Office Hour 1

2025.10.9

Finite Automaton

- 有限自动机（Finite Automaton），简称FA，是计算理论和形式语言理论中重要的数学模型之一. 描述为一种抽象的计算设备，用于处理输入符号序列并转换为输出结果的过程。
- 有限自动机由以下主要组成要素构成:

- 状态集合（Set of States）：有限自动机包含有限个状态（可能是初始状态、接受状态等）的集合。
- 输入字母表（Input Alphabet）：有限自动机接受的输入符号的有限集合，通常用Σ表示。
- 状态转移函数（Transition Function）：描述状态之间的转移关系，通常用δ表示，形如δ: Q × Σ → Q，表示从一个状态经过输入符号的转移得到另一个状态。
- 初始状态（Start State）：有限自动机开始接受输入时的初始状态。
- 接受状态（Accept States）：有限自动机在某些状态处停止接受输入并给出输出结果。

Finite Automaton

有限自动机可以分为以下两种类型:

确定性有限自动机（Deterministic Finite Automaton, DFA）：对于给定的状态和输入符号，存在唯一的下一个状态。即状态转移函数δ：Q × Σ → Q是确定的。DFA可以准确定义形式语言，适用于正则语言识别等。

非确定性有限自动机（Nondeterministic Finite Automaton, NFA）：对于给定的状态和输入符号，可能存在多种可能的下一个状态。NFA的状态转移函数δ：Q × Σ → 2^Q返回的是状态的子集。NFA通常需要通过ε-转移（空转移）来实现一些状态之间的跳转.

FA 的性质:

等价性（Equivalence）：两个有限自动机可以是等价的，即它们接受相同的输入语言。

最小化性（Minimization）：对于一个给定的有限自动机，可以通过状态的合并来得到一个状态数更少、结构更简洁的最小化自动机。最小化性质有助于简化自动机的描述和分析。

Regular Language

有限自动机可以构建任何正则语言，而正则语言可以是有限的或无限的集合。当然，还存在一些无限集合不是正则的。

确定性有限自动机（ DFA ）与 非确定性有限自动机（ NFA ）接受的是相同的语言，这个语言就是正则语言；

如果一个语言存在一个 有限自动机识别，那么称该语言是 正则语言(Regular Language)；

两种正则语言之间的运算：

1．并运算（ Union ）：将 A，B两个语言并在一起，就是将 A 语言的字符串，和 B 语言的字符串并在一起就可以；

2．串联运算（ Concatenation ）：将 A 语言的字符串 与 B 语言中的字符串串在一起，注意 A 语言字符串在前，B 字符串在后；

Regular Language

3．星运算（Star）：

$$A^{*}=\{x_1 x_2 \cdots x_k \mid x_i \in A\}(0 \leqslant i \leqslant k)$$

A 如果是一种语言，将 A 中的有限个字符串，串在一起，组成的集合，称为 A *

计算示例：

A 语言：A={001,10,111}

B 语言：B={ε,001}

并计算：将A,B 两个语言的集合，取并集，$A \cup B$={001,10,111,ε}

串联计算：A 语言中的任意字符，与B 语言中的任意字符串，串联在一起，A 语言中有3个字符串，B 语言中有两个字符串，所以，串联结果有6个。

Regular Language

星计算：A language的星计算，该集合一定是一个无限的集合，如果 A 语言不是空集，那么该 A * 集合个数是无限的，其可以由 K个字符串组成，K取值 0 到无穷大，[0 , + ∞ ）

$$A^* = \{\varepsilon, 001, 10, 111, \ldots\}$$

正则语言具有封闭性，正则语言组成的集合，在并运算，串联运算，星运算 中，都是封闭的；

A,B 都是正则语言，A 可以找到一个自动机识别该语言，B也可以找到一个自动机识别该语言，那么一定可以找到一个自动机分别可以识别 A ∪ B, A ◦ B, A *语言

- The complement of a regular language is also a regular language.
- The union of two regular languages is a regular language.
- The intersection of two regular languages is a regular language.
- The concatenation of two regular languages is a regular language.
- The star operation (concatenation repeated zero or more times) on a regular language is a regular language

确定性有限自动机（ DFA ）与 非确定性有限自动机（ NFA ）

确定性有限自动机（ DFA ）与 非确定性有限自动机（ NFA ）之间是相互等价的；

确定性的有限自动机（ DFA ）可以 看作是非确定性有限自动机（ NFA ）；

确定性有限自动机 给定一个输入，其输出时唯一的；

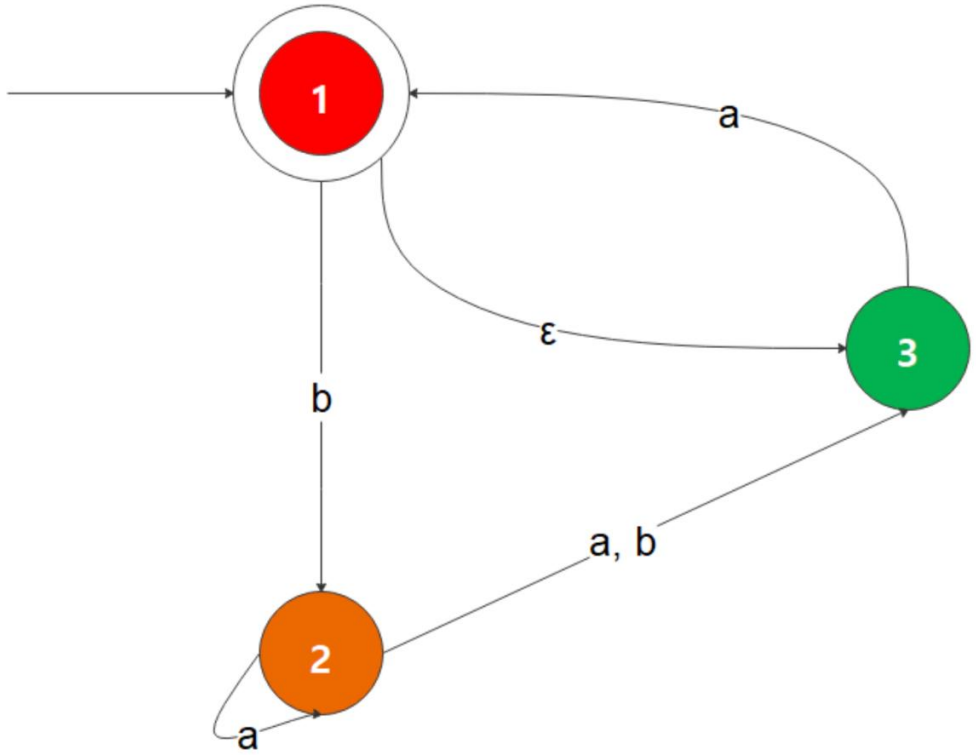
非确定性有限自动机的定义 包含 确定性有限自动机的定义中；

NFA 的后继状态 可以是 0 个，1 个 或 多个，DFA 每个状态只能有 1 个后继状态；

确定性有限自动机（ DFA ）就是 特殊的 非确定性有限自动机（ NFA ）；

非确定性有限自动机（ NFA ）转为 确定性有限自动机（ DFA ）

确定性有限自动机（ DFA ）与 非确定性有限自动机（ NFA ）转换



字符集： Σ ={a,b}

非确定性有限自动机转换成确定性有限自动机，需要将非确定性消除，只剩下确定性因素

转换过程如下：

可以使用表格法：绘制一个表格，表格列分别是a,b, 确定性有限自动机可以使用表格表示

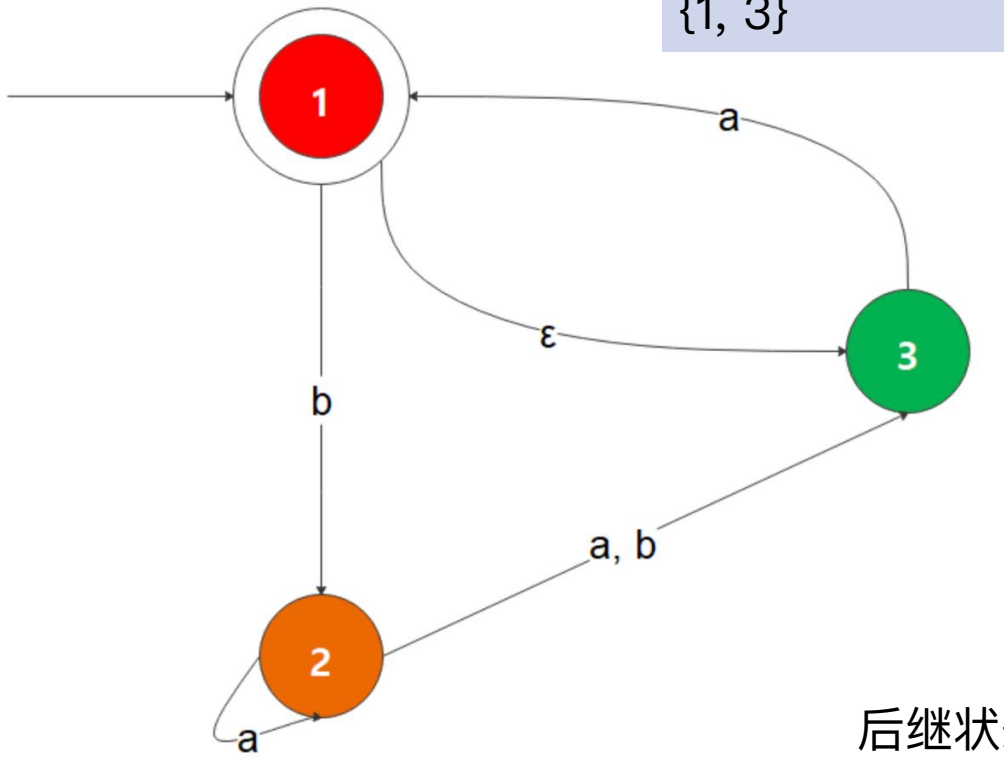
如图 非确定性有限自动机 开始状态为1，但是有一个空字符ε，指向状态3，读取到空字符ε 后会无条件跳转到后继状态 3，因此，该自动机 有2个开始状态 {1,3}.

将开始状态看做一个集合，将这两个状态组成的集合看作一个新的状态，得到如下表格

	a	b
{1, 3}		

确定性有限自动机（ DFA ）与 非确定性有限自动机（ NFA ）转换

	a	b
{1, 3}		



数值含义：该位置的含义是 { 1 , 3 }开始状态下，读取字符 a之后的后续状态集；

后继状态分析：分别考虑 1 状态下读取 a字符后的后继状态，3 状态下读取 a 字符后的后继状态；

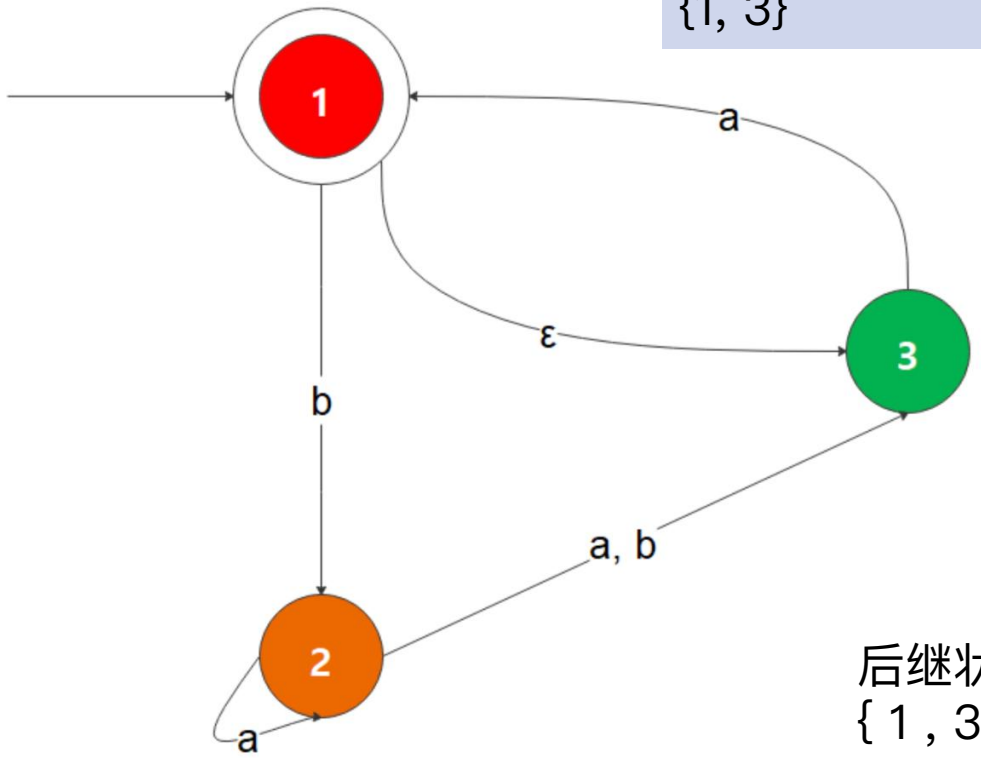
① 1 状态下读取 a 字符是空集 {∅ }

② 3 状态下读取 a 字符后继状态是 1，此时 1 状态又可以无条件跳转到 3 状态，可以与 1 状态并列，此时 3 状态 读取 a 字符的后继状态是 { 1 , 3 }

后继状态取并集：将上述的 两个 后继状态 { ∅ } 和 { 1 , 3 }，取并集，就是 { 1 , 3 }状态下读取 a字符的后继状态，取并集的结果是 { 1 , 3 } ；

	a	b
{1, 3}	{1, 3}	

确定性有限自动机（ DFA ）与 非确定性有限自动机（ NFA ）转换



	a	b
{1, 3}	{1, 3}	

数值含义：该位置的含义是 { 1 , 3 }开始状态下，读取字符 b之后的后续状态集；

后继状态分析：分别考虑 1 状态下读取 b字符后的后继状态，3 状态下读取 b 字符后的后继状态；

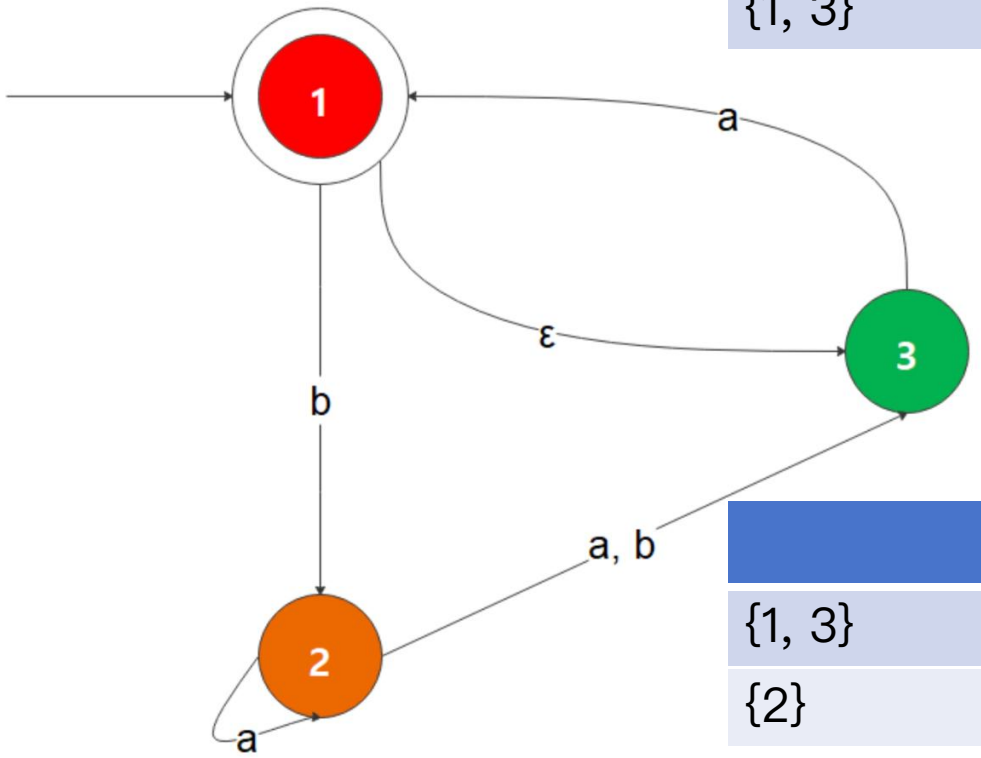
① 1 状态下读取 a 字符是2

② 3 状态下读取 b 字符后继状态是 { ∅ }

后继状态取并集：将上述的 两个 后继状态 { ∅ } 和 { 2 }，取并集，就是 { 1 , 3 }状态下读取 b字符的后继状态，取并集的结果是 {2} ；

	a	b
{1, 3}	{1, 3}	{2}

确定性有限自动机（ DFA ）与 非确定性有限自动机（ NFA ）转换



	a	b
{1, 3}	{1, 3}	{2}

在下面的分析中，需要对之前没出现过的状态逐一进行分析，所以对 状态2 进行分析

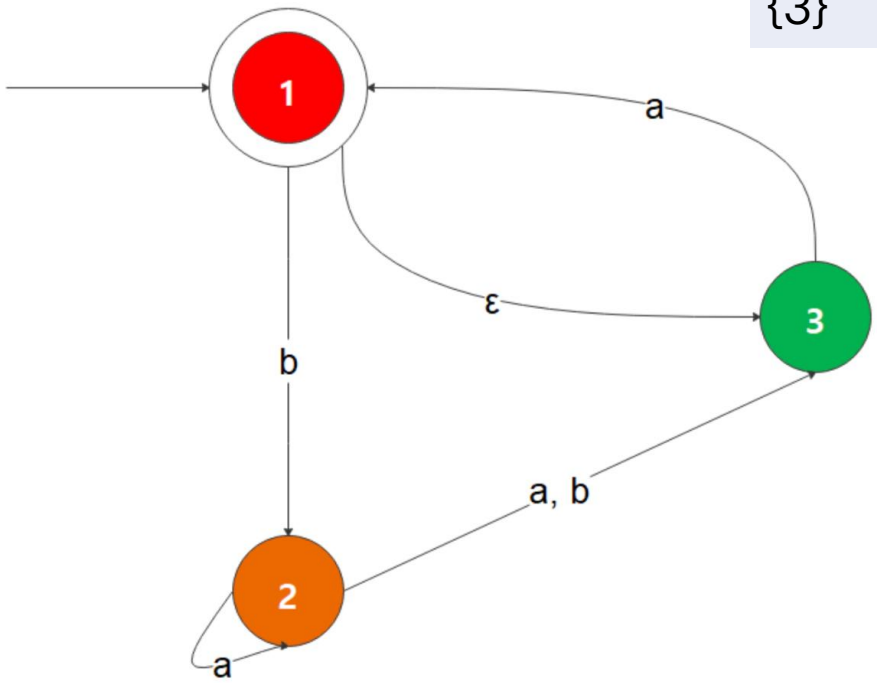
遵循上述方法，接下来开始分析以状态 2 开始之后的状态变化

	a	b
{1, 3}	{1, 3}	{2}
{2}	{2,3}	{3}

接下来，需要对{2,3},{3} 逐一进行分析

确定性有限自动机（ DFA ）与 非确定性有限自动机（ NFA ）转换

	a	b
{1, 3}	{1, 3}	{2}
{2}	{2,3}	{3}
{2,3}	{1,2,3}	{3}
{3}	{1,3}	{ ∅ }

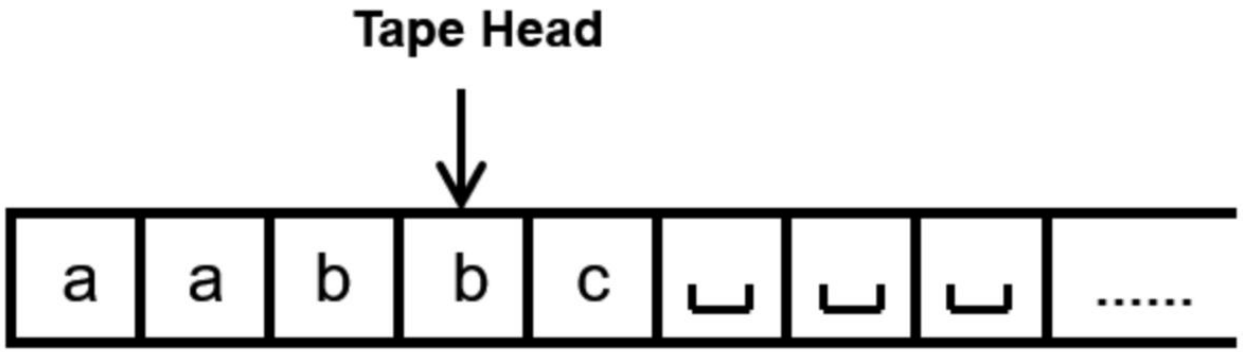


新状态

	a	b
{1, 3}	{1, 3}	{2}
{2}	{2,3}	{3}
{2,3}	{1,2,3}	{3}
{3}	{1,3}	{ ∅ }
{1,2,3}	{1,2,3}	{2,3}

图灵机

图灵机是一种自动机，图灵机会在状态转换过程中操作一个无限的tape，tape的样子如下图所示，tape里面包含字符，其中后面那个两竖一横的符号是blank的意思，它是一个特殊的字符，用来填满无限的tape



操作行为类似于有限状态机

有初始状态和终止状态有两种：accept state 和reject state

- 可以对tape head指向的字符进行读/扫描操作
- 可以对tape head指向的字符进行写/更新操作
- 将tape head左移一格
- 将tape head右移一格

计算结束后状态会变成下列三种中的一种：

- 终止并且接受(accept)
- 终止并且拒绝(reject)
- 循环(loop，就是说没有终止)

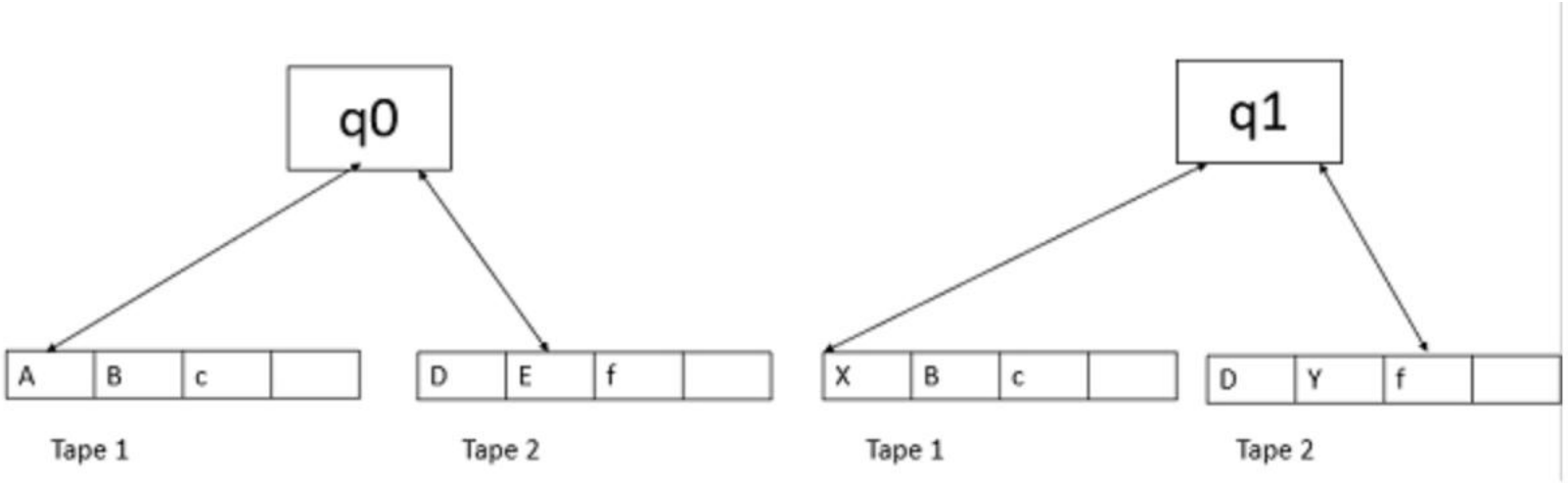
图灵机

图灵机可以表示为一个七元组(S , Σ , Γ , δ , S_0 , b , F)

S 是一组状态的集合
Σ 是输入字母表
Γ 是tape的字母表
δ 是转换函数
S_0 是初始状态
b 是空符号
F 是终止状态的集合(包括accept或reject)

多带图灵机

多个带子的图灵机 指的是 图灵机不止一个带子，如下图所示



多磁带图灵机有多个磁带，每个磁带都有一个单独的磁头。每个头可以独立于其他头移动。最初输入在磁带1上，其他的是空白的。首先，第一个磁带被输入占用，其他磁带保持空白。接下来，机器在其头部下读取连续的符号，TM在每个纸带上打印一个符号并移动它的头部。

P问题、NP问题、NPC问题、NP-hard问题

1. P 问题（P Class Problems）

定义：

P问题是指可以在多项式时间内通过一个确定性算法求解的问题。

对于一个问题，如果存在一个算法可以在输入规模增加时，其运行时间随着输入规模的增大以多项式级别增长（例如，O(n), O(n^2), O(n^3)等），那么这个问题就属于 P 类问题

特点：

可在多项式时间内解决：P 问题的解法非常高效。即使问题规模增大，算法的运行时间也不会增长得太快，通常是合理可接受的。

属于计算问题的“简单”类别：与更为复杂的 NP 或 NP-hard 问题相比，P 问题的解法是直接且高效的

常见的 P 问题示例：

排序问题：比如归并排序、快速排序、堆排序等，都可以在多项式时间内解决。

查找问题：如在已排序数组中查找元素（使用二分查找算法，时间复杂度是 O(logn)）。

图的遍历：如深度优先搜索（DFS）和广度优先搜索（BFS），它们可以在 O(V+E)的时间复杂度内完成（其中 V是图中的顶点数，EEE 是图中的边数）。

P问题、NP问题、NPC问题、NP-hard问题

2. NP 问题（NP Class Problems）

定义：

NP问题（Non-deterministic Polynomial time）是指可以在多项式时间内验证解的正确性的决策问题

虽然某些 NP 问题的解可能非常难找到，但一旦给出一个解（猜测的解），我们可以在多项式时间内验证这个解是否正确。

特点：

验证解的正确性高效：对于一个 NP 问题，如果给定一个候选解，我们可以迅速验证该解是否正确，验证过程可以在多项式时间内完成。

未知解法的复杂性：尽管能够验证解的正确性，但找到解本身可能非常困难，尤其是当问题规模变大时。

常见的问题：

旅行商问题（TSP）：给定一组城市，找出一条最短路径，使得每个城市都被访问一次并且最终回到起点。

0-1 背包问题：给定一组物品，每个物品都有一个重量和价值，背包有一个容量限制，目标是选择一些物品，使得它们的总价值最大且总重量不超过背包的容量。

图着色问题：给定一个图，要求用最少的颜色给图的每个节点着色，确保相邻的节点颜色不同。

P问题、NP问题、NPC问题、NP-hard问题

3. NP-Complete（NPC）问题

定义：
NP-Complete（NPC）问题是 NP 类问题中的最难的子集，这些问题不仅是 NP 问题，而且是NP-hard的。NPC 问题是 NP 类中最“难”解决的问题，并且具有以下两个特征：
属于 NP 类：即解可以在多项式时间内验证。
所有 NP 问题都可以通过多项式时间归约到该问题

特点：
NP-Complete 问题是 NP 类问题中最难的问题，因为它们不仅需要能验证解，还能够与所有其他 NP 问题互相转换。
如果解决了一个 NPC 问题的多项式时间算法，那么所有 NP 问题都能在多项式时间内解决。这就是著名的 P = NP 问题，如果我们找到了一个 NP-complete 问题的多项式时间算法，就等于解决了整个 NP 类问题。

常见的 NP-complete 问题：
旅行商问题（TSP）是 NP-complete 问题的一种常见形式。
3-SAT 问题：给定一个布尔公式，判断是否存在一个变量赋值使得公式为真。
图着色问题：判断是否能用 kkk 种颜色给图着色，使得相邻的节点颜色不同，是一个典型的 NP-complete 问题。

P问题、NP问题、NPC问题、NP-hard问题

4. NP-hard 问题

定义：

所有 NP 问题都可以多项式时间归约到 NP-hard 问题，但 NP-hard 问题不一定属于 NP 类，即它们的解不一定能在多项式时间内验证。NP-hard 问题可能连解的验证过程也非常复杂，甚至是不可验证的。

更广泛的类别：NP-hard 问题包括所有 NP-complete 问题，但它的定义更为广泛，既可以是 NP 类问题，也可以是无法在多项式时间内验证解的更复杂问题。

不一定属于 NP 类：例如，一些 NP-hard 问题的解可能无法在多项式时间内验证，这意味着它们不属于 NP 类。

常见的 问题：

哈密尔顿回路问题（Hamiltonian Cycle Problem）：给定一个图，判断是否存在一条回路，遍历每个节点且仅遍历一次，最终回到起点。

整数线性规划（Integer Linear Programming）：给定一个线性方程组，求解其中的整数解。

所有 NP-complete 问题：因为每个 NP-complete 问题都可以归约为 NP-hard 问题。