طراحی الگوریتم ها

ملکی مجد

مباحث

NPC

مبحث NPC از فصل 34 کتاب CLRS تدریس می شود.

- Almost all the algorithms we have studied thus far have been polynomial-time algorithms:

  on inputs of size n, their worst-case running time is O(n^k) for some constant k.

can all problems be solved in polynomial time??

Generally, we think of problems that are solvable by polynomial-time algorithms as being **tractable**, or **easy**, and problems that require superpolynomial time as being **intractable**, or **hard.**

The status of "NP-complete" problems is unknown

This so-called P $\neq$ NP question has been one of the deepest, most perplexing open research problems in theoretical computer science since it was first posed in 1971.

Solvable in polynomial time VS NP-complete

- Shortest vs. longest simple paths
- Euler tour vs. hamiltonian cycle
- 2-CNF satisfiability vs. 3-CNF satisfiability

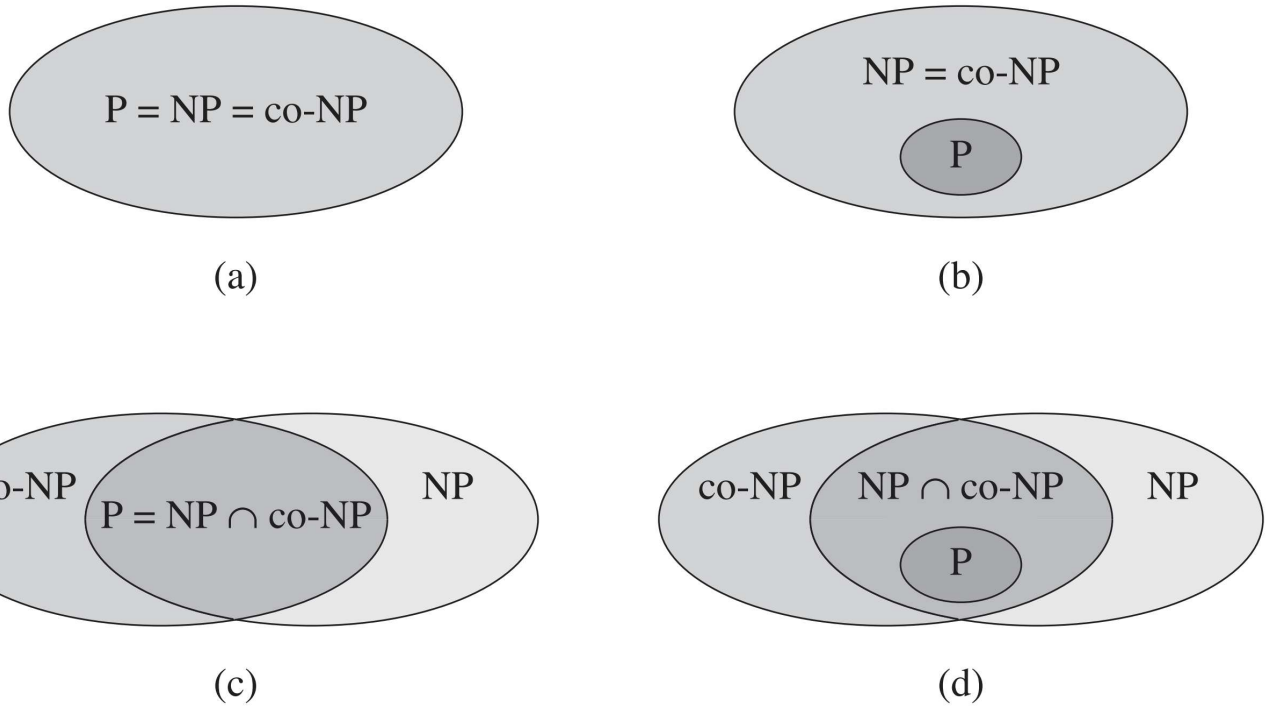
2-CNF satisfiability vs. 3-CNF satisfiability: A boolean formula contains variables whose values are 0 or 1; boolean connectives such as $\wedge$ (AND), $\vee$ (OR), and $\neg$ (NOT); and parentheses. A boolean formula is ***satisfiable*** if there exists some assignment of the values 0 and 1 to its variables that causes it to evaluate to 1. We shall define terms more formally later in this chapter, but informally, a boolean formula is in ***k-conjunctive normal form***, or k-CNF, if it is the AND of clauses of ORs of exactly k variables or their negations. For example, the boolean formula $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$ is in 2-CNF. (It has the satisfying assignment $x_1 = 1, x_2 = 0, x_3 = 1$.) Although we can determine in polynomial time whether a 2-CNF formula is satisfiable, we shall see later in this chapter that determining whether a 3-CNF formula is satisfiable is NP-complete.

Informal classification

- Class P
  - consists of those problems that are solvable in polynomial time. Class NP
- Class NP
  - consists of those problems that are “verifiable” in polynomial time.
  - If we were somehow given a “certificate” of a solution, then we could verify that the certificate is correct in time polynomial in the size of the input to the problem.
  - As an example, for 3-CNF satisfiability, a certificate would be an assignment of values to variables. We could check in polynomial time that this assignment satisfies the boolean formula.

  we believe P⊆ NP. The open question is whether or not P is a proper subset of NP!
- Class NPC
  - A problem is in the class NPC if it is in NP and is as “hard” as any problem in NP

Four possibilities for relationships among complexity classes



- To become a good algorithm designer:
  - must understand the basics of the theory of NP-completeness.

- If establish a problem as NP-complete
  - provide good evidence for its intractability.
  - spend your time developing an approximation algorithm
  - or solving a tractable special case, rather than searching for a fast algorithm that solves the problem exactly.

- Many natural and interesting problems that on the surface seem no harder than sorting, graph searching, or network flow are in fact NP-complete.

Stuff we need

When we demonstrate that a problem is NP-complete, we are making a statement about how hard it is rather than about how easy it is.

- Decision problems vs. optimization problems
- Reduction

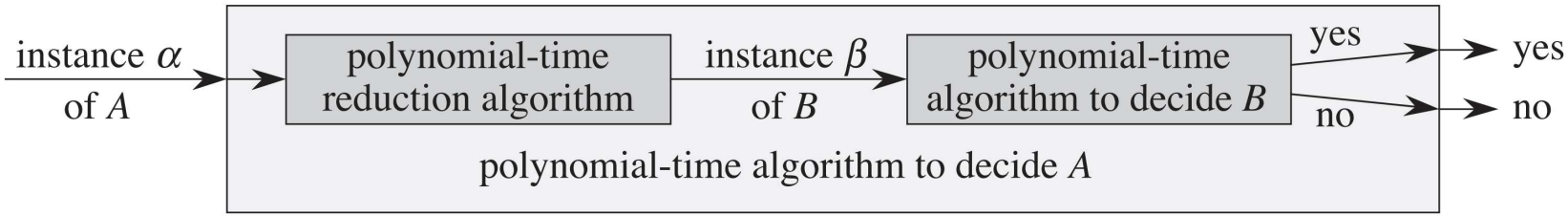
Decision problems vs. optimization problems

- Many problems of interest are optimization problems, in which each feasible (i.e., "legal") solution has an associated value, and we wish to find a feasible solution with the best value.
  - E.g., shortest path
- Answer of decision problems is simply "yes" or "no" (or, more formally, "1" or "0").
  - E.g., a path consisting at most k edges

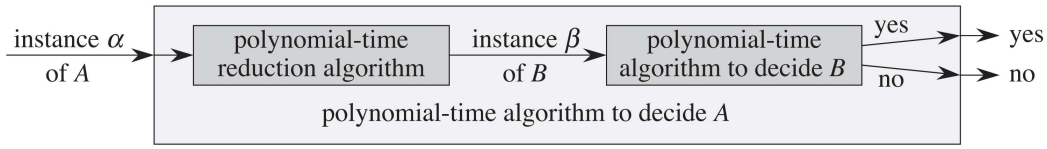
NP-complete problems are confined to the realm of **decision** problems, however, we can take advantage of the **relationship** between optimization problems and decision problems

Reduction algorithm

- By “reducing” solving problem A to solving problem B, we use the “easiness” of B to prove the “easiness” of A.



Reduction algorithm



- we use polynomial-time reductions in the opposite way to show that a problem is NP-complete.
  - Suppose we have a decision problem A for which we already know that no polynomial-time algorithm can exist.
  - Suppose further that we have a polynomial-time reduction transforming instances of A to instances of B.
  - we can use a simple proof by contradiction to show that no polynomial time algorithm can exist for B.

For NP-completeness, we cannot assume that there is absolutely no polynomialtime algorithm for problem A. We prove that problem B is NP-complete on the assumption that problem A is also NP-complete.

Prove P

- A problem Q can be reduced to another problem Q0 if any instance of Q can be “easily rephrased” as an instance of Q0 , the solution to which provides a solution to the instance of Q.

- Polynomial-time reductions give us a powerful tool for proving that various languages belong to P.

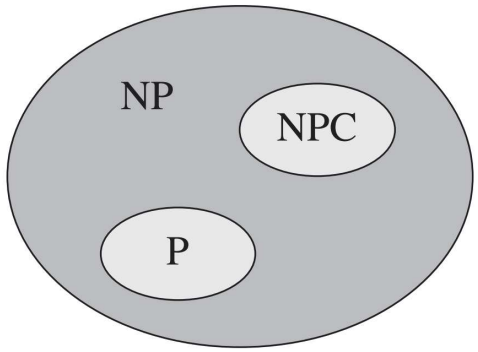
NP-complete
formal definition

A language $L \subseteq \{0,1\}^*$ is ***NP-complete*** if

1. $L \in \text{NP}$, and
2. $L' \leq_{\text{P}} L$ for every $L' \in \text{NP}$.

If a language L satisfies property 2, but not necessarily property 1, we say that L is ***NP-hard***. We also define NPC to be the class of NP-complete languages.

How most theoretical computer scientists view the relationships among P, NP, and NPC



First NP-complete Problem

The circuit-satisfiability problem is, “Given a boolean combinational circuit composed of AND, OR, and NOT gates, is it satisfiable?”

The circuit-satisfiability problem belongs to the class NP. (Proof not cover)

The circuit-satisfiability problem is NP-hard. (Proof not cover- directly reduce any NP problem to it)

The circuit-satisfiability problem is NP-complete.

This problem has the historical honor of being the first problem ever shown to be NP-complete.

- We prove that a problem is NP-complete without directly reducing every problem in NP to the given problem.

Lemma 34.8

If L is a language such that $L' \leq_P L$ for some $L' \in$ NPC, then L is NP-hard. If, in addition, $L \in$ NP, then $L \in$ NPC.

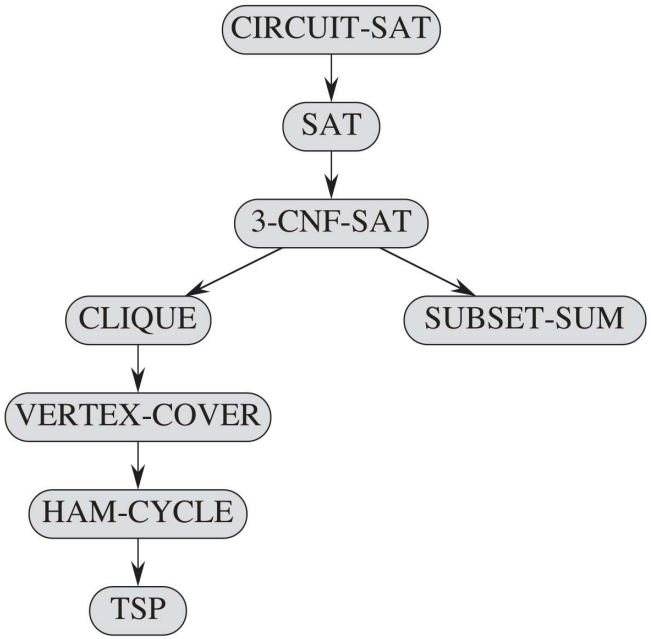
Proof Since L' is NP-complete, for all $L'' \in$ NP, we have $L'' \leq_P L'$. By supposition, $L' \leq_P L$, and thus by transitivity (Exercise 34.3-2), we have $L'' \leq_P L$, which shows that L is NP-hard. If $L \in$ NP, we also have $L \in$ NPC. ■

NPC reductions

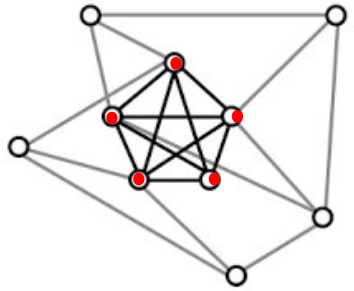
Lemma. If L is language s.t. $L' \leq_p L$ where $L' \in$ NPC, then L is NP-hard. If $L \in$ NP, then $L \in$ NPC.

This gives us a recipe for proving any $L \in$ NPC:

1. Prove $L \in$ NP
2. Select $L' \in$ NPC
3. Describe algorithm to compute f mapping every input x of L' to input $f(x)$ of L
4. Prove f satisfies $x \in L'$ iff $f(x) \in L$, for all $x \in \{0, 1\}^*$
5. Prove computing f takes p-time


CIRCUIT-SAT
SAT
3-CNF-SAT
CLIQUE
SUBSET-SUM
VERTEX-COVER
HAM-CYCLE
TSP

clique



A ***clique*** in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E. In other words, a clique is a complete subgraph of G. The ***size*** of a clique is the number of vertices it contains.

clique problem is the optimization problem of finding a clique of maximum size in a graph.

As a decision problem, we ask simply whether a clique of a given size k exists in the graph.

CLIQUE $\in$ NPC

CLIQUE = $\{\langle G, k\rangle$: graph $G = (V, E)$ has clique of size $k\}$

Proof. Show 3-CNF-SAT $\leq_p$ CLIQUE

Recall recipe for NPC proofs

1. Prove $L \in$ NP
2. Select $L' \in$ NPC
3. Describe algorithm to compute f mapping every input x of L' to input $f(x)$ of L
4. Prove f satisfies $x \in L'$ iff $f(x) \in L$, for all $x \in \{0, 1\}^*$
5. Prove computing f takes p-time

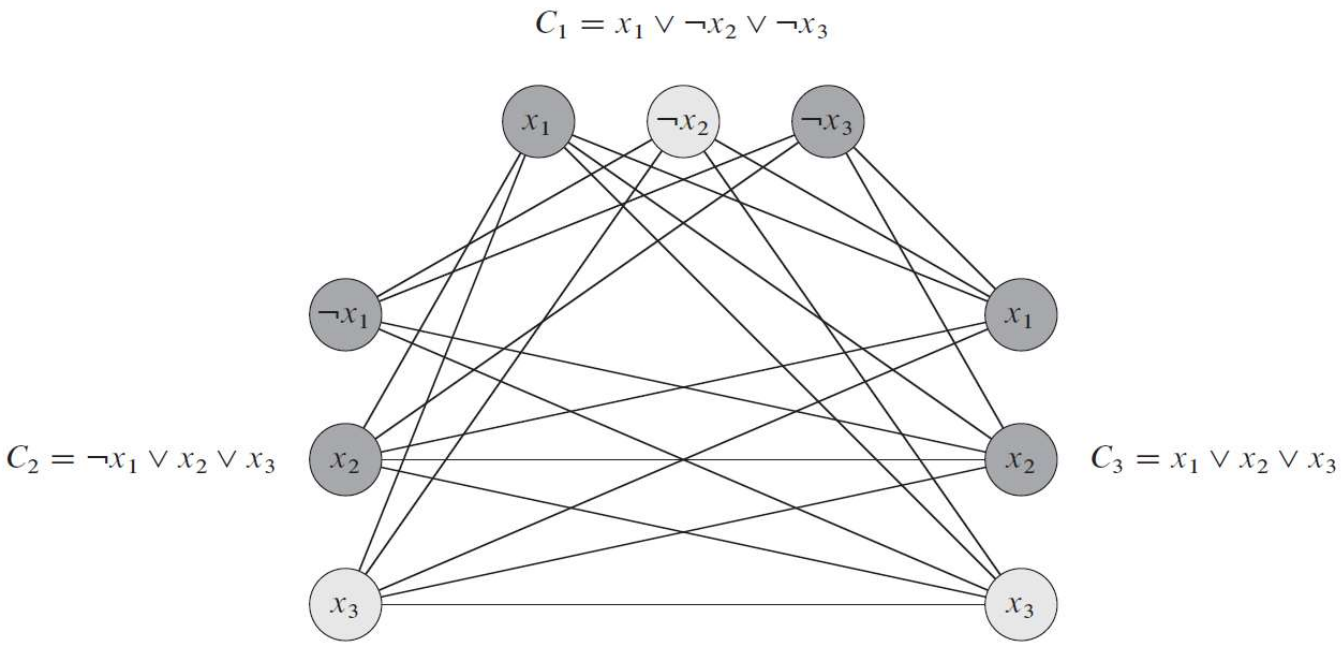
Given formula $\phi = c_1 \wedge c_2 \wedge \ldots \wedge c_k$, construct input of CLIQUE:

For each $c_r = (l_1^r \vee l_2^r \vee l_3^r)$, place v_1^r, v_2^r, v_3^r in V

Add edge between v_i^r and v_j^s if r $\neq$ s and corresponding literals are consistent

example

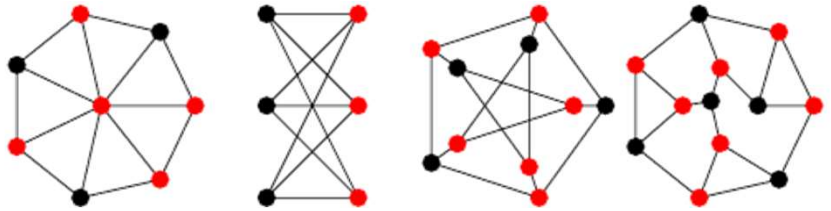
$$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$



If ϕ is satisfiable, at least one literal in each c_r is 1 $\Rightarrow$ set of k vertices that are completely connected

If G has clique of size k, contains exactly one vertex per clause $\Rightarrow$ ϕ satisfied by assigning 1 to corresponding literals

Vertex cover



a ***vertex cover*** of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if (u, v) is an edge of G, then either $u \in V'$ or $v \in V'$ (or both). The size of a vertex cover is the number of vertices in it.

The ***vertex-cover problem*** is to find a vertex cover of minimum size in a given undirected graph.

Restating this optimization problem as a decision problem, we wish to determine whether a graph has a vertex cover of a given size k.

VERTEX-COVER $\in$ NPC

VERTEX-COVER = $\{\langle G, k\rangle$: graph $G = (V, E)$ has vertex cover of size $k\}$

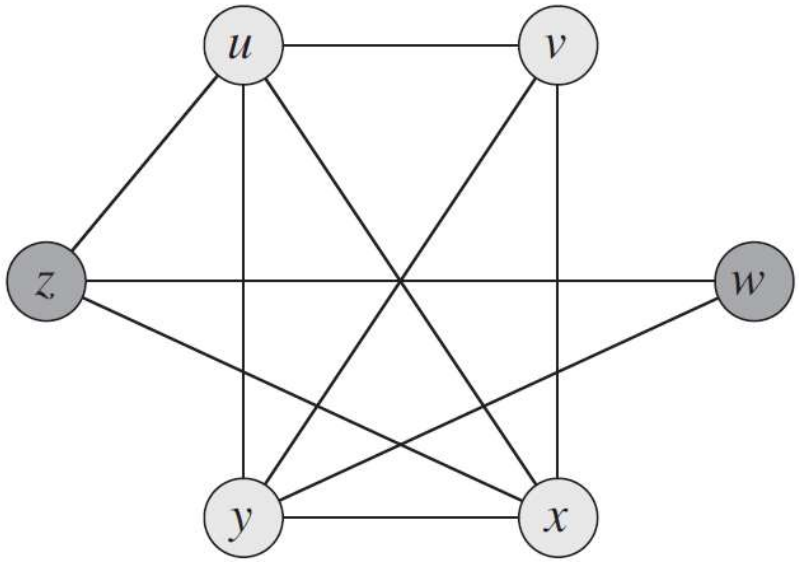
Vertex cover is $V' \subseteq V$ s.t. if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ or both

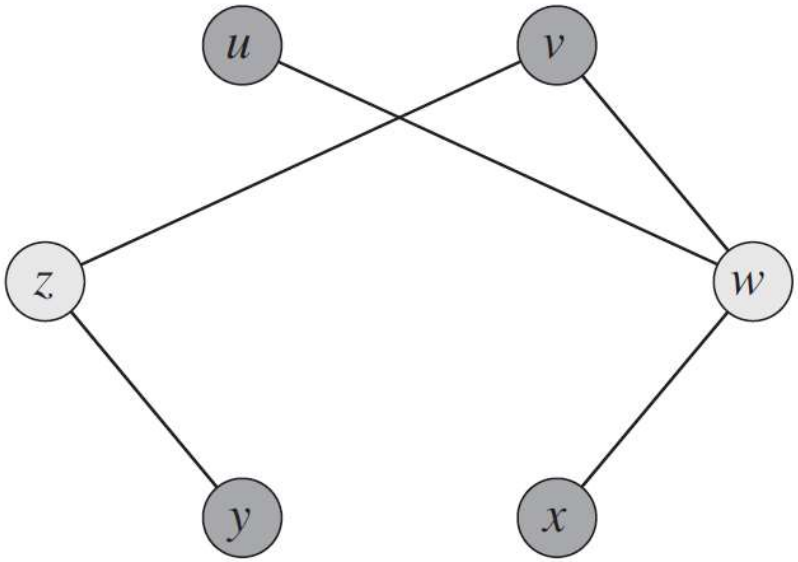
Proof. Show CLIQUE $\leq_p$ VERTEX-COVER

+ Verification algorithm can be done in polynomial time!

Given input $\langle G, k \rangle$ of CLIQUE, construct input of VERTEX-COVER:

$\langle \bar{G}, |V| - k \rangle$, where $\bar{G} = (V, \bar{E})$





Given input $\langle G, k \rangle$ of CLIQUE, construct input of VERTEX-COVER:

$\langle \bar{G}, |V| - k \rangle$, where $\bar{G} = (V, \bar{E})$

If G has clique V', $|V'| = k$, then $V - V'$ is vertex cover of $\bar{G}$:

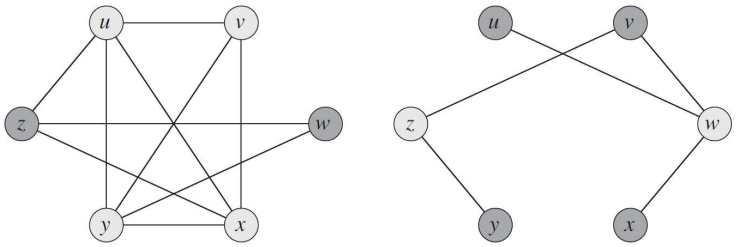
$(u, v) \in \bar{E} \Rightarrow$ either u or v not in V', since $(u, v) \notin E$

$\Rightarrow$ at least one of u or v in $V - V'$, so covered

If $\bar{G}$ has vertex cover $V' \subseteq V$, $|V'| = |V| - k$, then $V - V'$ is clique of G of size k

$(u, v) \in \bar{E} \Rightarrow u \in V'$ or $v \in V'$ or both

if $u \notin V'$ and $v \notin V'$, then $(u, v) \in E$



An undirected graph $G = (V, E)$ with clique $V' = \{u, v, x, y\}$.
The graph $\overline{G}$ produced by the reduction algorithm that has vertex cover $V - V' = \{w, z\}$.

The subset-sum problem

we are given a finite set S of positive integers and an integer ***target*** $t > 0$.
whether there exists a subset $S' \subseteq S$ whose elements sum to t.

For example,
if $S = \{1, 2, 7, 14, 49, 98, 343, 686, 2409, 2793, 16808, 17206, 117705, 117993\}$
and $t = 138457$,
then the subset $S' = \{1, 2, 7, 98, 343, 686, 2409, 17206, 117705\}$ is a solution.

SUBSET-SUM $\in$ NPC

SUBSET-SUM = $\{\langle S, t\rangle : S \subset \mathbf{N}, t \in \mathbf{N}$ and $\exists$ a subset $S' \subseteq S$ s.t. $t = \Sigma_{s \in S'} s\}$

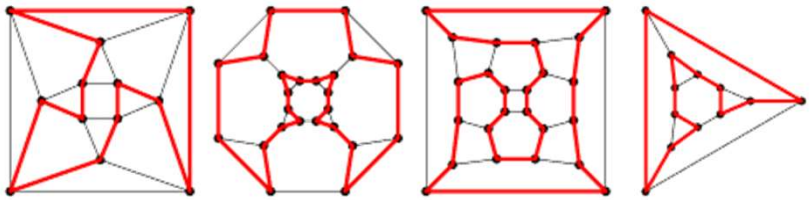
Integers encoded in binary! If t encoded in unary, can solve SUBSET-SUM in p-time, i.e. **weakly NPC** (vs. **strongly NPC**)

Proof. Show 3-CNF-SAT $\leq_p$ SUBSET-SUM

Hamiltonian cycle problem

A ***Hamiltonian cycle*** of an undirected graph $G(V, E)$ is a simple cycle that contains each vertex in V.

A graph that contains a Hamiltonian cycle is said to be ***Hamiltonian***; otherwise, it is ***non-Hamiltonian***.



Theorem 34.13

The hamiltonian cycle problem is NP-complete.

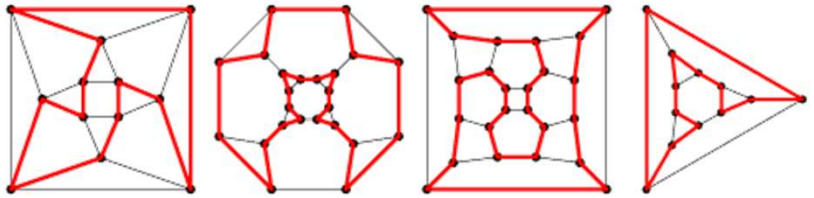
Hamiltonian cycle problem

The longest-simple-cycle problem is the problem of determining a simple cycle (no repeated vertices)of maximum length in a graph. Show that this problem is NP-Complete

A ***Hamiltonian cycle*** of an undirected graph $G(V, E)$ is a simple cycle that contains each vertex in V.

A graph that contains a Hamiltonian cycle is said to be ***Hamiltonian***; otherwise, it is ***non-Hamiltonian***.

Theorem 34.13
The hamiltonian cycle problem is NP-complete.



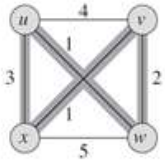
The traveling salesman problem

In the ***traveling-salesman problem***, is closely related to the hamiltonian-cycle problem, a salesman must visit n cities.

Modeling the problem as a complete graph with n vertices, we can say that the salesman wishes to make a ***tour***, or hamiltonian cycle, visiting each city exactly once and finishing at the city he starts from.

> The salesman incurs a nonnegative integer cost $c(i, j)$ to travel from city i to city j, and the salesman wishes to make the tour whose total cost is minimum, where the total cost is the sum of the individual costs along the edges of the tour.

For example, a minimum-cost tour is $\langle u, w, v, x, u \rangle$, with cost 7.



The formal language for the corresponding decision problem is

$$\begin{aligned}\text{TSP} = \{\langle G, c, k\rangle : \ & G = (V, E) \text{ is a complete graph,} \\ & c \text{ is a function from } V \times V \to \mathbb{Z}, \\ & k \in \mathbb{Z}, \text{ and} \\ & G \text{ has a traveling-salesman tour with cost at most } k\}\ .\end{aligned}$$

Theorem 34.14

The traveling-salesman problem is NP-complete.

34.5-1

The ***subgraph-isomorphism problem*** takes two undirected graphs G_1 and G_2, and it asks whether G_1 is isomorphic to a subgraph of G_2. Show that the subgraph-isomorphism problem is NP-complete.

- راهنمایی:
- از مساله clique استفاده شود.