Разбираемся с упрощенным алгоритмом Крошмора

Никита Сивухин

17.04.2017

1 О чем пойдет речь?

Мы поговорим про детерминированный алгоритм линейного поиска шаблона t в строке s, использующий $O(1)$ дополнительной памяти. Алгоритм не будет менять исходные строки (т.е. t, s — `read-only`) и реально дополнительно хранит только $O(1)$ машинных слов.

Также рассмотрим модификацию алгоритма, который находит все вхождения максимального префикса паттерна, который хоть раз встречается в строке s (т.е. если паттерн встречается в s, то алгоритм совпадает с алгоритмом поиска полных вхождений).

После постараемся понять, в каких задачах данный алгоритм может быть использован в качестве подзадачи.

Материал лекции является вольным пересказом статьи [1].

2 Начнём

Я начну рассказ с парочки базовых алгоритмов и утверждений, чтобы потом можно было легко собрать конечный алгоритм, представленный в заявленной выше статье.

Основным алгоритмом в данной области можно считать алгоритм Кнута-Морриса-Пратта. Можно мыслить себе этот алгоритм как процесс обработки строки неявным автоматом. Но полезно также иметь ввиду следующий простой факт:

Утверждение. Если $lcp(s_i, t) = l$, а период l равен p, то $|lcp(s_{i+k}, t)| < |l|$ для всех $k < p$.

С помощью этого факта мы можем запросто перескочить p позиций, не потеряв ни одного вхождения шаблона в текст, а также не потеряв ни одного вхождения максимального префикса в текст (данный факт будет использоваться в модификации алгоритма).

Алгоритм 1 Алгоритм Морриса–Пратта

```
start ← 1, lcp ← 0;
while start ≤ n do
    while start + lcp < n and s[start+lcp+1] = t[lcp+1] do
        lcp += 1
    if lcp = m then ReportOccurence(start)
    if lcp = 0 then start += 1
    else start += π(t[1 ... lcp]), lcp −= π(t[1 ... lcp])
```

Данная интерпретация работы алгоритма хороша тем, что нам необязательно точно знать значение периода прочитанного префикса паттерна. Например:

- если мы будем уметь точно определять период p строки u, в случае если $p \leq |u|/k$
- и уметь отличать случай когда $p > |u|/k$ (однако знать значение p мы уже не обязаны)

то несложно построить линейный алгоритм на основе КМП, который ищет все вхождений шаблона в строку (на самом деле алгоритм будет работать за $O(nk)$).

Алгоритм 2 Алгоритм Морриса–Пратта использующий оценку периода

$start \leftarrow 1, lcp \leftarrow 0$
while $start \leq n$ **do**
  while $start + lcp < n$ **and** $s[start{+}lcp{+}1] = t[lcp{+}1]$ **do**
    $lcp \mathrel{+}= 1$
  if $lcp = m$ **then** `ReportOccurence`($start$)
  $known\_period, p \leftarrow$ `EstimatePeriod`($t[1 \ldots lcp]$)
  if $lcp = 0$ **then** $start \mathrel{+}= 1$
  else if $known\_period$ **then** $start \mathrel{+}= p, lcp \mathrel{-}= p$
  else
    $start \mathrel{+}= lcp/k$
    $lcp \leftarrow 0$

Можно рассмотреть полуинвариант $k \cdot start + lcp$, который неубывает, а в случае успешного сравнение символа — возрастает.

3 Учимся искать периоды

Искать период строки используя $O(1)$ дополнительной памяти нетривиальная задача, поэтому мы научимся искать периоды в **несложных** строках, а потом научимся искать **несложные** строки в шаблоне и найдем взаимосвязь между ними.

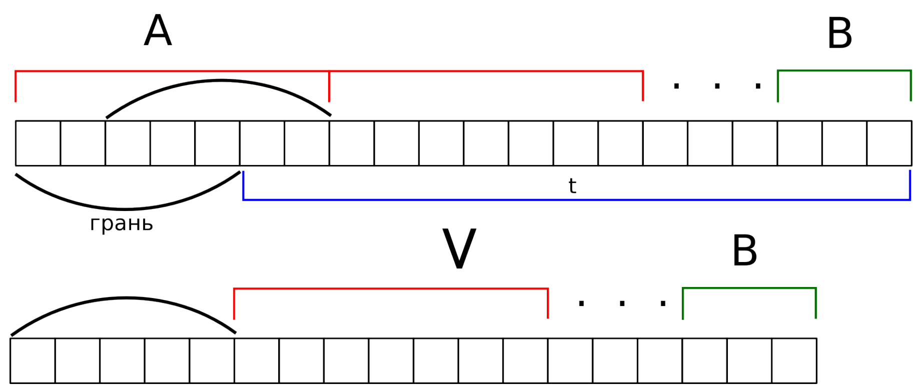
Присмотримся повнимательнее к следующему классу строк: u такие, что u больше любого своего собственного суффикса (будем называть этот класс — класс *само-максимальных* строк (*self-maximal*))

Данный класс строк очень сильно напоминает слова Линдона. Кажется, что от знака неравенства не должна меняться структура слов и данный класс изоморфен словам Линдона. Однако это не так, потому что можно считать, что в конце любой строки стоит символ, который меньше чем любой символ алфавита. Поэтому нарушается симметрия между неравенствами и класс *само-максимальных* строк шире, чем класс слов Линдона.

Видно например, что *само-максимальная* строка может иметь нетривиальный период($baba$), что отличает их от слов Линдона. Однако периодичность строк данного класса специфична. Докажем следующую лемму:

Лемма 1. *Если* $s = a^k b$ *—* само-максимальная *строка, где* a *— её минимальный период и* $|b| < |a|$*, то* a *— безграничная строка.*

Доказательство. Ну тут показывать надо.



Будем доказывать от противного. Пусть существует грань: $a = pu$ и u — префикс a. Тогда, т.к. s — *само-максимальная*, то $ua^{k-1}b < a^k b$.

- Если неравенство «строгое» (т.е. меньшая строка не является префиксом большей), то «отрезав» u от обеих строк мы получим, что $a^{k-1}b < t$, из чего следует, что $a^k b$ не является *само-максимальной* (т.к. неравенство опять строгое).
- Если же меньшая строка является префиксом большей, то p — период всей строки, что невозможно, т.к. a — минимальный.

□

Заметим, что любой префикс само-максимальной строки — тоже само-максимальный. Из этого наблюдения и леммы следует достаточно сильное утверждение:

Утверждение. Если Sa — *само-максимальная* строка, то либо $\pi(S) = \pi(Sa)$, либо $\pi(Sa) = |Sa|$

Таким образом, только что мы научились искать период само-максимальных строк за $O(n)$ используя $O(1)$ дополнительной памяти.

Алгоритм 3 Алгоритм поиска периода само-максимальной строки s

$p \leftarrow 1$ ▷ Период обработанного префикса
for $(i \leftarrow 2;\ i \leq |s|;\ i \leftarrow i+1)$ **do**
  if $s[i] \neq s[i-p]$ **then** $p \leftarrow i$

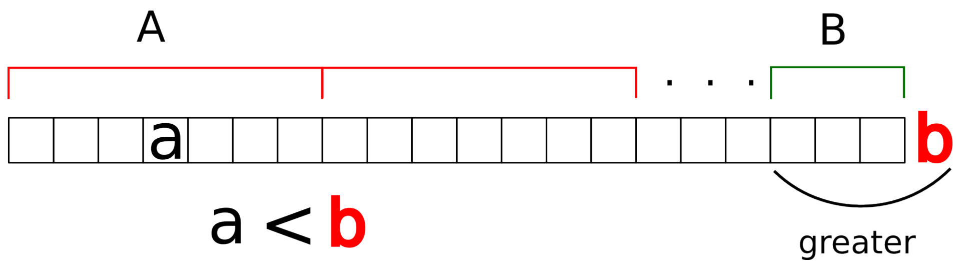
По сути класс само-максимальных слов представляет из себя следующий по «простоте» класс после слов Линдона — каждое слово данного класса представляется в виде степени одного слова Линдона (т.е. имеет простую декомпозицию Линдона, при условии, что порядок алфавита реверснут).

Поймем, в каких случаях слово перестает быть само-максимальным в случае дописывания символа справа.

Лемма 2. *Если S — само-максимальная строка, период которой p, то Sc — само-максимальная тогда и только тогда, когда $S[-p] \geq c$.*

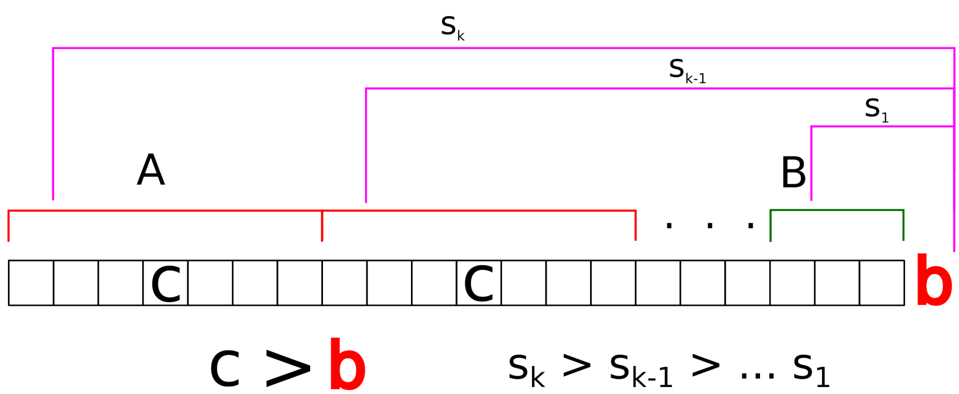
Доказательство.

$\Rightarrow$ Предположим противное: строка осталась само-максимальной, но $S[-p] < c$.



Видно, что тогда если $S = a^k b$, то bc больше чем Sc, значит строка больше не является самомаксимальной. Причем заметим, что все суффиксы левее bc меньше, чем S.

$\Leftarrow$



Предположим, что нашелся суффикс, больший чем вся строка. Тогда он обязательно начинается в первом блоке периода a. Но суффикс, соответствующий этой позиции в строке S меньшем «строго» меньше чем вся строка, т.к. в противном случае у строки был бы меньший период. Но тогда наше предположение неверно.

□

4 Ищем несложные слова с сложных

Раз мы изучили некоторый класс слов, нужно научиться быстро его искать в произвольных строках. Давайте попробуем найти максимальный суффикс строки, который является само-максимальным.

Более формально — мы хотим за линейное время(суммарно) найти для каждого префикса строки лексикографически максимальный суффикс и его период.

Ясно, что максимальный суффикс обработанного префикса ползет вправо. Причем, если после добавления нового символа, старый максимальный суффикс остался само-максимальным, то он является максимальным суффиксом новой строки.

Значит нужно понять, как меняется максимальный суффикс, когда старый суффикс перестает быть само-максимальным.

Воспользуемся уже имеющимися знаниями и в случае поломки максимального суффикса, запустим наш алгоритм рекурсивно на меньшем суффиксе строки bc, если считать что поломка произошла при добавлении символа c к строке $S = Pa^kb$ (a^kb — наш старый максимальный суффикс, можно посмотреть на картинку из леммы 2).

Алгоритм 4 Перечисление максимальных суффиксов префиксов строки s

```
function MaxSuffixes(s)                          ▷ Возвращает пары: (start, period)
    yield return (1, 1)
    reported, start, p ← 1, 1, 1
    for (i ← 2; i ≤ |s|; i ← i + 1) do
        if s[i − p] > s[i] then p ← i − start + 1
        else if s[i − p] < s[i] then
            i ← i − (i − start)%p
            start ← i
            p ← 1
        if i > reported then
            reported ← i
            yield return (start, p)
end function
```

Видно, что полуинвариант $start + i$ неубывает, а значит данный алгоритм работает линейное время. Корректность работы следует из вышедоказанных утверждений. Также полезно сделать следующее наблюдение насчет времени работы одной итерации (между двумя последовательными **return**-ами) алгоритма:

Наблюдение. Алгоритм работает за $O(\Delta s)$ на один шаг.

Мне почему-то показалось, что генераторную функцию понять проще, но в дальнейшей работе будет удобно использовать «развернутый» аналог алгоритма 4, который выполняет один шаг между последовательными **yield return**-ами.

Алгоритм 5 Алгоритм поиска максимального суффикса вместе с его периодом

```
function UpdateMS(len, start, p)
    if len = 0 then return (1, 1)        ▷ инициализируем значения для префикса длины 1
    i ← len + 1     ▷ длина префикса, для которого хотим обновить максимальный суффикс
    while i ≤ len + 1 do
        if s[i − p] > s[i] then
            p ← i − start + 1
        else if s[i − p] < s[i] then
            i ← i − (i − start)%p
            start ← i
            p ← 1
        i ← i + 1
    return (start, p)
end function
```

Таким образом, мы научились искать **несложные** слова в сложных за линейное время используя при этом $O(1)$ (**!**) дополнительной памяти.

5 Ищем связь между периодами

Осталось понять, существует ли связь между периодом лексикографически максимального суффикса и периодом всего слова.

Лемма 3. *Пусть $s = Pa^k b$, где $a^k b$ — максимальный суффикс строки s и a — его период. Докажем следующие утверждения:*

1. $|P| < \pi(s)$
2. $\pi(s) = \pi(a^k b) \Leftrightarrow P$ *— суффикс* a
3. *если s — 3-периодична ($3\pi(s) \leq |s|$), то $\pi(s) = \pi(a^k b)$*

Доказательство.

1. Если бы $|P| \geq \pi(s)$, то $a^k b$ встречалось бы на $\pi(s)$ позиций левее — противоречие с максимальностью
2. $\Rightarrow$ По определению периода $s[i] = s[i + |a|]$ для $1 \leq i \leq |P|$.

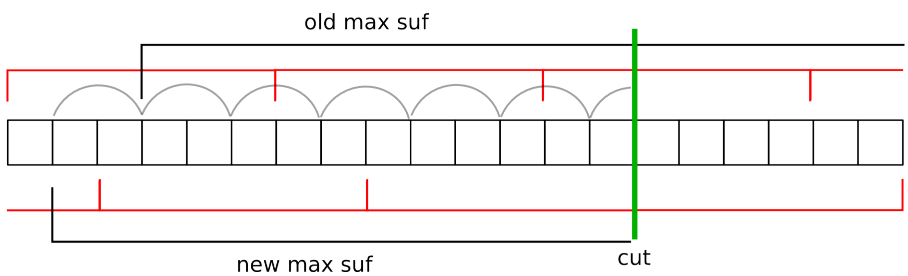
   $\Leftarrow$ Т.к. $\pi(s)$ не может быть меньше, чем $\pi(a^k b)$, а в случае, если P суффикс a видно, что у строки s есть период $|a|$, то этот период минимален.
3. От противного: пусть $\pi(s) > \pi(a^k b)$. Применим теорему Файна-Вильфа к строке $a^k b$. У нее есть период $p = \gcd(\pi(s), \pi(a^k b))$. Но тогда $\pi(s)$ — не минимальный период, т.к. p делит его.

□

Докажем ещё одну важную лемму:

Лемма 4. *Если u — 3-периодическая строка, а (s, p) — начало и период её максимального суффикса, то для строки $u[1 \ldots |u| - p]$ позиция и период максимального суффикса не поменяются.*

Доказательство. От противного — пусть после уменьшения строки появился новый максимальный суффикс.



Тогда этот суффикс появился левее старого. Но тогда он имеет периоды p и $s - s'$ (s' — новая позиция суффикса). Можно воспользоваться теоремой Файна-Вильфа. Тогда получим, что p — не минимальный период старого суффикса, т.к. существует меньший период g, который делит p.

□

Значит мы научились оценивать период всей строки, зная период её максимального суффикса. Уже сейчас мы можем написать алгоритм поиска шаблона в тексте с $O(1)$ дополнительной памяти:

Алгоритм 6 Упрощенный алгоритм Крошмора

$i \leftarrow 1$
$l \leftarrow 0$
$p \leftarrow 0$, $start \leftarrow 0$
while $i \leq n$ **do**
    while $l < m$ **and** $i + l \leq n$ **and** $s[i + l] = t[1 + l]$ **do**
        $start, p \leftarrow UpdateMS(l, start, p)$
        $l \leftarrow l + 1$
    if $l = m$ **then** `ReportOccurence`(i)
    if $p \leq \frac{1}{3}l$ **and** $t[1 \ldots start - 1] = t[1 + p \ldots start - 1 + p]$ **then**
        $l \leftarrow l - p$
        $i \leftarrow i + p$
    else
        $i \leftarrow i + \lfloor l/3 \rfloor + 1$
        $l, p, start \leftarrow 0, 0, 0$

Покажем для начала, что этот алгоритм работает за $O(n)$. Есть только два места, которые могут выполняться дольше — это сравнение $t[1 \ldots start-1] = t[1+p \ldots start-1+p]$ и $UpdateMS$. Оценим сначала сравнения. Для этого разделим их на два класса: успешные и неуспешные.

Заметим, что суммарная длина сравненных частей в случае успеха не превосходит $O(n)$, т.к. после каждого такого сравнения i сдвигается на p позиций вправо, а мы сравнивали подстроки длиной $s < p$.

В случае же неуспеха мы увеличиваем i на $\lfloor l/3 \rfloor + 1$, что больше, чем $s/3$, значит суммарно таких сравнений будет не больше чем $3n$.

Чтобы показать, что $UpdateMS$ работает линейное время достаточно рассмотреть полуинвариант $3i+s$ и вспомнить, что $UpdateMS$ работает за $O(\Delta s)$. Т.к. полуинвариант неуменьшается, получаем что эта часть тоже линейна.

6 Расширения

В статье [1] приводится две задачи, которые можно решить с помощью незначительных модификаций приведенного выше алгоритма:

Задача 1. Поиск максимального префикса

Найти все вхождения максимального префикса паттерна, которые встречается в тексте хотя бы раз.

Задача 2. Разреженный поиск максимального префикса

Для заданного упорядоченного списка позиций Pos найти все вхождения максимального префикса, которые начинаются в одной из этих позиций.

Для решения первой задачи, достаточно поддерживать максимальный префикс, которые мы до этого встречали и множество позиций, где он встречался. Тогда вместо проверки

$$\textbf{if } l = m \; \texttt{ReportOccurence}(i)$$

достаточно обновить значение максимального префикса текущим значением lcp и добавить новую позицию при необходимости. Имеет смысл запустить алгоритм два раза — сначала посчитать значение максимального префикса и только потом получить все позиции.

Для решения второй задачи, достаточно сразу пропускать те позиции i, которые не находятся во множестве Pos. Все остальное — как в первой задаче.

Список литературы

[1] Juha Kärkkäinen, Dominik Kempa, and Simon J. Puglisi *Crochemore's String Matching Algorithm: Simplification, Extensions, Applications.*