УДК 004.421.6

О НОВЫХ АЛГОРИТМАХ ВЗАИМНОГО ПРЕОБРАЗОВАНИЯ СТРОКОВЫХ СХЕМ

С. Д. Махортов

Воронежский государственный университет

Поступила в редакцию 27.01.2017 г.

Аннотация. Алгоритмы вычислений на строках широко востребованы в различных прикладных областях, связанных с обработкой больших символьных данных. Их актуальность существенно повысилась в последние десятилетия, когда появились новые предметные области, требующие *эффективной* обработки *гигантских* символьных последовательностей. К таким областям, в частности, относятся, компьютерное зрение, информационный поиск в глобальных сетях, вычислительная геномика и некоторые другие.

Многие из таких алгоритмов основаны на использовании схем описания структуры строк. В статье рассматриваются две наиболее эффективные схемы анализа и описания структуры строк — массивы граней и Z–блоков. Обсуждаются возможности их взаимного преобразования. Предлагается и обосновывается новый линейный алгоритм преобразования массива Z–блоков в массив граней.

Ключевые слова: строка, поиск вхождений, препроцессинг, префикс-функция, Z-функция, взаимные преобразования, анализ сложности.

ON NEW ALGORITHMS OF MUTUAL TRANSFORMATION OF STRING SCHEMES

S. D. Makhortov

Abstract. The algorithms of computation on the strings are widely claimed in various applied areas related to the processing of big character data. Their relevance has significantly increased in recent decades with the appearance of new subject areas that require efficient processing of giant character sequences. These areas, in particular, include computer vision, information search in global networks, computational genomics and some others.

Many of these algorithms are based on the use of schemes of the description of strings structure. The article considers two most effective schemes for the analysis and description of strings structure — border arrays and Z–blocks. The possibilities of their mutual transformation are discussed. A new linear algorithm for transforming an array of Z–blocks into an array of borders is proposed and justified.

Keywords: string, string matching, preprocessing, prefix-function, Z-function, complexity analysis.

ВВЕДЕНИЕ

Алгоритмы вычислений на строках находят широкое применение в различных разделах прикладных наук. В последние десятилетия появились новые предметные области, требующие *эффективной* обработки *гигантских* символьных последовательностей. К ним, в частности, относятся компьютерное зрение, информационный поиск в глобальных сетях, вычислительная геномика. Например, в задачах исследования цепочек ДНК [1] встречаются символьные строки длиной порядка 3000000, и при их обработке даже алгоритмы с квадратичной сложностью оказываются слишком дорогостоящими. Отсюда возникает потребность в

математическом описании структуры последовательностей символов с целью получения и обоснования более эффективных алгоритмов вычислений на строках.

Одна из наиболее известных задач обработки строк — вычисление всех вхождений образца P (заданной подстроки) длины m в текст T (большую строку) длины n. "Наивный", то есть непосредственный, алгоритм ее решения, очевидно, характеризуется вычислительной сложностью $O(mn)$. Как было подчеркнуто выше, подобный результат неприемлем для некоторых современных прикладных областей.

Задолго до возникновения подобной проблемы были предложены алгоритмы, решающие указанную задачу за время $O(n)$ (например, Кнута-Морриса-Пратта, Бойера-Мура и т. д. [2]). Использование же структуры "суффиксное дерево" (П. Вайнер) [1], получаемой за линейное по n время, позволяет выявить все вхождения подстроки выполнением $O(m$ + число_вхождений) операций. Ранее такие алгоритмы имели лишь теоретическое значение, однако теперь их изучение стало весьма актуальным для практики. Аналогичное справедливо и в случае некоторых других задач на строках (общая подстрока, оптимальное выравнивание и т. д. [1]), требующих точного или даже приближенного решения.

В основе многих эффективных алгоритмов анализа текстов лежит *препроцессинг* — предварительное исследование структуры строки. В результате такого исследования, подобно индексу базы данных, строится некоторая вспомогательная конструкция, позволяющая в дальнейшем существенно сократить количество избыточных сравнений символов. На первой стадии обычно исследуется только одна из обрабатываемых строк (например, образец P или текст T), а другая строка может даже оставаться неизвестной. При этом важно, что препроцессинг занимает приемлемое время, обычно — линейное. Далее следует стадия непосредственного поиска или анализа подстрок, на которой полученная ранее информация о строке используется для ускорения работы.

В настоящей статье представлены две наиболее известные схемы предварительного описания структуры строк — массивы граней и Z-блоков [1, 3]. Рассматриваются возможности их взаимного преобразования. Подобные вопросы уже изучались в предыдущих публикациях ([1, 4]). Однако здесь предлагается новый линейный алгоритм преобразования массива Z-блоков в массив граней, по мнению автора, имеющий более простое обоснование и отличающийся лучшей читабельностью.

1. БАЗОВЫЕ ОПРЕДЕЛЕНИЯ

Напомним основные понятия и обозначения, связанные со строками [1–3].

Пусть задан *алфавит* — фиксированное множество различимых элементов (*символов*). *Строка* на алфавите A — это любой конечный упорядоченный набор символов, принадлежащих A. Через $n = |S|$ обозначается *длина* (число символов) строки S. Формально вводится также понятие *пустой* строки ε (нулевой длины), которая не содержит символов.

С позиций информатики строку S можно интерпретировать как массив символов $S[0..n-1]$. Под *равенством* (или *совпадением*) двух строк $S_1 = S_2$, определенных на общем алфавите, подразумевается посимвольное равенство двух соответствующих массивов одинаковой длины.

Подстрока некоторой строки — это строка, полученная как непрерывная подпоследовательность исходной строки. Для любой строки S через $S[i..j]$ обозначается подстрока в массиве $S[0..n\text{-}1]$, которая начинается в позиции i и заканчивается в позиции j. В частности, $S[0..i]$ называется *префиксом*, а $S[j..|S|-1]$ — *суффиксом* строки S.

При $i > j$ подстрока $S[i..j]$ пуста. Очевидно, длина непустой подстроки $S[i..j]$ равна $j-i+1$. Подстрока называется *собственной*, если она не равна S. По определению любая непустая строка в качестве собственной подстроки в любой позиции содержит пустую строку.

Наиболее известны две эффективные схемы предварительного анализа строк – нахождение

граней и Z-блоков. Они рассматриваются в следующем разделе.

2. ГРАНИ И Z-БЛОКИ СТРОК

Определение. Гранью строки называется любой собственный префикс этой строки, совпадающий с ее же суффиксом.

Ограничение "собственный" исключает из рассмотрения грань, совпадающую со всей строкой. Любая непустая строка имеет пустую грань (длины 0). Например, строка $\underline{ABAAB}ABA\underline{ABAAB}$ содержит две непустые грани: AB и $ABAAB$.

Выделяют *наибольшую грань* строки, то есть имеющую среди всех ее граней наибольшую длину (в предыдущем примере — $ABAAB$). Строка $ABAABAAB$ имеет в точности те же грани, но в этом случае наибольшая грань $ABAAB$ частично перекрывается (точнее, перекрываются префикс и суффикс, образующие эту грань).

Очевидно, что известная длина грани конкретной строки несет полную информацию о самой этой грани. Поэтому иногда, говоря о гранях строки, подразумевают их длины (если это не вызывает недоразумений).

В приложениях к исследованию строк существенную роль играет *массив граней* (*border of prefixes array*) $bp[0..n-1]$, содержащий длины наибольших граней для всех подстрок $S[0..i]$, $i = 0, 1, \ldots, n-1$, то есть для префиксов в S. Например, строке $ABAABABAABAAB$ соответствует массив 0, 0, 1, 1, 2, 3, 2, 3, 4, 5, 6, 4, 5. В ряде источников (например, [2]) для такого массива используется другое название — "префикс-функция".

Существует эффективный алгоритм вычисления массива граней, работающий за линейное время [3]. Он основан на следующих достаточно очевидных свойствах граней и их массива.

$$\underline{\boldsymbol{AB}AA\boldsymbol{AB}}ACBCA\underline{\boldsymbol{AB}AA\boldsymbol{AB}}$$

1. $bp[0] = 0$, поскольку строка длины 1 имеет единственную собственную подстроку — ε.

2. Если подстрока $S[0..i]$ $(0 < i < n)$ имеет грань длины $k > 0$, то подстрока $S[0..i-1]$ имеет, по крайней мере, грань длины $k-1$. Отсюда следует, что при переходе от $i-1$ к i значение $bp[i]$ увеличивается не более чем на 1, то есть $bp[\mathrm{i}] \leqslant bp[i-1]+1$.

3. Равенство $bp[\mathrm{i}] = bp[i-1]+1$ выполнено тогда и только тогда, когда $S[i] = S[bp[i-1]]$ (поскольку $bp[i-1]$ — позиция символа справа от префикса $S[0..bp[i-1]-1]$, который является наибольшей гранью подстроки $S[0..i-1]$).

4. Если b — грань строки S, а $b\prime$ — грань подстроки b, то $b\prime$ есть грань строки S. Другими словами, грань грани строки также является гранью этой строки. Отсюда следует, что, поскольку $bp[i]$ — длина наибольшей грани строки $S[0..i]$, то $bp[bp[i]-1]$ — длина второй по величине грани строки $S[0..i]$ и т. д. Эта строго убывающая конечная последовательность заканчивается нулем.

Замечание 1. Согласно свойствам 2–3, массив bp состоит из нескольких *серий* возрастающих на 1 целочисленных подпоследовательностей, возможно, разделенных нулями.

Перечисленные выше свойства дают возможность вычислить $bp[i]$ на основе значений $bp[0]$, $bp[1]$, …, $bp[i-1]$. Согласно свойству 2, положительная величина $bp[i]$ получается увеличением на 1 длины некоторой грани предыдущей строки — $S[0..i-1]$. Если $S[i] = S[bp[i-1]]$, то (свойство 3) следует положить $bp[\mathrm{i}] = bp[i-1]+1$. В противном случае надо рассмотреть вторую по величине грань строки $S[0..i-1]$, то есть (свойство 4) проверить равенство $S[i] = S[bp[bp[i-1]-1]]$. Если это равенство выполнено, взять $bp[\mathrm{i}] = bp[bp[i-1]-1]+1$. При необходимости рассматриваются следующие по уменьшающейся величине грани подстроки $S[0..i-1]$. Процесс останавливается при достижении равенства символов либо когда очередная по длине грань окажется пустой. В последнем случае в качестве $bp[\mathrm{i}]$ берется 1 или 0, в зависимости от истинности равенства $S[i] = S[0]$.

Приведем обновленное (по сравнению с [3]) формальное представление этого алгоритма — оно потребуется в следующем разделе. Здесь и в дальнейшем алгоритм записан с помощью псевдокода — подмножества языка С без указания типов данных.

```
Prefix-Border-Array (S, bp)
{
  n = strlen(S); bp[0] = 0;
  for (i = 1; i < n; ++i)
  { // i - длина рассматриваемого префикса
    bpRight = bp[i-1]; // Позиция справа от предыдущей (i-1) грани
    while (bpRight && (S[i]!=S[bpRight])) bpRight = bp[bpRight-1];
    bp[i] = (S[i] == S[bpRight]) ? bpRight + 1 : 0;
  }
}
```

Вычислительная сложность этого алгоритма равна $O(n)$. С обоснованием данного факта можно ознакомиться, например, в [3].

Построение массива граней позволяет, в частности, более эффективно найти вхождения образца P в текст T. Введем новый символ $\#$, не принадлежащий алфавиту A, и составим строку $S = P\#T$. Для строки S по алгоритму *Prefix-Border-Array* за линейное время вычислим массив *bp*. Далее в этом массиве достаточно выбрать все значения, равные m (длине P). Индекс i каждого такого значения (уменьшенный на $m+1$ — длину левой "добавки" к тексту в S) укажет правую координату вхождения P в T.

Заметим, что при построенной таким образом строке S в алгоритме *Prefix-Border-Array* все обращения к граням меньшей длины (цикл *while*) будут приводить внутрь P. Поэтому для решения поставленной задачи достаточно сохранять лишь элементы массива *bp* с индексами $1, \ldots, m-1$, а другие его элементы на каждом шаге цикла *for* перезаписывать в скалярную переменную.

С учетом сказанного выше, вычислительная сложность данного алгоритма поиска всех вхождений равна $O(m+n)$.

Z-блоки представляют альтернативный метод описания структуры строк [1].

Определение. Для строки S и ее позиции $i > 0$ определяется значение $Z_i(S)$ (короче — Z_i) как длина наибольшей подстроки, начинающейся в i и совпадающей с префиксом строки S. Сама такая подстрока (иногда — ее длина) называется *Z-блоком*. При $i = 0$ полагают $Z_0(S) = 0$.

Например, если $S = AABCAABXAAZ$, то
$Z_4(S) = 3$ ($AAB\boldsymbol{C}\ldots AAB\boldsymbol{X}\ldots$),
$Z_5(S) = 1$ ($A\boldsymbol{A}\ldots A\boldsymbol{B}\ldots$),
$Z_6(S) = Z_7(S) = 0$,
$Z_8(S) = 2$ ($AA\boldsymbol{B}\ldots AA\boldsymbol{Z}$).

В [1] описан и обоснован линейный алгоритм нахождения массива Z-значений (другими словами — "Z-функции") заданной строки S.

Вычисление массива Z-блоков, подобно массиву граней, дает эффективный способ решения задачи поиска вхождений образца P в текст T. Как и ранее, определим символ $\#$, не принадлежащий исходному алфавиту, и построим строку $S = P\#T$. Для S за линейное время выполним алгоритм вычисления массива *zp*. В нем каждое значение $zp[i]$, равное m (то есть длине P), дает индекс i начальной координаты вхождения P в T(его необходимо скорректировать на m+1). Для поиска вхождений образца достаточно хранить лишь элементы массива *zp* с индексами 1, ..., $m-1$.

Сложность данного алгоритма — $O(m+n)$.

3. ВЗАИМНЫЕ ПРЕОБРАЗОВАНИЯ СХЕМ ОПИСАНИЯ СТРУКТУРЫ СТРОК

В предыдущем разделе были представлены две схемы описания структуры строк — массивы граней и Z-блоков. Показаны также примеры их непосредственного применения к оптимизации поиска вхождений образца в текст.

Нетрудно заметить, что эти два метода взаимосвязаны: оба массива содержат длины подстрок в S, совпадающих с префиксами S. Существенное отличие в том, что в zp отмечаются начальные индексы таких подстрок, а в bp — конечные. При этом справедливо следующее очевидное утверждение.

Замечание 2. Если для позиции $i-1$ строки S справедливо $bp[i-1]=r$, то $zp[i-r]\geqslant r$.

В фундаментальной монографии [1] Д. Гасфилд, автор концепции Z-блоков, формально обосновал возможность преобразования массива Z-блоков в массив граней с помощью линейного алгоритма. В нем не используются непосредственные сравнения символов исходной строки. Назовем эту задачу "прямой". Представленный результат автор монографии возвел в статус теоремы.

Обратное же преобразование массивов (решение "обратной" задачи) в [1] не представлено. Подобных рассмотрений, по-видимому, не было и в других общедоступных научно-технических печатных изданиях. Следует заметить, что решения обратной задачи, то есть алгоритмы получения массива Z-значений из массива граней, можно обнаружить в Интернете. Однако некоторые из них не являются непосредственными (например, предварительно осуществляется стадия восстановления варианта исходной строки S [5]), другие не имеют строгого обоснования [6–7]). Последний факт признает и сам автор публикации [6]. Алгоритм же [7] представляет собой модификацию алгоритма [6].

Обоснованный линейный алгоритм вычисления массива Z-блоков из массива граней впервые представлен в недавней публикации [4] автора настоящей статьи. Для указанной цели предложен и использован специальный подход. Он состоит в том, что за основу берется первоначальный алгоритм непосредственного нахождения массива Z-блоков по строке $S[3]$. Однако из этого алгоритма исключаются все сравнения символов исходной строки S. Вместо этих сравнений используются эквивалентные свойства массива граней bp (см. Замечание 1). Такой алгоритм работает за линейное время и имеет достаточно простое обоснование, с учетом обоснованности свойств исходного алгоритма.

Полученный в [4] положительный результат приводит к мысли о применении аналогичного подхода и к решению *прямой* задачи. Возьмем в качестве основы алгоритм *Prefix-Border-Array* (п. 2), который вычисляет массив граней непосредственно из строки S. В этом алгоритме используются сравнения пары символов этой строки, а именно — $S[i]$ и $S[bpRight]$, где i — некоторая текущая позиция в S, а $bpRight$ — наибольшая грань в позиции $i-1$, то есть $bpRight=bp[i-1]$. При этом величина $bp[i-1]$ уже получена на предыдущем шаге. Можно ли в описанной ситуации сопоставить символы $S[i]$ и $S[bpRight]$, используя вместо строки S массив Z-блоков, если он известен? Ответ дает следующее утверждение.

Лемма. Пусть i — некоторая позиция строки S, $r=bp[i-1]$ — наибольшая грань в позиции $i-1$, $zp[i-r]$ — величина Z-блока в позиции $i-r$. Тогда равенство $S[i]=S[r]$ эквивалентно неравенству $zp[i-r]>r$.

Доказательство. Предположим, что в условиях леммы справедливо $S[i]=S[r]$. Без учета этого равенства, применяя к условиям леммы замечание 2, имеем $zp[i-r]\geqslant r$. По определению Z-блока в позиции $i-r$ это означает, что символы $S[i-r+0]$, $S[i-r+1]$, …, $S[i-r+r-1]=S[i-1]$ попарно совпадают с соответствующими символами $S[0]$, $S[1]$, …, $S[r-1]$. Тогда дополнительное равенство $S[i]=S[r]$ продлевает указанные 2 подпоследовательности еще на одну позицию вправо, то есть увеличивает значение Z-блока в позиции $i-r$. Таким образом,

будет выполнено требуемое неравенство $zp[i-r] > r$.

Обратно, пусть выполнены условия леммы и имеет место неравенство $zp[i-r] > r$. Данное неравенство по определению Z-блока в позиции $i-r$ означает попарное совпадение соответствующих элементов приведенных выше 2-х последовательностей вплоть до символов $S[i]$ и $S[r]$, то есть имеем $S[i] = S[r]$.

Таким образом, утверждение леммы доказано.

Используя предыдущую лемму, заменим в алгоритме *Prefix-Border-Array* сравнения символов $S[i]$ и $S[bpRight]$ проверкой неравенства $zp[i-bpRight] > bpRight$. Получим следующий новый алгоритм преобразования массива Z-блоков в массив граней, по вычислительной сложности эквивалентный алгоритму *Prefix-Border-Array*.

```
ZP-to-BP (bp, zp, n)
{
  bp[0] = 0;
  for(i = 1; i < n; ++i)
  {// i - длина рассматриваемого префикса
    bpRight = bp[i-1]; //Позиция справа от предыдущей (i-1) грани
    while (bpRight && (zp[i-bpRight] <= bpRight) )
      bpRight = bp[bpRight - 1];
    bp[i] = (zp[i-bpRight] > bpRight) ? bpRight + 1 : 0;
  }
}
```

Поскольку базовый алгоритм *Prefix-Border-Array* ранее подробно обоснован и изучен ([2, 3]), то полученный на его основе алгоритм *ZP-to-BP* может составить конкуренцию известному алгоритму преобразования массива Z-блоков в массив граней [3] в плане простоты и читабельности. Кроме того, он имеет более простое обоснование.

ЗАКЛЮЧЕНИЕ

В настоящей статье рассмотрены две наиболее известные схемы предварительного анализа и описания структуры строк — массивы граней и Z-блоков. Обсуждены возможности их взаимного преобразования. Предложен и обоснован новый линейный алгоритм преобразования массива Z-блоков в массив граней, отличающийся простотой, читабельностью, наглядностью обоснования.

Представленные результаты могут быть применены при практическом исследовании больших символьных последовательностей, в частности, в области биоинформатики [8].

СПИСОК ЛИТЕРАТУРЫ

1. Гасфилд, Д. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология / Д. Гасфилд. — СПб : Невский диалект, 2003. — 654 с.

2. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. — М. : Вильямс, 2016. — 1328 с.

3. Смит, Б. Методы и алгоритмы вычислений на строках / Б. Смит. — М. : Вильямс, 2006. — 496 с.

4. Махортов, С. Д. О выразительных возможностях схем описания структуры строк / С. Д. Махортов // Программная инженерия. — 2018. — № 5. — С. 235–240.

5. Code Review. Prefix- and z-functions in C++ (string algorithms) [Электронный ресурс]. — URL: https://codereview.stackexchange.com/questions/87349/prefix-and-z-functions-in-c -string-algorithms

6. Codeforces. Переход между Z- и префикс- функциями [Электронный ресурс]. — URL: http://codeforces.ru/blog/entry/9612/

7. Университет ИТМО. Z-функция [Электронный ресурс]. — URL: https://neerc.ifmo.ru/wiki/index.php?title=Z-функция

8. Quantitative detection of low-abundance somatic structural variants in normal cells by high-throughput sequencing / W. Quispe-Tintaya, T. Gorbacheva, M. Lee et. al. // Nature Methods. — 2016. — № 13. — P. 584–586.

REFERENCES

1. Gusfield D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. [Gasfild D. Stroki, derev'ya i posledovatel'nosti v algoritmax: Informatika i vychislitel'naya biologiya]. SPb, 2003, 654 p.

2. Cormen T., Leiserson C., Rivest R., Stein C. Introduction to Algorithms. [Kormen T., Leyjzerson Ch., Rivest R., Shtayjn K. Algoritmy: postroenie i analiz]. Moscow, 2016, 1328 p.

3. Smyth B. Computing Patterns in Strings. [Smit B. Metody i algoritmy vychisleniyj na strokax]. Moscow, 2006, 496 p.

4. Makhortov S.D. On expressive possibilities of schemes for strings structure description. [Maxortov S.D. O vyrazitel'nyx vozmozhnostyax sxem opisaniya struktury strok]. *Programmnaya inzheneriya — Software Engineering*, 2018, no. 5, pp. 235–240.

5. Code Review: Prefix- and z-functions in C++ (string algorithms). Retrieved from https://codereview.stackexchange.com/questions/87349/prefix-and-z-functions-in-c-string-algorithms.

6. Codeforces: Conversion between Z-function and Prefix-function. Retrieved from http://codeforces.ru/blog/entry/9612/.

7. ITMO University: Z-function. Retrieved from https://neerc.ifmo.ru/wiki/index.php?title=Z-функция.

8. Quispe-Tintaya W., Gorbacheva T., Lee M., Makhortov S.D., Popov V.N., Vijg J., Maslov A.Y. Quantitative detection of low-abundance somatic structural variants in normal cells by high-throughput sequencing, Nature Methods, 2016, no. 13, pp. 584–586.

Махортов С. Д., заведующий кафедрой, д.ф.-м.н., Воронежский государственный университет, Воронеж, Российская федерация
E-mail: msd_exp@outlook.com

Makhortov S. D., Head of Department, Doctor of Science, Voronezh State University, Voronezh, Russian Federation
E-mail: msd_exp@outlook.com