Лекция 9

Конечные автоматы. Задача о подстроке

9.1 Формальные языки

Будем называть *алфавитом* произвольное конечное множество (например, $\{0,1\}$ — алфавит). Строкой в алфавите Σ будет называться конечная последовательность его элементов (пустой строкой, обозначаемой ϵ, будет называться последовательность из нуля элементов). Для строки α будем обозначать α_i (а иногда — $\alpha[i]$) ее i-й символ, $\alpha[i..j]$ — ее подстроку с i-го по j-й символ включительно, а $|\alpha|$ — ее длину (количество символов).

Языком в алфавите Σ называется множество некоторых строк в алфавите Σ. Например: $\{\epsilon, 1, 00, 01\}$. Или: $\{\underbrace{0\ldots0}_{n}\underbrace{1\ldots1}_{n}|n \in \mathbb{N}\}$.

9.2 Конечные автоматы

[Полностью определенный] детерминированный конечный автомат — упорядоченная пятерка $(Q, \Sigma, q_S, F, \delta)$, где

- Q — конечное множество состояний,
- Σ — алфавит,
- $q_S \in Q$ — стартовое состояние,
- $F \subseteq Q$ — множество конечных состояний,
- $\delta : Q \times \Sigma \to Q$ — функция перехода.

Автомату дают строку в алфавите Σ; он, по очереди считывая ее символы, переходит из одного состояния в другое. Именно, считав символ $a \in \Sigma$, он переходит из текущего состояния q (начинает он с $q = q_S$) в состояние $\delta(q, a)$. На следующем шаге он будет считывать следующий символ. Если, считав входную строку полностью, он попадает в одно из конечных состояний, то говорят, что автомат *принимает* данную строку. Формально говоря, автомат принимает строку $s = s_1 \ldots s_k$ (где $\forall i\ s_i \in \Sigma$), если существуют состояния $q_1, q_2, \ldots, q_{k+1}$, такие, что $q_1 = q_S$, $q_{k+1} \in F$ и $\forall i \leqslant k\ q_{i+1} = \delta(q_i, s_i)$.

Множество всех строк в алфавите Σ, принимаемых данным автоматом $\mathcal{A}$, называется *языком, принимаемым (задаваемым) этим автоматом*, и обозначается $L(\mathcal{A})$.

9.3 Задача о поиске подстроки (pattern matching)

Задача: даны строки p (образец — pattern) и t (текст — text); $|p| = m$, $|t| = n$. Вопрос: встречается ли подстрока p в строке t, то есть существует ли такое $i \geqslant 1$, что $i + m - 1 \leqslant n$ и $p = t_i t_{i+1} \dots t_{i+m-1}$?

Тривиальный алгоритм работает $O(mn)$ шагов. Мы построим алгоритм, которому достаточно $O(m + n)$ шагов.

9.3.1 Конечный автомат для поиска заданного образца

ПРОБЕЛ В КОНСПЕКТЕ.

Чтобы построить этот автомат, к сожалению, мы потратим $O(m|\Sigma|)$ шагов (см. ниже), но это стоит сделать, если[1] $m|\Sigma| < n$, поскольку исполнение конечного автомата делается за линейное количество шагов с чрезвычайно небольшой константой в $O(\cdot)$ и весьма эффективной реализацией каждого шага.

9.3.2 Линейный алгоритм

В алгоритме нам понадобится «таблица откатов» Π,

$$\Pi[q] = \max\{k \mid k < q,\ p[1..k] \text{ — суффикс } p[1..q]\}$$

(если таких k нет, $\Pi[q] = 0$). (Заметим, что $p[1..k]$ — суффикс $p[1..q]$, если $p[1] \dots p[k] = p[q-k+1] \dots p[q]$.) Как вычислить эту таблицу, мы узнаем чуть позже.

Основной алгоритм. У нас будет два «указателя» q и i; первый указывает на текущий элемент образца; второй — текста.

$q := 1;$
for $i := 1$ to n do
begin
(*) while $q > 1$ and $p[q] \neq t[i]$ do $q := \Pi[q-1] + 1;$
if $p[q] = t[i]$ then $q := q + 1;$
if $q = m + 1$ then «Нашли!»;
end;

Покажем, что этот алгоритм заканчивает свою работу за $O(n)$ шагов. Сомнения может вызывать лишь строка (*), так как в ней имеется вложенный цикл. Однако, в ней уменьшается q. Увеличиться же оно может лишь n раз (по одному разу для каждого i), причем всего на единицу. Следовательно, и тело цикла (*) не может выполниться более n раз за все время работы алгоритма.

Вычисление «таблицы откатов» Π. У нас снова будет два «указателя» k и q; на сей раз оба указывают на текущие элементы образца.

[1]Или если этот образец придется искать многократно в разных текстах: например, если в программе написано «найдти подстроку “error” в строке `next_string`».

$k := 1$;
$\Pi[1] := 0$;
for $q := 2$ to m do
begin
  while $k > 1$ and $p[k] \neq p[q]$ do $k := \Pi[k-1]+1$;
  if $p[k] = p[q]$ then $k := k+1$;
  $\Pi[q] := k-1$;
end;

То, что таблица будет вычислена за $O(m)$ шагов, показывается аналогично тому, как это было сделано для основного алгоритма (только теперь мы следим за «указателем» k).

Замечание 9.1. Эту таблицу откатов, разумеется, можно перестроить в упоминавшийся выше конечный автомат:

ПРОБЕЛ В КОНСПЕКТЕ.

9.4 Конечные автоматы: продолжение

Регулярные языки — синоним для языков, принимаемых конечными автоматами (чуть позже мы увидим, почему).

Лемма 9.1 (лемма о разрастании для регулярных языков (pumping lemma)). *Пусть L — регулярный язык. Тогда существует константа c, такая, что любую строку $x \in L$ длины не менее c можно разбить на три части $x = u \cdot v \cdot w$, такие что $0 < |v| \leqslant c$ и $\forall i \geqslant 0$ $u \cdot v^i \cdot w \in L$.*

Доказательство. Рассмотрим детерминированный конечный полностью определенный автомат для языка L. Пусть $c = |Q|+1$. Посмотрим, как он работает на цепочке x: $q_s \to q_1 \to \ldots \to q_k \in F$. На каждом шаге считывается некоторый символ. Поскольку $|x| \geqslant c$, мы должны были побывать в каком-то состоянии дважды, и в нашем пути есть циклы. Выберем несамопересекающийся цикл; пусть до первого прохождения по нему считывалась строка u, при прохождении по нему считывалась строка v (ее длина не превосходит c, поскольку цикл — несамопересекающийся), а после прохождения по нему (в том числе, если по нему пошли еще раз) — строка w. Очевидно, наш автомат примет любую строку вида uv^iw. □

С помощью этой леммы можно доказывать, что какой-нибудь язык не является регулярным.

Пример 9.1. $L = \{0^n1^n | n \in \mathbb{N} \cup 0\}$ не является регулярным.
Доказательство. Пусть регулярный. Рассмотрим варианты подстроки v из леммы.

1. В строку попадают только нули $\Rightarrow$ в uv^2w количество 0 увеличится, а количество 1 останется неизменным $\Rightarrow$ строка не будет принадлежать L.
2. В строку попадают только единицы — аналогично.
3. $v = 0^i1^j \Rightarrow$ в uv^2w после 1 будет идти 0 $\Rightarrow$ строка опять не будет принадлежать языку.

□

9.5 Регулярные выражения

Определим *регулярные выражения* в алфавите Σ. Они будут определяться индуктивно:

- $\emptyset$ — регулярное выражение;
- $\{\epsilon\}$ — регулярное выражение;
- $\{a\}$ — регулярное выражение (для каждого $a \in \Sigma$);
- Если A, B — регулярные выражения, то $A \cup B$ — тоже регулярное выражение;
- Если A, B — регулярные выражения, то $A \cdot B$ — тоже регулярное выражение;
- Если A — регулярное выражение, то A^* — тоже регулярное выражение.

Это определение исчерпывает все возможные регулярные выражения.

Каждое регулярное выражение определяет некоторый язык. Для большинства пунктов определения очевидно, какой язык имеется в виду; оставшиеся пункты:

- для данных языков A и B язык $A \cdot B$ состоит из строк вида ab, где $a \in A$, $b \in B$ (значок операции “точка” — конкатенации строк — часто опускается);
- $A^* = \{\epsilon\} \cup A \cup A \cdot A \cup A \cdot A \cdot A \ldots$ (все конечные $A \cdot \ldots \cdot A$).

Например, $(\{0\} \cup \{11\})^* \cdot \{000\}$ обозначает множество всех последовательностей нулей и пар единиц, заканчивающихся на три нуля.

Теорема 9.1. *Множества языков, задаваемых*
(1) конечными автоматами,
(2) регулярными выражениями
в одном и том же алфавите Σ, совпадают.

Задача 9.1. Доказать теорему 9.1.

9.6 Замкнутость регулярных языков относительно некоторых операций и разрешимые проблемы, связанные с регулярными языками

Полезные свойства.

1. Множество всех регулярных языков в данном алфавите замкнуто относительно операций, которые их порождают: $\cup$, $\cdot$, *.
2. **Лемма 9.2.** *Множество всех регулярных языков в данном алфавите замкнуто относительно дополнения.*

   Доказательство. Рассмотрим детерминированный конечный полностью определенный автомат, задающий язык L. Поменяем местами его конечные состояния с остальными: $F' := Q \setminus F$. Полученный автомат задает язык $\overline{L}$. □

3. **Следствие 9.1.** *…и замкнуто относительно пересечения.*

Разрешимые проблемы.

1. Принадлежность. Рассмотрим детерминированный конечный полностью определенный автомат, принимающий данный язык. Чтобы узнать, принадлежит ли этому языку некоторая строка, запустим наш автомат на этой строке x. После $|x|$ переходов мы поймем, принадлежит ли она языку.

2. Пустота языка. Чтобы решить эту проблему, достаточно проверить, достижимо ли какое-нибудь конечное состояние автомата из начального. Эта задача, очевидно, алгоритмически разрешима.
3. Равенство языков. Чтобы построить алгоритм для этой задачи, достаточно заметить, что

$$L_1 = L_2 \Leftrightarrow L_1 \cap \overline{L_2} = \emptyset \wedge L_2 \cap \overline{L_1} = \emptyset \text{ (или: } (L_1 \cap \overline{L_2}) \cup (L_2 \cap \overline{L_1}) = \emptyset).$$

4. Включение языков. Чтобы построить алгоритм для этой задачи, достаточно заметить, что

$$L_1 \subseteq L_2 \Leftrightarrow (L_1 \cap \overline{L_2} = \emptyset) \text{ (или: } (L_1 \cap L_2 = L_1)).$$

9.7 Простой алгоритм для поиска подстроки, использующий случайные числа

ПРОБЕЛ В КОНСПЕКТЕ.