数据结构与算法设计

周　可

Mail：zhke@hust.edu.cn

华中科技大学，武汉光电国家研究中心

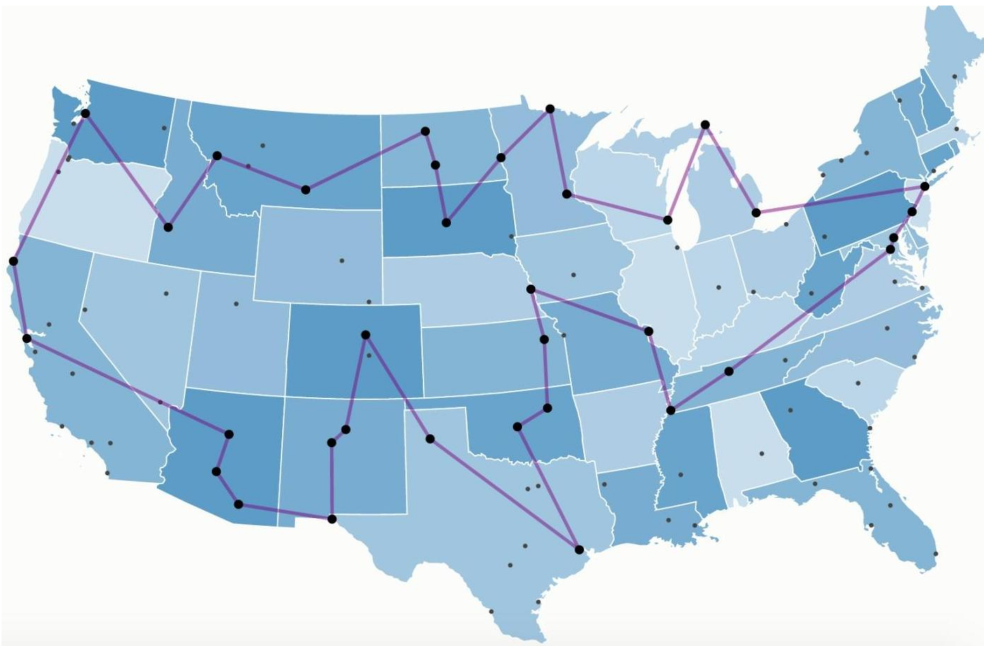
NP-Completeness

1. **Polynomial time → P**
2. **Polynomial time verification → NP**
3. **NP-Completeness → NPC**
4. **NP-hard**

引入：千禧难题

P=NP? 是千禧年大奖难题（世界七大数学难题）之首。

TSP，即旅行商问题，是数学领域著名问题之一，也是NP问题。



10个城市为例：

10！=3628800

贪心算法求解

算法简单，时间/空间复杂度低

时间复杂度的简单分类

（1）多项式时间 ⟸ $2n$，$n\log n$，$3n^2+4n$

（2）非多项式时间 ⟸ 2^n，$n!$，n^n

已知：$2n$，2^n，$n\log n$，$n!$，$3n^2+4n$

Q1：该如何归类？

Q2：多项式时间算法一定比非多项式时间算法快吗？

为什么？

Background

Almost all the problem we have studied thus far have been tractable or easy problem that are solvable by ***polynomial-time*** algorithms.

(1) How to determine whether a problem is easy or not, and tractable or not ?

(2) Whether all problems can be solved in polynomial time ?

Definition

- A concrete problem is ***polynomial-time solvable*** if there exists an algorithm to solve it in time $O(n^k)$ for some constant k.
- ***The complexity class P*** is the set of concrete decision problems that are polynomial-time solvable.

Q: how to describe a concrete problem and a concrete decision problem.

Definition

Before describe ***a concrete problem***, define ***an abstract problem*** first.

Problem Instances

Abstract problem

$$Q = < I, S >$$

Problem solution

For example

An instance for SHORTEST-PATH is a triple consisting of a graph and two vertices.

$$I_{SHORTEST-PATH} = < G, u, v >$$

A solution for SHORTEST-PATH is a sequence of vertices in the graph, with perhaps the empty sequence denoting that no path exists.

$$S_{SHORTEST-PATH} = \begin{cases} (u, \ldots x_i, \ldots v) & x_i \in G \\ \varnothing & no\ path\ exists \end{cases}$$

Definition

- ***Optimization problem***: those require some value to minimized or maximized.

e.g. Given undirected graph G and vertices u and v, find a path from u to v that uses the fewest edges.

easy to recast

- ***Decision problem***: those having a yes/no solution.

e.g. Given a undirected graph G, vertices u and v and an integer k, does a path exist from u to v consisting at most k edges?

For example (*Optimization problem*)

An instance for SHORTEST-PATH is a triple consisting of a graph and two vertices.

$$I_{SHORTEST-PATH} = \ < G, u, v >$$

A solution for SHORTEST-PATH is a sequence of vertices in the graph, with perhaps the empty sequence denoting that no path exists.

$$S_{SHORTEST-PATH} = \begin{cases} (u, \ldots x_i, \ldots v) & x_i \in G \\ \varnothing & no \ \ path \ exists \end{cases}$$

For example (*Decision problem*)

An instance for SHORTEST-PATH decision problem.

$$I_{SHORTEST-PATH} = < G, u, v, k >$$

A solution for SHORTEST-PATH decision problem.

$$S_{SHORTEST-PATH} = \begin{cases} 1 & yes \\ 0 & no \end{cases}$$

Definition

- ***A concrete problem*** is a problem whose instance set is the set of binary string.
- An abstract problem can be represented into a concrete problem with ***encoding***.
- An encoding of a set S of abstract object is a mapping *e* from S to the set of binary.

e.g. $N = \{0,1,2,3\ldots\}$

$\text{String} = \{0,1,10,11,100,..\}$

$\underline{e(17)} = 10001$

encoding

Tips

- With different encoding, the algorithm runs in either polynomial or superpolynomial time.

e.g. Support that an integer k is to be provided as the sole input to an algorithm and suppose that the running time of the algorithm is $O(k)$.

Input way	Input size	Running time
Unary	n	$O(n)$
Binary	$N = \lfloor \lg k \rfloor + 1$	$O(2^n)$

standard encoding: assume that the encoding of an integer is polynomially related to its binary representation.

e.g. <G> denotes the standard encoding of a graph G.

Summary

- An algorithm solves a concrete problem in time $O(T(n))$, if when it is provided an instance i length of n = $|i|$, the algorithm can produce the solution in $O(T(n))$ time.

- A concrete problem is polynomial-time solvable, if there exists an algorithm to solve it in time $O(n^k)$ for some constant k.

Optional—formal-language Definition

- **Define any decision problem Q** as a language L over $\sum = \{0, 1\}$, where

$$L = \{x \in \sum^{*} : Q(x) = 1\}$$

e.g. SHORTES-PATH= {<G, u, v, k>: G = (V, E) is an undirected graph, u, v $\in$ V, k $\geq$ 0 is an integer, and there exists a path from u to v in consisting of at most k edges}

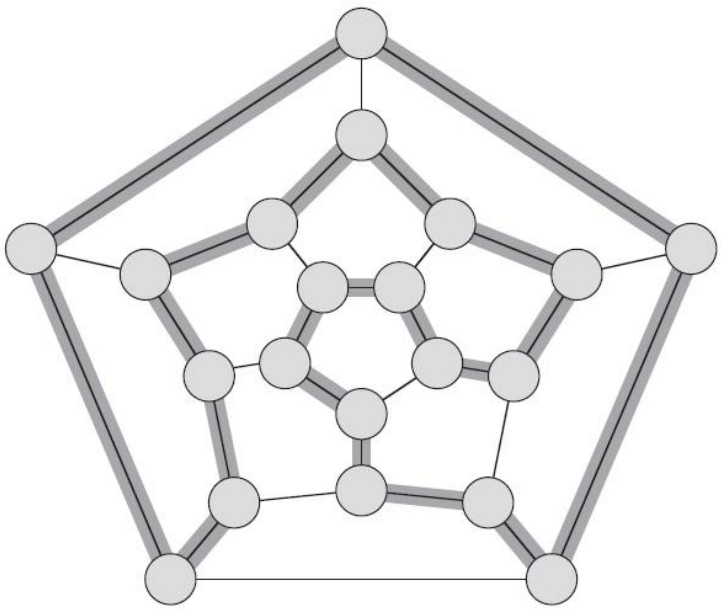
- Define **the complexity class P**:

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

Chapter10 NP-Completeness

1. Polynomial time
2. **Polynomial time verification**
3. NP-Completeness
4. NP-hard

Background—Hamiltonian cycles



A Hamiltonian cycle is a simple cycle that contains each vertex in an undirected graph.

It's name honors W.R. Hamilton, who described a mathematical game on the dodecahedron in which one player sticks five pins in any five consecutive vertices and other player must complete the path to form a cycle containing all the vertices.

***It is NP problem*.**

HAM-CYCLE = {<*G*>: *G* is a Hamiltonian graph}.

Hamiltonian cycles

HAM-CYCLE = {<G>: G is a Hamiltonian graph}.

1. Choose encoding: adjacency matrix (input size is n)
2. Get vertices number: $m = \sqrt{n}$
3. Get possible permutations of the vertices: $m!$
4. Calculate running time: $\Omega(m!) = \Omega(\sqrt{n}\,!) = \Omega(2^{\sqrt{n}})$

$$\Omega(2^{\sqrt{n}}) >> O(n^k)$$

HAM-CYCLE isn't P problem but can be verified in polynomial time. It's NP problem.

Definition

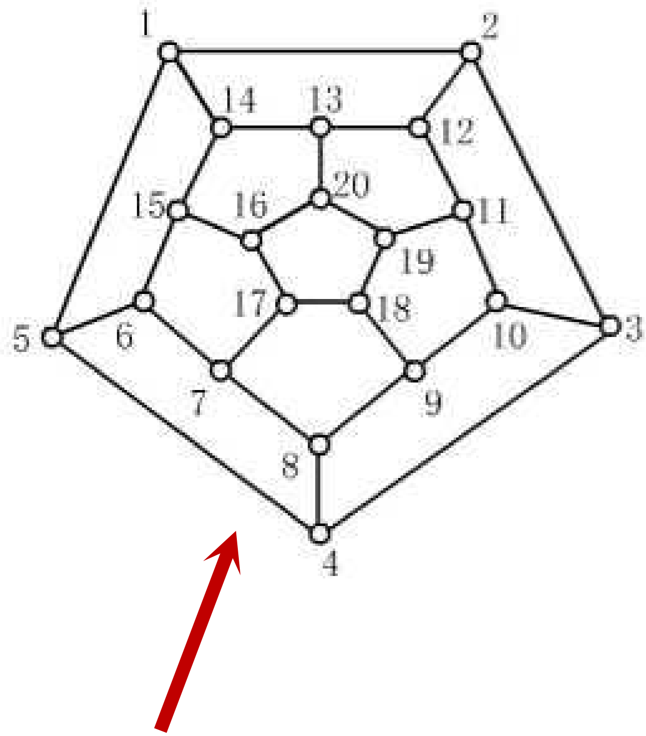
- The language verified by a ***verification algorithm*** A is:

$$L = \{x \in \{0,1\}^*: \text{there exists } y \in \{0,1\}^* \text{ such that } A(x, \underbrace{y}_{certificate})=1\}$$

An algorithm A verifies a language L if for any string $x \in L$, there exists a ***certificate y*** that A can use to prove that $x \in L$.

For example

Verification algorithm of Hamiltonian cycles.



Step 1: checking whether **certificate *y*** is a permutation of vertices of V.

Step 2: checking whether each of consecutive edges along the **certificate *y*** is exists in the graph.

***Certificate y* = {1,2,3,4…20}**

Definition

- ***The complexity class NP*** of languages that can be ***verified by a polynomial-time algorithm***.

- (optional) A language L belong to NP if and only if exist a two-input polynomial-time algorithm A and a constant c such that:

$L = \{x \in \{0,1\}^*$: there exists a certificate y with $|y| = O(|x|^c)$ such that $A(x, y) = 1\}$

Q: P = NP?

Obviously, P $\subseteq$ NP, but it is unknown whether P = NP.

- Intuitively, P problems can be solved quickly, NP problems can be verified quickly.
- the existence of NP-complete problems show compelling evidence that P $\neq$ NP.

What is a NP-complete problem? What is the relationship between P and NP?

NP-Completeness

1. Polynomial time

2. Polynomial time verification

3. NP-Completeness

4. NP-hard

Background

Why theoretical computer scientists believe that P≠NP come from the existence of the class of ***NP-complete problems***.

(1) If **any** NP-complete problem can be solved in polynomial time, then **every** NP problem has a polynomial time solution.

(2)Despite years of study, **no polynomial-time algorithm** has ever been discovered for any NP-complete problem.

Definition

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

1. $L \in$ NP, and

2. $L` \leq_P L$ for every $L` \in$ NP.

Polynomial-time reducible

Reducibility is tools to make decision.

Tips： if a language L satisfies property 2, but not meets 1, we say that L is **NP-hard**.

Reducibility

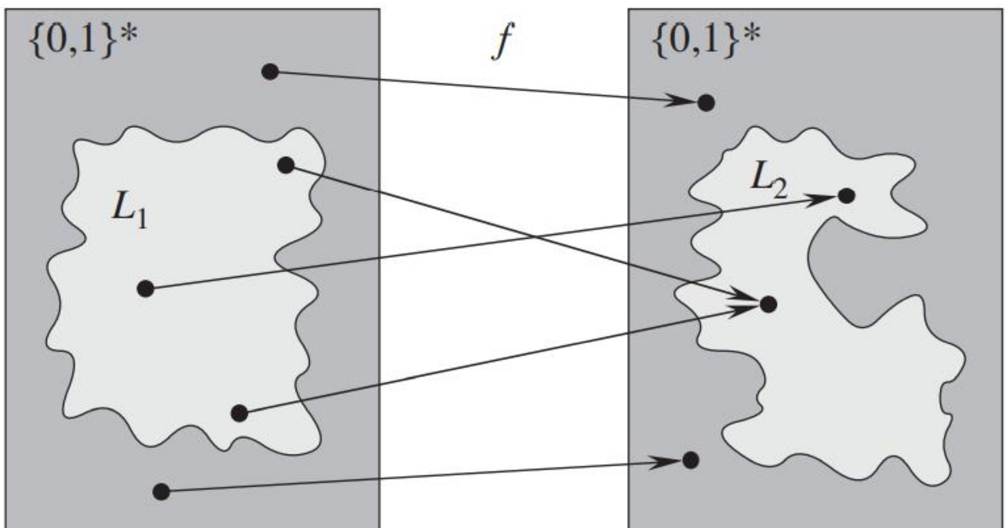
Intuitively, a problem Q can be ***reduced*** to another problem $Q`$, if any instance of $Q$ can be ***easily rephrased*** as an instance of $Q`$, the solution to which provides a solution to the instance of Q.

$$ax + b = 0 \xrightarrow{\textbf{Reduce to}} 0x^2 + ax + b = 0$$

Definition

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

$$x \in L_1 \text{ if and only if } f(x) \in L_2$$

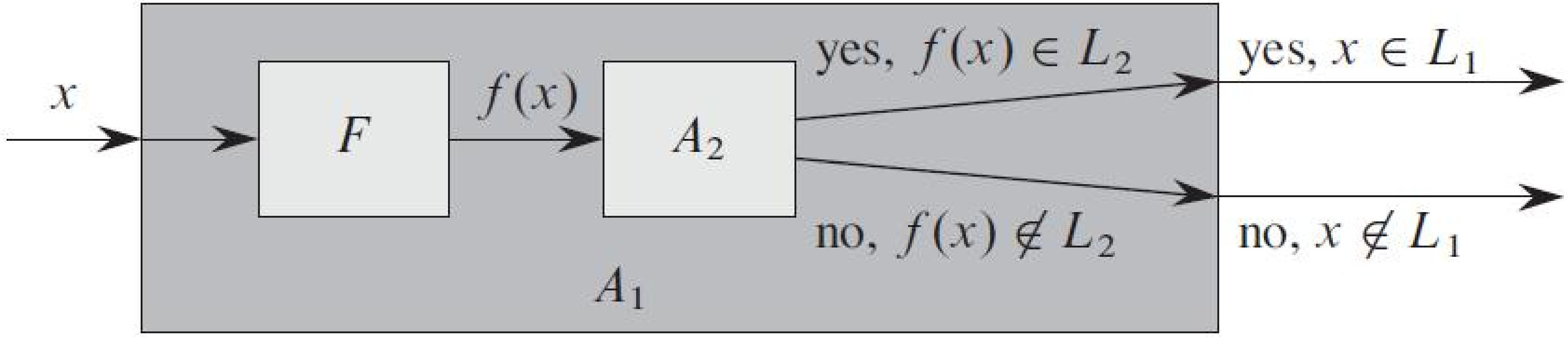


For any input $x \in \{0,1\}$, the question of whether $x \in L_1$ has the same answer as the question of whether $f(x) \in L_2$.

Lemma

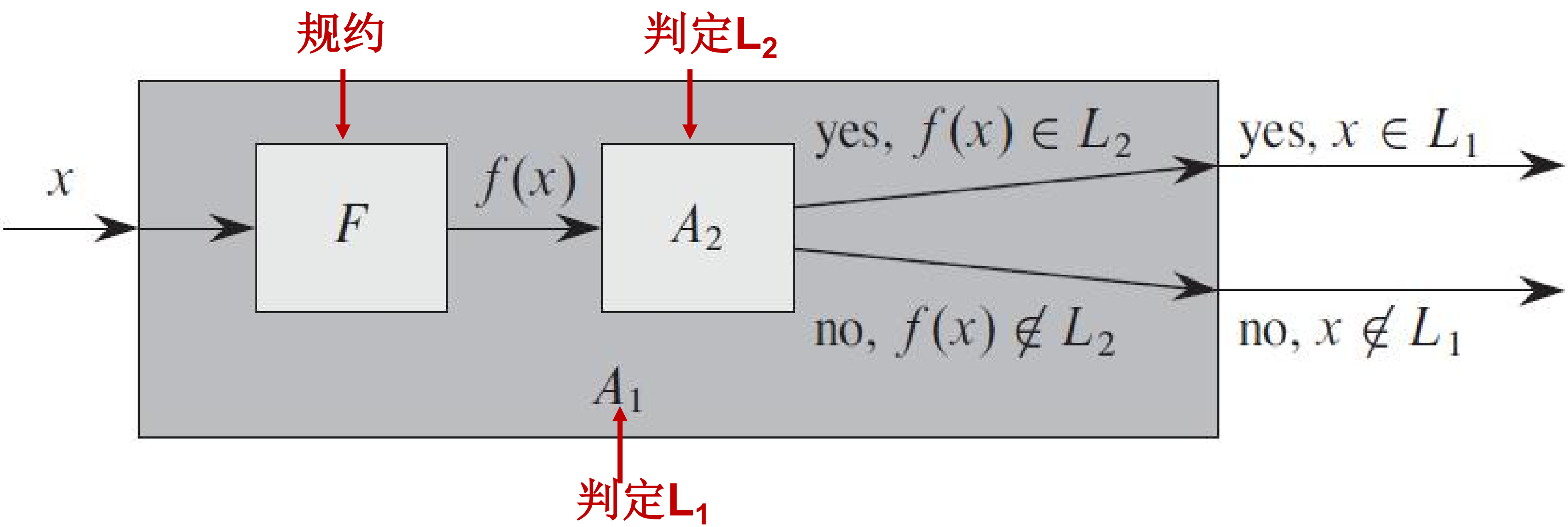
If $L_1, L_2 \subseteq \{0,1\}^*$ are languages such that $L_1 \leq_P L_2$, then $L_2 \in P$ implies $L_1 \in P$.

Proof Let A_2 be a polynomial-time algorithm that decides L_2, and let F be a polynomial-time reduction algorithm that computes the reduction function f. We shall construct a polynomial-time algorithm A_1 that decides L_1.



Practice:

Please write the meaning of the following figure.

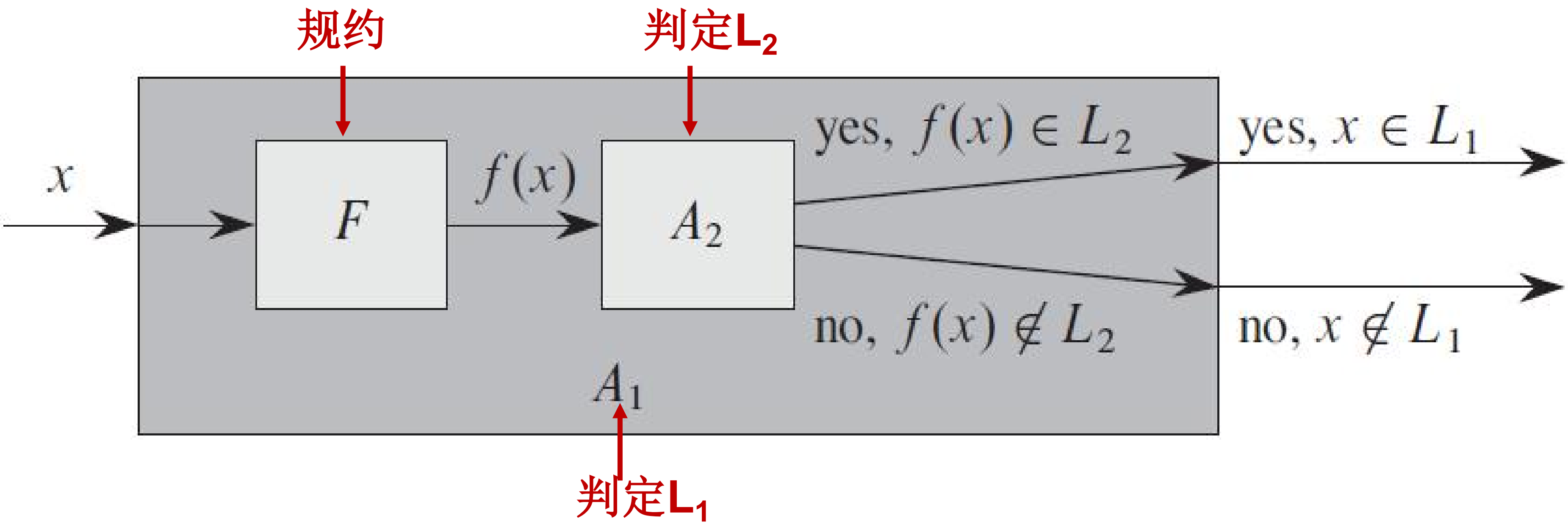


算法F是一个规约算法，它在多项式时间内计算出从L_1到L_2的规约函数f，A_2是一个能判定L_2的多项式时间算法。

算法A_1利用F将任何输入x转换为$f(x)$，再利用A_2来判定是否有$f(x) \in L_2$，最终判定是否有$x \in L_1$.

Practice:

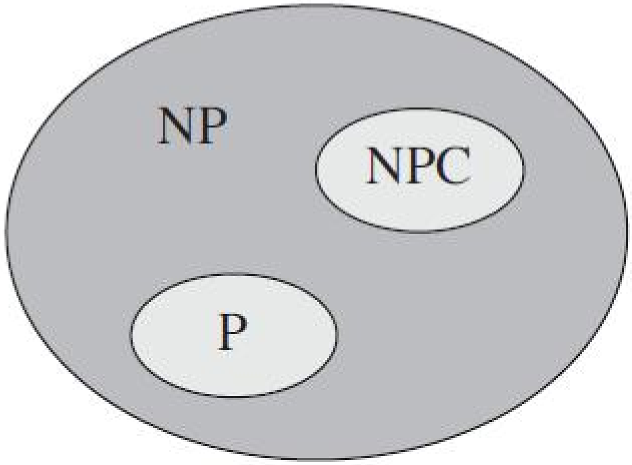
Please write the meaning of the following figure.



（多项式时间）判定L_1? ⇨ 规约 +（多项式时间）判定L_2

Lemma

IF any NP-complete problem is polynomial-time solvable, then P = NP. Equivalently, if any problem in NP is not polynomial-time solvable, then no NP-complete problem is polynomial-time solvable.



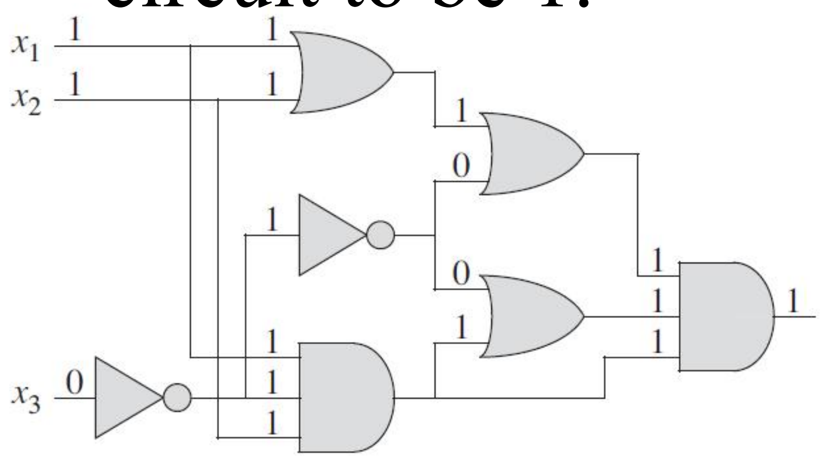
It is for this reason that research into the $P \neq NP$ question centers around the NP-complete problems.

For all we known, someone may yet come up with a polynomial-time algorithm for an NP-complete problem, thus proving P = NP.

Circuit satisfiability

- A ***truth assignment*** for a Boolean combinational circuit is a set of Boolean input values.

- A one-output ***Boolean combinational circuit*** is ***satisfiable*** if it has a ***satisfying assignment***: a truth assignment that cases the output of the circuit to be 1.



Circuit satisfiability has the historical honor of being the first problem ever shown to NPC.

NP-completeness Proofs

Lemma

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

① Prove $L \in$ NP.

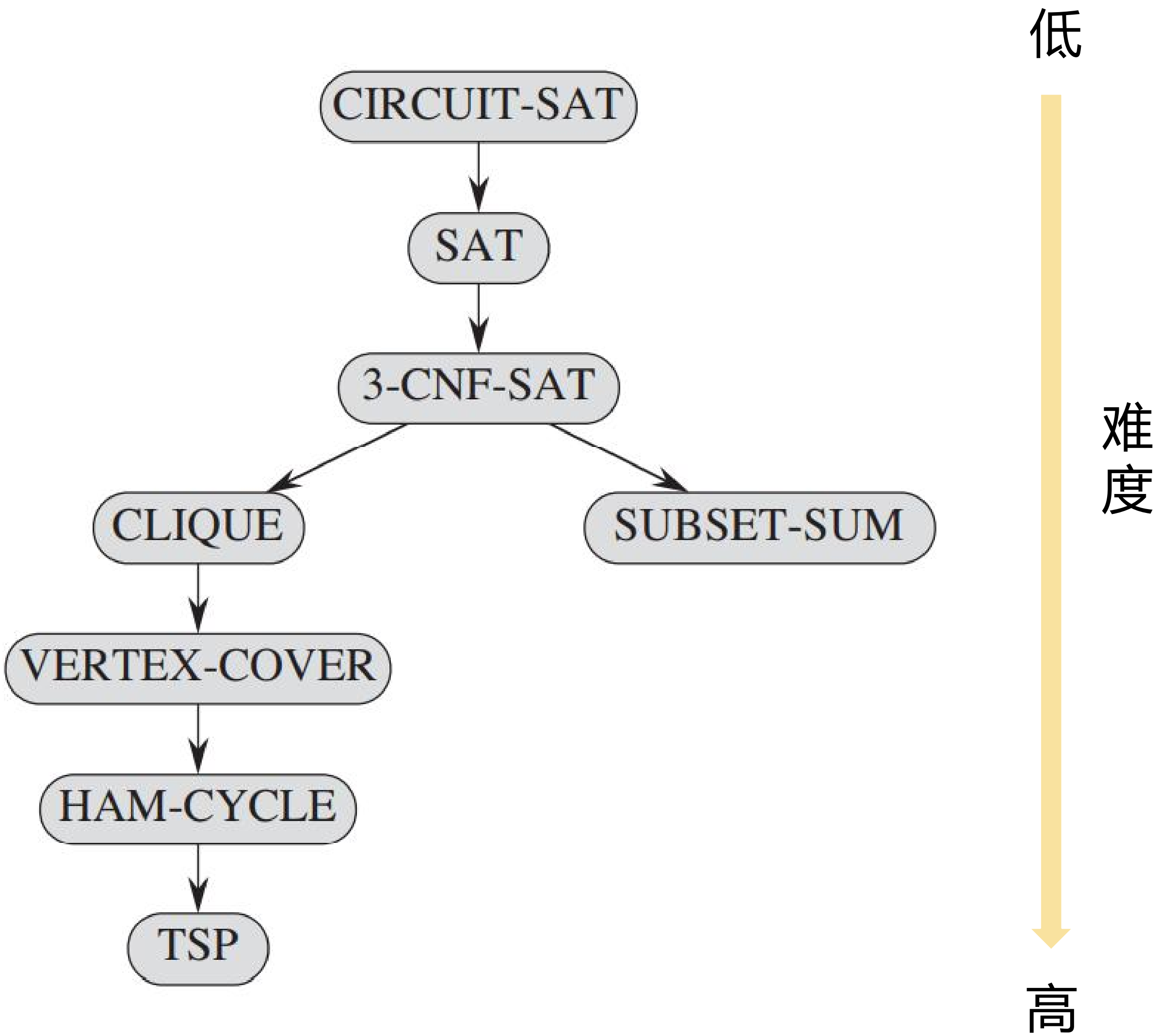
② Select a known NP-complete language $L`$.

③ Describe an algorithm that computes a function f mapping every instance $x \in \{0,1\}^*$ of L` to an instance $f(x)$ of L.

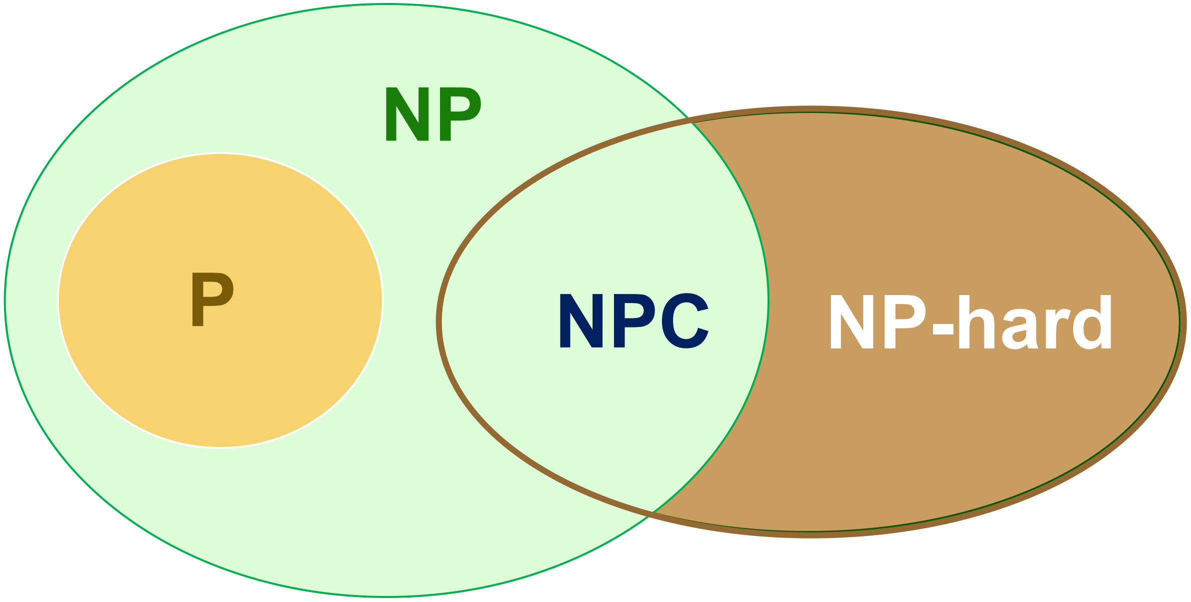
④ Prove that the function f satisfies $x \in L`$ and only if $f(x) \in L$ for all $x \in \{0,1\}^*$.

⑤ Prove that the algorithm computing f runs in polynomial time.

NPC部分问题



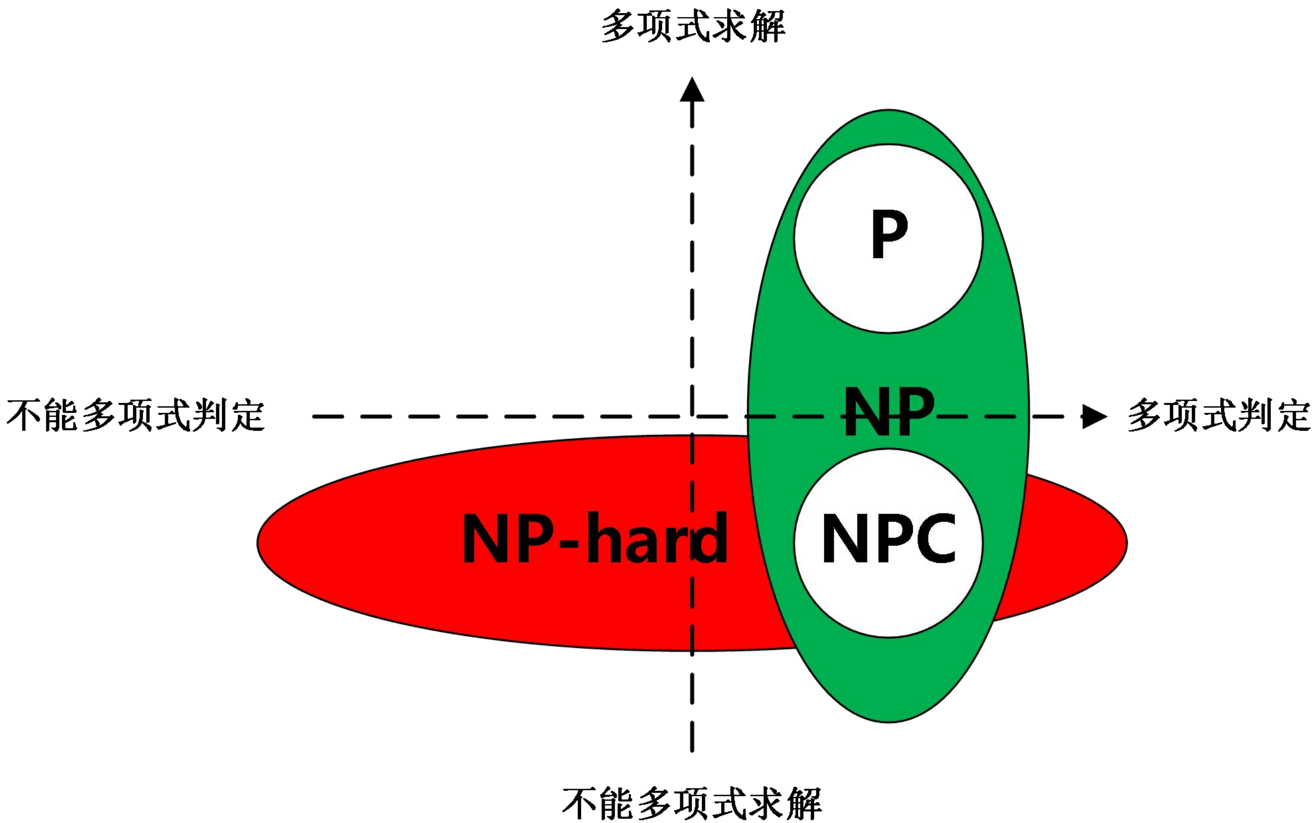
低
CIRCUIT-SAT
SAT
3-CNF-SAT
CLIQUE
SUBSET-SUM
VERTEX-COVER
HAM-CYCLE
TSP
难
度
高



1. P $\subseteq$NP

2. NP-hard问题，不一定是NP问题

2. NPC问题: 是NP问题，且是NP-hard问题



1. P $\subseteq$NP

2. NP-hard问题，不一定是NP问题

2. NPC问题: 是NP问题，且是NP-hard问题

拓展：量子计算机

国际最新进展

求解数学算法高斯玻色取样只需要两百秒，比用目前世界上最快的SC“富岳”要快一百万亿倍。

62比特的超导量子计算原型机**“祖冲之号”**
（2021.5，中科大）
（2021.10，“祖冲之二号”，66比特）

量子计算关键词：
量子叠加态，
并行，概率性…

76个光子的量子计算原型机**“九章”**
（2020.12，中国在全球第二个实现量子霸权）
（2021，“九章二号”，113个光子）

- 发展三阶段：
  量子计算（量子优越性/量子霸权）→量子模拟机→可编程通用量子计算机
- **指数增长复杂度（电子计算机）→多项式增长复杂度（量子计算机）**

Thank You!

Q&A