Computing Theory

COMP 147

Last Time

- Class NP
- NP completeness

NP-Completeness

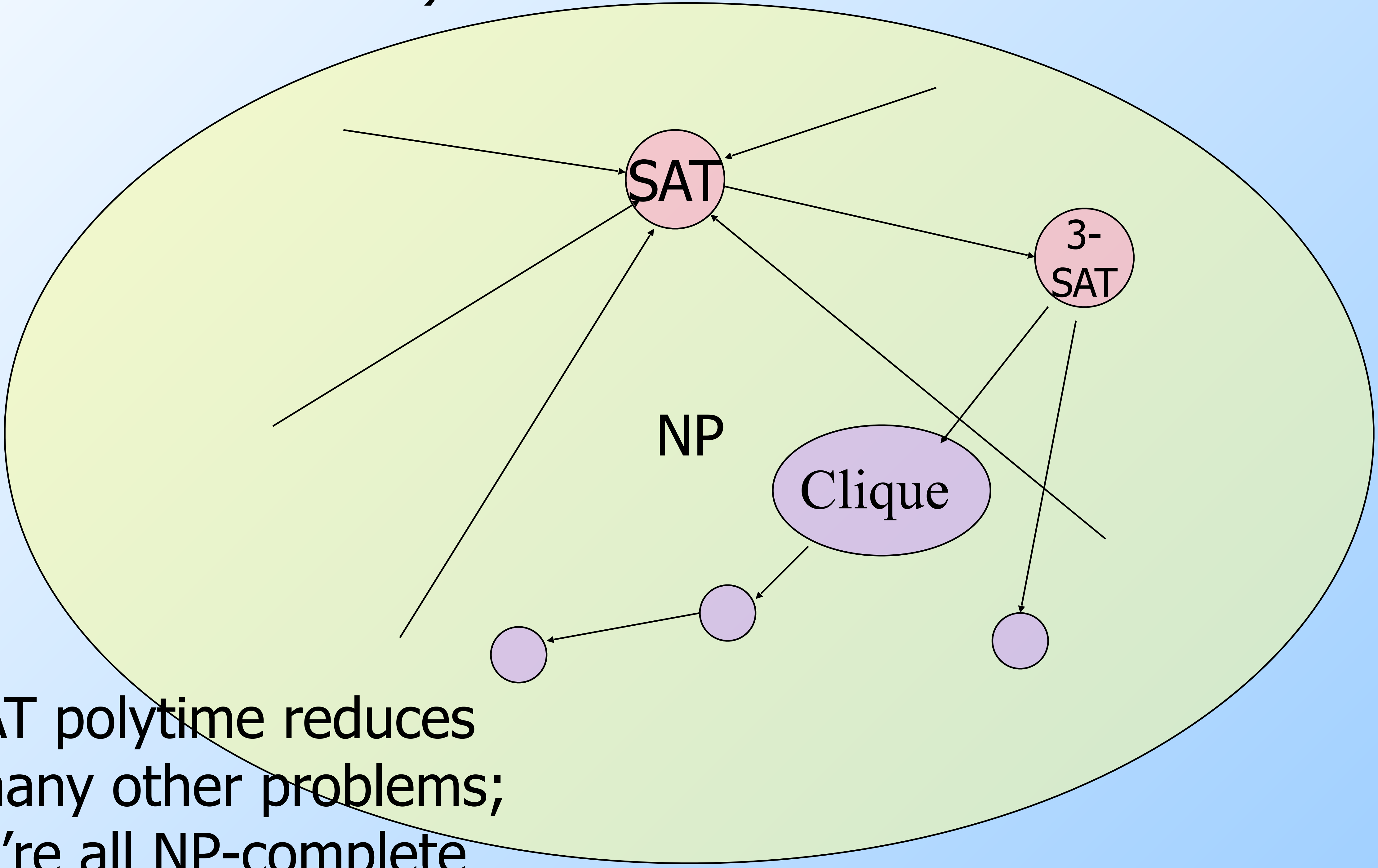
- A problem L is NP-complete if
  - 1. $L \in$ NP, and
  - 2. Every problem L' $\in$ NP,
    L' is polytime reducible to L in polynomial time

- L is as hard as any problem in NP

The Plan

All of **NP** polytime reduces to SAT, which is therefore NP-complete (Cook-Levin Theorem)

SAT polytime reduces to 3-SAT



3-SAT polytime reduces to many other problems; they're all NP-complete

Satisfiability (*SAT*)

SAT = { <φ> | φ is a satisfiable Boolean formula }

Boolean formula: an expression involving Boolean variables and operations (and ∧, or ∨, not)
Satisfiable: there is an assignment of values to the variables that makes the expression true.

Example: $\varphi = (\bar{x} \wedge y) \vee (x \wedge \bar{z})$

Satisfiable: $x=0, y=1, z=0$

Initial NP-complete Language

- Theorem 7.37: $\boldsymbol{SAT}$ is NP-complete

- Proof Idea:
  - 1. Show that $SAT \in$ NP (easy)
  - 2. Show that <u>any</u> language $A \in$ NP
    is ptime-reducible to SAT

  Given <A,w> we'll construct a Boolean formula
  ϕ that simulates the NP machine M for A on w.
  M accepts $w \Leftrightarrow \phi$ is satisfiable
  M doesn't accept $w \Leftrightarrow \phi$ is not satisfiable

Given that AND, OR and NOT are the basic components of digital computers,
it's not surprising that we can simulate a TM with a logical formula.
However, the devil is in the details.

3SAT

- A boolean formula is in Conjunctive Normal Form (CNF), it is the AND of clauses and each clause consists of the OR of literals.

- Any boolean expression can be written in CNF

- The problem k-SAT is SAT restricted to expressions in k-CNF.

- Example instance of 3SAT (k =3)

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$$

3SAT

- *3SAT* = { <φ> | φ is a satisfiable 3cnf formula }

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$$

- Each clause must have at least one true literal

- Example has 64 possible assignments

3SAT is NP-complete

- 3-SAT is in NP
- show 3-SAT is NP-complete by polytime reduction from SAT
- show SAT $\leq_P$ 3-SAT

3SAT – (2)

- First idea try to convert any boolean expression to 3-CNF
    - It is not true that every Boolean expression can be converted to an equivalent 3-CNF
- But we don't need equivalence
- We need to reduce every CNF expression E to some 3-CNF expression that is satisfiable if and only if E is.

Reducing SAT to 3-SAT

Suppose a clause contains k literals:

if $k = 1$ (meaning $C_i = \{z_1\}$), we can add in two new variables v_1 and v_2, and transform this into 4 clauses:

$\{v_1, v_2, z_1\}$ $\{v_1, \neg v_2, z_1\}$ $\{\neg v_1, v_2, z_1\}$ $\{\neg v_1, \neg v_2, z_1\}$

if $k = 2$ ($C_i = \{z_1, z_2\}$), we can add in one variable v_1 and 2 new clauses: $\{v_1, z_1, z_2\}$ $\{\neg v_1, z_1, z_2\}$

if $k = 3$ ($C_i = \{z_1, z_2, z_3\}$), we move this clause as-is.

Continuing the Reduction….

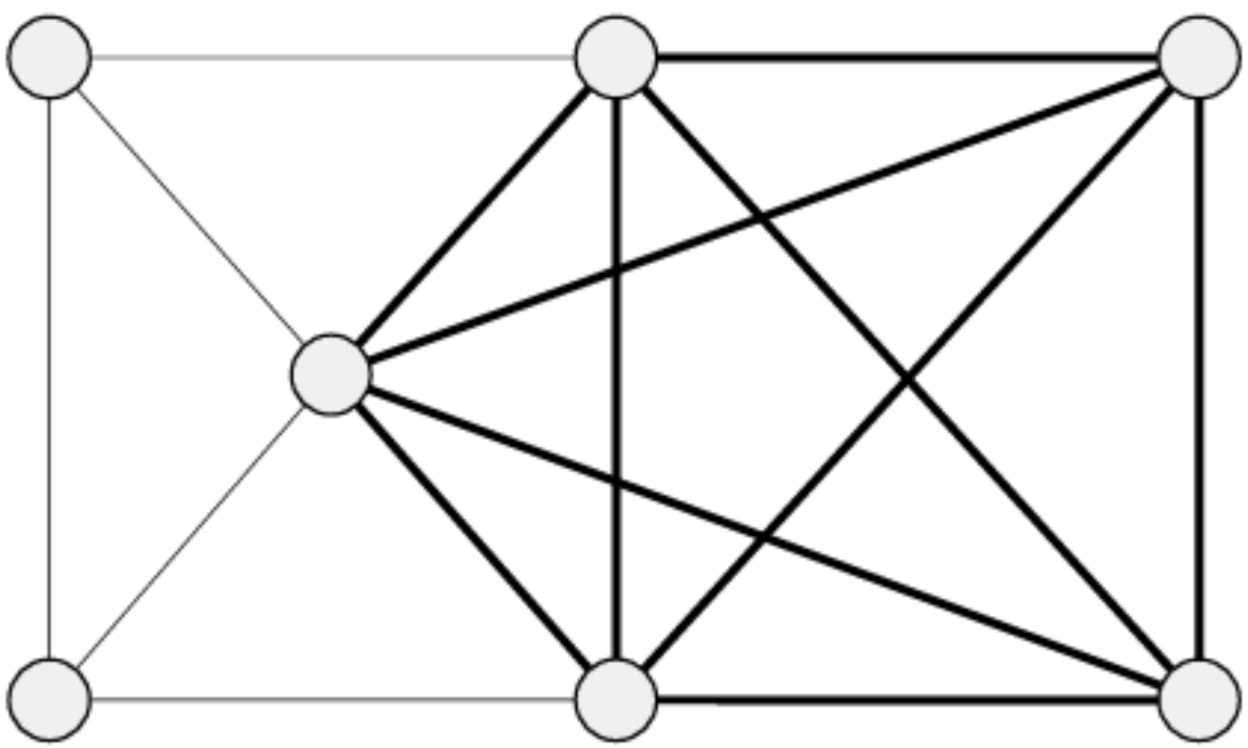
if $k > 3$ ($C_i = \{z_1, z_2, \ldots, z_k\}$) we can add in $k - 3$ new variables ($v_1, \ldots, v_{k-3}$) and $k - 2$ clauses:

$$\{z_1, z_2, v_1\} \ \{\neg v_1, z_3, v_2\} \ \{\neg v_2, z_4, v_3\} \ \ldots \ \{\neg v_{k-3}, z_{k-1}, z_k\}$$

Thus, in the worst case, n clauses will be turned into n^2 clauses. This cannot move us from polynomial to exponential time.

CLIQUE

- $CLIQUE = \{ \langle G,k \rangle \mid G$ is an undirected graph with a k-clique $\}$
  - clique = fully connected subgraph
  - k-clique = fully connected subgraph of k nodes

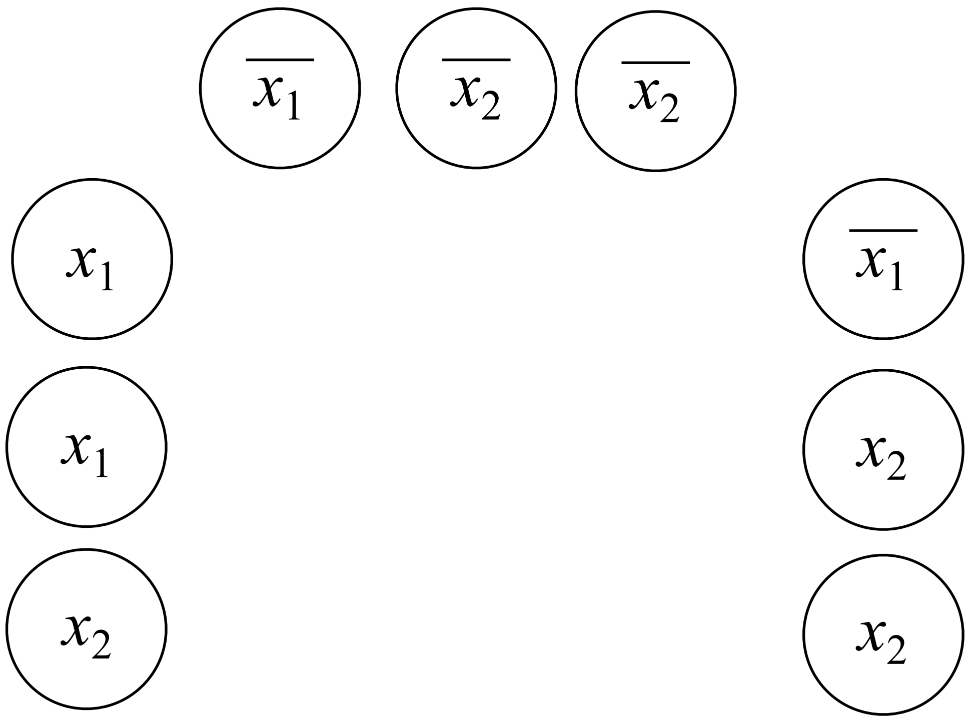


Graph with a 5-clique

$3SAT \leq_P CLIQUE$

- Given a 3cnf-formula with k clauses, generate a graph with $3k$ nodes, such that the formula is satisfiable iff the graph has a k-clique.

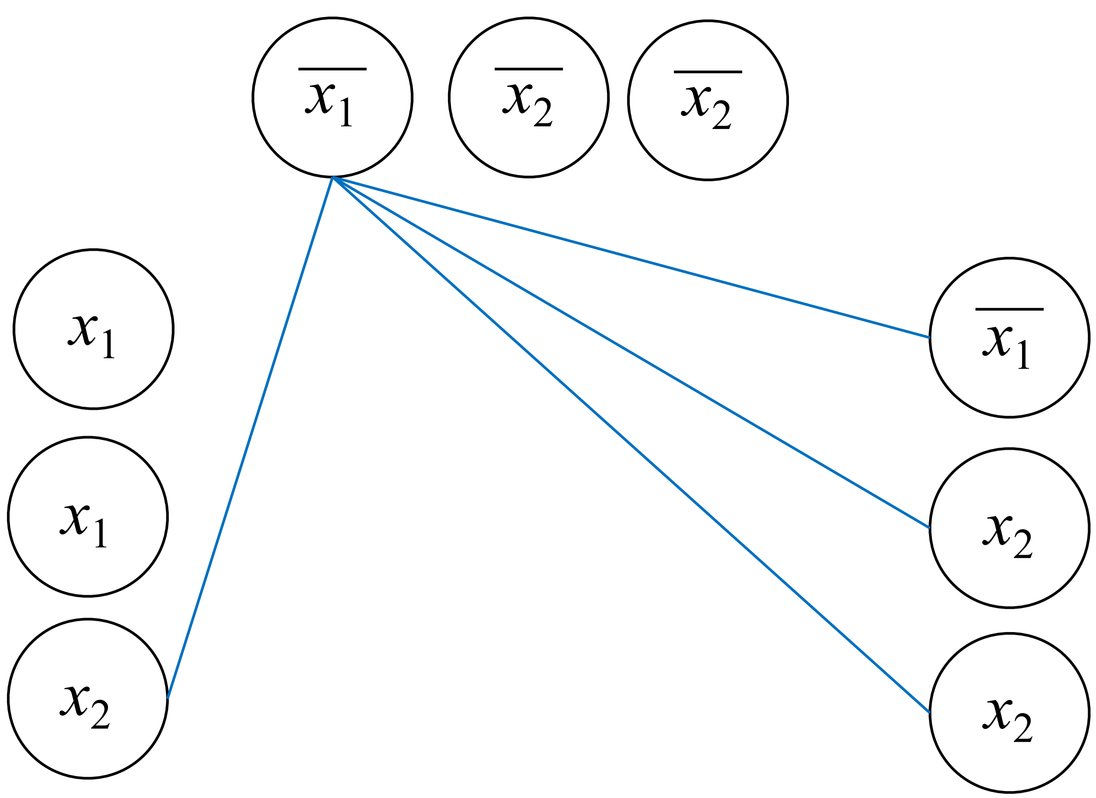
$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



3SAT $\leq_P$ CLIQUE

- Connect each pair of nodes, unless:
  - they are in the same clause, or
  - they have contradictory (negated) labels

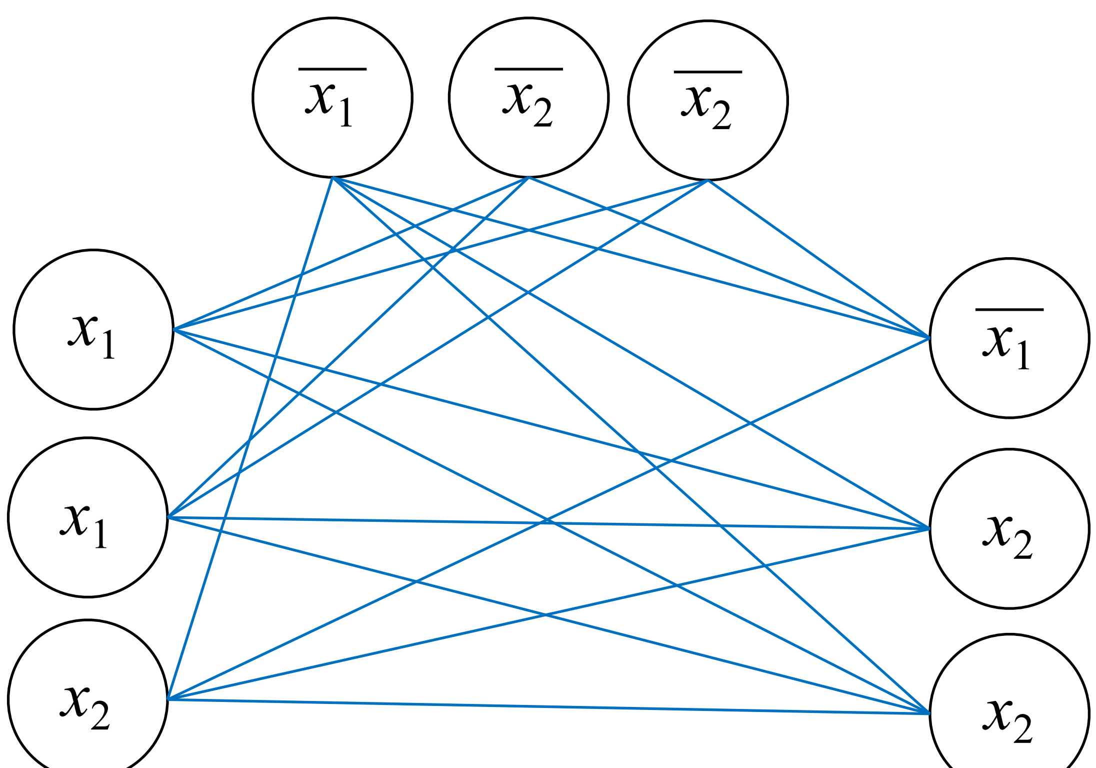
$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



$3SAT \leq_P CLIQUE$

- Connect each pair of nodes, unless:
  - they are in the same clause, or
  - they have contradictory (negated) labels

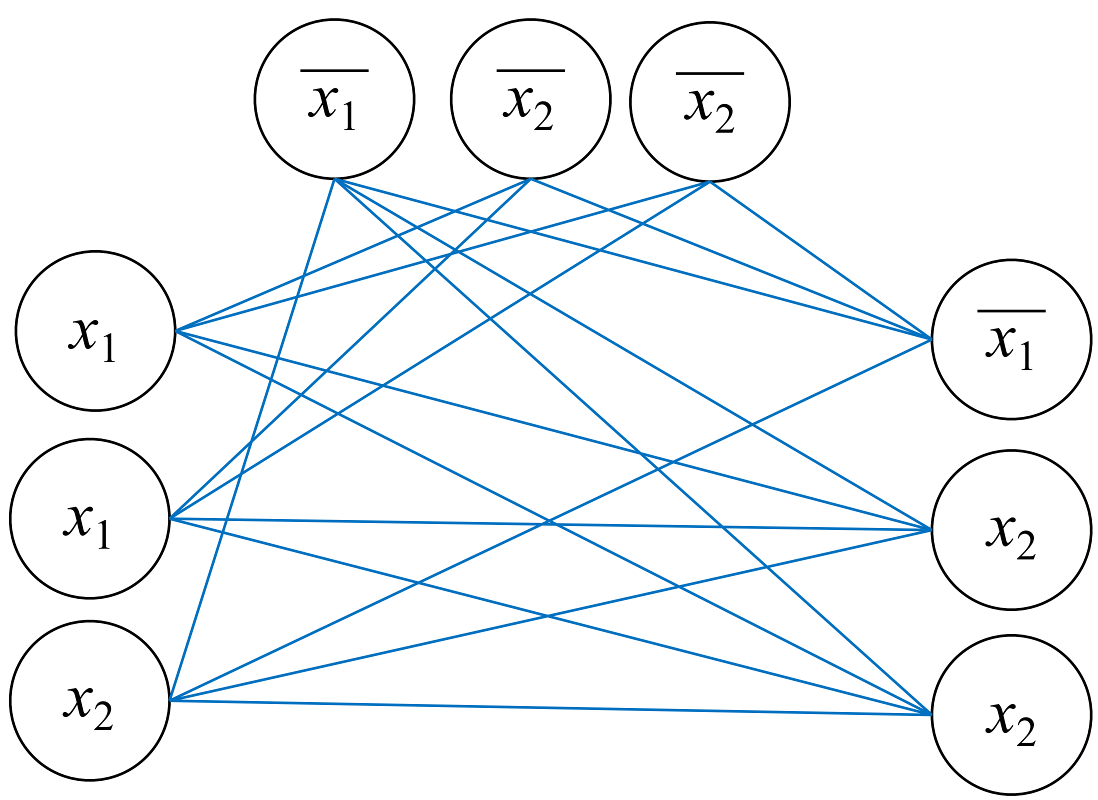
$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



$3SAT \leq_P CLIQUE$

- Any k-clique
  - has at most one node from each clause → has exactly one node from each clause
  - Does not contain contradictory assignments

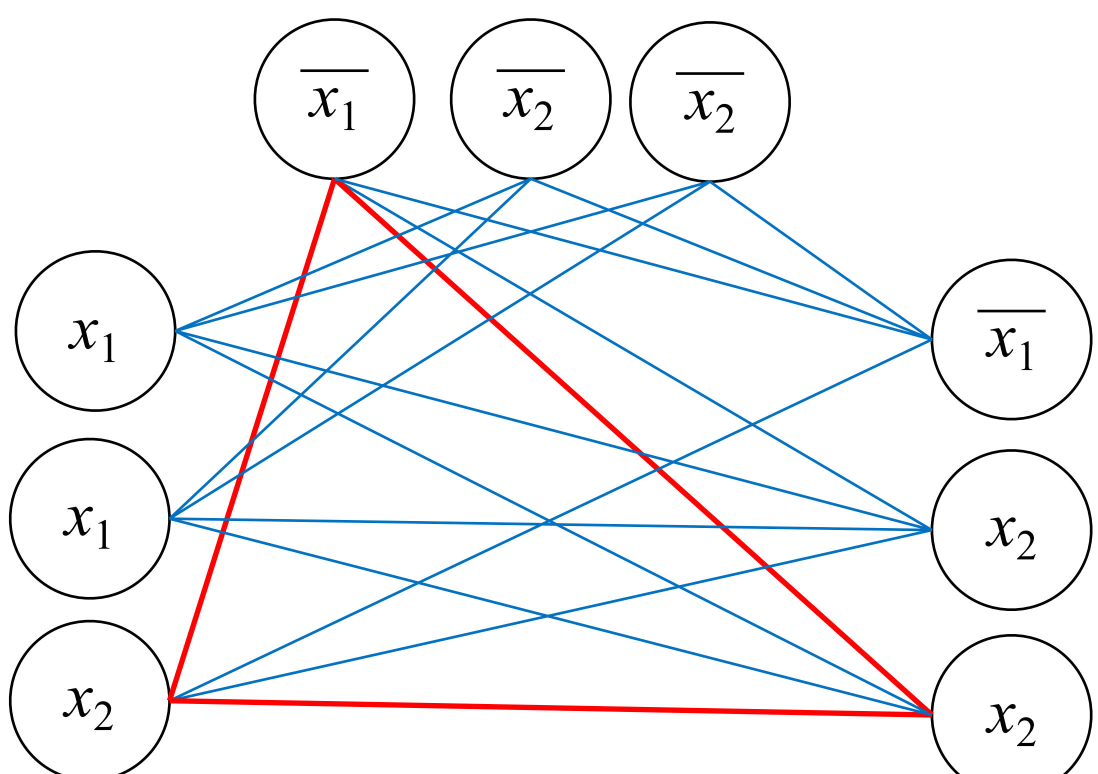
$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



$3SAT \leq_P CLIQUE$

- Any k-clique
  - has at most one node from each clause → has exactly one node from each clause
  - Does not contain contradictory assignments

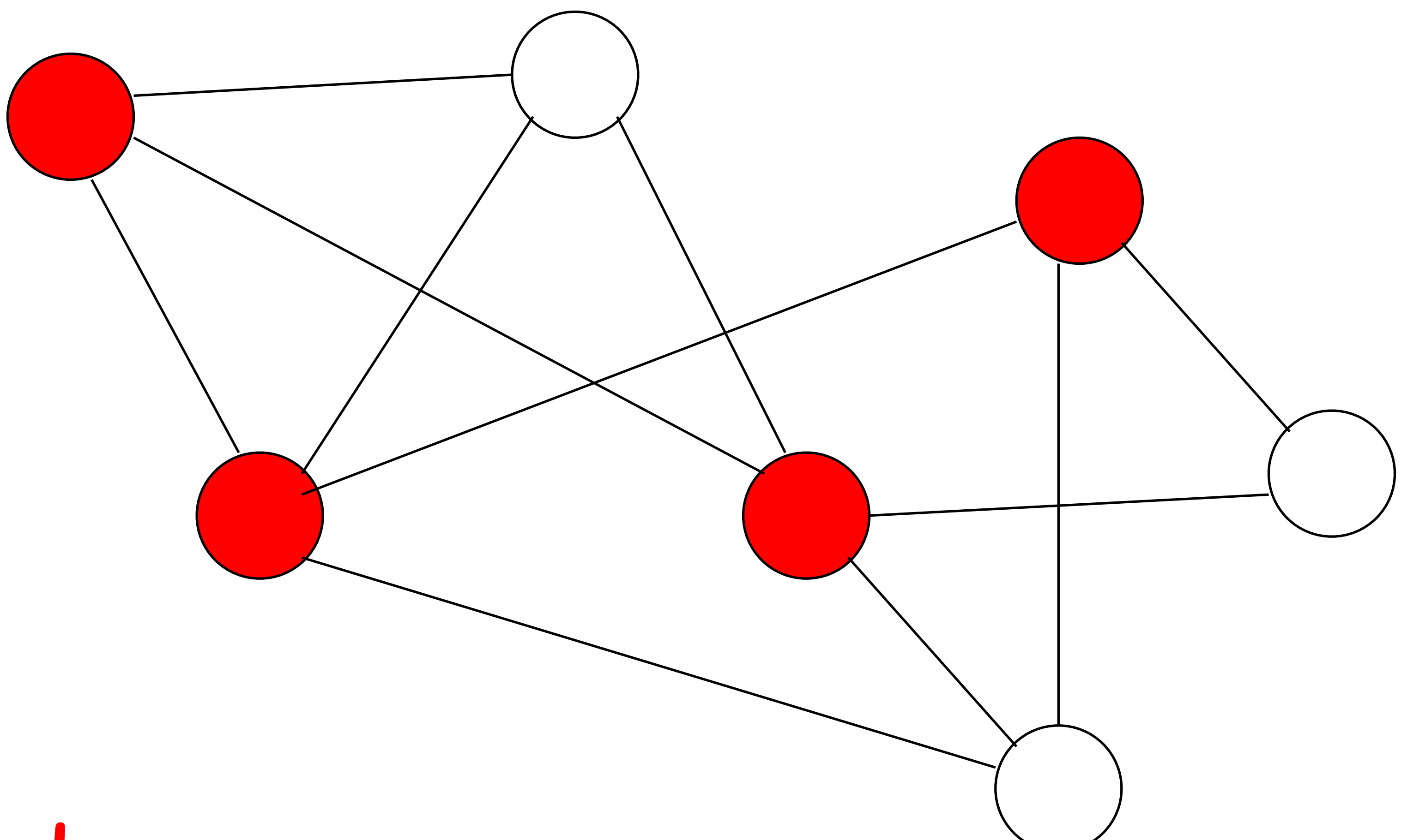
$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



Vertex Cover

Vertex cover of a graph is a subset of nodes S such that every edge in the graph touches one node in S

Example:

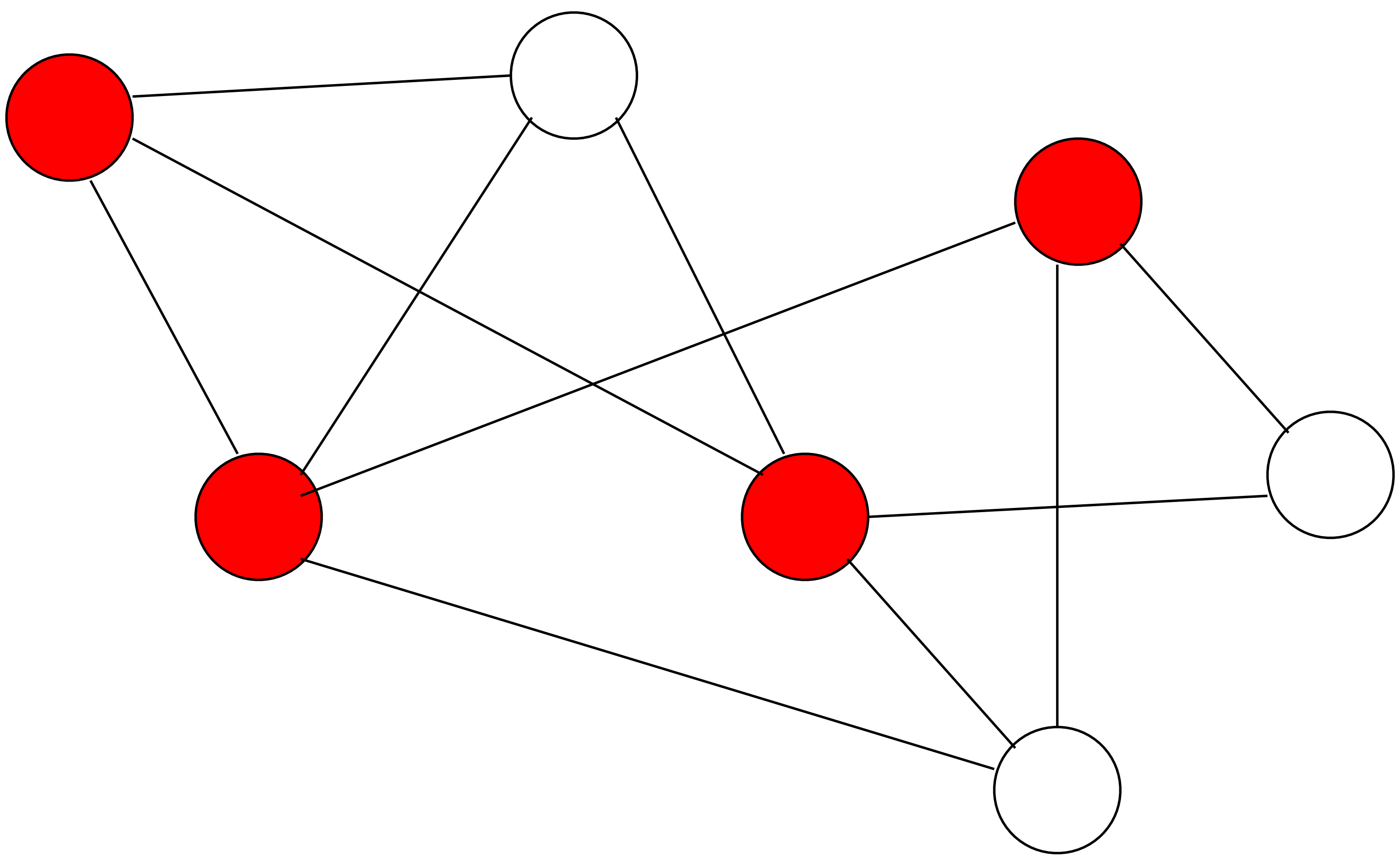


S = red nodes

Size of vertex-cover

is the number of nodes in the cover

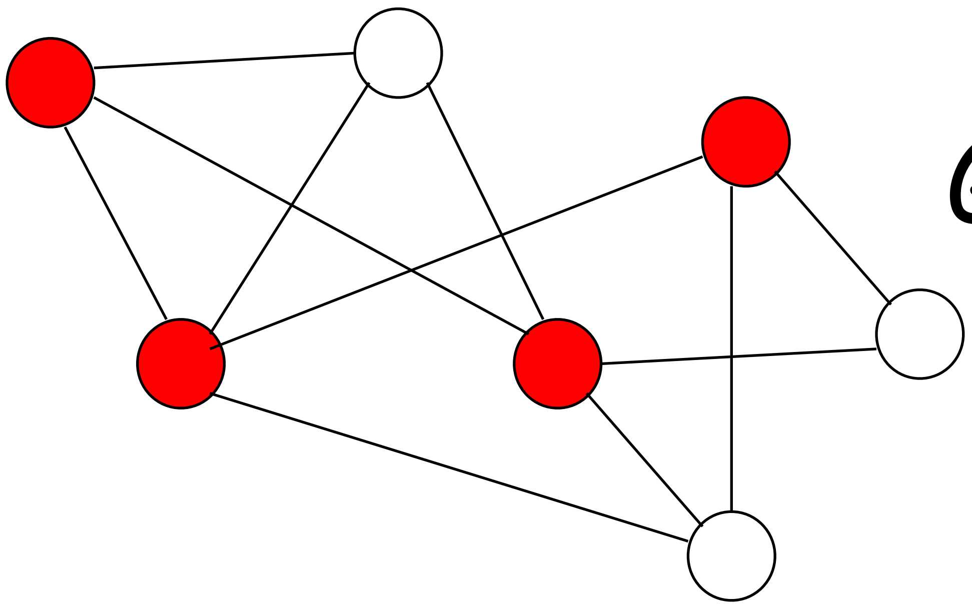
Example: $|S|=4$



Corresponding language:

VERTEX-COVER = { $\langle G, k \rangle$: graph G contains a vertex cover of size k }

Example:



G'

$\langle G', 4 \rangle \in \text{VERTEX-COVER}$

Theorem: VERTEX-COVER is NP-complete

Proof:

1. VERTEX-COVER is in NP

   Can be easily proven

2. We will reduce in polynomial time

   CLIQUE to VERTEX-COVER

   (NP-complete)

NP-Completeness Proof

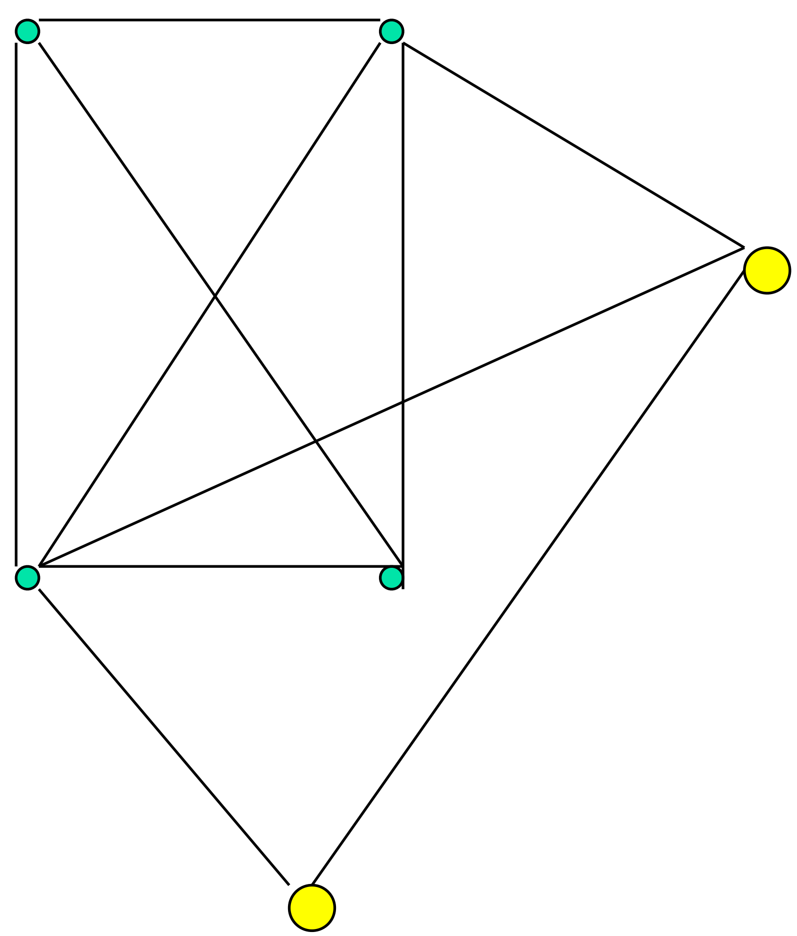
- Vertex Cover(VC): Given undirected $G=(V, E)$ and integer k, does G have a vertex cover with $\leq k$ vertices?
- CLIQUE: Does G contain a clique of size $\geq k$?

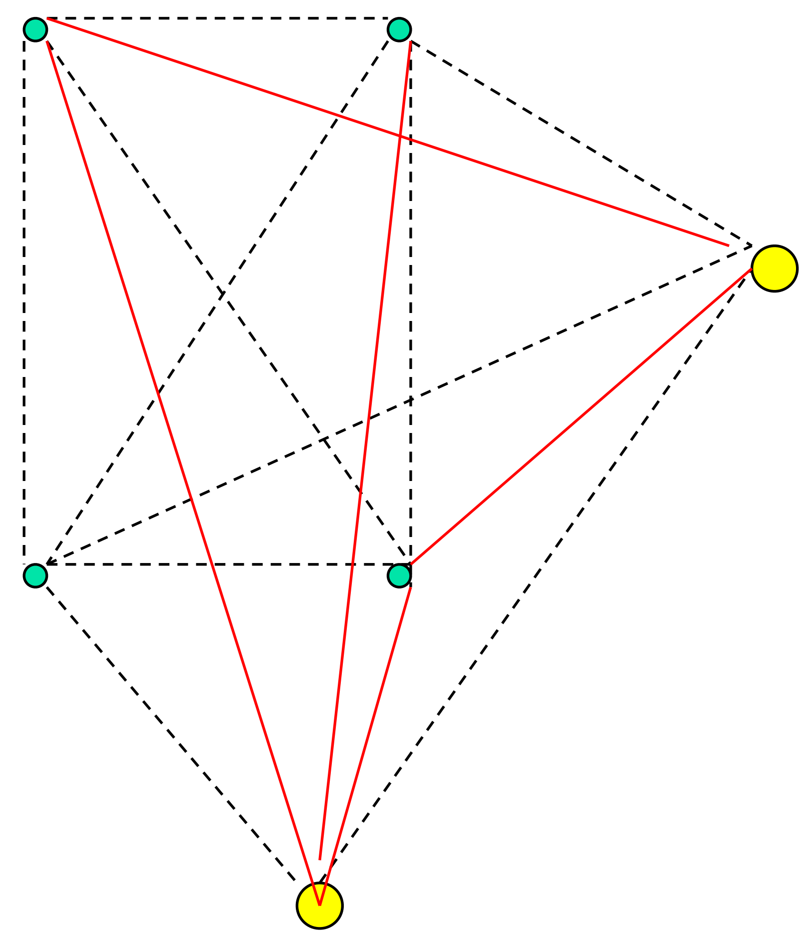
NP-Completeness Proof: Vertex Cover(VC)

- Problem: Given undirected G=(V, E) and integer k, does G have a vertex cover with ≤k vertices?
- Theorem: the VC problem is NP-complete.
- Proof: (Reduction from CLIQUE)
  - **VC is in NP**.
  - **Goal:** Transform arbitrary CLIQUE instance into VC instance such that CLIQUE answer is “yes” **iff** VC answer is “yes”.

NP-Completeness Proof: Vertex Cover(VC)

- **Claim:** CLIQUE(G, k) has **same** answer as VC ($\overline{G}$, n-k), where n = |V|.
- **Observe:** There is a clique of size k in G iff there is a VC of size n-k in $\overline{G}$.





NP-Completeness Proof: Vertex Cover(VC)

- **Observe:** If D is a VC in $\overline{G}$, then $\overline{G}$ has no edge between vertices in V-D.
  So, we have k-clique in G $\Leftrightarrow$ n-k VC in $\overline{G}$
- Can transform in polynomial time.

- You are working on a problem and you can't find a polynomial time algorithm

- Google is not being helpful

- What do you do?

    - Show NP-Completeness (List of NP-Complete Problems)

        - Look for similar problems

        - Example: Bioinformatics or merging source code similar to longest common subsequence

Dealing with NP-completeness

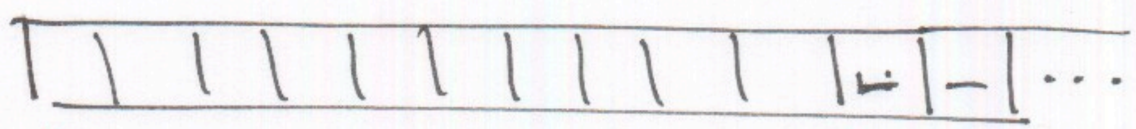
- Not all exponential time algorithmsS
    - Size of instance matters
    - Find “smart exponential” time algorithms
    - $2^{\sqrt{n}}$ represents a trillion improvement over 2^n when n = 50 (or 1.1^n)
    - Approximation algorithms
    - Randomized Algorithms

Note on PSPACE

SPACE COMPLEXITY

How to measure?

The number of cells on the tape that we visit.



THESE CELLS GOT USED AT LEAST ONCE.

NOT EVER VISITED

Note on PSPACE

- What is the relationship between P and PSPACE

- An algorithm that uses 30 tape cells must use at least 30 time steps.
- An algorithm that uses 30 tape cells may use many more steps.

$\boxed{P \subseteq PSPACE}$

WE KNOW:

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq EXPSPACE$$

$$P \subset EXPTIME$$

$$PSPACE \subset EXPSPACE$$