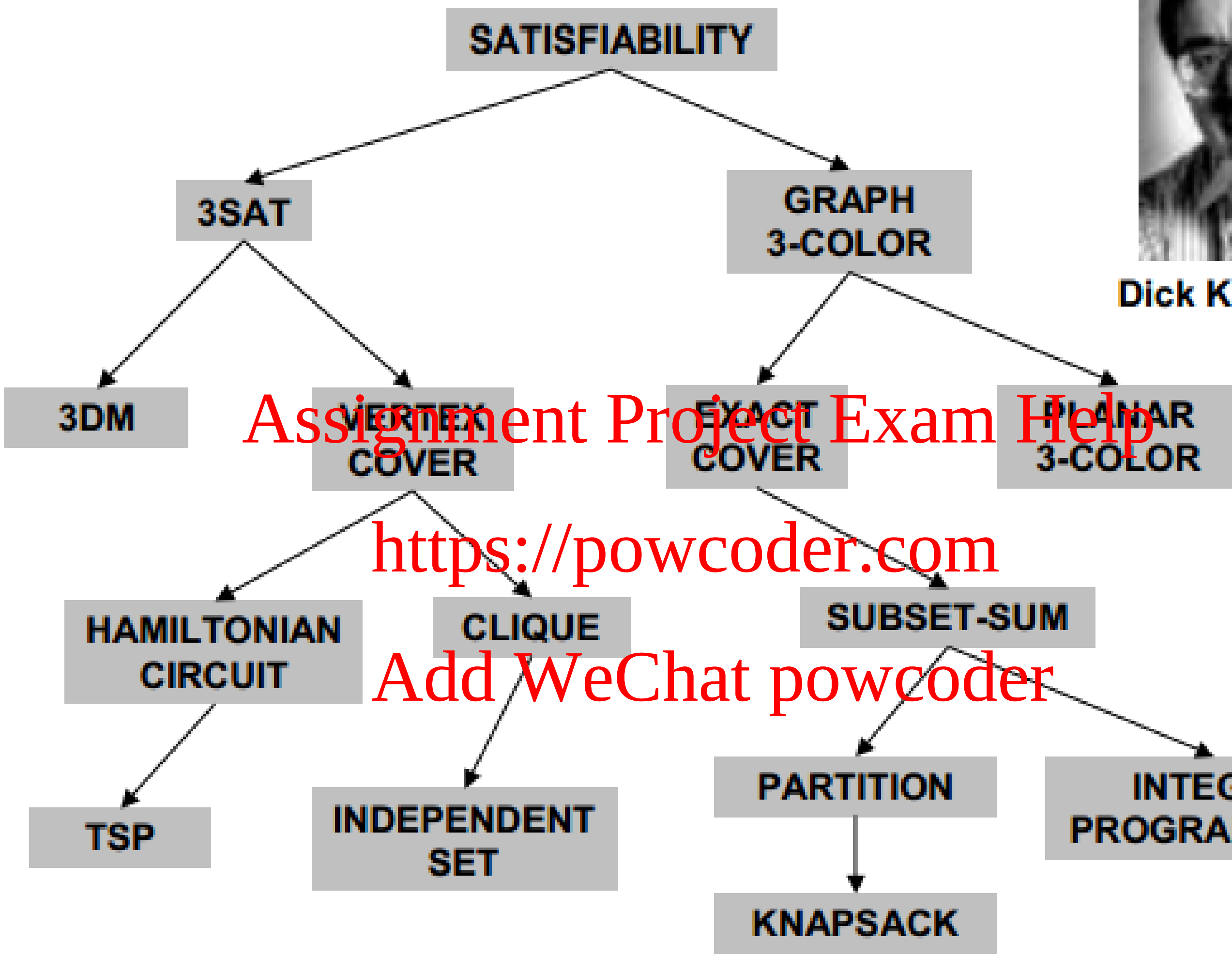
Theorem. CNF SAT is NP-complete.

No proof…

Cook received a Turing Award for his work.

You are not responsible for knowing the proof.

Reduction



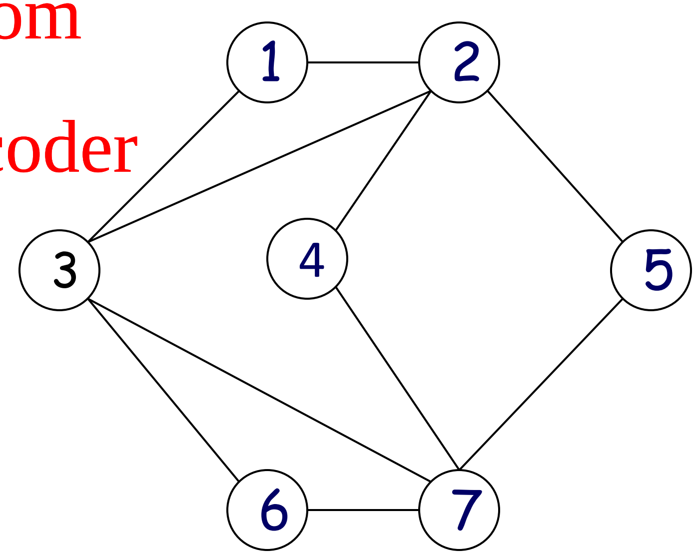
Dick Karp (1972)

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.



Independent Set

Optimization Version:

Given a graph, find the largest independent set.

Decision Version:

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 at least 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.

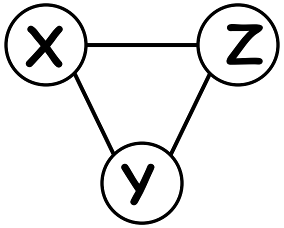
3-SAT is SAT where each clause has at most 3 literals.

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 $(X \vee Y \vee Z)$, we will be using a special gadget:

Next, we need to connect gadgets.

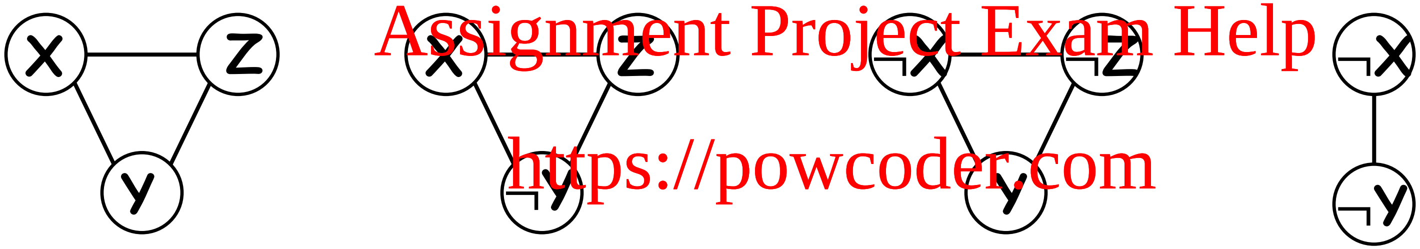


As an example, consider the following instance:

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

3SAT $\leq_p$ IndSet

$(X \lor Y \lor Z) \land (X \lor \neg Y \lor Z) \land (\neg X \lor Y \lor \neg Z) \land (\neg X \lor \neg Y)$

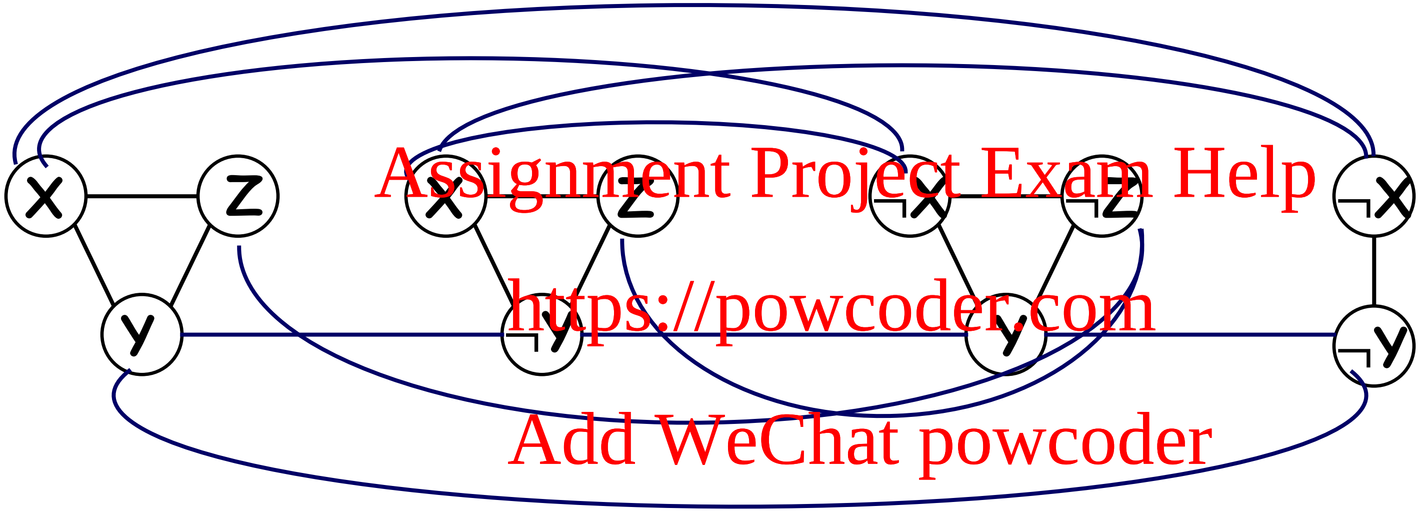


How do we connect gadgets?

<u>Claim:</u>

$3SAT \leq_p IndSet$

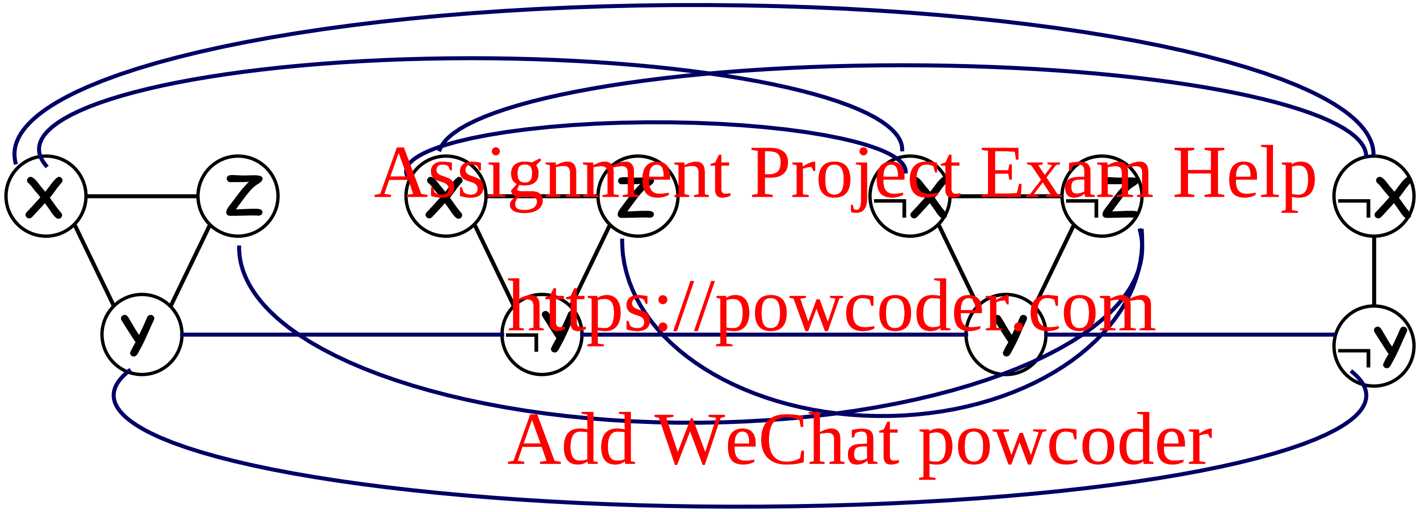
$(X \lor Y \lor Z) \land (X \lor \neg Y \lor Z) \land (\neg X \lor Y \lor \neg Z) \land (\neg X \lor \neg Y)$



Proof. $\Rightarrow$)

3SAT $\leq_p$ IndSet

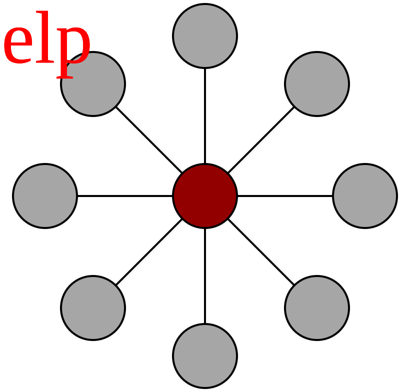
$(X \vee Y \vee Z) \wedge (X \vee \neg Y \vee Z) \wedge (\neg X \vee Y \vee \neg Z) \wedge (\neg X \vee \neg Y)$

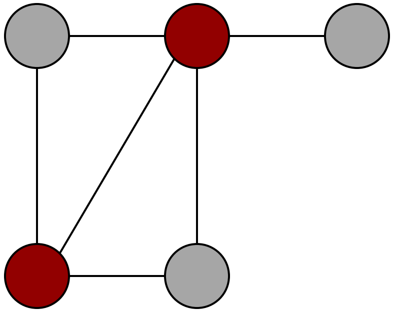


Proof. $\Leftarrow$)

Vertex Cover

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





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. $\Rightarrow$)

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. $\Leftarrow$)

Vertex Cover in NP-Complete

<u>Claim</u>: *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

Vertex Cover $\leq_p$ Ind. Set

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 $\leq_p$ VC-even

VC $\leq_p$ VC-even

Hamiltonian Cycle Problem

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

<u>Problem Statement:</u>

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.