浅谈有限状态自动机及其应用

杭州学军中学 徐哲安

摘 要

有限状态自动机是目前信息学竞赛中相对冷门的内容。随着时代的进步，有限状态自动机相关的内容越来越多地作为难题出现在各类算法竞赛中。此外，学习有限状态自动机相关的内容有助于我们深入理解某些问题与算法的本质。本文系统地简述了有限自动机的基础理论及算法，引出了一个正则表达式匹配的高效算法，以及有限状态自动机在信息学竞赛中的丰富应用。希望本文能推动有限状态自动机在信息学竞赛界的普及。

1 引言

计算理论是理论计算机科学的分支，也是一门本科生课程。计算理论研究在某种计算模型下，怎样的问题是可解决的，通过算法能多少有效地解决这个问题。目的是回答：计算机的基本能力和局限是什么？

有限状态自动机作为一种基础的计算模型，因其结构简单，相关理论成熟的特点，不仅在现实生活中有广泛的应用，而且更能协助我们解决各类信息学竞赛中的问题，具有广泛的应用前景。

但令人遗憾的是，目前在信息学竞赛界缺乏相关的系统的学习资料，目前仅有的资料是一些信息学竞赛活动中的讲课课件 [5, 6, 7]。由此，作者对有限状态自动机进行了深入的探究，将相关的理论加以梳理，并总结了有限状态自动机在信息学竞赛中的丰富应用，写作本文。

本文共由六个章节组成。

第二节主要介绍了两类有限状态自动机——确定性有限状态自动机与非确定性有限状态自动机，与正则语言。本节揭示了两类有限状态自动机在计算能力上的等价性，以及它们之间的不同之处，并介绍了它们计算的算法，提出了具有启发意义的优化方法。

第三节主要介绍了正则语言的另一种表达——正则表达式，以及正则语言具有的大量性质。

第四节主要介绍了确定性有限状态自动机的最小化，揭示了正则语言与确定性有限状态自动机更加本质的关系。本节也介绍了用于确定性有限状态自动机最小化的高效算法——Hopcroft 算法。

第五节主要介绍了有限状态自动机的相关理论在信息学竞赛中的应用。本节内容是作者深入探究有限状态自动机的相关理论，并对各类算法竞赛中的相关试题加以研究后整理得到的。作者引出了一个正则表达式匹配的高效算法，并提出了一些个人的见解。

第六节作简要的总结。

2 有限状态自动机与正则语言

2.1 确定性有限状态自动机

有限状态自动机（*Finite State Machine*, FSM）是最简单的一类计算模型，体现在它的描述能力与资源都极其有限。我们首先给出确定性有限状态自动机的形式化定义。

定义 2.1 **确定性有限状态自动机**（*Deterministic Finite Automaton*, DFA）是一个五元组 $(Q, \Sigma, \delta, q_0, F)$，其中

- Q 是一个有限**状态集合**；
- Σ 是一个有限**字符集**；
- $\delta : Q \times \Sigma \to Q$ 是**转移函数**；
- $q_0 \in Q$ 是**开始状态**；
- $F \subset Q$ 是**接受状态集合**。

我们可以使用**状态图**来直观地描述一个 DFA。

要让 DFA 发挥作用，我们也需要给出另一些定义。

定义 2.2 设 $M = (Q, \Sigma, \delta, q_0, F)$ 是一个 DFA，$w = w_1 w_2 \cdots w_n$ $(w_i \in \Sigma)$（也记作 $w \in \Sigma^*$）是一个串，若存在 Q 中的状态序列 $r_0, r_1, \cdots, r_n$ 满足

- $r_0 = q_0$；
- $\delta(r_i, w_{i+1}) = r_{i+1}, \quad i \in \{0, 1, \cdots, n-1\}$；
- $r_n \in F$。

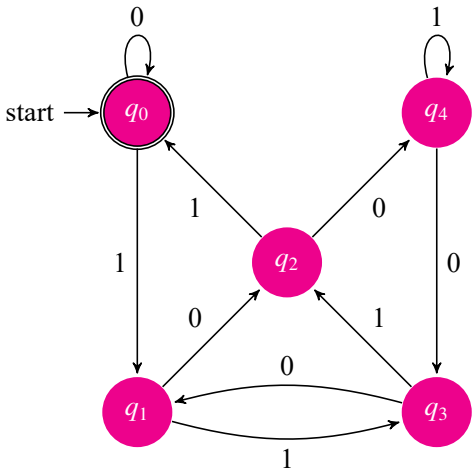
则称 M **接受** w。

定义**形式语言**（简称**语言**）是任意 Σ 上串的集合（集合大小可以为无限）。令语言 $L(M) = \{w \mid M \text{ 接受 } w\}$，则称 M **识别** $L(M)$。

如果一个语言能被某个 DFA 识别，则称它为**正则语言**（*Regular Language*）。

为了方便说明，我们将求出输入串 w 在 DFA 中的状态序列，并判断其是否被接受的过程称为**计算**。

考虑这样一个例子：设计一个 DFA 能识别 5 的倍数的二进制串。注意到，我们只需要考虑已读入的串模 5 后的余数，由此构造状态 q_0, q_1, q_2, q_3, q_4。若当前在状态 q_r 读入字符 c 后，转移到状态 $q_{(2r+c) \bmod 5}$ 即可。此外，开始状态和接受状态均为 q_0。



2.2 非确定性有限状态自动机

非确定性是确定性的自然推广，在非确定性机中，任何一个点和某个转移字符可能存在多个后继。下面我们用 $\mathcal{P}(Q)$ 表示 Q 的幂集（所有子集的集合），令 $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$（$\epsilon$ 表示空串，即在串中能任意加删），并给出非确定性有限状态自动机的形式化定义。

定义 2.3 **非确定性有限状态自动机**（*Nondeterministic Finite Automaton*, NFA）是一个五元组 $(Q, \Sigma, \delta, q_0, F)$，其中

- Q 是一个有限**状态集合**；
- Σ 是一个有限**字符集**；
- $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ 是**转移函数**；
- $q_0 \in Q$ 是**开始状态**；
- $F \subset Q$ 是**接受状态集合**。

我们同样给出 NFA 计算的形式化定义，需要注意，只要最终任何一个状态是接受状态，整个输入串就会被接受。

定义 2.4 设 $N = (Q, \Sigma, \delta, q_0, F)$ 是一个 NFA，串 w 可以被表示为序列 $y_1 y_2 \cdots y_m$ ($y_i \in \Sigma_\epsilon$)，若存在 Q 中的状态序列 $r_0, r_1, \cdots, r_m$ 满足

- $r_0 = q_0$；
- $r_{i+1} \in \delta(r_i, w_{i+1}), \quad i \in \{0, 1, \cdots, m-1\}$；
- $r_m \in F$。

则 N **接受** w。

NFA 是 DFA 的扩展，似乎一个可能的推论是 NFA 比 DFA 能力强，能识别更多的语言类？

2.3 DFA 与 NFA 的等价性

实则不然，下面我们将说明 DFA 与 NFA 的计算能力等价性。

定理 2.1 每一个 NFA 都等价于某一个 DFA，反之亦然。两个机器等价，即它们能识别的语言类相同。

证明. 显然每个 DFA 都可以直接转换为一个 NFA，考虑将 NFA 转化为一个模拟它的 DFA。我们使用一种被称作**幂集构造**（*Powerset construction*）的方法，下面给出形式化描述。

设 $N = (Q, \Sigma, \delta, q_0, F)$，定义 $E(q)$ 表示从状态 q 出发，只沿 ϵ 转移能到达的状态集合。

我们构造 $M = (Q', \Sigma, \delta', E(q_0), F')$，其中

- 状态集合：$Q' = \mathcal{P}(Q)$；
- 转移函数：$\delta' : Q' \times \Sigma \to Q'$；
- 其中 $\delta'(S, c) = \bigcup_{q \in S, q' \in \delta(q,c)} E(q')$；
- 终止状态集合：$F' = \{S \subset Q \mid S \cap F \neq \emptyset\}$。

显然计算的每一步，M 所在的状态对应 N 所处的状态集合。 □

鉴于 DFA 与 NFA 的计算能力等价性，我们可以得到一个简单推论。

推论 2.1 一个语言是正则的，当且仅当存在一个 NFA 能识别它。

虽然 NFA 与 DFA 能力相同，但我们认为 NFA 是有用的。这是因为对于某些正则语言，用 NFA 表示所需的状态数远小于 DFA 所需的状态数。而且，我们也不难构造出一个状态数为 n 的 NFA 使得它对应的最小 DFA 状态数是 $\Theta(2^n)$ 的。

2.4 DFA 与 NFA 的计算

两类 FSM 的差异还体现在计算一个给定串的时间复杂度上。接下来，我们设串长为 n，FSM 状态数为 s，字符集大小为常数。

由于状态和转移的唯一性，显然 DFA 计算这个串的时间复杂度为 $O(n)$。为了方便处理 NFA 的转移，我们可以先将 ϵ 转移消除。在 NFA 的计算过程中，最坏可能包含所有 s 个状态，同时每个状态可能至多存在 s 个后继，所以其时间复杂度为 $O(ns^2)$。接下来，我们将给出两个有用的优化。

一是通过 bitset 来优化复杂度。具体来说，预处理出每个状态 u 沿 c 转移边会到达的集合，NFA 计算过程中求出新的状态集合就只需要将这些集合或起来即可。时间复杂度被优化为 $O(\frac{ns^2}{\omega})$。

二是使用 **Method of Four Russians**。将 NFA 的状态分块，设块大小为 T，全体状态被分成了 $O(\frac{s}{T})$ 块。对于每个块，我们枚举 2^T 种状态子集，再枚举所有转移 c，处理出这个集合沿 c 转移边所形成的新的状态集合。在 NFA 计算过程中，我们只需要利用每个块内预处理的结果就能求出新的状态集合。时间复杂度为 $O(\frac{s^2\cdot 2^T}{\omega\cdot T}+\frac{ns^2}{\omega\cdot T})$，取 $T=O(\log n)$ 即可做到 $O(\frac{ns^2}{\omega\cdot\log n})$ 的时间复杂度了。

3 正则表达式与正则语言

3.1 正则表达式

正则表达式是另一种常用的正则语言表达，下面给出形式化定义。

定义 3.1 对于一个**正则表达式**（*Regular Expression*）R，称 $L(R)$ 为正则表达式 R 对应的形式语言。正则表达式 R 可以是

1. c $(c\in\Sigma)$，表示语言 $L(R)=\{c\}$；
2. ϵ，表示语言 $L(R)=\{\epsilon\}$；
3. $\emptyset$，表示语言 $L(R)$ 为空语言；
4. (R_1+R_2)，表示语言 $L(R)=L(R_1)\cup L(R_2)$；
5. (R_1R_2)，表示语言 $L(R)=\{uv\mid u\in L(R_1),v\in L(R_2)\}$；
6. (R_1^*)，表示语言 $L(R)=\{u_1u_2\cdots u_n\mid u_i\in L(R_1),n\in\mathbb{N}^+\}\cup\{\epsilon\}$。

3.2 正则表达式与 FSM 的等价性

看上去正则表达式与 FSM 相当地不同，但是，它们所能表述的语言类是相同的。

定理 3.1 一个语言是正则的，当且仅当可以用正则表达式表示它。

这个定理有两个方向，我们可以写作两个引理分别加以证明。

引理 3.1 如果一个语言可以用正则表达式描述，那么它是正则的。

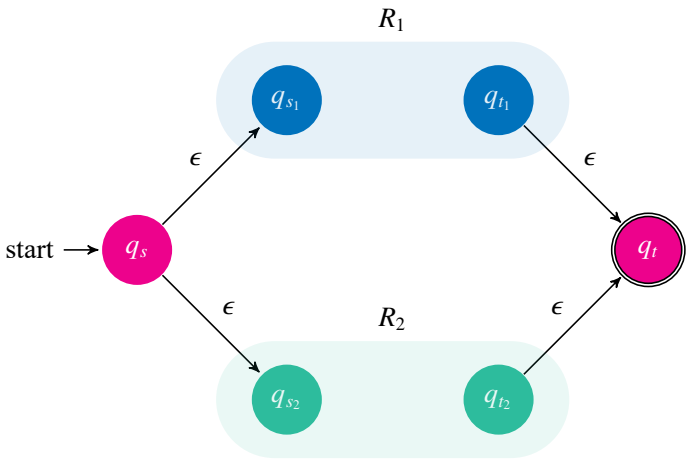
证明. 发现我们只需要将一个正则表达式转化为一个 NFA 即可。我们可以使用 **Thompson 构造法**（*Thompson's construction*），根据正则表达式 R 的构成，有如下六种情况：

1. 若 $R = c\ (c \in \Sigma)$，

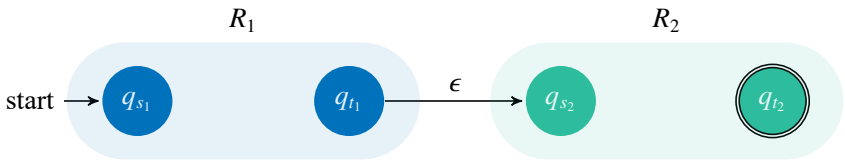
2. 若 $R = \epsilon$，

3. 若 $R = \emptyset$，

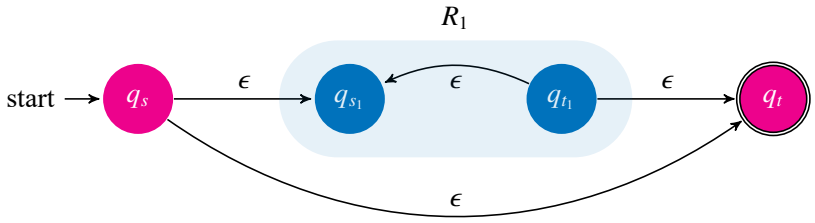
4. 若 $R = (R_1 + R_2)$，



5. 若 $R = (R_1 R_2)$，



6. 若 $R = (R_1^*)$，



□

引理 3.2 如果一个语言是正则的，那么可以用正则表达式描述它。

证明. 显然我们只需要找到一种将一个 DFA 转化为正则表达式的方法，首先引入一种新型 FSM。

定义 3.2 广义非确定性有限状态自动机（*Generalized Nondeterministic Finite Automaton*, GNFA）是一个五元组 $(Q, \Sigma, \delta, q_s, q_t)$，其中

1. Q 是一个有限**状态集合**；
2. Σ 是一个有限**字符集**；
3. $\delta : (Q \backslash \{q_t\}) \times (Q \backslash \{q_s\}) \to \mathcal{R}$ 是**转移函数**，即每条边上是一个正则表达式；
4. $q_s \in Q$ 是**开始状态**；
5. $q_t \in Q$ 是**接受状态**。

如果串 w 可写作 $w = w_1 w_2 \cdots w_k$ $(w_i \in \Sigma^*)$，且存在状态序列 $q_0, q_1, \cdots, q_k$ 使得

1. $q_0 = q_s$ 是开始状态；
2. $q_k = q_t$ 是接受状态；
3. $\forall i = 1, 2, \cdots, k,\ w_i \in L(\delta(q_{i-1}, q_i))$。

则称这个 GNFA **接受**串 w。

我们可以通过增加开始状态，接受状态，以及一些必要的转移箭头，使得一个 DFA 转化为等价的 GNFA。

假设当前的 GNFA 存在 k 个状态，且 $k > 2$。我们任取一状态 $q_r \in Q \backslash \{q_s\} \backslash \{q_t\}$，并构造一个新的 GNFA $(Q', \Sigma, \delta', q_s, q_t)$。其中 $Q' = Q \backslash \{q_r\}$，对于任意 $q_i \in Q' \backslash \{q_t\}, q_j \in Q' \backslash \{q_s\}$，我们令

$$\delta'(q_i, q_j) = ((\delta(q_i, q_r)\delta(q_r, q_r)^*\delta(q_r, q_j)) + (\delta(q_i, q_j)))$$

考虑串在原来 GNFA 中的路径会经过若干次 q_r，我们通过构造使得它被转移边等效替代了。所以新的 GNFA 与原来的 GNFA 等价，并且状态数恰好减一。

通过不断减少 GNFA 的状态数，我们能使得最终的 GNFA 状态数为 2，此时 q_s 到 q_t 转移边上的正则表达式就是我们的目标。

□

建立起了正则表达式与正则语言之间的联系，我们就可以从代数视角考虑正则语言运算的性质。

3.3 正则语言的代数定律

在本小节中，我们不考虑具体的正则表达式，转而考虑以变量为参数的正则表达式（变量可以为任意正则语言）。运用正则表达式的代数定律有助于化简正则表达式。

定理 3.2 *以下正则表达式的代数定律成立：*

1. *并的交换律：* $L + M = M + L$;
2. *并的结合律：* $(L + M) + N = L + (M + N)$;
3. *连接的结合律：* $(LM)N = L(MN)$;
4. $\emptyset$ *是并运算的单位元：* $\emptyset + L = L + \emptyset = L$;
5. ϵ *是连接运算的单位元：* $\epsilon L = L\epsilon = L$;
6. $\emptyset$ *是连接运算的零元：* $\emptyset L = L\emptyset = \emptyset$;
7. *分配律：* $L(M + N) = LM + LN, (M + N)L = ML + NL$;
8. *并的幂等律：* $L + L = L$;
9. *闭包相关的定律：* $(L^*)^* = L^*, \emptyset^* = \epsilon, \epsilon^* = \epsilon$。

以上定律的证明均较为简单，故在此略去。

由此延伸出这样一个问题：我们如何判断一个正则表达式的“定律”是否成立？

接下来给出了一个一般性的方法，但非常有趣的是，它紧密依赖于正则表达式的运算符性质，不能推广到含某些其他运算符（例如交运算）的表达式。我们给出一个重要引理：

引理 3.3 设 E 是包含变量 $L_1, L_2, \cdots, L_m$ 的正则表达式。将所有 L_i 替换为标志符 a_i，得到**形式正则表达式** $T(E)$。

令 $L_1 = L_1', L_2 = L_2', \cdots, L_m = L_m'$（注意 L_i 是变量，而 L_i' 是具体的语言），得到具体的正则表达式 E'。我们可以按如下方式构造全体 $L(E')$：对于任意 $A = a_{j_1}a_{j_2}\cdots a_{j_k} \in L(T(E))$，我们将 a_{j_t} 替换为 L_{j_t}' 中的任意元素，并将所有得到的串加入到 $L(E')$，记这些串的集合为 $w(A)$。

证明. 考虑对正则表达式进行归纳，不包含运算符的基础情况是显然的。

- 若 $E = F+G$，根据定义有 $T(E) = T(F)+T(G)$。对于任意串 $w \in w(A)$ $(w(A) \subset L(E'))$，存在对应的 $A \in L(T(E))$ 有 $A \in L(T(F))$ 或者 $A \in L(T(G))$，也即 $w \in L(F')$ 或者 $w \in L(G')$。

  反之，对于 $w \in L(F')$，存在对应的 $A \in L(T(F))$，那么 $A \in L(T(E))$ 即 $w \in L(E')$。

- 若 $E = FG$，根据定义有 $T(E) = T(F)T(G)$。对于任意串 $w = uv,\ u \in w(A), v \in w(B)$ $(w(A) \subset L(F'), w(B) \subset L(G'))$，它们存在对应的 $A \in L(T(F)), B \in L(T(G))$ 有 $u \in L(F'), v \in L(G')$。

  反之，对于 $u \in L(F'), v \in L(G')$，存在对应的 $A \in L(T(F)), B \in L(T(G))$，那么 $AB \in L(T(E))$ 即 $uv \in L(E')$。

- 若 $E = F^*$，其证明也是类似的，这里不再赘述。

□

如果扩展正则表达式引入交运算，可以发现上述的论证不再成立，也就不再适用接下来的定理。这里举出一个简单的例子，对于含变量的正则表达式 $E = L_1 \cap L_2$，有 $T(E) = a_1 \cap a_2 = \emptyset$。而如果我们令正则语言 $L_1' = \{a, b\}, L_2' = \{a, c\}$，带入 $L_1 = L_1', L_2 = L_2'$，得到 $L(E') = \{a\}$，而使用上述过程会得到空语言。

定理 3.3 对于一对带有相同变量集合的正则表达式 E 和 F。将每个变量替换为一个标识符，得到 $T(E), T(F)$，那么 $E = F$ 当且仅当 $L(T(E)) = L(T(F))$。

证明. 我们要证明：对于任意替换变量为具体语言的方案，均有 $L(E') = L(F')$ 当且仅当 $L(T(E)) = L(T(F))$。

假设对于所有替换选择，均有 $L(E') = L(F')$。那么替换为标识符也是一种替换选择，令 $E = T(E), F = T(F)$，即 $L(T(E)) = L(T(F))$。

假设 $L(T(E)) = L(T(F))$，根据**引理 3.3**，将 $L(T(E)), L(T(F))$ 中的标识符替换为对应的语言，就构造出了 $L(E')$ 和 $L(F')$，显然它们是相等的。 □

定理 3.3 为我们提供了一种判断两个含变量的正则表达式是否相同的方法。只需要不断将同一变量替换为未曾出现过的字符，将它们变成具体的正则表达式，再判断两个具体的正则表达式对应的语言是否相同。这个问题我们将在后续的章节中解决。

3.4 正则语言的封闭性

正则语言的封闭性是其重要的性质，这些性质使得我们能通过一定的运算，构造能够识别另一些语言的 FSM。简而言之，封闭性可以作为构造复杂 FSM 的工具。

定理 3.4 *关于正则语言的封闭性，我们有：*

1. *两个正则语言的并是正则的；*
2. *两个正则语言的连接是正则的；*
3. *正则语言的闭包是正则的；*
4. *正则语言的补是正则的；*
5. *两个正则语言的交是正则的；*
6. *两个正则语言的差是正则的；*
7. *正则语言的反转是正则的；*
8. *正则语言的同态（相同串替代符号）是正则的；*
9. *正则语言的逆同态是正则的。*

这里简单说明这些性质的正确性。前三条性质就是正则表达式的运算符；性质四只需要将接受状态取作补集；性质五六可以构造两个 DFA 的笛卡尔积得到；性质七可以认为是反转所有转移边的方向，构造 NFA 得到。性质八九的证明较为繁琐，以上这些性质的详细证明都能在 [1] 中找到，此处略去。

3.5 正则语言的泵引理

引理 3.4 **正则语言的泵引理** *设 L 是正则语言，存在与 L 相关的常数 n 满足：对于任意 L 中的串 w，如果 $|w| \geq n$，则我们能将 w 表示为 xyz，满足：*

1. $y \neq \epsilon$；
2. $|xy| \leq n$；

3. $\forall k \geq 0,\ xy^k z \in L$。

证明. 对于任意正则语言 L，存在某个 DFA 使得 $L = L(A)$。假设 A 有 n 个状态，考虑长度不小于 n 的串 $w = a_1 a_2 \cdots a_m$ 其中 $m \geq n$ 且 a_i 均为输入符号。对于 $i = 0, 1, \cdots, n$ 定义 p_i 为读入 w 前 i 个字符后所处的状态。

根据抽屉原理，必然存在 $0 \leq i < j \leq n$ 使得 $p_i = p_j$，我们构造

1. $x = a_1 a_2 \cdots a_i$;

2. $y = a_{i+1} a_{i+2} \cdots a_j$;

3. $z = a_{j+1} a_{j+2} \cdots a_m$;

发现串 y 对应的路径构成了一个环，如果我们在环上走 k 次，就能构造出串 $xy^k z$。 □

泵引理描述了正则语言一个共有的基本属性。对于一个语言，如果我们能找到一个足够长的串使其不满足泵引理，就足以说明这不是正则语言。泵引理是我们证明一个语言的非正则性的有力工具。不过需要注意的是，泵引理的逆命题并不成立，也即存在某些非正则语言满足泵引理。

4 DFA 的等价类与最小化

我们定义两个 DFA 等价当且仅当它们识别相同的正则语言，全体 DFA 被划分为了无穷多个等价类。本节介绍了一些通用的算法，来寻找某个 DFA 所属等价类中的最小 DFA。

4.1 DFA 的最小化

定义 4.1 对于 DFA 的两个状态 p, q，我们定义关系 $p \sim q$ 当且仅当：对于任意输入串 $w = w_1 w_2 \cdots w_k$ $(k \geq 0)$，$\hat{\delta}(p, w) = \delta(\cdots \delta(\delta(p, w_1), w_2) \cdots, w_k)$ 是接受状态当且仅当 $\hat{\delta}(q, w)$ 是接受状态。

从直观上理解，关系 $p \sim q$ 相当于不存在输入串 w 能区分状态 p 和 q。稍加观察不难发现，关系 $\sim$ 构成了一个**等价关系**（自反性与对称性显然，传递性易反证得到）。也就是说对于一个特定的 DFA，等价关系 $\sim$ 将 DFA 的状态划分为若干个等价类。

剔除初始状态无法到达的状态后，假设我们得到了 DFA A 的等价类 $S_1, S_2, \cdots, S_m$，考虑构造一个新的 DFA B。我们将 A 中的等价类 S_i 作为 B 中的状态 i，那么 B 恰好包含 m 个状态一一对应 A 中的等价类。对于等价类 S_i，若存在状态 $u \in S_i$ 和转移 c，使得 $\delta_A(u, c) \in S_j$，则对于任意状态 $v \in S_i$ 均有 $\delta_A(v, c) \in S_j$（使用反证法易得）。那么，我们不妨令 B 中的转移 $\delta_B(i, c) = j$。显然，对于 S_i 中的所有状态，要么均为接受状态，要么均不是接受状态。若 S_i 中的状态均为接受状态，我们令 B 中的状态 i 为接受状态，否则不是接受状态。最后，我们

令包含 A 的初始状态的等价类 S_i 对应的状态 i 为 B 的初始状态。不难验证，我们构造出的 DFA B 与 DFA A 等价。

那么，我们剩下的问题就是找出一个 DFA A 所有的等价类。

接下来的算法基于不断对等价类进行划分。我们扩展等价关系 $\sim$ 的定义，令 $p \sim_k q$ 表示对于任意长度 $\leq k$ 的串 w，均有 $\hat{\delta}(p,w)$ 是接受状态当且仅当 $\hat{\delta}(q,w)$ 是接受状态。不难发现 $\sim_k$ 也是一个等价关系。由关系 $\sim_k$，我们的 DFA A 被划分成了等价类集合 Π_k。显然 $\Pi_0 = \{Q\backslash F, F\}$，我们也不难从 Π_k 推出 Π_{k+1}，具体算法如下：

Algorithm 1: 等价类划分算法

Input: DFA $A = (Q, \Sigma, \delta, q_0, F)$

Output: 等价类集合 $\Pi_0, \Pi_1, \Pi_2, \cdots$

```
Π_0 ← {Q\F, F};
m ← 0;
repeat
    Π_{m+1} ← Π_m;
    foreach c ∈ Σ do
        Π' ← Π_{m+1};
        foreach G ∈ Π_{m+1} ∧ |G| > 1 do
            if ∃ u, v ∈ G, δ(u, c) 与 δ(v, c) 属于 Π_m 的不同组 then
                根据转移后不同的组，对 G 进一步划分得到 G*;
                Π' ← Π'\{G} ∪ G*;
            end
        end
        Π_{m+1} ← Π';
    end
    m ← m + 1;
until Π_m = Π_{m−1};
```

上述算法依次求出了等价类集合 $\Pi_0, \Pi_1, \cdots, \Pi_m$，其中 $\Pi_{m-1} = \Pi_m$。由于划分方案仅依赖于前一个等价类集合，我们断言 $\Pi_{m-1} = \Pi_m = \Pi_{m+1} = \Pi_{m+2} = \cdots$，根据定义，$\Pi_m$ 就是我们所求的等价类集合。值得一提的是，如果我们稍加修改上述算法，每次找出**任意**一个能被分裂的集合并不断迭代，不难证明这同样也是正确（首先证明等价的一对状态不会被分开；再考虑最小的 k 使得存在一对处于同一等价类中且不满足 $\sim_k$ 的状态，一定能继续划分）。

在具体实现的时候，对于每种转移 c，我们遍历所有等价类集合，再遍历其中的元素。我们使用一个 Hash 表来维护当前等价类中的 u 对应的 $\delta(u,c)$ 所在 Π_m 中的等价类，据此来继续划分集合。该算法的时间复杂度为 $O(n^2|\Sigma|)$。值得注意的是，根据 DFA 随机生成的方式不同，该算法的平均迭代次数能达到 $O(\log n)$ 甚至更低。这是一个实践中非常优秀的算法。

4.2 Myhill-Nerode 定理

给定一个语言 L，对于任意一对串 x, y，我们定义关系 $x \equiv_L y$ 当且仅当：对于任意串 z，$xz \in L$ 当且仅当 $yz \in L$。不难验证这是一个等价关系，也即 Σ^* 被划分为了若干等价类。我们有定理：

定理 4.1 **Myhill-Nerode theorem** *语言 L 是正则的，当且仅当关系 $\equiv_L$ 对应的等价类个数有限。描述正则语言 L 的最小 DFA 唯一（忽略状态标号），且它的状态数量等于关系 $\equiv_L$ 对应的等价类数量。*

该定理包含两个方向，那么我们同样分作两个引理分别证明。

引理 4.1 *对于任意正则语言 L，都有关系 $\equiv_L$ 对应的等价类个数有限，且不超过能识别 L 的最小 DFA 状态数。*

证明. 取一个识别语言 L 的最小 DFA A，假设关系 $\equiv_L$ 对应的等价类个数大于 A 的状态数。根据抽屉原理，必然存在不属于同一个等价类的串 x, y 会到达 A 中的同一个状态，也即 $x \equiv_L y$，矛盾。 □

引理 4.2 *对于任意语言 L，关系 $\equiv_L$ 对应的等价类个数有限，我们都能构造唯一的 DFA 使得其状态数等于等价类个数。*

证明. 考虑构造一个 DFA A 使得每个等价类恰好对应一个状态。DFA 的开始状态对应空串所属的等价类，DFA 的接受状态对应全体元素属于 L 的等价类。对于每个等价类和字符 c，任意选择一个代表元求出沿 c 转移边到达的等价类，在这对等价类对应的状态之间加入 c 转移边。不难验证这个 DFA 识别语言 L。

假设存在另一个与 A 不同构的 DFA B 识别 L 且它的大小不超过等价类个数。那么，必然存在不属于同一个等价类的串 x, y 会到达 B 中的同一个状态，也即 $x \equiv_L y$，矛盾。 □

显然最小化后的 DFA 的状态一一对应等价类，所以最小化后的 DFA 就是最小 DFA。

Myhill-Nerode 定理也为我们提供了新的判断一个语言非正则性的方法，显然有推论

推论 4.1 *对于一个语言 L，若存在任意多个串 $w_1, w_2, \cdots$ 使得它们两两不等价，那么 L 不是正则语言。*

由于最小 DFA 的唯一性，我们也找到了判断两个 DFA 是否等价的方法。只需将两个 DFA 分别最小化，判断是否同构即可。

4.3 Hopcroft 算法

Hopcroft 算法 是一个寻找 DFA 等价类更加高效的算法 [3]，基于**启发式分裂**来优化时间复杂度。算法描述如下：

Algorithm 2: Hopcroft's algorithm

Input: DFA $A = (Q, \Sigma, \delta, q_0, F)$
Output: 等价类集合 P

```
P ← {F, Q\F};
W ← {F};
while W ≠ ∅ do
    从 W 中任取一个集合 A，并将其从 W 中删去;
    foreach c ∈ Σ do
        X ← {u ∈ Q | δ(u, c) ∈ A};
        foreach Y ∈ {u ∈ P | u ∩ X ≠ ∅, u\X ≠ ∅} do
            P ← P\{Y} ∪ {Y ∩ X} ∪ {Y\X};
            if Y ∈ W then
                W ← W\{Y} ∪ {Y ∩ X} ∪ {Y\X}
            else
                if |Y ∩ X| ≤ |Y\X| then
                    W ← W ∪ {Y ∩ X};
                else
                    W ← W ∪ {Y\X};
                end
            end
        end
    end
end
```

如何证明上述算法的正确性？稍加修改后的等价类划分算法可以看做是以下过程的不断重复：选择等价类 Y, A 以及字符 c，求出能沿着 c 转移边到达 A 的状态集合 X，满足 $Y \cap X$ 与 $Y \backslash X$ 均非空，那么我们就能将等价类 Y 拆分为两个等价类。我们称等价类 A 是划分等价类 Y 的**证据**。Hopcroft 算法实际上维护了一个集合 W 表示还未用于划分等价类的证据。显然用证据 A 将现有的等价类划分完后，证据 A 就失去了意义。

在我们将等价类 Y 划分为等价类 U, V 之后有两种情况。一种是 Y 还未作为证据使用，显然 Y 的作用可以被 U 和 V 共同作用代替。另一种是 Y 已经作为证据使用，我们可以证明，仅保留 U 或者 V 作为证据使用均能得到正确的结果。假设存在等价类 Z 使得 U, V 中仅有 U 能作为证据将其继续划分。那么必然存在一对状态 $a, b \in Z$ 和转移 c 使得 $\delta(a, c) \in U, \delta(b, c) \notin U$。如果 $\delta(b, c) \notin Y$，那么 Y 作为证据一定会将 Z 继续划分，否则 $\delta(b, c) \in V$ 即 V 同样能作为划分 Z 的证据。矛盾，故结论成立。

接下来的问题是，如何高效实现该算法？不难发现，时间复杂度的瓶颈在于快速寻找到

与 X 有交的 Y。我们不妨遍历 X 中的所有元素，在其所属的等价类打上标记，最后将所有符合条件 Y 进一步处理。如果 $|Y\cap X|$ 较小，那么我们将 $Y\cap X$ 单独提出，并保留这些元素在原来的集合中（之后再次遍历到这个集合时就忽略这些元素，并重构整个集合），否则意味着遍历集合 Y 的时间复杂度上界为 $O(|X|)$。注意到时间复杂度依赖于所有 $|X|$ 与 $\min(|Y\cap X|, |Y\backslash X|)$ 之和，因为每个元素对时间复杂度产生贡献时，其所在等价类规模至少除以 2，故 Hopcroft 算法的时间复杂度为 $O(n|\Sigma|\log n)$。对于某些特定方式随机生成的 DFA，该算法的平均时间复杂度能达到 $O(n|\Sigma|\log\log n)$。

5 FSM 在 OI 中的应用

虽然 FSM 是 OI 中相对冷门的内容，但是学习 FSM 有助于我们深入了解一些问题的本质，以及提出更加优秀的算法。接下来，我们将略举几例来展现其丰富的应用。

5.1 DFA 的构造与设计

在 OI 中，我们有时会遇到这样的计数问题：“给定一个串，问其存在多少子串满足某条件”，或者“给定一个包含 0, 1, ? 的串，问存在多少种替换 ? 为 0 或者 1 的方案使其满足某条件”等等。如果满足条件的全体串构成了一个正则语言，我们往往能通过某种方式来构造一个 DFA，接着再通过 DP 解决。

为了解决这样的问题，我们需要设计一个 DFA 来识别特定的语言。没有什么机械的方法或者公式能帮助我们直接构造出一个 DFA，但是，在设计 DFA 的时候仍然有一般性的原则。

DFA 的局限在于它的状态数必须是有限的，而可能的输入种类却是无限的。这就需要我们用某些**关键信息**来描述状态，使得所有输入被划分为有限类。第二节提到的例子就是将已读入的数模 5 作为关键信息。

例题 5.1 Foreigner[1] 我们定义好数：

- 所有 1 至 9 的数是好的；
- 如果一个整数 $x \geq 10$ 是好的，它需要 $\lfloor x/10 \rfloor$ 是好的；除此之外，设 $\lfloor x/10 \rfloor$ 是第 k 个好数，那么 x 是好的仅当 $x \bmod 10 < k \bmod 11$。

前若干个好数为 $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 21, 30, 31, 32\cdots$

给定数字串 s，求有多少个 s 的子串不含前导 0 且是好数。

数据范围：$1 \leq |s| \leq 10^5$

[1] Codeforces Round #549 (Div. 1), Problem D

考虑构造一个 DFA，使之能识别全体好数构成的集合。对于一个好数 x，我们维护三个参数：a_x 表示小于等于 x 的好数个数，b_x 表示小于 x 且与 x 长度相同的好数个数，c_x 表示大于 x 且与 x 长度相同的好数个数。在 x 后追加一个数字得到 y，我们不难判断 y 是否是一个好数，同时容易得到新的 a_y, b_y, c_y。利用性质 $(0+1+\cdots+10) \bmod 11 = 0$，我们只需要知道 a_x, b_x, c_x 模 11 的值。由此，我们可以构造一个包含 11^3 个状态的 DFA，在这个 DFA 上 DP 就能解决本题。值得注意的是，这个 DFA 有很多状态是不可达的，删去不可达的状态可以大大提高程序的效率。

事实上，上述做法构造的 DFA 包含很多冗余、重复的状态，我们能通过进一步分析问题性质得到更优的解法。方便起见，我们认为 0 也是好数，同时它的排名为 0。观察好数序列发现从 10 开始所有的好数都是在更小好数的末尾添加 $0, 1, \cdots, 9$ 得到的，同时能添加上的数字个数是 $0, 1, \cdots, 10$ 循环的。假设好数 x 与 $\lfloor x/10 \rfloor$ 的排名分别为 u, v，那么有

$$u = 9 + 55\lfloor v/11 \rfloor + (0 + 1 + \cdots + (v-1) \bmod 11) + (x \bmod 10) + 1$$

发现我们只需要考虑模 11 的值，也即 $u \equiv v(v-1)/2 + 10 + (x \bmod 10) \pmod{11}$。我们可以构造一个仅包含 11 个状态的 DFA。之后使用 DP 计数即可，时间复杂度为 $O(11n)$。

5.1.1 DP 套 DP

DP 套 DP 是陈立杰于 WC 2015 提出的一类问题的解决方法 [9]。基本的问题形式是：问存在多少种特定的对象（例如串，序列，矩阵等）满足某个特定的条件。如果判断该对象是否符合条件可以通过 DP 判断，那么我们可以“通过一个外层的 DP 来计算使得另一个 DP 方程（子 DP）最终结果为特定值的输入数。由于我们只对 DP 方程的最终结果感兴趣，我们并不需要记录这前 i 位都是什么，只需要记录对这前 i 位进行转移以后，DP 方程关于每个状态的值就可以了。也即外层 DP 的状态是所有子 DP 的状态的值”。

DP 套 DP 的内层可以看做是记录“全体 DP 状态及值”的 DFA。对于某些此类问题，其内层的 DP 仅有有限种情况（或者可以通过一些方法合并为有限种情况），这时全体符合条件的串构成了正则语言，可以使用 DFA 描述。对于其他此类问题，其内层 DP 可能有无限种情况（也即等价类个数随串长的增长而增长），如果我们固定一个上界 N，仅考虑能正确识别长度不超过 N 的串，同样可以使用一个 DFA 来描述。

5.2 DFA 最小化：减少 DP 状态数

将 DP 套 DP 或者其它一些问题引入 DFA 的好处在于，我们可以使用有限状态自动机相关的理论来帮助我们解决问题。下面举出一个例子：

例题 5.2 Equanimous[2]

[2]XIX Open Cup Grand Prix of China, Problem E，另见 Codeforces Round #472 (Div. 1), Problem F

定义 $f(n)$ 表示将十进制数 n 所有数码之间填入加号或者减号，最终得到的值的绝对值最小值。

T 组询问，给定 l, r，对于所有 $k = 0, 1, \cdots, 9$，求所有 m 之和满足 $l \le m \le r$ 且 $f(m) = k$。答案对 $10^9 + 7$ 取模。

数据范围：$1 \le T \le 10^4, 1 \le l \le r \le 10^{100}$。

考虑构造一个 DFA 能对于给定的 m 计算 $f(m)$。令 $dp(i, j)$ 表示 m 的前 i 位插入加减号后能否变成绝对值为 j 的数。转移就是 $dp(i-1, j) \to dp(i, j + d_i), dp(i, |j - d_i|)$，其中 d_i 就是 m 的第 i 位数。不难发现，DP 过程中状态可能会达到 $9 \cdot \log m$ 的级别，时间复杂度无法承受。实际上，我们能设定一个阈值 K，使得我们不需要考虑 $j \ge K$ 的 DP 信息，同样能够得到正确的结果。为了得到更加一般化的结论，我们使用 w ($w \ge 2$) 替代题中的 9。

首先，我们需要用到以下引理

引理 5.1 对于任意两个大小为 n 的多重集，只包含元素 $1, 2, \cdots, n$，一定能各自找到一个子集，使得这两个子集之和相同。

证明. 假设两个集合分别为 $\{a_1, a_2, \cdots, a_n\}$ 与 $\{b_1, b_2, \cdots, b_n\}$。设 $A_i = \sum_{j=1}^{i} a_j$，$B_i = \sum_{j=1}^{i} b_j$。不失一般性，设 $A_n \le B_n$。

令 p_i 为满足 $B_{p_i} \ge A_i$ 的最小值。令 $R_i = B_{p_i} - A_i$，显然有 $R_i \in [0, n-1]$。

若存在 $R_i = 0$，那么子集 $a[1 \cdots i], b[1 \cdots p_i]$ 即符合条件。

否则根据抽屉原理，必然存在 $1 \le i < j \le n$ 使得 $R_i = R_j$。此时子集 $a[(i+1) \cdots j], b[(p_i + 1) \cdots p_j]$ 符合条件。 □

由此可以得到下述推论

推论 5.1 对于任意两个多重集 A, B，只包含元素 $1, 2, \cdots w$ 且元素之和 $\ge w(w-1)$，一定能各自找到一个非空子集，使得这两个子集之和相同。

证明. 不妨设 $|A| \le |B|$。对于满足上述条件的多重集 A, B，若元素个数不超过 $w-1$，一定只能是 $w-1$ 个 w。

若 A, B 均包含 $\ge w$ 个元素，那么根据**引理 5.1**，结论成立。

若 A, B 均只包含 $w-1$ 个元素，显然结论同样成立。

若 A 包含 $w-1$ 个 w，B 包含 $\ge w$ 个元素。任取 B 中的 w 个元素 $x_1, x_2, \cdots, x_w$，构造 $s_i = (\sum_{j=1}^{i} x_j) \bmod w$。若存在 $s_i = 0$，取 $x[1 \cdots i]$ 即可。否则必然存在 $1 \le i < j \le w$ 使得 $s_i = s_j$，取 $x[(i+1) \cdots j]$ 即可。结论同样成立。 □

接下来的定理给出了本题可行的阈值

定理 5.1 只保留上述 DP 在 $0 \cdots w^2 - 1$ 处的信息，同样能得到正确结果。

证明. 设 $m = \overline{d_1 d_2 \cdots d_k}$ ($0 \le d_i \le w$)，则上述定理等价于对于任意正整数 m，我们能找到序列 $s_0, s_1, \cdots, s_k$ 使得：

- $s_0 = 0$;
- $|s_i - s_{i-1}| = d_i$ $(i = 1, 2, \cdots, k)$;
- $|s_k| = f(m)$;
- $|s_i| \le w^2 - 1$ $(i = 1, 2, \cdots, k)$.

设存在最小的 p 使得 $|s_p| \ge w^2$，不失一般性，设 $s_p \ge w^2$。

显然我们有 $|s_k| = f(m) \le w$，即 $s_p - s_k \ge w(w-1)$。设 $d_{p+1}, d_{p+2}, \cdots, d_k$ 中取减号的所有元素构成集合 B，那么 B 中所有元素之和一定 $\ge w(w-1)$。

考虑 $d_1, d_2, \cdots, d_p$ 中取加号的元素（包括 d_1）依次为 $d_{t_1}, d_{t_2}, \cdots, d_{t_q}$。取最大的 l 满足 $\sum_{j=l}^{q} d_{t_j} \ge w(w-1)$。令集合 $A = \{d_{t_l}, d_{t_{l+1}}, \cdots, d_{t_q}\}$，集合 A 中的所有元素之和 $\in [w(w-1), w^2-1]$。

对于集合 A, B，我们应用**推论 5.1**，一定能找到 $U \in A, V \in B$ 使得它们的元素之和相同。

之后将所有 U 中元素对应的加号改成减号，将所有 V 中元素对应的减号改成加号。不难发现 $|s_k|$ 仍然保持不变，同时对于 $i = 1, 2, \cdots, p-1$ 仍然有 $|s_i| \le w^2 - 1$。此外 $|s_p|$ 一定会变小。

不断重复上述操作，此后一定会终止，最终的序列就是符合要求的了。

□

实际上，本题的阈值通过更加细致的分析能达到 $(9-1)\times 9 + 1 = 73$，到达上界的方式为构造连续 8 个 9 和 9 个 8。我们将状态看做一个长度为 73 的 bitset，并搜索出所有可达的状态，这样的状态数量只有数万个。我们将状态对应的 f 值分为 10 类，并运行 Hopcroft 算法，最终的最小 DFA 只有 715 个状态。通过 DP 预处理足够的信息，我们能 $O(10 \cdot (\log l + \log r))$ 回答一次询问。

5.3 利用等价关系构造 DFA

例题 5.3 Median Replace Hard[3]

给定一个长度为 8 的二进制串 $P = P_0 P_1 \cdots P_7$。定义一个长度为 n 的二进制串 X 是好的，当且仅当能够通过执行 $(n-1)/2$ 次下述操作变为串“1”：

- 选择 X 的连续三个比特 (X_i, X_{i+1}, X_{i+2})，将它们替换为 P 的第 $(X_i + 2X_{i+1} + 4X_{i+2})$ 个比特。

共 T 组询问。给定一个包含 $0, 1, ?$ 的串 S，问存在多少个将 $?$ 替换为 $0, 1$ 的方案，使得最后的串为好串。答案对 $10^9 + 7$ 取模。

数据范围：$1 \le T \le 256, 1 \le |S|, \sum |S| \le 300000$，$|S|$ 为奇数

[3] XX Open Cup Grand Prix of Tokyo, Problem J

实际上对于所有 256 个可能的 P，我们都能构造一个 DFA 识别全体好串。找到这样的 DFA 之后，我们就能够通过一个简单的 DP 解决本题。

回忆 Myhill–Nerode 定理中的等价关系，我们认为串 x, y 等价当且仅当对于任意的串 z，均有 xz 是好串当且仅当 yz 是好串。我们能根据等价类直接构造出 DFA。但是枚举所有的串 z 来判断是否等价是不可能的，考虑仅枚举长度不超过 L 的串 z 来判断等价性（同时通过记忆化 DFS 判断一个串是否为好串）。

不妨将所有等价类中任意长度最小的串视作代表元，我们可以使用 BFS 找出所有的代表元。先将空串入队，每次从队首弹出一个串 x，并判断串 x 和已有的代表元是否等价。如果均不等价，那么它一定能成为某个等价类的代表元。假设存在某个等价类的代表元没有被找到，同时该串去掉最后一个字符得到串 t，那么串 t 所在等价类的代表元加入这个字符就得到与之等价的串。观察这个过程，我们也能发现所有代表元形成了一个树形结构。

接下来的问题是我们怎样验证这个 DFA 是否符合条件？假设存在一个 DFA $d(k)$ 能正确识别长度不超过 k 的好串。据此可以构造出一个 NFA 能正确识别长度不超过 $k+2$ 的好串，再将其转化为 DFA $d(k+2)$，并最小化。如果 $d(k)$ 等价于 $d(k+2)$，我们就能得到 $d(k) = d(k+2) = d(k+4) = \cdots$，这也就是我们所要求的 DFA。通过计算机验证，取 $L = 10$ 就能满足条件。最坏情况下 DFA 的大小为 35，可以通过本题。

5.4 OI 字符串理论中的 DFA

由于 DFA 自身的特性，它成为了信息学竞赛中的许多字符串算法的基础结构。Trie 是一个能识别多串的简单结构，AC 自动机在此基础上引入失配边，使之能不断回退最终找到匹配。序列自动机是贪心的产物，结构简单却能有效处理子序列相关的问题。回文树是解决回文串问题有力算法，进一步的讨论可以参见 [11]。

在 OI 字符串理论中，成果最为丰富，应用最为广泛的当属后缀数据结构。后缀 Trie 是最基本的结构，在此基础上进行压缩（缩去出入度均为 1 的点）就得到了后缀树，而后缀自动机就是最小化后的后缀 Trie。同时对后缀 Tire 进行最小化和压缩能够得到压缩后缀自动机等更进一步的结果。从 DFA 的角度思考，有助于我们更加深刻地理解这些结构。上述内容的进一步讨论可以参见 [8, 10, 12]。接下来的这道例题假定读者已经掌握了后缀自动机的基础知识。

例题 5.4 Beautiful Automata[4]

构造一个仅包含小写字母的串 s，使得 s 的后缀自动机的转移图（仅保留其有向图的结构）与给定的 DAG 同构。如果不存在，输出 -1，否则输出字典序最小的解。

数据范围：$1 \le n \le 2000, 1 \le m \le 3000$。

显然只有开始状态入度为 0，记为 S。根据后缀自动机的性质，出度为 0 的点只能有一个，记为 T。也即图中的任意一个点都能从 S 到达，也都能到达 T。因为后缀自动机中的一

[4]XIX Open Cup Grand Prix of Baltic Sea, Problem G

个节点 u 是 right 集合相同的串的集合，所以所有从 S 到 u 的路径长度互不相同且构成了一个区间。

假设所有从 S 到 T 的路径长度分别为 $K+1, K+2, \cdots n$，分别代表从 $1, 2, \cdots, n-K$ 开始的后缀。因为 S 的转移边对应字符互不相同，所以我们根据这 $n-K$ 条路径的经过的第一条边，就能贪心确定 s 的前 $n-K$ 个字符。

剩下的 K 个字符在枚举 T 的后缀连接 P（即 parent 树上的父亲）后能唯一确定。我们要求从 S 到 P 的最长路恰好为 K，对应 s 长度为 K 的后缀。假设存在一条 P 到 T 的路径长度为 l，那么就有 $S[n-K+1:n] = S[n-K+1-l:n-l]$。由此便能唯一确定整个串 s。

如何判断一个串 s 是否符合条件？考虑恢复 DAG 的转移边字符。观察到对于一个后缀自动机，我们从 T 出发的任意一条反向路径构成了 s 反串的前缀。所以我们可以从 T 开始 BFS，就能唯一确定 DAG 的转移边了。接着再求出 s 的真实后缀自动机，判断两者是否同构即可。

时间复杂度为 $O(nm|\Sigma|)$。

5.5 正则表达式匹配的算法

关于正则表达式一个常见的问题是，判断一个串是否属于它的正则语言（设正则语言规模为 s，串长为 n）。我们不难想到使用 Thompson 构造法将正则表达式转化为一个 NFA，变成 NFA 的计算问题。接下来有两个方向可以继续。

一个方向是将 NFA 通过幂集构造法转化为 DFA，计算时可以 $O(1)$ 完成一次转移。这在正则表达式规模较小的时候比较适用。但是，我们构造正则表达式 $(a+b)^*a(a+b)(a+b)(a+b)\cdots$ 就能将对应状态数最少的 DFA 卡到指数级。所以这个做法的时间复杂度为 $O(s\cdot 2^s+n)$。

另一个方向是直接在 NFA 上计算。我们套用前文的 NFA 计算算法，即可做到 $O(\frac{n\cdot s^2}{\omega\cdot\log n})$ 的时间复杂度。实际上，利用正则表达式转化后的 NFA 的特殊性，我们能做到更优的时间复杂度。

注意到 Thompson 构造法每次只会增加常数条转移边。如果我们不将 ϵ 转移边消去，总转移边数就是 $O(s)$ 的。我们可以借鉴幂集构造方法的思想，在算法的开始我们求出 $E(q_0)$（即从初始状态沿 ϵ 转移到达的状态）作为初始的状态集合。在计算过程中，我们求出当前状态集合 S 中的元素，沿 c 转移边能够到达的新的状态集合 S'。之后再求出沿 ϵ 转移边能到达的所有状态（这一部分可以通过队列实现 BFS 完成）。发现每一步转移只需要遍历所有 $O(s)$ 个状态与转移边，所以该算法的时间复杂度为 $O(ns)$。

5.5.1 基于 Method of Four Russians 的高效算法

事实上，我们做到更优的时间复杂度。由于下述算法较为复杂，本文仅做一个简单的介绍，感兴趣的读者可以参考 [4] 获取详细内容。

首先建出正则表达式的表达式树，一个性质是每个节点至多只有两个子节点。考虑将表达式树分块，设阈值为 K，这里介绍一种分块的方法：每次从根节点开始，不断移向较大的一棵子树，直到子树的大小 $\leq K$，将该子树看做一个块，并使用一个节点来替代该子树。因为每个节点至多只有两个子节点，所以子树大小至多除 2，除根节点外，剩余的块大小一定在 $K/2$ 到 K 之间。表达式树被分作 $O(s/K)$ 块。我们建出正则表达式对应的 NFA，强制 Thompson 构造法中的每个运算符都新建两个节点（即 $R = (R_1R_2)$ 时，也新建一对节点），不难发现构造出 NFA 形成了嵌套结构，每块 NFA 通过一对节点与上一层的 NFA 产生联系。假设 $K \leq \omega$，我们可以使用一个二进制数维护每块 NFA 中的状态。

之后的转移分为两部分，首先考虑字符 c 的转移。根据 Thompson 构造法得到的 NFA 的性质，我们只需要考虑正则表达式树中的叶子节点。对于所有 $O(s/K)$ 个块和转移字符，我们预处理 $O(2^K)$ 种情况转以后得到的结果，就能在 $O(s/K)$ 内完成转移。

接下来考虑 ϵ 的转移。假设不存在闭包运算（即 * 运算），不难发现我们的 NFA 构成了一个 DAG，使得我们能够按照拓扑序来处理转移。这种情况下，我们可以按照正则表达式树 DFS 序的顺序来处理，同样预处理所有 $O(2^K)$ 种情况，我们能 $O(1)$ 完成一次块间的转移。存在闭包运算的情况更为复杂，因为存在回退的 ϵ 边，不过我们有以下结论

引理 5.2 *由 Thompson 构造法得到的 NFA 中，任意仅由 ϵ 组成的无环路径，至多经过一次闭包运算中的回退边。*

相信读者不难独立给出这个结论的证明（只需对正则表达式进行归纳即可）。由于只会回退一次，我们可以同样通过预处理来加速回退边的转移。接着，只需要再次按正则表达式树的 DFS 序处理一遍即可。本算法可以看做是 Method of Four Russians 又一强大的运用，正则表达式匹配的时间复杂度被优化为了 $O(\frac{ns}{\log n})$。

例题 5.5 *数字搜索*[5]

给定一个字符集大小为 10 的正则表达式（长度为 s），以及一个长度为 n 的字符串。

问该字符串存在多少前缀使得：该前缀存在一个后缀与正则表达式匹配？

数据范围：$1 \leq s \leq 500, 1 \leq n \leq 10^7$。

注意到本题与正则表达式匹配非常相似。同样考虑使用 NFA 来处理，我们只需要每一步转移之后，将开始状态加入状态集合。每次判断该前缀是否存在后缀与其匹配，判断状态集合是否包含接受状态即可。套用普通的正则表达式匹配算法即可做到 $O(ns)$，使用基于 Method of Four Russians 的高效算法即可做到 $O(\frac{ns}{\log n})$ 的时间复杂度。

本题在网络中现存的解题报告[6]与通过程序时间复杂度都是错误的。这些做法都是将当前的状态集合与转移对应的新的状态集合记录下来，再次访问到的时候就能 $O(\frac{s}{\omega})$ 完成一次转移。由于本题的测试数据强度较低，使得 NFA 在计算过程中有大量状态集合会重复出现。实际上，我们只需要构造正则表达式为 $(0+1)^*1(0+1)(0+1)\cdots(0+1)$（总长度不超过 500），

[5]CTSC 2004

[6]参见 https://www.docin.com/p-2262533912.html，或 https://www.docin.com/p-2268504832.html

并使用强度较高的随机数生成器产生一个长度为 10^7 的随机 01 序列，就能使得所有通过程序无法在时限的十倍时间内计算出答案。

如果有读者提出了更加优秀的算法，欢迎与作者交流。

6 总结

本文系统地简述了有限自动机的基础理论及算法，并展现了相关理论在信息学竞赛中的丰富应用。

有限状态自动机是一个非常简单的结构，但是我们能发掘大量的性质与许多有趣的应用，这就是它的魅力所在。作者相信，有限状态自动机仍然有着巨大的潜力，等待我们去进一步探索更加有趣的理论与应用。希望本文能起到抛砖引玉的作用，吸引更多的选手来学习和研究有限状态自动机。

致谢

感谢中国计算机学会提供学习和交流的平台。

感谢徐先友老师的关心和指导。

感谢周航锐同学、唐靖哲前辈对我的启发和写作本文的帮助。

感谢陈立言同学、周航锐同学为本文验稿。

感谢父母对我多年来的关心与支持。

参考文献

[1] Hopcroft J E, Motwani R, Ullman J D. Introduction to Automata Theory, Languages, and Computation, 3rd Edition[M]. Addison Wesley, 2006.

[2] Sipser M. Introduction to the Theory of Computation[M]. Cengage learning, 2012.

[3] Hopcroft J. An N Log N Algorithm for Minimizing States in a Finite Automaton[M]. Academic Press, 1971.

[4] Myers G. A four russians algorithm for regular expression pattern matching[J]. Journal of the ACM (JACM), 1992, 39(2): 432-448.

[5] 乔明达. 有限状态自动机 [R]. NOI WC 讲课, 2014.

[6] 杜瑜皓. FSM 相关问题选讲 [R]. ZJOI 讲课, 2015.

[7] 茹逸中. 形式语言与自动机 [R]. NOI WC 讲课, 2016.

[8] 张云帆. 从 DFA 到后缀自动机 [R]. APIO 讲课, 2018.

[9] 陈立杰. 计数问题选讲 [R] NOI WC 讲课, 2015.

[10] 徐翊轩. 浅谈压缩后缀自动机 [J]. IOI 中国国家候选队论文集, 2020.

[11] 翁文涛. 回文树及其应用 [J]. IOI 中国国家候选队论文集, 2017.

[12] Wikipedia. Suffix automaton[OL].
https://en.wikipedia.org/wiki/Suffix_automaton.

[13] Wikipedia. Myhill–Nerode theorem[OL].
https://en.wikipedia.org/wiki/Myhill-Nerode_theorem.

[14] Wikipedia. DFA minimization[OL].
https://en.wikipedia.org/wiki/DFA_minimization.