

# NP and NP-Complete

Mohammad Javad Dousti

# Changelog

## □ Rev. 1

- Added quiz question to slides 65 – 67.
- Updated TSP definition in slide 60. TSP takes a complete weighted graph.

# Overview

- ❑ Introduction
- ❑ P vs. NP
- ❑ Reduction
- ❑ NP-Complete
- ❑ NP-Complete Examples
- ❑ Sample Problems

# Introduction

- ❑ What has this semester been about?
  - We've taken problems you probably knew how to solve slowly, and we figured out how to solve them faster.
- ❑ In some sense, that's the job of a computer scientist. Figure out how to take our problems and make the computer do the hard work for us.
- ❑ Let's take a big step back, and try to break problems into three types:
  1. Those for which a computer might be able to help.
  2. Those which would take so long to solve even on a computer we wouldn't expect to solve them.
  3. Those which a computer cannot solve regardless of how long we wait.
- ❑ There are problems we could solve in finite time...but we'll all be long dead before our computer tells us the answer.

# Efficient (کارا یا کارآمد)

- We'll consider a problem *efficiently solvable*, if it has a polynomial time algorithm.
  - In other words, there's an algorithm which runs in  $O(n^k)$ , where  $k$  is a constant.
  - Are these algorithms always actually *optimum* (بهینه)?
    - Not necessarily!
- Your  $n^{10000}$  algorithm or even your  $2^{2^{2^2}}$ .  $n^3$  algorithm probably aren't going to finish anytime soon.
  - But these edge cases are rare, and polynomial time is good as a low bar.
  - If we can't even find an  $n^{10000}$  algorithm, we should probably rethink our strategy.

# Decision Problem vs. Optimization Problem

## Optimization Problem

- Try to optimize
- More complex
- Examples
  - **Max flow**: Find the maximum flow
  - **Shortest-path**: Find the shortest path
  - **Knapsack**: Find the maximum possible value.

## Decision Problem

- Output is *yes* or *no*
- Simple
- Examples
  - **Max flow**: Is there any feasible flow of size  $k$ ?
  - **Shortest-path**: Is there any path of length less than or equal to  $k$ ?
  - **Knapsack**: Is there any solution with the value of at least  $k$ ?

# Class P

A decision problem  $Q$  is in **P** if there is a **polynomial-time algorithm**  $A$  called **decider** such that for all inputs  $x$ :

- if  $x \in Q$  then  $A(x) = YES$ ,
- if  $x \notin Q$  then  $A(x) = NO$ ,

$$Q \in \mathbf{P} \leftrightarrow [\exists A \text{ such that } \forall x: x \in Q \leftrightarrow A(x) = \text{yes}]$$

## Examples:

- Connectivity problem
- Shortest path problem
- Summation

**Input:** Undirected graph  $G$

**Output:** Is  $G$  a connected graph?

**Input:** Directed weighted graph  $G$  and two vertices  $s$  and  $t$  and a value of  $k$

**Output:** Is there any path between  $s$  and  $t$  with the length of at most  $k$ ?

**Input:** Three numbers  $x$ ,  $y$ , and  $z$

**Output:** Is  $x + y = z$ ?

# Class NP (Non-deterministic Polynomial)

A decision problem  $Q$  is in **NP** if there is a **polynomial-time algorithm**  $V$  called **verifier** such that for all inputs  $x$ :

- if  $x \in Q$  then there is a **certificate**  $y$  such that  $V(x, y) = YES$ ,
- if  $x \notin Q$  then for all certificates  $y$  we have  $V(x, y) = NO$ ,

Size of  $y$  should be a polynomial of size of  $x$

$Q \in \mathbf{NP} \leftrightarrow [\exists V \text{ such that } \forall x: x \in Q \leftrightarrow (\exists y \text{ such that } V(x, y) = \text{yes})]$

## Examples:

- Traveling Sales Man
- Clique
- Longest path

**Input:** Directed weighted graph  $G$  and a value of  $k$   
**Output:** Is there any tour of length at most  $k$ ?

**Input:** Undirected graph  $G$  and a value of  $k$   
**Output:** Can we find  $k$  vertices in graph  $G$  such that they are all adjacent to each other?

**Input:** Directed weighted graph  $G$  and two vertices  $s$  and  $t$  and a value of  $k$   
**Output:** Is there any path between  $s$  and  $t$  with the length of at least  $k$ ?



# What if the $|y|$ is not polynomial in $|x|$ ?

- ❑ Note that the runtime of a verifier (or generally any algorithm) is **defined in terms of the input size**.
- ❑ If  $|y|$  is arbitrarily large, then there is an algorithm that can verify a given NP problem, in polynomial time w.r.t.  $|y|$ , but not necessarily polynomial in  $|x|$ .
- ❑ Does a given program  $P$  halts (finishes) in less than or equal to  $2^n$  steps?
  - This problem can be shown not to be in NP.
  - If we could use an arbitrarily large certificate, one can pass this certificate:  $\langle c_0, c_1, c_2, \dots, c_m \rangle$ , where  $c_i$  is the configuration of the program after each step.
  - A simple verifier can walk through these certificates and check the following properties:
    1. All  $c_i$ 's are legit configurations.
    2. Any  $c_i \rightarrow c_{i+1}$  is a legit step.
    3.  $m \leq 2^n$

# The \$1M Question

## □ The Clay Mathematics Institute: [Millennium Prize Problems](#)

1. Birch and Swinnerton-Dyer Conjecture
2. Hodge Conjecture
3. Navier-Stokes Equations
- 4. P vs. NP**
5. Poincaré Conjecture ← **Solved in 2002 by Grigori Perelman**
6. Riemann Hypothesis
7. Yang-Mills Theory



# The P versus NP problem


- ❑ Is one of the biggest open problems in computer science (and mathematics) today.
- ❑ It's currently unknown whether there exist polynomial time algorithms for NP-complete problems
  - That is, does  $P = NP$ ?
  - People generally believe  $P \neq NP$ , but no proof yet.
- ❑ But what is the P-NP problem?

# $P \subseteq NP$ Proof

A decision problem  $Q$  is in **P** if there is an **polynomial-time algorithm**  $A$  called **decider** such that for all inputs  $x$ :

- if  $x \in Q$  then  $A(x) = YES$ ,
- if  $x \notin Q$  then  $A(x) = NO$ ,

$$Q \in \mathbf{P} \leftrightarrow [\exists A \text{ such that } \forall x: x \in Q \leftrightarrow A(x) = \text{yes}]$$


$$V(x, y) = A(x)$$

A decision problem  $Q$  is in **NP** if there is an **polynomial-time algorithm**  $V$  called **verifier** such that for all inputs  $x$ :

- if  $x \in Q$  then there is a **certificate**  $y$  such that  $V(x, y) = YES$ ,
- if  $x \notin Q$  then for all certificates  $y$  we have  $V(x, y) = NO$ ,

$$Q \in \mathbf{NP} \leftrightarrow [\exists V \text{ such that } \forall x: x \in Q \leftrightarrow (\exists y \text{ such that } V(x, y) = \text{yes})]$$

Is **P** a proper subset of **NP**?

# Polynomial-Time Reductions (تقلیل یا تحویل)

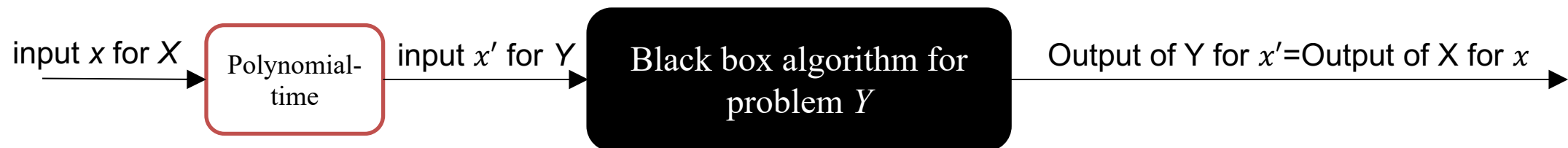
- ❑ The purpose of a reduction is to show that *some problem is at least as hard as some other problem*.
- ❑ If problem  $X$  reduces to problem  $Y$ , then solving  $Y$  implies solving  $X$ .
  - $Y$  is at least as hard as  $X$ , denoted  $X \leq Y$ .
- ❑ Reduction types:
  - **Karp reduction**
    - We use it in this course to prove NP-hardness of problems.
  - **Cook reduction**
    - We don't talk about it in this course.
  - **Levin reduction**
    - We don't talk about it in this course.

# Karp Reduction

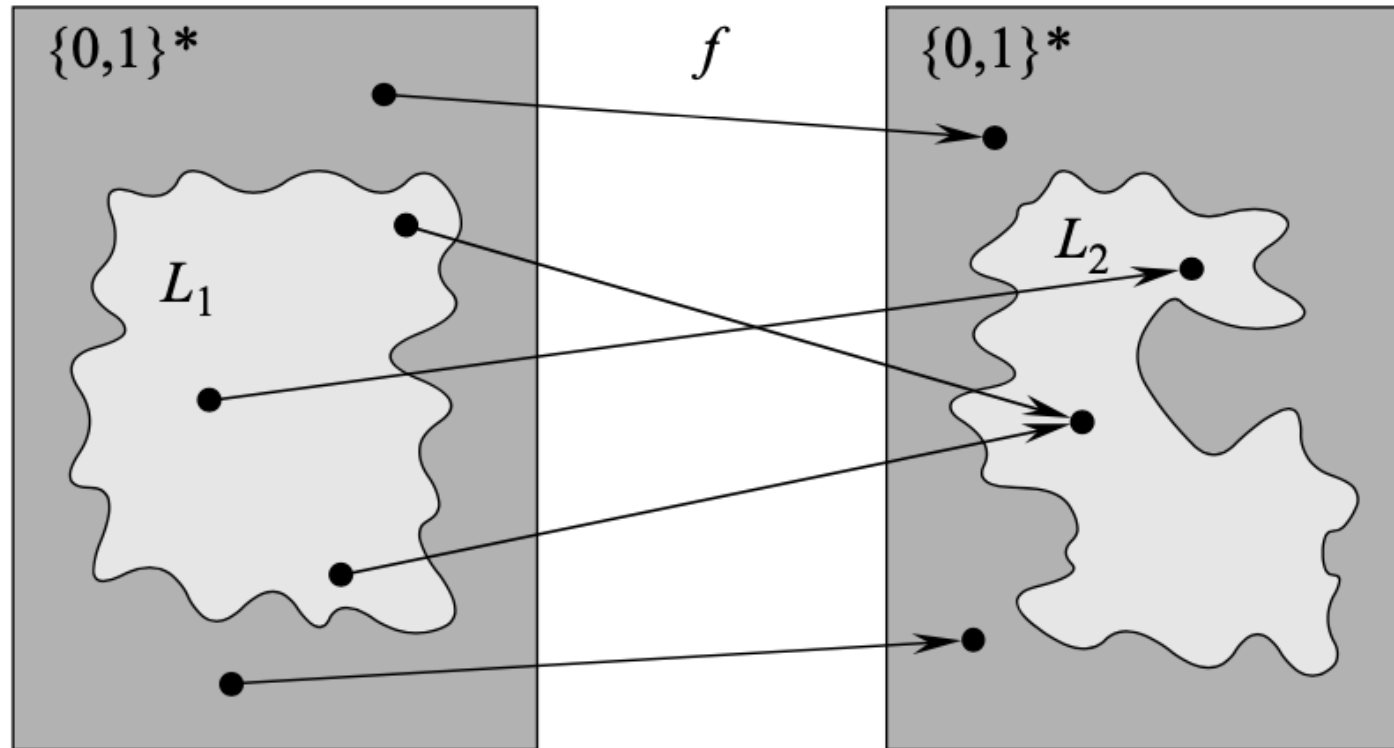
- A polynomial time reduction from a decision problem  $X$  to a decision problem  $Y$  is an algorithm  $f$  that has the following properties:
  - Given an instance  $I_X$  of  $X$ ,  $f$  produces an instance  $I_Y$  of  $Y$ .
  - $f$  runs in polynomial time w.r.t.  $|I_X|$ . This implies that  $|I_Y|$  (size of  $I_Y$ ) is polynomial in  $|I_X|$ .
  - Answer to  $I_X$  YES iff answer to  $I_Y$  is YES. In other words,  $x \in X \iff f(x) \in Y$ .
- **Karp reduction** is also called **many-one reduction** and **polynomial transformations**.
- **Notation:**  $X \leq_P Y$  (or  $X \leq_m^P Y$ ) if  $X$  reduces to  $Y$ .
- **Proposition:** If  $X \leq_P Y$ , then a polynomial time algorithm for  $Y$  implies a polynomial time algorithm for  $X$ .



Richard M. Karp



# Karp Reduction (cont'd)



# Karp Reduction: Example 1

## Matching

**Input:** Undirected bipartite graph  $G$  and a value of  $k$

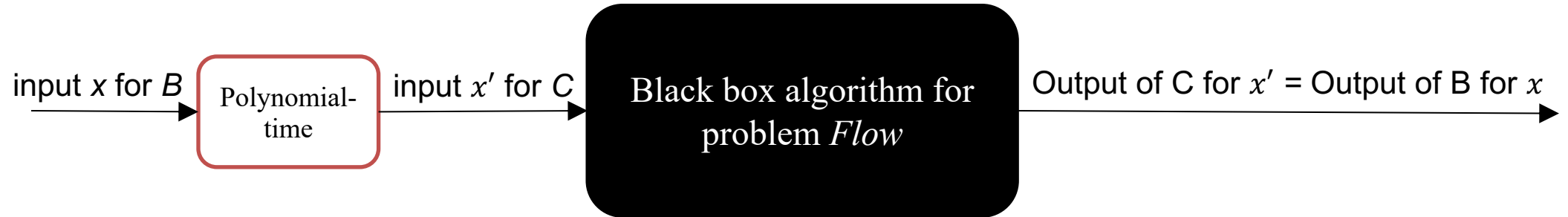
**Output:** Does  $G$  have a matching of size at least  $k$ ?

$\text{Matching} \leq_p \text{Flow}$

## Flow

**Input:** Network flow  $G$  and a value of  $k$

**Output:** Does  $G$  has a feasible flow of at least  $k$ ?





# Karp Reduction: Example 2

## Independent Set

**Input:** Undirected graph  $G$  and a value of  $k$

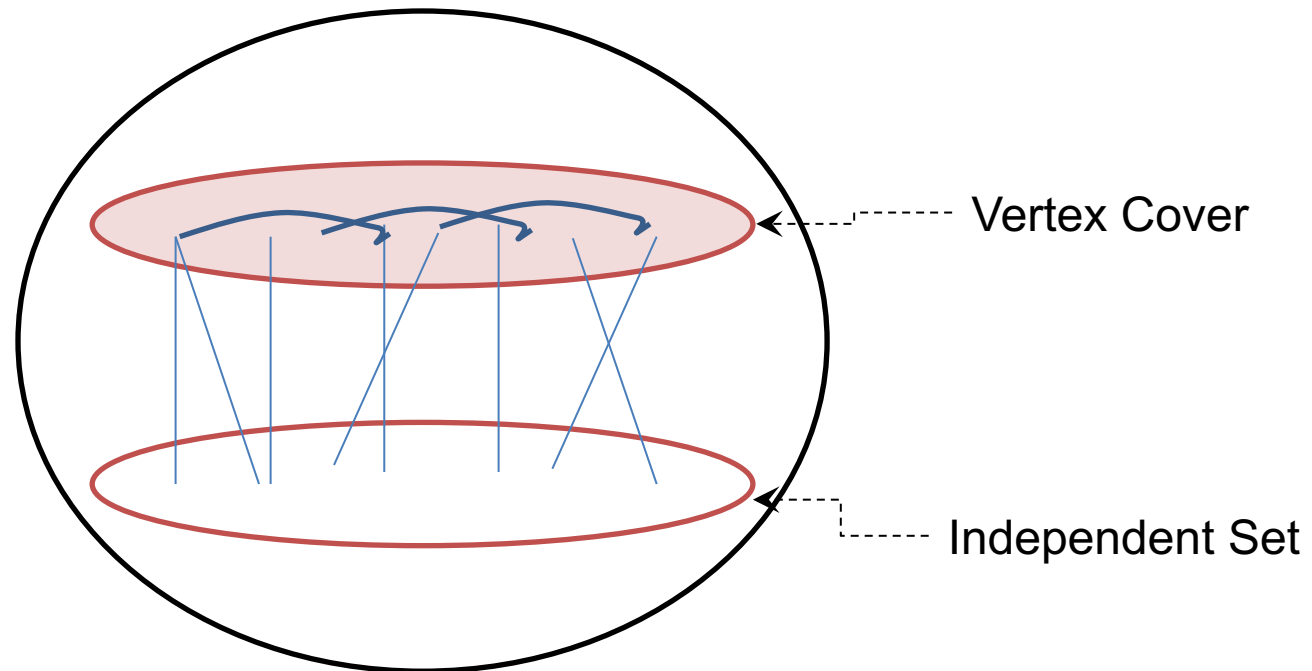
**Output:** Is there any set of vertices of size  $k$  in  $G$  that none of them are adjacent?

## Vertex Cover

**Input:** Undirected graph  $G$  and a value of  $k$

**Output:** Can we color  $k$  vertices of  $G$  such that for each edge one of its endpoints is colored?

Independent Set  $\leq_P$  Vertex Cover



# Karp Reduction: Example 2 (cont'd)

## Independent Set

**Input:** Undirected graph  $G$  and a value of  $k$

**Output:** Is there any set of vertices of size  $k$  in  $G$  that none of them are adjacent?

## Vertex Cover

**Input:** Undirected graph  $G$  and a value of  $k$

**Output:** Can we color  $k$  vertices of  $G$  such that for each edge one of its endpoints is colored?

Independent Set  $\leq_P$  Vertex Cover



# Polynomial-Time Karp Reduction Transitivity

□ **Theorem:** If  $A \leq_p B$  and  $B \leq_p C$  then  $A \leq_p C$ .

□ **Proof:**

- Per definition,  $\exists f, g$  such that  $x \in A \iff f(x) \in B$  and  $y \in B \iff g(y) \in C$ .
- $x \in A \iff f(x) \in B \iff g(f(x)) \in C$ .
- $g(f(.))$  is polynomial because  $f(x)$  is polynomial in  $x$ .

# NP-Complete and NP-Hard

NP-Complete

- The most difficult problems in NP to solve
- If we can solve an NP-complete problem in polynomial time, we can solve all NP problems in polynomial time.

**$Q \in \text{NP} - \text{Complete}:$**

- $Q \in \text{NP}$
- $\forall Q' \in \text{NP}, \text{ we have } Q' \leq_p Q$

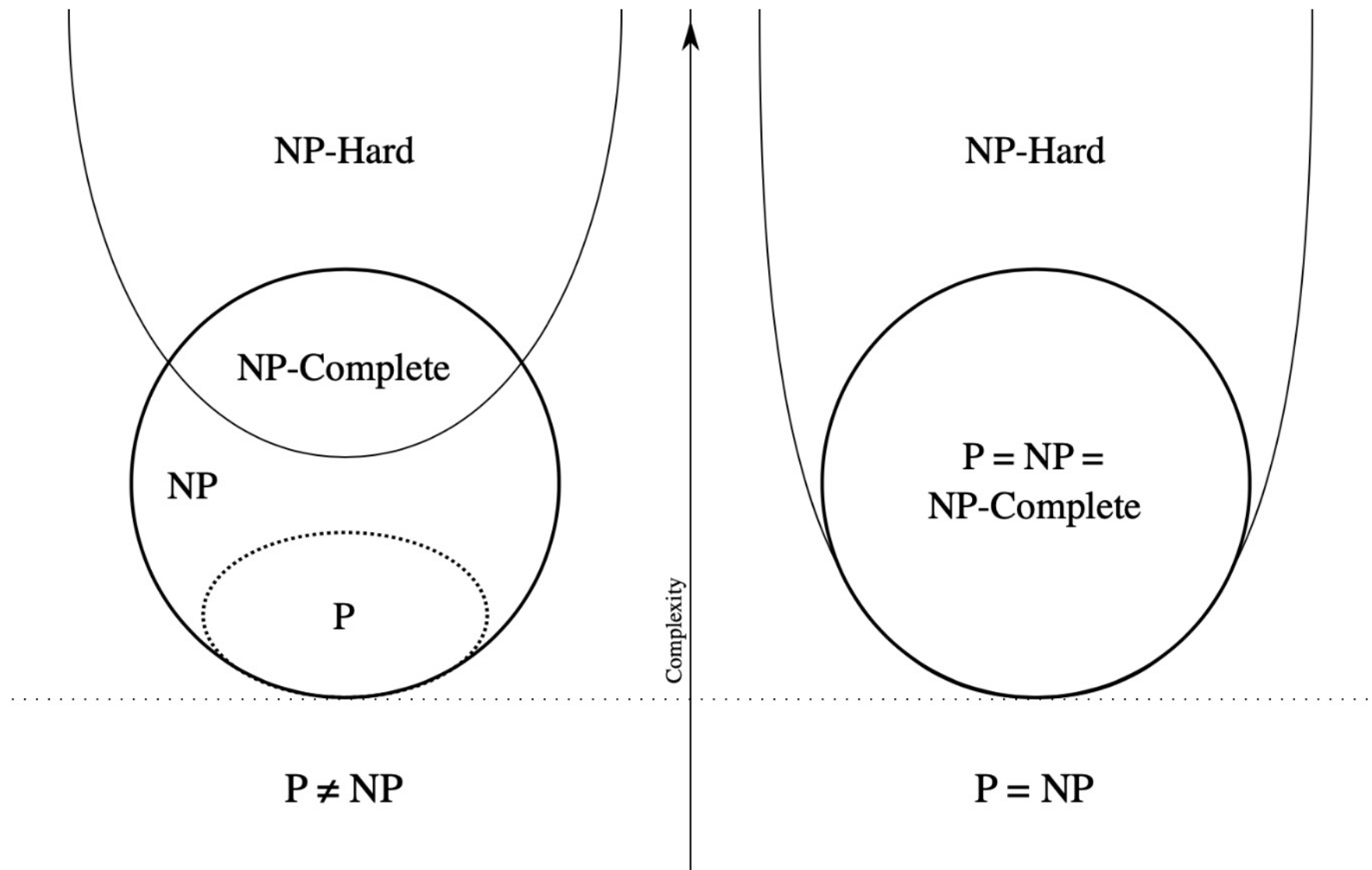
NP-Hard

- If we can solve an NP-hard problem in polynomial time, we can solve all NP problems in polynomial time.
- They are not necessarily in NP.

**$Q \in \text{NP} - \text{hard}:$**

- $\forall Q' \in \text{NP}, \text{ we have } Q' \leq_p Q$

# NP-Complete and NP-Hard

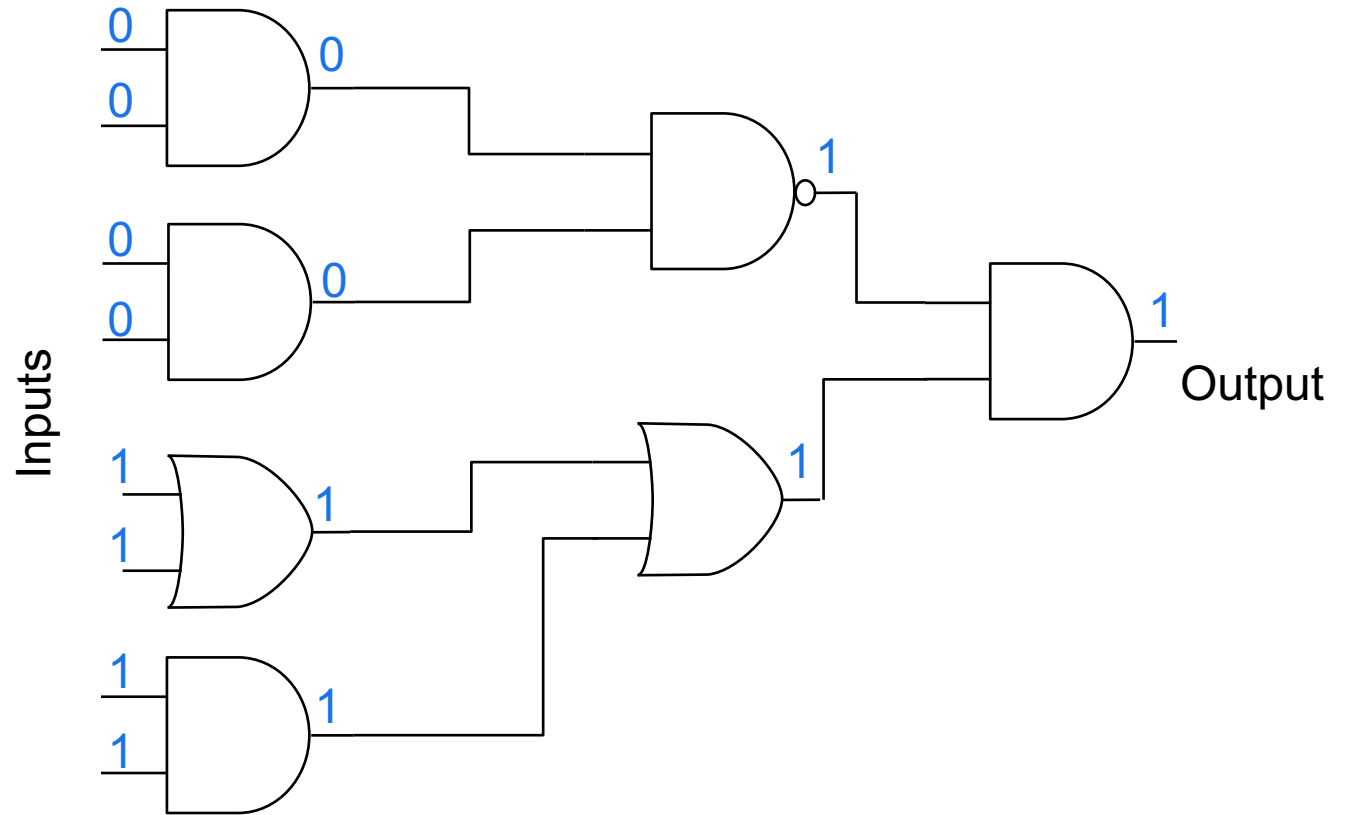


# The First NP-Complete Problem

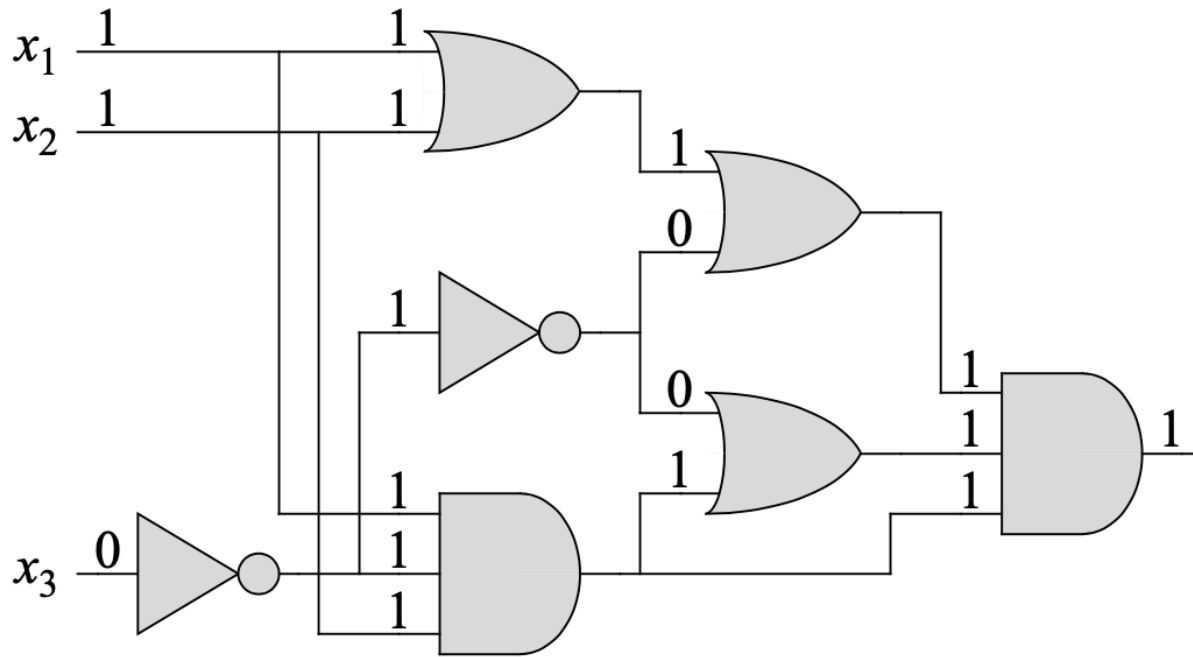
## Circuit Satisfiability (CS)

**Input:** Logical circuit with AND, OR, and NOT gates with  $n$  inputs,  $m$  gates and one output

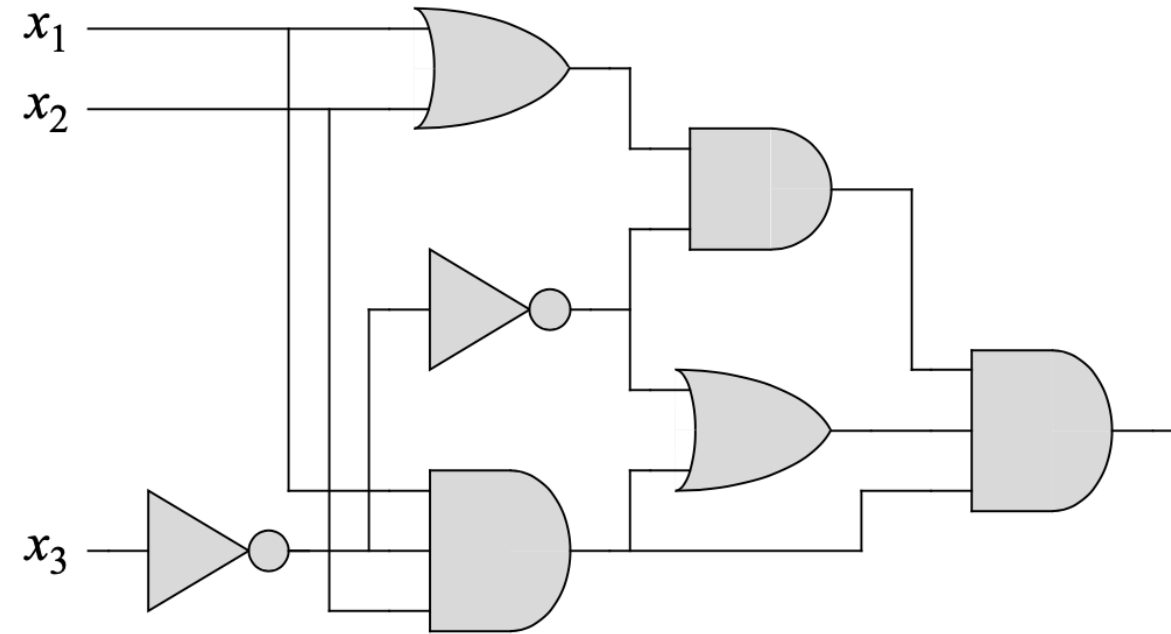
**Output:** Can we set  $n$  inputs such that the output becomes 1?



# CS Problem: Example



Satisfiable



Unsatisfiable

# NP-Completeness of CS Problem: Proof

- This is the first NP-complete problem we prove.
  - Two steps are required to prove the theorem:
    - $\text{Circuit-SAT} \in \mathbf{NP}$
    - $\forall Q \in \mathbf{NP}$ , we have  $Q \leq_p \text{Circuit-SAT}$
- $Q \in \mathbf{NP} \leftrightarrow [\exists A \text{ such that } \forall x: x \in Q \leftrightarrow (\exists y \text{ such that } A(x, y) = \text{yes})]$

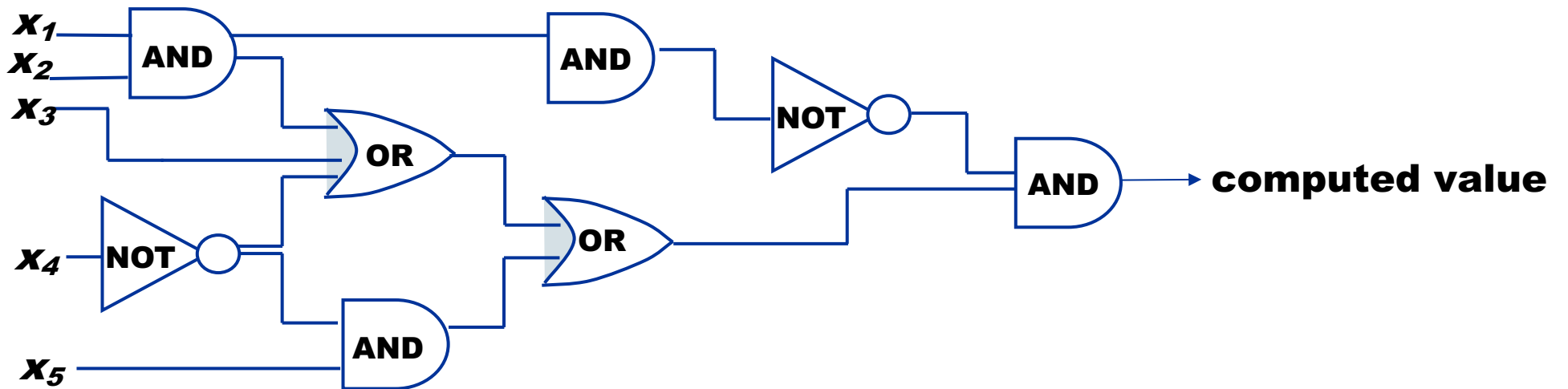


# CS Problem is NP

□ **Lemma 1:** Circuit-SAT is in NP

□ **Proof:**

- Must show that there exists a polynomial-time verifier.
- We can easily check in polynomial time if truth assignment produces TRUE.



# CS Problem is NP-Hard (1)

□ **Lemma 2:** For every problem  $Q$  in NP, we have:  $Q \leq_p \text{Circuit-SAT}$

➤ Or Circuit-SAT is NP-hard.

□ **Proof:** (sketch of the proof; for full sketch, see CLRS 34.3)

- If an algorithm runs in polynomial time, then there is a polynomial-size Boolean circuit that “implements” the algorithm, and such a circuit can be constructed in polynomial time.
- **Idea:** Algorithm runs on computer that is essentially a Boolean circuit.
- To complete the proof, we should create a reduction function  $f$  such that for any  $x \in Q$ , we have  $f(x) \in \text{Circuit-SAT}$ , and vice versa.

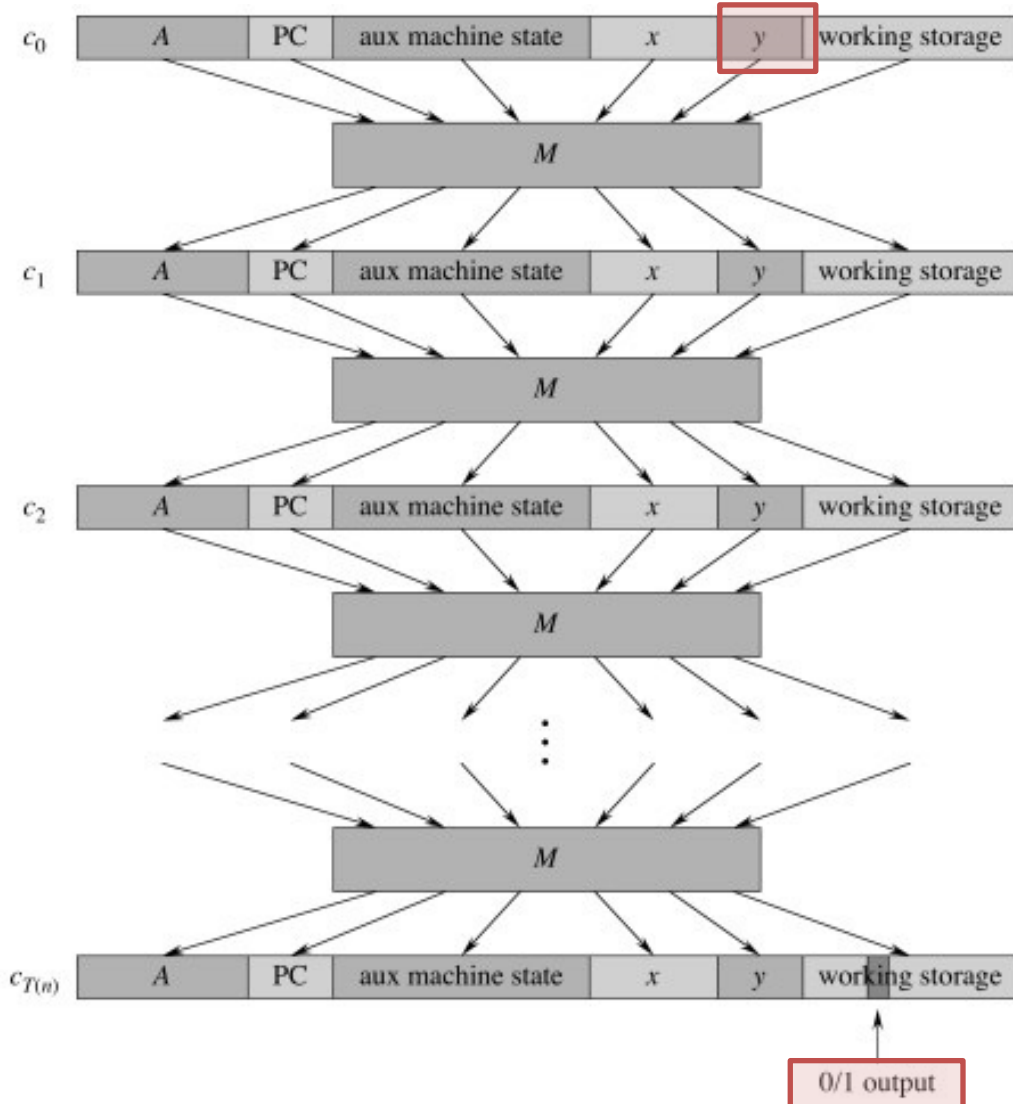
# CS Problem is NP-Hard (2)

□ **Step 2:** Describe algorithm that performs reduction:

- $Q$  is in NP, so  $Q$  has a polynomial-time verification algorithm  $A(x,y)$  that checks if  $x$  is a “yes”-instance using certificate  $y$ .
- Construct circuit implementing  $A$ .
  - Circuit runs in polynomial time.
    - Input has size polynomial.
    - A combinational circuit implementing a mapping on the polynomial-size input has size polynomial.
    - A polynomial-sized circuit can run in polynomial time.
- “Fix” the variables corresponding to  $x$  according to the given input
- Run Circuit-SAT:
  - Circuit-SAT returns “yes” iff  $x$  is a “yes” instance for problem  $Q$ .

# CS Problem is NP-Hard (3)

Fix input  $x$ , we want to decide whether  $x \in Q$



If runtime of Algorithm  $A$  for  $x$  is  $T(n)$ , then the size of our logical circuit is polynomial of  $T(n)$ .

# NP-Complete Examples

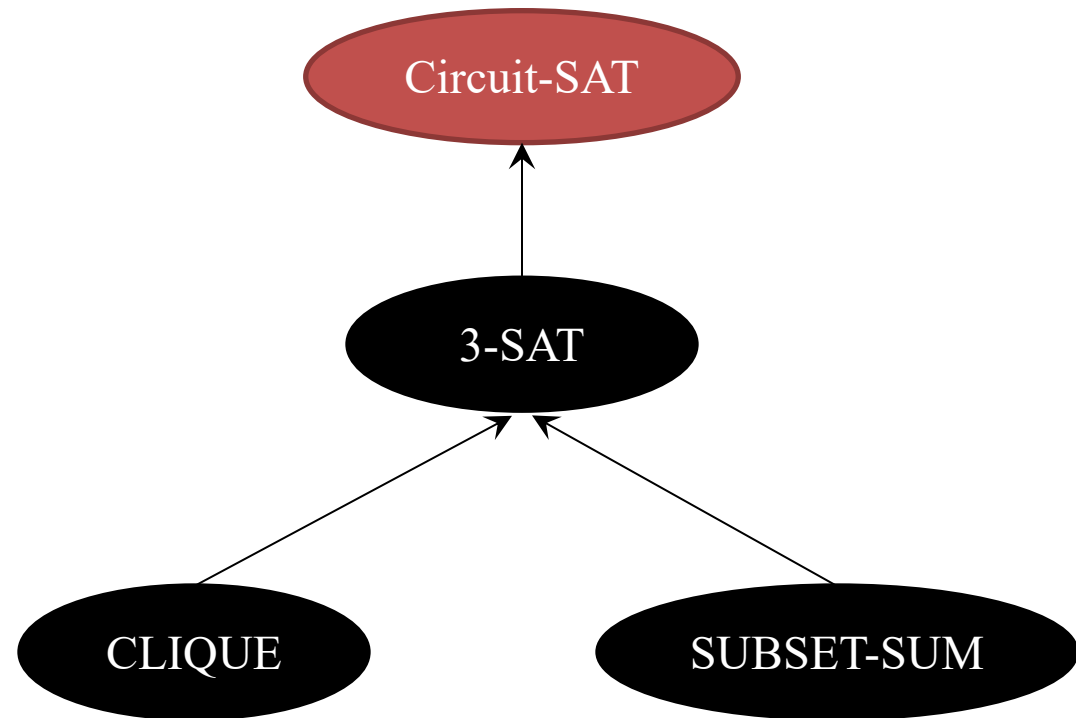
# Structure of NP-Completeness Proofs

Now, we have a strong tool to prove  
a new problem  $Q$  is NP-Complete

↓

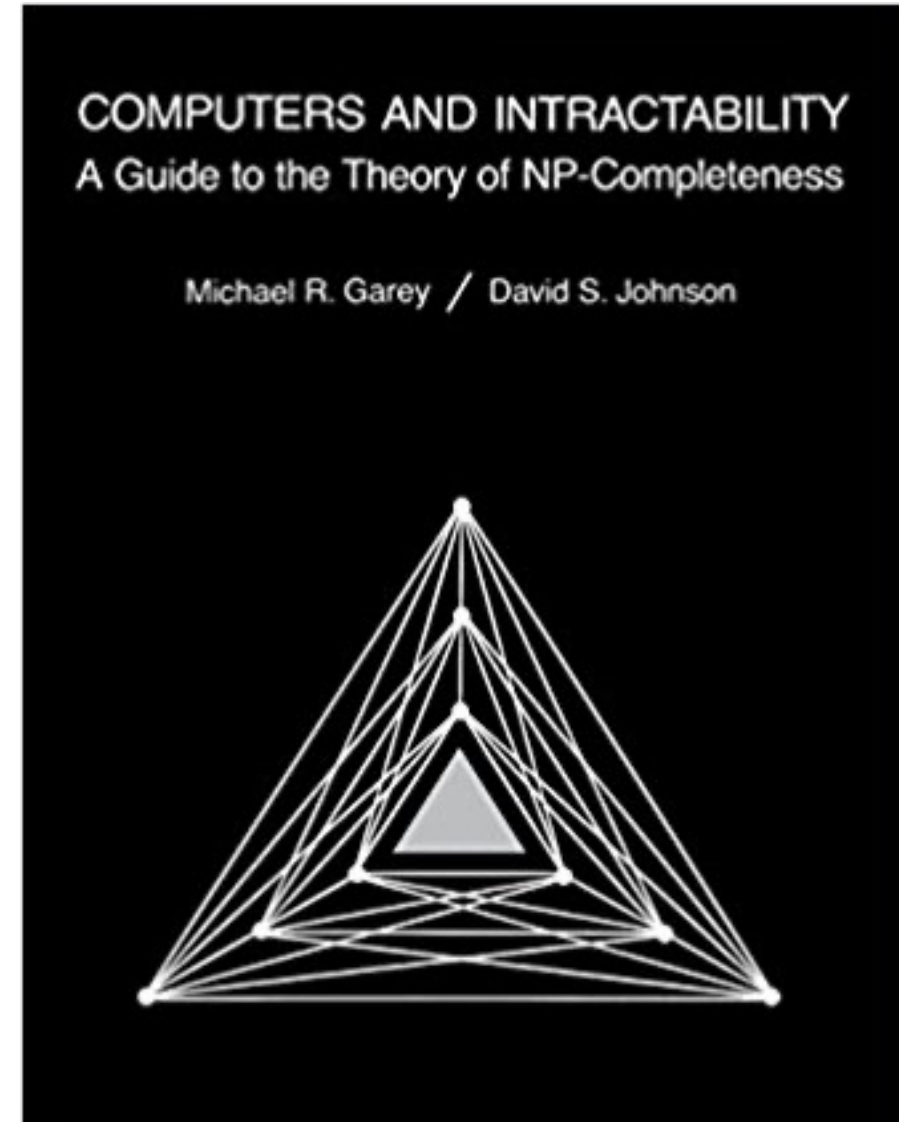
We just need to prove:

- $Q \in \text{NP}$
- $CS \leq_p Q$



# NP-Complete Problems

- ❑ By 1979, at least 300 problems had been proven NP-complete.
- ❑ **Garey** and **Johnson** put a list of all the NP-complete problems they could find at the time in this textbook.
- ❑ Took them almost 100 pages to just list them all.
- ❑ No one has made a comprehensive list since.



# 3-SAT Problem is NP-Complete



# 3-SAT (3-Conjunctive Normal Form or 3-CNF) Problem

## Satisfiability (SAT) and 3-SAT

**Input:**  $m$  Boolean clauses and  $n$  Boolean variable.  
Each clause is like  $x_1 \vee \overline{x_2} \vee x_4$  (with only  $\vee$  and **NOT** operators)  
**Output:** Can we find an assignment such that all Boolean clauses become true?

- $x_1 \vee \overline{x_2} \vee x_3$
- $\overline{x_1} \vee \overline{x_2} \vee x_3$
- $\overline{x_1} \vee x_2 \vee \overline{x_3}$
- $x_1 \vee x_2 \vee \overline{x_3}$
- $\overline{x_1} \vee \overline{x_2} \vee x_3$

- $x_1 = 1$
- $x_2 = 0$
- $x_3 = 0$

True, Satisfiable

- $\overline{x_1} \vee \overline{x_2}$
- $\overline{x_1} \vee x_2$
- $x_1 \vee x_2$
- $x_1 \vee \overline{x_2}$

Not Satisfiable

## 3-SAT Example

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

- $x_1 = 1$
- $x_2 = 0$
- $x_3 = 1$

True, Satisfiable

# 3-SAT is NP

Two steps:

- **3-SAT  $\in$  NP**
- $CS \leq_p 3\text{-SAT}$  (it means  $\forall Q \in NP$  we have  $Q \leq_p 3\text{-SAT}$ )

→ How to design a poly-time verifier for 3-SAT to prove  $3\text{-SAT} \in \mathbf{NP}$ ?

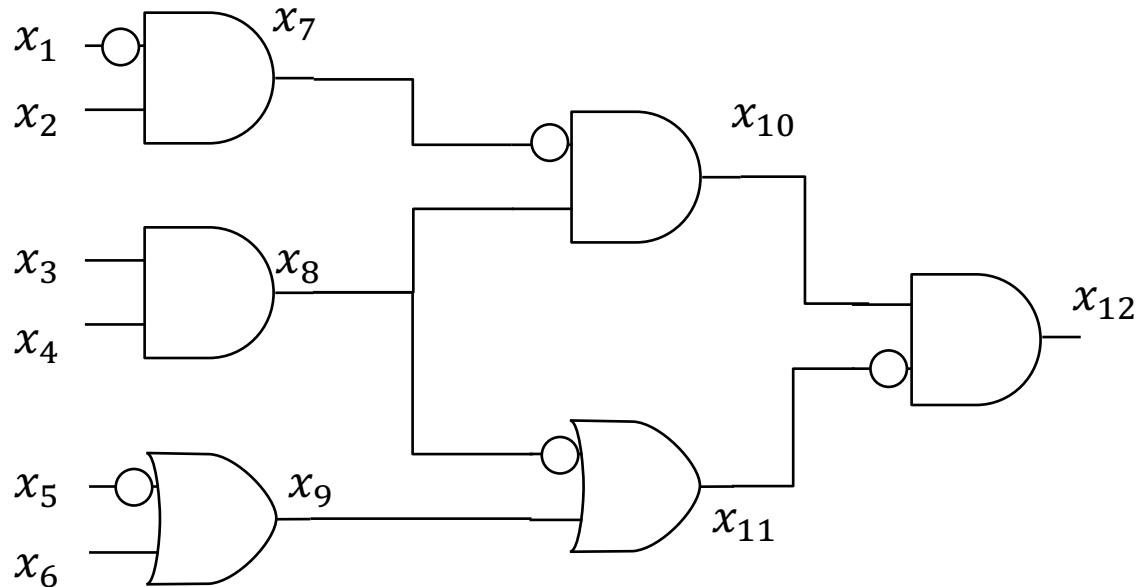
# 3-SAT is NP-Hard (1)

Two steps:

- $3\text{-SAT} \in \text{NP}$
- $CS \leq_p 3\text{-SAT}$  (it means  $\forall Q \in \text{NP}$  we have  $Q \leq_p 3\text{-SAT}$ )

For any Boolean circuit, one can:

1. Break each Boolean gates into 2-input gates.
2. Each of the intermediate results are stored in a variable.
3. Boolean equation equivalent of the circuit is written.
4. The formula is satisfiable when all of intermediate equations are satisfied. Hence, we can AND them together.



- $x_7 \leftrightarrow \overline{x_1} \wedge x_2$
- $x_8 \leftrightarrow x_3 \wedge x_4$
- $x_9 \leftrightarrow \overline{x_5} \vee x_6$
- $x_{10} \leftrightarrow \overline{x_7} \wedge x_8$
- $x_{11} \leftrightarrow \overline{x_8} \vee x_9$
- $x_{12} \leftrightarrow \overline{x_{11}} \wedge x_{10}$
- $x_{12}$



$$(x_7 \leftrightarrow \overline{x_1} \wedge x_2) \wedge (x_8 \leftrightarrow x_3 \wedge x_4) \wedge (x_9 \leftrightarrow \overline{x_5} \vee x_6) \wedge (x_{10} \leftrightarrow \overline{x_7} \wedge x_8) \wedge (x_{11} \leftrightarrow \overline{x_8} \vee x_9) \wedge (x_{12} \leftrightarrow \overline{x_{11}} \wedge x_{10}) \wedge x_{12}$$

# 3-SAT is NP-Hard (2)

Two steps:

- 3-SAT  $\in$  NP
- $CS \leq_p 3\text{-SAT}$  (it means  $\forall Q \in NP$  we have  $Q \leq_p 3\text{-SAT}$ )

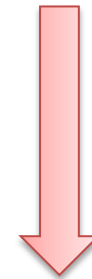
$x_1$	$x_2$	$x_3$	$\phi = x_3 \leftrightarrow x_1 \wedge \overline{x_2}$
1	1	1	0
1	0	1	1
0	1	1	0
0	0	1	0
1	1	0	1
1	0	0	0
0	1	0	1
0	0	0	1



$$\bar{\phi} = (x_1 \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge x_2 \wedge x_3) \vee (\overline{x_1} \wedge \overline{x_2} \wedge x_3) \vee (x_1 \wedge \overline{x_2} \wedge \overline{x_3})$$

DeMorgan's law:

- $\overline{(a \wedge b)} = \bar{a} \vee \bar{b}$
- $\overline{(a \vee b)} = \bar{a} \wedge \bar{b}$



$$\phi = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$$

# 3-SAT is NP-Hard (4)

Two steps:

- $3\text{-SAT} \in \text{NP}$
- $CS \leq_p 3\text{-SAT}$  (it means  $\forall Q \in \text{NP}$  we have  $Q \leq_p 3\text{-SAT}$ )

Last step: How to convert everything to clauses with 3 variables?

- If a clause has 2 literals, it can be converted to 3 literals as follows:
  - $(l_1 \vee l_2) \rightarrow (l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \bar{p})$
- If a clause has 1 literal, it can be converted to 3 literals as follows:
  - $l \rightarrow (l \vee p \vee q) \wedge (l \vee p \vee \bar{q}) \wedge (l \vee \bar{p} \vee q) \wedge (l \vee \bar{p} \vee \bar{q})$

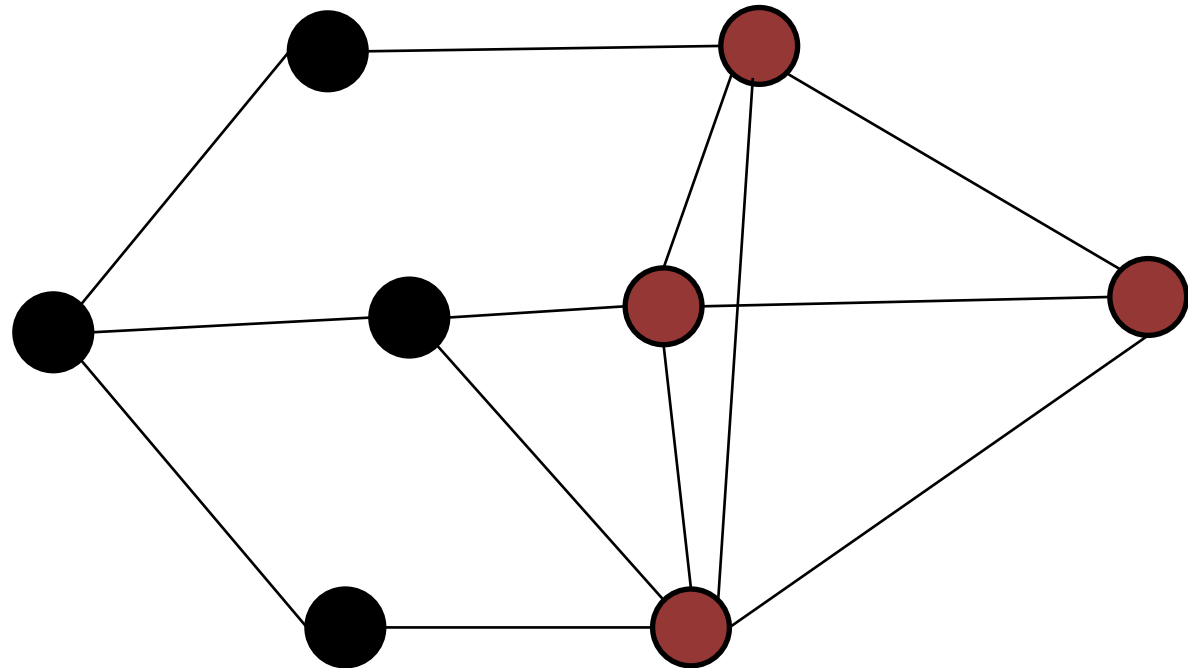
# CLIQUE Problem is NP-Complete

# CLIQUE problem

## CLIQUE

**Input:** Undirected graph  $G$  and a value of  $k$

**Output:** Can we find  $k$  vertices in graph  $G$  such that there are all adjacent to each other?



Has a clique of size 4 but not 5

# CLIQUE is NP

Two steps:

- **CLIQUE  $\in$  NP**
- $3\text{-SAT} \leq_p \text{CLIQUE}$  (it means  $\forall Q \in NP$  we have  $Q \leq_p \text{CLIQUE}$ )

→ How to design a verifier to prove **CLIQUE  $\in$  NP**?



# CLIQUE is NP-Hard

Two steps:

- $\text{CLIQUE} \in \text{NP}$
- $3\text{-SAT} \leq_p \text{CLIQUE}$  (it means  $\forall Q \in \text{NP}$  we have  $Q \leq_p \text{CLIQUE}$ )

□ Construct graph  $G = (V, E)$  as follows:

- Introduce a node for each *literal* in each clause.
- Put edge between each pair of nodes such that
  - Nodes are in different clauses.
  - Nodes are not each other's opposite.

# CLIQUE Problem Reduction Example

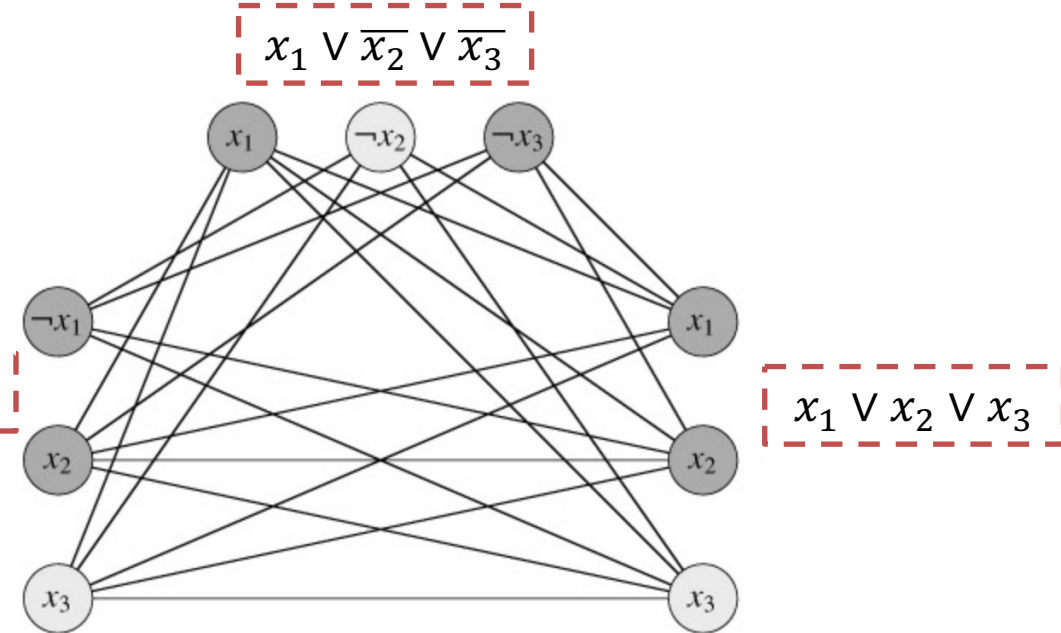
Two steps:

- $\text{CLIQUE} \in \text{NP}$
- $3\text{-SAT} \leq_p \text{CLIQUE}$  (it means  $\forall Q \in \text{NP}$  we have  $Q \leq_p \text{CLIQUE}$ )

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



$$\overline{x_1} \vee x_2 \vee x_3$$



Does it have a clique of size  $k$ ?

- Should select exactly one vertex from each clause.
- Set the value of selected vertex to 1 in 3-SAT instance

# Formula Is Satisfiable $\Leftrightarrow$ A Clique of Size $k$ Exists

## □ Formula is satisfiable $\Rightarrow$ A clique of size $k$ exists

- Assume the formula with  $k$  clauses is satisfiable.
- For each clause, select the **TRUE** node.
- Then these nodes must form a clique.

## □ A clique of size $k$ exists $\Rightarrow$ Formula is satisfiable

- Assume  $G$  has a clique of size at least  $k$ .
- Set variables such that these nodes evaluate to **TRUE**.
- Must be a consistent setting that makes formula satisfiable true.

## □ It suffices to show that CLIQUE problem is NP-hard in this special case. Why?

- If we had a polynomial-time algorithm that solved clique on general graphs, it would also solve CLIQUE on restricted graphs.

# SUBSET-SUM Problem is NP-Complete

# Subset-Sum problem

## SUBSET-SUM

**Input:** Set  $S = \{x_1, x_2, \dots, x_n\}$  with integer values and a value of  $t$

**Output:** Is there any subset of  $S$  such that sum of its elements is equal to  $t$ ?

- $S = \{1, 2, 5, 10, 11\}$  and  $t = 17$
- Answer is *yes* because of  $\{2, 5, 10\}$

- $S = \{1, 2, 5, 10, 11\}$  and  $t = 19$
- Answer is *yes* because of  $\{1, 2, 5, 11\}$

- $S = \{1, 2, 5, 10, 11\}$  and  $t = 20$
- Answer is *no*

# Subset-Sum is NP

Two steps:

- **SUBSET-SUM  $\in$  NP**
- $3\text{-SAT} \leq_p \text{SUBSET-SUM}$  (it means  $\forall Q \in NP$  we have  $Q \leq_p \text{SUBSET-SUM}$ )

→ How to design a verifier to prove SUBSET-SUM  $\in$  NP?

# SUBSET-SUM is NP-Hard (1)

Two steps:

- SUBSET-SUM  $\in$  NP
- 3-SAT  $\leq_p$  SUBSET-SUM (it means  $\forall Q \in NP$  we have  $Q \leq_p$  SUBSET-SUM)

□  $n$  variables  $x_i$  and  $m$  clauses  $C_j$

□ For each variable  $x_i$ , construct numbers  $v_i$  and  $v'_i$  of  $n + m$  digits:

- The  $i$ -th digit of  $v_i$  and  $v'_i$  is equal to 1.
- For  $n + 1 \leq j \leq n + m$ , the  $j$ -th digit of  $v_i$  is equal to 1 if  $x_i$  is in clause  $C_{j-n}$
- For  $n + 1 \leq j \leq n + m$ , the  $j$ -th digit of  $v'_i$  is equal to 1 if  $\bar{x}_i$  is in clause  $C_{j-n}$

□ All other digits of  $v_i$  and  $v'_i$  are 0.

□ Example:

- $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$

	$x_1$	$x_2$	$x_3$	$C_1$	$C_2$	$C_3$	$C_4$
$v_1$	1	0	0	1	0	0	1
$v'_1$	1	0	0	0	1	1	0
$v_2$	0	1	0	0	0	0	1
$v'_2$	0	1	0	1	1	1	0
$v_3$	0	0	1	0	0	1	1
$v'_3$	0	0	1	1	1	0	0

# Subset-Sum is NP-Hard (2)

Two steps:

- SUBSET-SUM  $\in$  NP
- $3\text{-SAT} \leq_p \text{SUBSET-SUM}$  (it means  $\forall Q \in \text{NP}$  we have  $Q \leq_p \text{SUBSET-SUM}$ )

□ For each clause  $C_j$ , construct *slack variables*  $s_j$  and  $s'_j$  of  $n + m$  digits:

- The  $(n + j)$ -th digit of  $s_j$  is equal to 1.
- The  $(n + j)$ -th digit of  $s'_j$  is equal to 2.
- All other digits of  $s_j$  and  $s'_j$  are 0.

□ Finally, construct a sum number  $t$  of  $n + m$  digits:

- For  $1 \leq j \leq n$ , the  $j$ -th digit of  $t$  is equal to 1.
- For  $n + 1 \leq j \leq n + m$ , the  $j$ -th digit of  $t$  is equal to 4.

□ Example:

- $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$

	$x_1$	$x_2$	$x_3$	$C_1$	$C_2$	$C_3$	$C_4$
$s_1$	0	0	0	1	0	0	0
$s'_1$	0	0	0	2	0	0	0
$s_2$	0	0	0	0	1	0	0
$s'_2$	0	0	0	0	2	0	0
$s_3$	0	0	0	0	0	1	0
$s'_3$	0	0	0	0	0	2	0
$s_4$	0	0	0	0	0	0	1
$s'_4$	0	0	0	0	0	0	2
$t$	1	1	1	4	4	4	4



# Subset-Sum is NP-Hard (3)

- $C_1 = x_1 \vee \overline{x_2} \vee \overline{x_3}$
- $C_2 = \overline{x_1} \vee \overline{x_2} \vee \overline{x_3}$
- $C_3 = \overline{x_1} \vee \overline{x_2} \vee x_3$
- $C_4 = x_1 \vee x_2 \vee x_3$

Two numbers for each variable

Two numbers for each clause

Subset =  $\{v'_1, v'_2, v_3, s_1, s'_1, s'_2, s_3, s_4, s'_4\}$   
 $= \{1000110, 101110, 10011, 1000, 2000, 200, 10, 1, 2\}$   
 $t = 1114444$

	$x_1$	$x_2$	$x_3$	$C_1$	$C_2$	$C_3$	$C_4$
$v_1$	1	0	0	1	0	0	1
$v'_1$	1	0	0	0	1	1	0
$v_2$	0	1	0	0	0	0	1
$v'_2$	0	1	0	1	1	1	0
$v_3$	0	0	1	0	0	1	1
$v'_3$	0	0	1	1	1	0	0
$s_1$	0	0	0	1	0	0	0
$s'_1$	0	0	0	2	0	0	0
$s_2$	0	0	0	0	1	0	0
$s'_2$	0	0	0	0	2	0	0
$s_3$	0	0	0	0	0	1	0
$s'_3$	0	0	0	0	0	2	0
$s_4$	0	0	0	0	0	0	1
$s'_4$	0	0	0	0	0	0	2
<b>t</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>4</b>

# Formula Satisfiable $\Rightarrow$ Subset Exists

- Take  $v_i$  if  $x_i$  is **true**.
- Take  $v_i'$  if  $x_i$  is **false**.
- Take both  $s_j$  and  $s_j'$  if number of true literals in  $C_j$  is **1**.
- Take  $s_j'$  if number of true literals in  $C_j$  is **2**.
- Take  $s_j$  if number of true literals in  $C_j$  is **3**.
- Example:
  - $(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3)$
  - $x_1 = x_2 = x_3 = \text{TRUE}$
  - Subset =  $\{v_1, v_2, v_3, s_1, s_2, s_2', s_3, s_3', s_4'\}$

	$x_1$	$x_2$	$x_3$	$C_1$	$C_2$	$C_3$	$C_4$
$v_1$	1	0	0	1	0	0	1
$v_2$	0	1	0	1	0	1	0
$v_3$	0	0	1	1	1	0	1
$s_1$	0	0	0	1	0	0	0
$s_2$	0	0	0	0	1	0	0
$s_2'$	0	0	0	0	2	0	0
$s_3$	0	0	0	0	0	1	0
$s_3'$	0	0	0	0	0	2	0
$s_4'$	0	0	0	0	0	0	2
t	1	1	1	4	4	4	4

# Subset Exists $\Rightarrow$ Formula Satisfiable

- ❑ Assign value **TRUE** to  $x_i$  if  $v_i$  is in subset.
- ❑ Assign value **FALSE** to  $x_i$  if  $v'_i$  is in subset.
- ❑ Exactly one number per variable must be in the subset .
  - Otherwise one of first  $n$  digits of the sum is not equal to **1**.
- ❑ At least one variable number corresponding to a literal in a clause must be in the subset.
  - Otherwise one of next  $m$  digits of the sum is smaller than **4**.
- ❑ Each clause is satisfied.

# An Undecidable Problem: HALTING Problem

## HALTING Problem

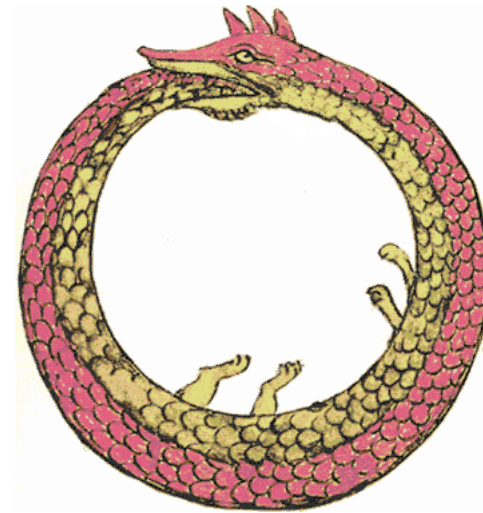
**Input:** Program  $P$  and input  $I$

**Output:** Returns **yes** if program  $P$  halts on input  $I$  and **no** otherwise

Assume program  $H(P, I)$  decides the **HALTING Problem**.

```
G(x) {  
    if H(x,x) = yes  
        Loop forever;  
    else  
        Halt;  
}
```

**Contradiction:**  
What is the result of  $G(G)$ ?



**Ouroboros:** a dragon that continually consumes itself

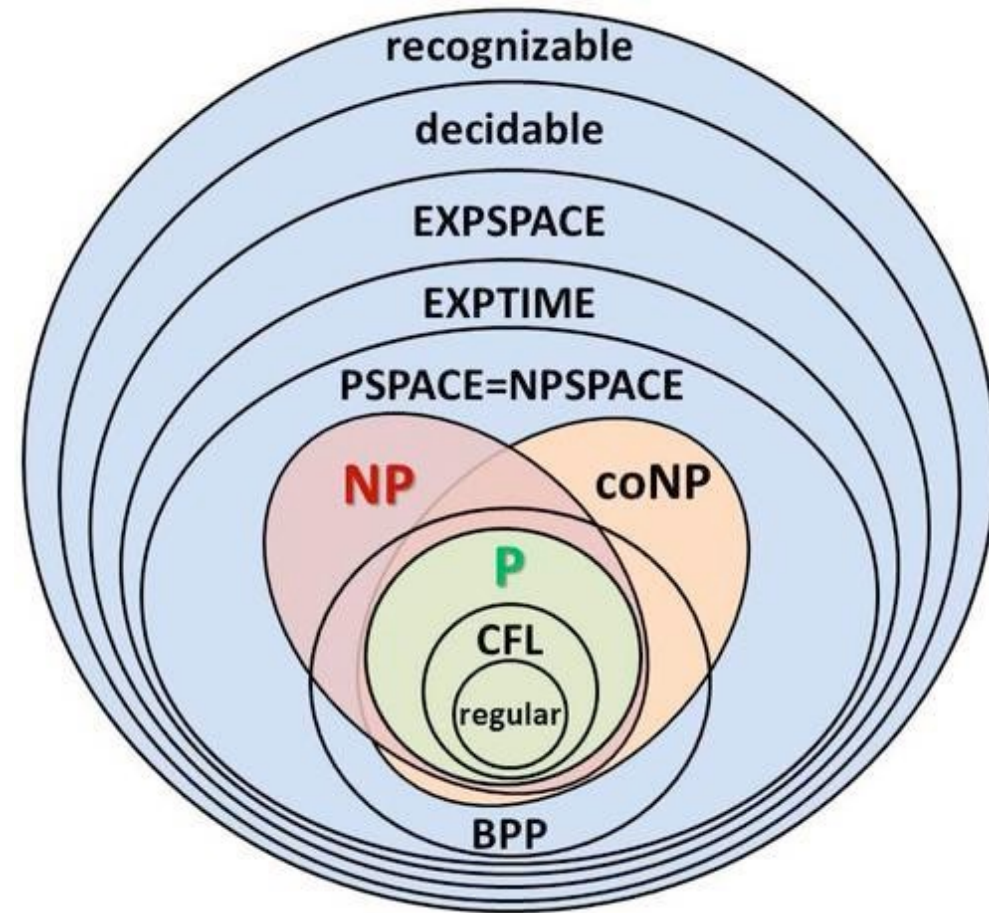
# Why did we study about complexity classes?

## ❑ As a scientist:

- You need to understand complexity classes.
- If you establish a problem as **NP-complete**, it's a good evidence for its intractability.
- There are MANY MANY more classes we didn't discuss in the class.
  - In the CS theory field, many researchers are actively working on this subject.

## ❑ As an engineer:

- Find an approximate algorithm instead of trying to solve the problem exactly.
- Solve a tractable special case.



# Sample Problems

# True or False?

1. **NP** is the class of problems that are verifiable in polynomial time.
2. It is not known whether  $\mathbf{P} \neq \mathbf{NP}$  or  $\mathbf{P} = \mathbf{NP}$ .
3. If a problem is not in **P**, it should be in **NP-complete**.
4. If a problem is in **NP**, it must also be in **P**.
5. If a problem is **NP-complete**, it must not be in **P**.
6. **NP-complete** problems cannot be decided efficiently.
7. **NP-complete** problems are the hardest decision problems.

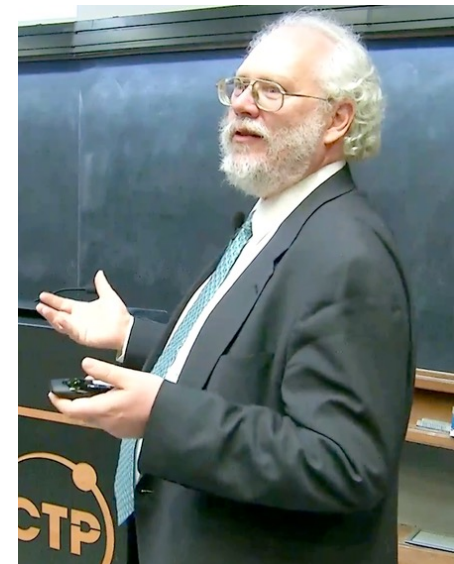
# True or False? (cont'd)

8. Assume  $\mathbf{P} \neq \mathbf{NP}$ . Let  $A$  and  $B$  be decision problems. If  $A$  is in **NP-complete** and  $A \leq_p B$ , then  $B$  is not in  $\mathbf{P}$ .
9. There exists a decision problem  $X$  such that for all  $Y$  in **NP**,  $Y$  is polynomial-time reducible to  $X$ .
10. If  $\mathbf{P} = \mathbf{NP}$ , then  $\mathbf{NP} = \mathbf{NP-complete}$ .
11. If a problem is not in  $\mathbf{P}$ , then it must be in **NP**.
12. **NP** is the class of problems that are not decidable in polynomial time.



# Integer Factorization Problem

- ❑ *Integer factorization* is the **decomposition of a composite number into a product of smaller integers greater than 1**.
  - If these factors are further restricted to prime numbers, the process is called *prime factorization*.
- ❑ No efficient (*non-quantum*) integer factorization algorithm is known.
  - However, it has not been proven that no efficient algorithm exists.
  - The presumed difficulty of this problem is at the heart of widely used algorithms in cryptography such as RSA.
    - Take a course on computer security or cryptography to learn more about it.
  - *Peter Shor* came up with an algorithm in 1994 which could factorize integers in polynomial-time on **quantum computers**.
    - Take a course on quantum computing/information processing to learn more about it.



Peter Shor

# Polynomial-Time Solution for Integer Factorization!

- ❑ We have learned that no algorithm has been published that can factor any integer in polynomial time.
- ❑ I claim that I can come up with a polynomial-time algorithm though!

```
factorize(n) {  
    for i = 2 to n - 1 {  
        if n % i == 0 {  
            return i, n / i  
        }  
    }  
    return n + " is prime."  
}
```

Prove or disprove whether this algorithm factorizes  $n$  in polynomial time.

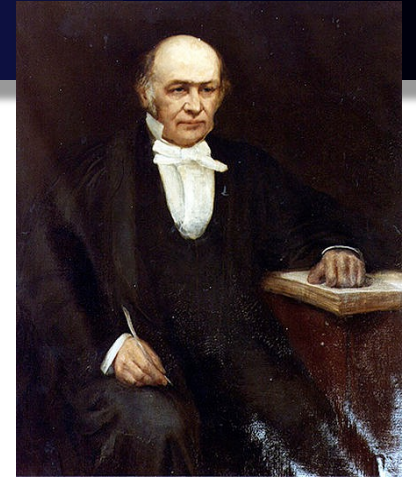
# Traveling Salesman Problem (TSP)

□ *Hamiltonian cycle* is a cycle which passes through all the vertices of the graph exactly once.

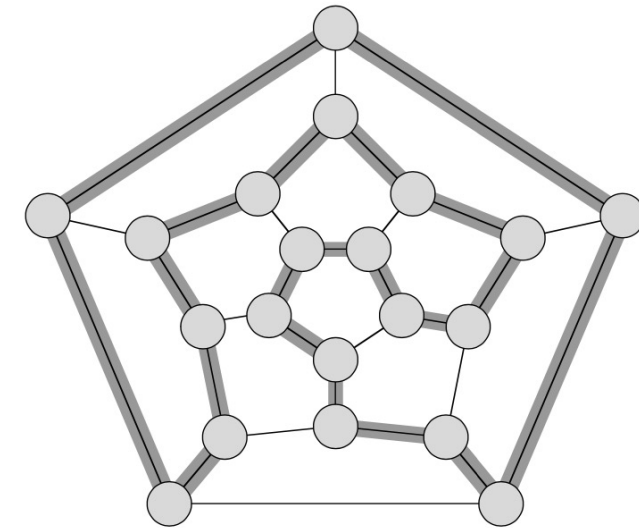
- Assume that deciding whether a graph has a Hamiltonian cycle (HAM-CYCLE) is **NP-complete**.
- See the NP-completeness proof in CLRS 34.5.3.

□ **Traveling salesman problem (TSP):**

- Given a weighted complete graph  $G$  with non-negative edges and integer  $k$ , decide whether the graph  $G$  contains a *tour* (or *Hamiltonian cycle*) of cost  $k$  or smaller.
- Prove that TSP is NP-complete.
- **Hint:** Show  $\text{HAM-CYCLE} \leq_p \text{TSP}$



William Rowan Hamilton

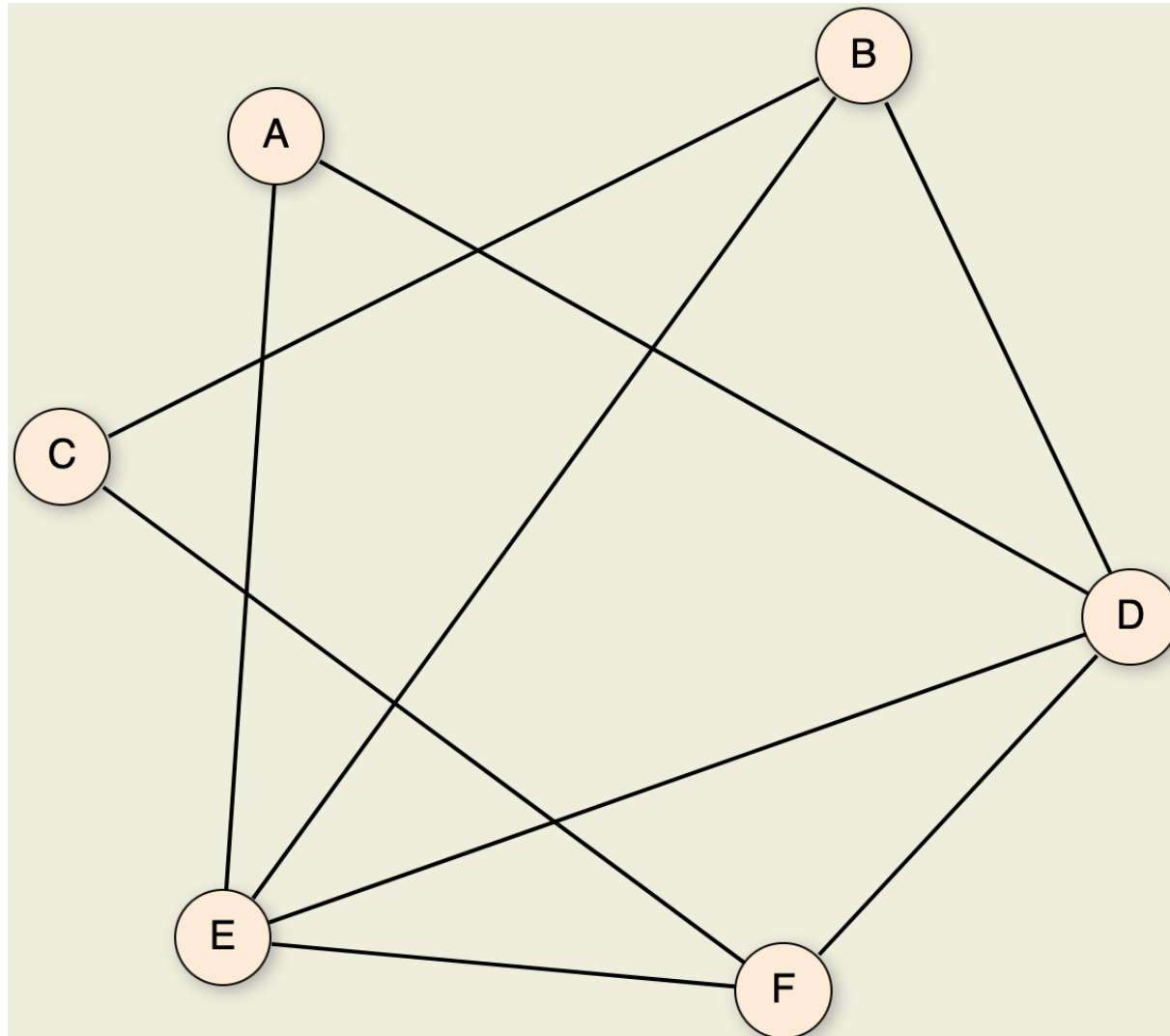


Hamiltonian cycle

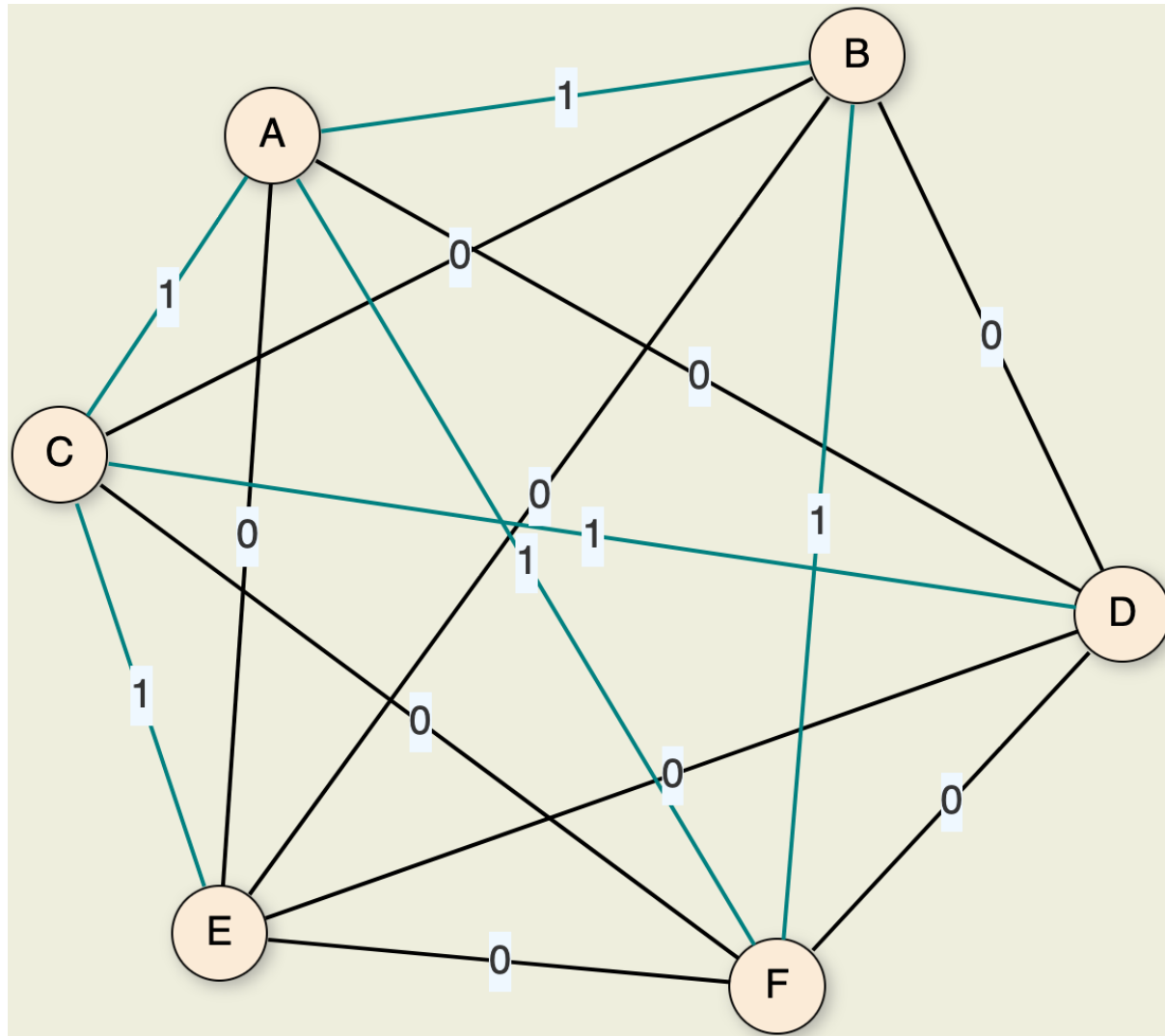
# TSP is NP

□ How to design a verifier to prove that  $\text{TSP} \in \mathbf{NP}$ ?

# Graph $G$



# Graph $G'$



# TSP is NP-Hard (cont'd)

- $G$  has a Hamiltonian cycle if and only if  $G'$  has a tour of cost at most 0.
  - If  $G$  has a Hamiltonian cycle,  $G'$  has a tour of cost at most 0.
  - If  $G'$  has a tour of cost at most 0,  $G$  has a Hamiltonian cycle.

Quiz Time!



# Quiz Rules

- ❑ Similar to those of previous exams' and quizzes' rules.
- ❑ The quiz is closed book, internet, friend, etc.
- ❑ Don't forget to write your full name and student ID on top of your answer sheet.
- ❑ The exam time is 15 min. There is *NO* grace period for late submissions.

□ **مسئله ۱:** گراف وزن دار  $G$  و راس  $r$  به عنوان ریشه به شما داده شده است. یک درخت پوشا در  $G$  که فاصله همه یال‌های آن از ریشه حداکثر ۲ یال باشد، **درخت پوشای دو گام** نامیده می‌شود. در **مسئله ی درخت پوشای دو گام**، می‌خواهیم بدانیم آیا درخت پوشای دو گامی در  $G$  می‌توان یافت که مجموع وزن یال‌های آن حداکثر  $k$  شود؟

□ **مسئله ۲:** برای افزایش کیفیت اینترنت خوابگاه، دانشگاه می‌خواهد در هر اتاق یک روتر نصب کرده یا اتاق را با کابل شبکه به اتاق دیگری که در آن روتر نصب شده متصل کند (اتصال کابل بین دو اتاق بدون روتر بی‌فایده است). هزینه نصب روتر در اتاق  $i$  برابر  $r_i$  و هزینه نصب کابل از اتاق  $i$  به اتاق  $j$  برابر  $c_{ij} = c_{ji}$  است. مسئول خوابگاه می‌خواهد بداند آیا می‌توان با (حداکثر) بودجه  $k$ ، اینترنت با کیفیت را به همه ی اتاق‌ها رساند؟

□ اگر بدانیم مسئله ۱، NP-hard است، ثابت کنید مسئله ۲ نیز NP-hard است.

# Recommended Website

- ❑ See Chapter 28 slides of this website for nice proves of different NP-complete problems:
  - <https://opensa-server.cs.vt.edu/ODSA/Books/Everything/html/index.html>
    - For instance, circuit satisfiability problem is detailed here:
      - <https://opensa-server.cs.vt.edu/ODSA/Books/Everything/html/circuitSAT.html>