


NP-Completeness

NP-Completeness


Recall:

❖ **Euler circuit problem:** Find a path that touches every edge exactly once.


❖ **Hamilton cycle problem:** Find a single cycle that contains every vertex.


❖ **Single-source unweighted shortest-path problem**


❖ **Single-source unweighted longest-path problem**


 **No** known algorithms are guaranteed to run in *polynomial* time.

☞ Easy vs. Hard

The *easiest*: $O(N)$ – since we have to read inputs at least once.

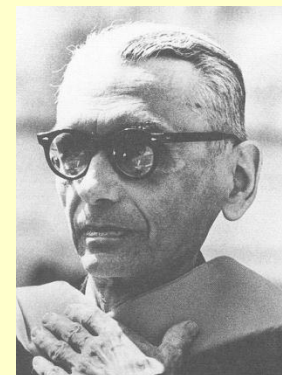
The *hardest*:

– undecidable problems.



The great mathematician **David Hilbert** at the 1900 International Congress of Mathematicians outlined 23 research problems to be investigated in the coming century. One of the problems is the question of **Decidability** — Could there exist, at least in principle, any definite method or process by which *all mathematical questions could be decided*?

Kurt Gödel proved in 1931 that not all true statements that evolve from an axiomatic system can be proven – *we can never know everything nor prove everything we discover.*



【Example】 **Halting problem**: Is it possible to have your C compiler detect all **infinite loops**?

Answer: No.

Proof: If there exists an infinite loop-checking program, then surely it could be used to **check itself**.

```

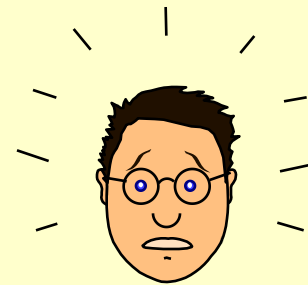
Loop( P )
{
  /* 1 */ if ( P(P) loops )    print (YES);
  /* 2 */ else infinite_loop();
}

```

Impossible to tell

What will happen to **Loop(Loop)** ?

- Terminates ➡ **/* 2 */** is true ➡ Loops
- Loops ➡ **/* 1 */** is true ➡ Terminates



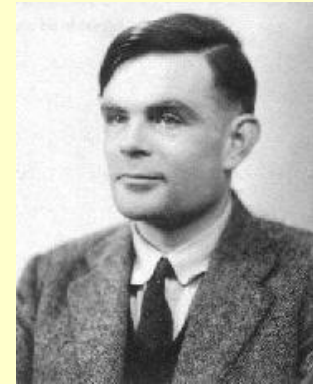
☞ The Class NP

TURING MACHINE

Alan Mathison Turing (June 21, 1912 – June 7, 1954)

Founder of Computer Science.

More details can be found at <http://www.turing.org.uk> .



Task

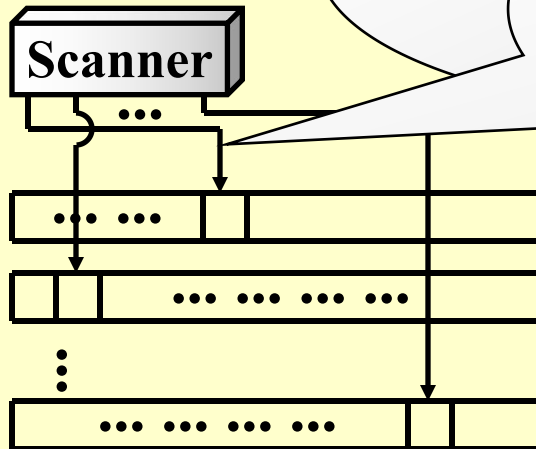
To simulate any kind of **computation** which a mathematician can do by some **arithmetical method** (assuming that the mathematician has infinite time, energy, paper and pen, and is completely dedicated to the work).

Components

and

Operations

**One head can point to
one and only one unit at each state,
and it may move at most one unit
to the left or right.**



- ③ Head moves one position to the right (**R**), or stays at its current position (**S**).

A **Deterministic Turing Machine** executes one instruction at each point in time. Then depending on the instruction, it goes to the next **unique** instruction.

A **Nondeterministic Turing Machine** is **free to choose** its next step from a finite set. And if one of these steps leads to a solution, it will **always choose the correct one**.

NP: **Nondeterministic polynomial-time**

The problem is **NP** if we can prove any solution is true *in polynomial time*.

[[**Example**] **Hamilton cycle problem**: Find a single cycle that contains every vertex – **does this simple circuit include all the vertices?** **NP** **undecidable**.]

Undecidable

problems are still

undecidable.

Note: Not all decidable problems are in NP. For example, consider the problem of determining whether a graph **does not** have a Hamiltonian cycle.

$$P \subseteq NP \quad \checkmark \qquad P \subset NP \quad ?$$

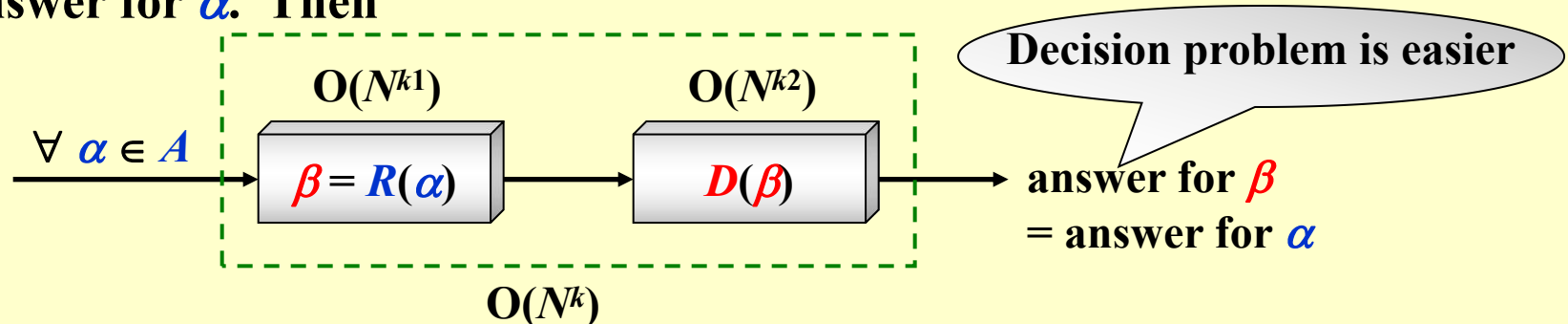
👉 NP-Complete Problems -- the **hardest**

An **NP-complete problem** has the property that any problem in NP can be **polynomially reduced** to it.



If we can solve **any** NP-complete problem in *polynomial* time, then we will be able to solve, in *polynomial* time, **all** the problems in NP!

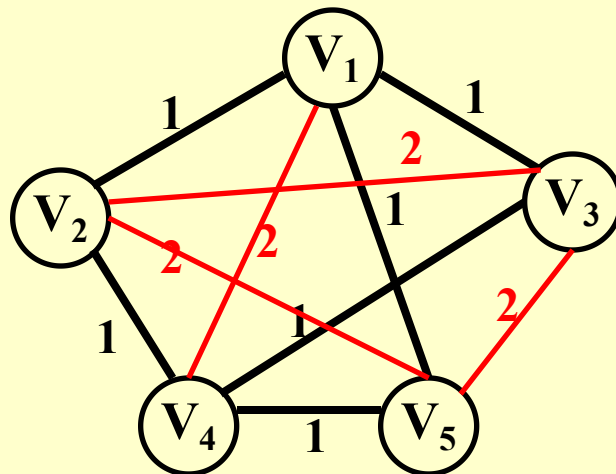
Given any instance $\alpha \in \text{Problem } A$, if we can find a program $R(\alpha) \rightarrow \beta \in \text{Problem } B$ with $T_R(N) = O(N^{k_1})$, and another program $D(\beta)$ to get an answer in time $O(N^{k_2})$. And more, if the answer for β is the same as the answer for α . Then



[[**Example**]] Suppose that we already know that the **Hamiltonian cycle problem** is NP-complete. Prove that the **traveling salesman problem** is NP-complete as well.

- ❖ **Hamiltonian cycle problem:** Given a graph $G=(V, E)$, is there a **simple cycle** that visits **all vertices**?
- ❖ **Traveling salesman problem:** Given a **complete** graph $G=(V, E)$, with edge costs, and an integer K , is there a **simple cycle** that visits **all vertices** and has **total cost $\leq K$** ?

Proof: TSP is obviously **in NP**, as its answer can be verified polynomially.



$$K = |V|$$

G has a Hamilton cycle **iff**
 G' has a traveling
 salesman tour of total
 weight $|V|$.



The first problem that was proven to be NP-complete was the *Satisfiability* problem (**Circuit-SAT**): Input a boolean expression and ask if it has an assignment to the variables that gives the expression a value of 1.

Cook showed in 1971 that all the problems in NP could be polynomially transformed to Satisfiability. He proved it by **solving this problem on a nondeterministic Turing machine in polynomial time.**

A Formal-language Framework

👉 Abstract Problem

an *abstract problem* Q is a binary relation on a set I of problem *instances* and a set S of problem *solutions*.

[[Example]] For **SHORTEST-PATH** problem

$I = \{ \langle G, u, v \rangle : G=(V, E) \text{ is an undirected graph; } u, v \in V \};$

$S = \{ \langle u, w_1, w_2, \dots, w_k, v \rangle : \langle u, w_1 \rangle, \dots, \langle w_k, v \rangle \in E \}.$

For every $i \in I$, **SHORTEST-PATH**(i) = $s \in S$.

For decision problem **PATH**:

$I = \{ \langle G, u, v, k \rangle : G=(V, E) \text{ is an undirected graph; } u, v \in V;$
 $k \geq 0 \text{ is an integer } \};$

$S = \{ 0, 1 \}.$

For every $i \in I$, **PATH**(i) = 1 or 0.

👉 Encodings

Map I into a binary string $\{ 0, 1 \}^* \rightarrow Q$ is a *concrete problem*.

☞ Formal-language Theory — *for decision problem*

- An *alphabet* Σ is a finite set of symbols $\{0, 1\}$
- A *language* L over Σ is any set of strings made up of symbols from Σ

$$L = \{x \in \Sigma^* : Q(x) = 1\}$$
- Denote *empty string* by ε
- Denote *empty language* by \emptyset
- Language of all strings over Σ is denoted by Σ^*
- The *complement* of L is denoted by $\Sigma^* - L$
- The *concatenation* of two languages L_1 and L_2 is the language

$$L = \{x_1x_2 : x_1 \in L_1 \text{ and } x_2 \in L_2\}.$$
- The *closure* or *Kleene star* of a language L is the language

$$L^* = \{\varepsilon\} \cup L \cup L^2 \cup L^3 \cup \dots,$$

where L^k is the language obtained by concatenating L to itself k times

- Algorithm A *accepts* a string $x \in \{0, 1\}^*$ if $A(x) = 1$
- Algorithm A *rejects* a string x if $A(x) = 0$
- A language L is *decided* by an algorithm A if every binary string *in* L is *accepted* by A and every binary string *not in* L is *rejected* by A
- To *accept* a language, an algorithm need only worry about strings in L , but to *decide* a language, it must correctly accept or reject every string in $\{0, 1\}^*$

$\mathbf{P} = \{ L \subseteq \{0, 1\}^* : \text{there exists an algorithm } A \text{ that } \mathbf{decides} \\ L \text{ in polynomial time } \}$

- A *verification algorithm* is a two-argument algorithm A , where one argument is an ordinary input string x and the other is a binary string y called a *certificate*.
- A two-argument algorithm A *verifies* an input string x if there exists a certificate y such that $A(x, y) = 1$.
- The *language* verified by a verification algorithm A is $L = \{ x \in \{0, 1\}^* : \text{there exists } y \in \{0, 1\}^* \text{ such that } A(x, y) = 1 \}$.

[[Example]] For SAT

$$x = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$$

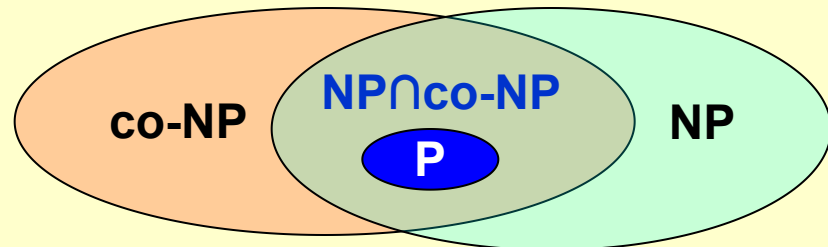
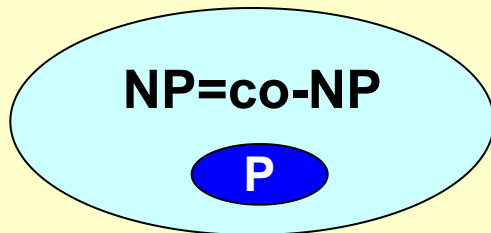
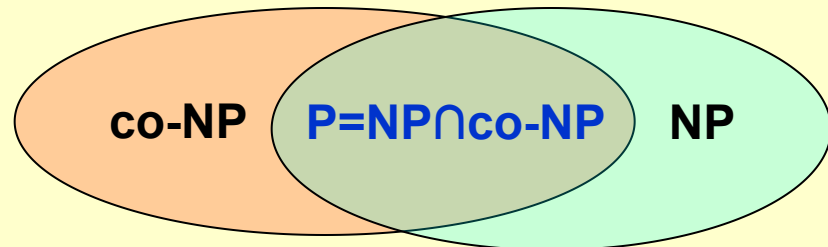
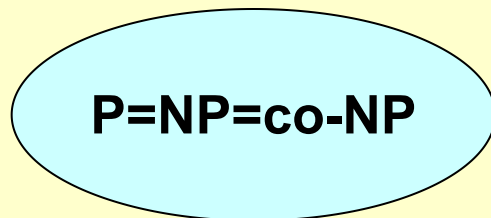
Certificate: $y = \{ x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1 \}$

A language L belongs to **NP** iff there exist a two-input polynomial-time algorithm A and a constant c such that $L = \{ x \in \{0, 1\}^* : \text{there exists a certificate } y \text{ with } |y| = O(|x|^c) \text{ such that } A(x, y) = 1 \}$. We say that algorithm A *verifies* language L in *polynomial time*.

$$L \in NP \xrightarrow{?} \bar{L} \in NP$$

complexity class **co-NP** = the set of languages L such that
 $\bar{L} \in NP$

Four possibilities:





no harder than

A language L_1 is *polynomial-time reducible* to a language L_2 ($L_1 \leq_p L_2$) if there exists a *polynomial-time computable* function $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$ such that for all $x \in \{0, 1\}^*$, $x \in L_1$ iff $f(x) \in L_2$.

We call the function f the *reduction function*, and a polynomial-time algorithm F that computes f is called a *reduction algorithm*.

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

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

[[**Example**]] Suppose that we already know that the **clique problem** is NP-complete. Prove that the **vertex cover problem** is NP-complete as well.

❖ **Clique problem:** Given an undirected graph $G = (V, E)$ and an integer K , does G contain a **complete subgraph** (*clique*) of (at least) K vertices?

$\text{CLIQUE} = \{ \langle G, K \rangle : G \text{ is a graph with a clique of size } K \}.$

❖ **Vertex cover problem:** Given an undirected graph $G = (V, E)$ and an integer K , does G contain a subset $V' \subseteq V$ such that $|V'|$ is (at most) K and every edge in G has a vertex in V' (*vertex cover*)?

$\text{VERTEX-COVER} = \{ \langle G, K \rangle : G \text{ has a vertex cover of size } K \}.$

Proof: ① $\text{VERTEX-COVER} \in \text{NP}$

Given any $x = \langle G, K \rangle$, take $V' \subseteq V$ as the certificate y .

Verification algorithm: check if $|V'| = K$; check if for each edge $(u, v) \in E$, that $u \in V'$ or $v \in V'$.

$O(N^3)$

Proof (con.): ② CLIQUE \leq_p VERTEX-COVER

G has a **clique** of size K iff \overline{G}
has a **vertex cover** of size $|V| - K$.

\Rightarrow G has a **clique** $V' \subseteq V$ of size K

Let (u, v) be any edge in \overline{E} \rightarrow

At least one of u or v does not belong to V'

At least one of u or v does belong to $V - V'$

Every edge of \overline{G} is covered by a vertex in $V - V'$

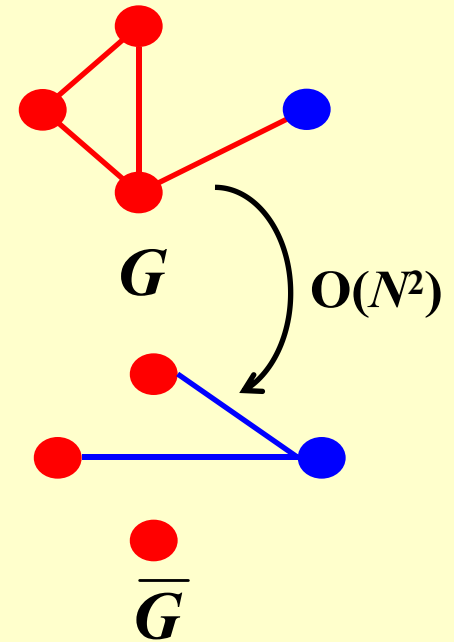
Hence, the set $V - V'$, which has size $|V| - K$ forms
a vertex cover for \overline{G}

\Leftarrow \overline{G} has a **vertex cover** $V' \subseteq V$ of size $|V| - K$

For all $u, v \in V$, if $(u, v) \notin E$, then $u \in V'$ or $v \in V'$ or both.

For all $u, v \in V$, if $u \notin V'$ AND $v \notin V'$, then $(u, v) \in E$.

$V - V'$ is a **clique** and it has size $|V| - |V'| = K$. ■



Reference:

Introduction to Algorithms, 3rd Edition: **Ch.34, p. 1048 - 1105**; *Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. The MIT Press. 2009*