Индексация и выявление символьных множеств в одномерных и двумерных словах

ДЖ. БЕЛАЗЗУГИ
Хельсинкский университет, Финляндия
e-mail: Djamal.Belazzougui@cs.helsinki.fi

Р. КОЛПАКОВ
Московский государственный университет им. М. В. Ломоносова
e-mail: foroman@mail.ru

М. РАФФИНО
Университет Париж VII им. Дени Дидро, Франция
e-mail: raffinot@liafa.univ-paris-diderot.fr

УДК 519.712.43

Ключевые слова: комбинаторные алгоритмы, сложность алгоритмов, буквенные составы, структуры данных.

Аннотация

В работе представлен детальный обзор результатов, полученных для решения сравнительно новой проблемы вычисления, индексации и выявления множеств различных символов, называемых буквенными составами, в фрагментах одномерных и двумерных символьных массивов, и поясняются основные идеи, используемые для получения этих результатов.

Abstract

D. Belazzougui, R. Kolpakov, M. Raffinot, Indexing and querying character sets in one- and two-dimensional words, Fundamentalnaya i prikladnaya matematika, vol. 20 (2015), no. 6, pp. 3—16.

We give a detailed review of results obtained for a relatively new problem of finding, indexing, and querying character sets, which are called fingerprints in fragments of one- and two-dimensional words, and explain basic ideas used for obtaining these results.

1. Одномерные буквенные составы

Пусть Σ — конечный упорядоченный алфавит из σ букв и $S = s_1 \dots s_n$ — строка из n букв над алфавитом Σ. Длина n строки S обозначается через $|S|$. Множество всех строк над Σ (включая пустую строку) обозначается через Σ^*. Для удобства мы упорядочим все буквы алфавита Σ, занумеровав их числами от 0 до $\sigma - 1$. Номер буквы a в алфавите Σ обозначается через $r_\Sigma(a)$. Подстрока $s_i s_{i+1} \dots s_j$ называется *фактором* строки S и обозначается через

Фундаментальная и прикладная математика, 2015, том 20, № 6, с. 3—16.

$S\langle i;j\rangle$. *Буквенный состав* $f(S)$ строки S — это множество всех различных букв, содержащихся в S. Аналогичным образом буквенный состав $f_S(i,j)$ — это множество $f(s_i s_{i+1}\dots s_j)$ всех различных букв, содержащихся в факторе $S\langle i;j\rangle$, при этом фактор $S\langle i;j\rangle$ называется *ареалом* буквенного состава $f_S(i,j)$. Ареал $S\langle i;j\rangle$ буквенного состава f называется *максимальным*, если

1) $s_{i-1}\notin f$ при $i>1$;
2) $s_{j+1}\notin f$ при $j<n$.

Отметим, что любой ареал буквенного состава однозначным образом расширяется до некоторого максимального ареала этого буквенного состава. Мы обозначаем через $\mathcal{F}(S)$ множество

$$\{f\subseteq\Sigma \mid \text{найдутся } i,\ j, \text{ для которых } f=f_S\langle i,j\rangle\}$$

всех различных буквенных составов факторов строки S, а через $\mathcal{L}(S)$ обозначаем множество всех максимальных ареалов для всех буквенных составов из $\mathcal{F}(S)$. Нетрудно показать, что $|\mathcal{F}(S)|\leqslant|\mathcal{L}(S)|\leqslant n|\Sigma|$.

В данной статье мы рассматриваем следующие три алгоритмические проблемы для заданной строки S.

1. Вычислить множество $\mathcal{F}(S)$.
2. Для заданного $f\subseteq\Sigma$ проверить, содержится ли f в $\mathcal{F}(S)$.
3. Для заданного $f\subseteq\Sigma$, если $f\in\mathcal{F}(S)$, найти все максимальные ареалы для f в строке S.

Эффективное решение рассматриваемых проблем имеет многочисленные приложения в различных областях: информационном поиске, вычислительной биологии, обработке текстов и т. д. Далее мы без ограничения общности полагаем, что в строке S содержатся все буквы алфавита Σ, т. е. $|\Sigma|\leqslant n$.

Первой задачей, возникающей в связи с изучением рассматриваемых проблем, является задача эффективного представления символьных множеств. Заметим, что любое множество $f\in\Sigma$ естественным образом представляется в виде двоичного массива F длины σ, называемого характеристическим набором f и определяемого следующим образом: для каждого $\alpha\in\Sigma$ выполняется $F[r_\Sigma(\alpha)]=1$, если $\alpha\in f$, и $F[r_\Sigma(\alpha)]=0$ иначе. Однако в наиболее интересном для приложений случае, когда алфавит Σ имеет большую мощность, такое представление требует слишком большого объёма памяти. Для более экономного в плане использования памяти представления символьных множеств в [2] был предложен новый метод, который мы будем называть методом наименований. В результате применения данного метода каждому буквенному составу из $\mathcal{F}(S)$ присваивается уникальный числовой идентификатор, называемый именем данного буквенного состава. Чтобы объяснить метод наименований, для простоты без ограничения общности будем полагать, что σ является степенью числа 2. Для вычисления имени произвольного буквенного состава f мы последовательно определяем $\log\sigma+1$ числовых наборов $F_0,F_1,\dots,F_{\log\sigma}$ (в данной работе все логарифмы рассматриваются по основанию 2). Набор F_0 является характеристическим набором f, состоящим только из значений 0 и 1. Каждый

набор F_i при $i > 0$ содержит вычисленные имена $2^{\sigma-i}$ последовательных фрагментов длины 2^i набора F_0, тем самым каждому имени фрагмента длины 2^i, содержащемуся в наборе F_i, соответствует содержащаяся в наборе F_{i-1} пара последовательных имён половинок этого фрагмента. Таким образом, вычисление набора F_i заключается в постановке в соответствие $2^{\sigma-i}$ последовательным парам имён из набора F_{i-1} новых имён так, что одинаковым парам соответствуют одинаковые имена. В итоге последний набор $F_{\log\sigma}$ содержит единственное имя, являющееся именем исходного буквенного состава f. На рис. 1 показан простой пример вычисления имени буквенного состава в случае $|\Sigma| = 8$.

<table>
<tr><td colspan="8">[7]</td></tr>
<tr><td colspan="4">[5]</td><td colspan="4">[6]</td></tr>
<tr><td colspan="2">[2]</td><td colspan="2">[2]</td><td colspan="2">[3]</td><td colspan="2">[4]</td></tr>
<tr><td>[1]</td><td>[0]</td><td>[1]</td><td>[0]</td><td>[1]</td><td>[1]</td><td>[0]</td><td>[0]</td></tr>
</table>

Рис. 1. Вычисление имени буквенного состава с характеристическим набором 10101100

Легко заметить, что метод наименований может быть применён для вычисления имён всех буквенных составов из $\mathcal{F}(S)$ так, чтобы всем одинаковым фрагментам характеристических наборов, используемым при вычислении этих имён, были присвоены одинаковые имена. В этом случае, если мы имеем два буквенных состава, различающихся вхождением только одной буквы, для вычисления имени f' исходя из имени f достаточно дополнительно вычислить не более чем σ новых имён для фрагментов характеристического набора f' (включая сам этот набор). С использованием этого наблюдения в [2] был предложен алгоритм решения проблемы 1 за время $O(n\sigma\log\sigma\log n)$. Для проблем 2 и 3 в [2] были предложены решения соответственно за время $O(\sigma\log n)$ и $O(|\Sigma|\log n + K)$, где K — размер выходных данных. В дальнейшем метод наименований был усовершенствован в [6]. Усовершенствование состоит в одновременном вычислении имён всех необходимых для определения имён буквенных составов из $\mathcal{F}(S)$ фрагментов одинаковой длины в характеристических наборах. Посредством данного усовершенствования время решения проблемы 1 может быть уменьшено до $O(\min\{n\sigma\log\sigma, n^2\})$.

Дальнейшие усиления результатов, связанных с решением рассматриваемых проблем, были получены в [7]. Одним из стимулов для этих усилений являлась независимость оценок, полученных в [2, 6] для времени решения рассматриваемых проблем, от мощности множеств $\mathcal{F}(S)$ и $\mathcal{L}(S)$, хотя для многочисленных известных семейств строк эти множества имеют сравнительно небольшую мощность. В качестве примера мы можем рассмотреть семейство строк $\{W_1, W_2, \ldots\}$, где W_k — строка над алфавитом $\Sigma_k = \{a_1, a_2, \ldots, a_k\}$, определяемая рекурсивными соотношениями $W_1 = a_1$ и $W_k = W_{k-1}a_kW_{k-1}$ для $k > 1$. Можно проверить, что $|W_k|\cdot|\Sigma_k| = k\cdot(2^k - 1)$ и $|\mathcal{L}(W_k)| = 2^{k+1} - (k+2)$, поэтому $|\mathcal{L}(W_k)| = o(|W_k|\cdot|\Sigma_k|)$. Чтобы описать алгоритм, предложенный в [7]

для решения проблемы 1, необходимо дать некоторые вспомогательные определения.

Без ограничения общности мы будем полагать, что исходная строка S не содержит двух последовательных вхождений одного и того же символа. Для дальнейшего удобства мы добавляем в качестве последнего символа в S некоторый специальный символ $s_{n+1} = \#$, не содержащийся в S, т. е. $S = s_1 \dots s_n \#_{n+1}$. Пусть i, j — две позиции в S, такие что $1 \leqslant i \leqslant j \leqslant n+1$. Обозначим через $\mathrm{fo}_S(i,j)$ строку, образованную конкатенацией крайних левых вхождений всех различных символов, содержащихся в факторе $S\langle i;j\rangle$, в порядке расположения этих вхождений слева направо. Например, если

$$S = a_1 b_2 a_3 c_4 e_5 a_6 b_7 a_8 c_9 d_{10} \#,$$

то $\mathrm{fo}_S(3,9) = aceb$ и $\mathrm{fo}_S(5,10) = eabcd$. Пусть i — некоторая позиция в S. Обозначим через j наименьшую позицию в S, такую что $i < j$ и $s_j = s_i$, если такая позиция существует, в противном случае положим $j = n+2$. Положим $\mathrm{lfo}_S(i) = \mathrm{fo}_S(i, j-1)$. Например, если

$$S = a_1 b_2 c_3 a_4 d_5 a_6 b_7 a_8 c_9 b_{10} e_{11} \#_{12},$$

то $\mathrm{lfo}_S(1) = abc$ и $\mathrm{lfo}_S(5) = dabce\#$. Через $\mathrm{lfo}'_S(i)$ мы обозначаем строку $\mathrm{lfo}_S(i)$ без последнего символа. Можно показать, что существует взаимно-однозначное соответствие между всеми максимальными ареалами из $\mathcal{L}(S)$ и префиксами всех строк $\mathrm{lfo}'_S(i)$, так что множество всех символов, содержащихся в префиксе, соответствующем максимальному ареалу, является буквенным составом данного максимального ареала. Таким образом, для выявления всех буквенных составов из $\mathcal{F}(S)$ можно вычислить все строки $\mathrm{lfo}_S(i)$ для S. Это может быть сделано последовательной обработкой всех букв строки S слева направо за время $O(|\mathcal{L}(S)| + n)$ (рис. 2). После этого мы можем вычислить за время $O\big((|\mathcal{L}(S)| + n)\log\sigma\big)$ имена всех буквенных составов из $\mathcal{F}(S)$, используя метод наименований из [6]. Общее время решения проблемы 1 составляет в таком случае $O\big((|\mathcal{L}(S)| + n)\log\sigma\big)$. Данная оценка времени решения проблемы 1 была усилена до $O(|\mathcal{L}(S)|\log\sigma + n)$ в [8].

В [7] были также усилены оценки для времени решения проблем 2 и 3 с помощью новой структуры данных, называемой *деревом буквенных составов*. Дерево буквенных составов строки S (обозначаемое через $\mathrm{FT}(S)$) — это бинарная древовидная структура, образованная характеристическими наборами всех буквенных составов из $\mathcal{F}(S)$, причём все совпадающие начальные сегменты этих наборов соединены вместе. Таким образом, каждому ребру дерева $\mathrm{FT}(S)$ соответствует некоторый сегмент характеристических наборов буквенных составов из $\mathcal{F}(S)$. Заметим, что такой сегмент может быть представлен в виде конкатенации суффикса U некоторого поименованного[1] фрагмента F_l и префикса V некоторого поименованного фрагмента F_r, при этом фрагменты F_l и F_r выбраны минимально возможными, тем самым $|U| > |F_l|$ и $|V| > |F_r|$. Таким образом,

[1] Фрагмент называется *поименованным*, если для него вычислено имя в процессе определения имён буквенных составов из $\mathcal{F}(S)$.

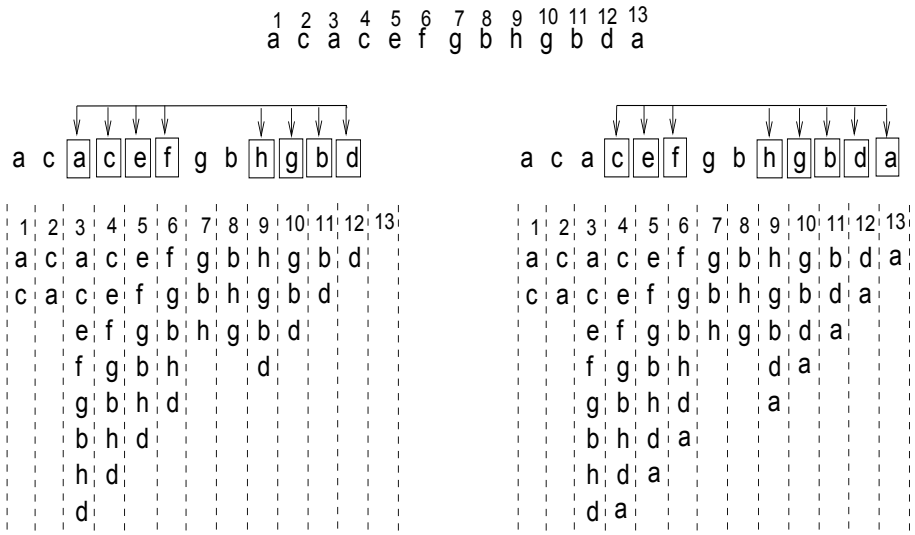

Рис. 2. Шаг обработки последней буквы a в процедуре вычисления строк $\mathrm{lfo}_S(i)$ для $S = acacefgbhgbda$. Мы добавляем символ a во вспомогательную таблицу, содержащую все строки $\mathrm{lfo}_S(i)$

сегмент, соответствующий ребру дерева $\mathrm{FT}(S)$, однозначно определяется набором чисел (n_l, n_r, l, r), где n_l, n_r — имена фрагментов F_l, F_r соответственно, а l, r — длины суффикса U и префикса V соответственно (рис. 3). Более того, исходя из набора (n_l, n_r, l, r) данный сегмент может быть вычислен за время $O(l + r)$, поэтому в дереве $\mathrm{FT}(S)$ соответствующее ребро помечается этим набором.

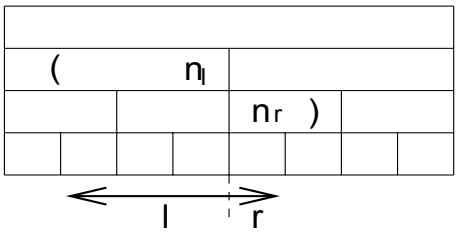

Рис. 3. Метка для ребра в $\mathrm{FT}(S)$. (n_l, n_r) — пара имён кратчайших смежных поименованных фрагментов, покрывающих суффикс U длины l и префикс V длины r

В [7] показано, что $\mathrm{FT}(S)$ может быть построено за время $O(|\mathcal{F}(S)| \log \sigma)$ и для заданного подмножества алфавита Σ можно проверить за время $O(\sigma)$, представлен ли характеристический набор этого подмножества в $\mathrm{FT}(S)$. Таким образом, проблемы 2 и 3 могут быть решены соответственно за время $O(\sigma)$ and $O(\sigma + K)$, где K — размер выходных данных.

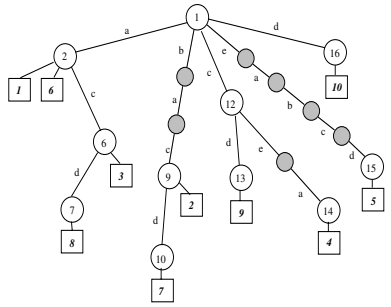

Рис. 4. Дерево участия для $S'' = abaceabacd\#$

Оценка времени сложности решения проблемы 1 в дальнейшем была усилена в [9]. Для описания этого результата введём следующие дополнительные понятия. Два максимальных ареала $S\langle i;j\rangle$ и $S\langle k;l\rangle$ в S будем называть *копиями*, если $s_i \dots s_j = s_k \dots s_l$. Например, в строке

$$S' = ab\underline{acadc}b\underline{acadc}e\underline{acdc}$$

подчёркнутые максимальные ареалы являются копиями. Заметим, что отношение между максимальными ареалами, являющимися копиями, является отношением эквивалентности, поэтому мы можем рассмотреть классы эквивалентности для этого отношения в множестве $\mathcal{L}(S)$. Обозначим через $\mathcal{L}_C(S)$ множество всех таких классов эквивалентности. Отметим, что мощность множества $\mathcal{L}_C(S)$ может быть существенно меньше мощности множества $\mathcal{L}(S)$. В качестве примера мы можем рассмотреть семейство строк $\{W'_1, W'_2, \dots\}$ над алфавитом $\{a_1, a_2, \dots\}$, где $W'_1 = a_1$ и $W'_k = W'_{k-1}(a_1 a_2 \dots a_k)^k$ при $k > 1$. Можно проверить, что

$$|W'_k| = \frac{1}{6}k(k+1)(2k+1),$$

$$|\mathcal{L}(W'_k)| = \frac{1}{12}k(3k^3 + 2k^2 - 9k + 16) = \Theta(|W'_k|^{4/3}),$$

$$|\mathcal{L}_C(W'_k)| = \frac{1}{6}k(k^2+5) = \Theta(|W'_k|),$$

поэтому $|\mathcal{L}_C(W'_k)| = o(|\mathcal{L}(W'_k)|)$.

Под *деревом участия* строки S (обозначаемым через $\mathrm{PT}(S)$) понимается древовидная структура данных, образованная всеми строками $\mathrm{lfo}'_S(i)$, $i = 1, 2, \dots, n$, где все общие начальные фрагменты этих строк соединены вместе (при этом для удобства все рёбра в $\mathrm{PT}(S)$ отмечены отдельными символами). На рис. 4 приведён пример дерева участия строки $S'' = abaceabacd\#$.

Поскольку все буквенные составы из $\mathcal{F}(S)$ соответствуют префиксам строк $\mathrm{lfo}'_S(i)$, дерево $\mathrm{PT}(S)$ является на самом деле сжатым представлением всех буквенных составов из $\mathcal{F}(S)$, т. е. для каждого буквенного состава f из $\mathcal{F}(S)$

в $\mathrm{PT}(S)$ существует хотя бы одна соответствующая вершина, такая что f является множеством всех символов, которыми помечены рёбра, содержащиеся в пути из корня $\mathrm{PT}(S)$ в данную вершину. С другой стороны, каждая вершина в $\mathrm{PT}(S)$ соответствует некоторому буквенному составу из $\mathcal{F}(S)$. Таким образом, множество $\mathcal{F}(S)$ может быть вычислено посредством построения дерева $\mathrm{PT}(S)$.

Чтобы построить $\mathrm{PT}(S)$, сначала мы строим суффиксное дерево строки S. Суффиксное дерево S (обозначаемое через $\mathrm{ST}(S)$) — это классическая древовидная структура данных, образованная всеми суффиксами строки S, где все общие префиксы этих суффиксов соединены вместе. Легко заметить, что каждому ребру дерева $\mathrm{ST}(S)$ соответствует некоторый фактор строки S, который однозначно определяется своими начальной и конечной позицией в S. Поэтому каждое ребро в $\mathrm{ST}(S)$ помечено начальной и конечной позициями некоторого фактора в S, соответствующего данному ребру. Таким образом, дерево $\mathrm{ST}(S)$ имеет размер $O(n)$ и может быть построено за время $O(n \log \sigma)$ (см., например, [10, 13, 14]). На рис. 5 приведён пример суффиксного дерева строки S''.

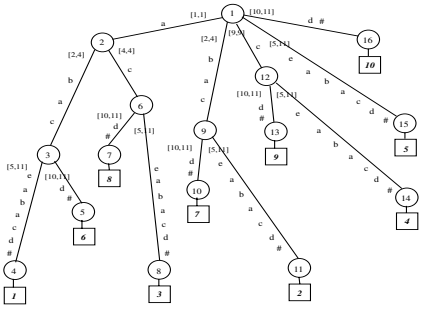

Рис. 5. Суффиксное дерево для $S'' = abaceabacd\#$

Для удобства под меткой ребра дерева $\mathrm{ST}(S)$ мы будем понимать соответствующий этому ребру фактор, рассматриваемый сам по себе как строка. Под *предком* ребра e из $\mathrm{ST}(S)$ понимается любое ребро, содержащееся в пути из корня $\mathrm{ST}(S)$ в ребро e (включая само ребро e), а под *главным предком* ребра e понимается первое ребро на этом пути. Пусть a', a'' — две (возможно, одинаковые) буквы, содержащиеся соответственно в метках некоторых рёбер e', e'' из $\mathrm{ST}(S)$. Мы говорим, что a' является *предком* a'', если либо e', e'' являются различными рёбрами и ребро e' является предком ребра e'', либо e', e'' являются одним и тем же ребром и буква a' находится перед буквой a'' в метке этого ребра. В этом случае a' называется *главным предком* a'', если e' является главным предком e'' и a' является первым символом в метке e'. Чтобы получить $\mathrm{PT}(S)$ из $\mathrm{ST}(S)$, мы удаляем из меток рёбер дерева $\mathrm{ST}(S)$ все символы, для которых имеются идентичные символы в качестве предков, и все символы, для которых имеются в качестве предков символы, идентичные главным предкам, но не являющиеся главными предками. Используя в качестве вспомогательных структур

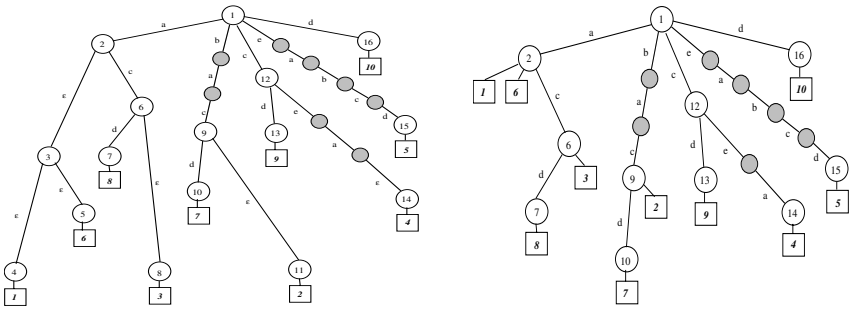

Рис. 6

АВЛ-деревья, посредством процедуры последовательной обработки меток рёбер из $\mathrm{ST}(S)$ в направлении от листьев к корню это может быть сделано за время $O(n \log \sigma + |\mathcal{L}_C(S)|)$. После этого мы удаляем в полученном дереве последние символы из меток всех висячих рёбер (под висячим ребром в данном случае понимается ребро с непустой меткой, являющееся предком, кроме самого этого ребра, только рёбер с пустыми метками) и все рёбра с пустыми метками посредством соединения концов этих ребер (очевидно, что данная процедура может быть произведена за время $O(n)$) и заменяем все рёбра, помеченные строками из нескольких символов, цепочками рёбер, помеченных отдельными символами (это может быть сделано за время $O(|\mathcal{L}_C(S)|)$. Для окончательного построения дерева $\mathrm{PT}(S)$ мы последовательно объединяем вместе в полученном дереве все рёбра, имеющие общие концевые вершины и помеченные одинаковыми символами, что может произведено за время $O(|\mathcal{L}_C(S)| \log \sigma)$. На рис. 6 проиллюстрировано вычисление $\mathrm{PT}(S'')$ из $\mathrm{ST}(S'')$ для строки S''. Таким образом, $\mathrm{PT}(S)$ может быть построено из $\mathrm{ST}(S)$ за общее время $O\big((n + |\mathcal{L}_C(S)|) \log \sigma\big)$, тем самым $\mathrm{PT}(S)$ может быть построено за время $O\big((n + |\mathcal{L}_C(S)|) \log \sigma\big)$. После этого за время $O(|\mathcal{L}_C(S)| \log \sigma)$ можно вычислить имена всех буквенных составов, представленных в $\mathrm{PT}(S)$, используя метод наименований из [6]. Таким образом, проблему 1 можно решить за время $O\big((n + |\mathcal{L}_C(S)|) \log \sigma\big)$.

Отметим также, что суффиксное дерево может быть смоделировано за время $O(n)$ посредством суффиксного массива [1], поэтому время построения $\mathrm{PT}(S)$ может быть уменьшено до $O(n + |\mathcal{L}_C(S)| \log \sigma)$. Таким образом, оценка для времени решения проблемы 1 была усилена в [3] до $O(n + |\mathcal{L}_C(S)| \log \sigma)$.

Дальнейшие продвижения в решении проблем 2 и 3 были сделаны в [5]. В данной работе авторы используют дополнительную структуру данных, называемую *деревом обратного поиска*. Построение данного дерева основано на сравнительно простом наблюдении, заключающемся в том, что в множестве $\mathcal{F}(S)$ для любого буквенного состава f найдётся (возможно, пустой) буквенный состав f', получающийся из f удалением ровно одного символа. Поэтому каждому буквенному составу из $\mathcal{F}(S)$ можно поставить в соответствие дугу

из f в f', помеченную удаляемым символом. Деревом обратного поиска называется полученное таким образом ориентированное дерево, корнем которого является пустой буквенный состав. Данное дерево можно построить за время $O(|\mathcal{L}(S)|)$. В [5] показано, что, применяя в качестве вспомогательных структур дерево обратного поиска и упрощённый вариант дерева буквенных составов («лекси-строковое трай-дерево») для строки S, можно решить проблему 2 (проблему 3) за время $O\bigl(|f|\log(\sigma/|f|)\bigr)$ с использованием $O(|\mathcal{F}(S)|)$ памяти (за время $O(|f|\log(\sigma/|f|)+K)$, где K — размер выходных данных, с использованием $O(|\mathcal{L}(S)|)$ памяти).

В [3] изучается применение техники хеширования для решения рассматриваемых проблем. Заметим, что при некотором выбранном подходящим образом простом числе P подмножества f алфавита Σ могут быть идентифицированы хеш-функциями

$$h_X(f)=\sum_{a\in f} X^{r_\Sigma(a)} \pmod P,$$

где $X\in\{1,2,\ldots,P-1\}$. Можно показать, что для любого натурального c значение $h_X(f)$ может быть вычислено за время $O(ct)$ с использованием предварительно вычисленного массива размера $O(c\sigma^{1/c})$. Пусть

$$H_P=\{h_X \mid X\in\{0,1,\ldots,P-1\}\}.$$

Тогда можно доказать следующий факт.

Лемма 1. *Для заданного семейства M, состоящего из m подмножеств алфавита Σ, выбранная случайным образом хеш-функция $h_X\in H_P$ при $P\geqslant m^2\sigma$ является инъективным отображением семейства M в множество $\{0,1,\ldots,P-1\}$ с вероятностью, не меньшей $1/2$.*

Мы также используем следующий результат, приведённый в [12].

Лемма 2. *Произвольное заданное кардинальное дерево, состоящее из N вершин над алфавитом размера σ, может быть представлено в виде структуры данных, использующей $N\bigl(\log\sigma+\log e+o(1)\bigr)$ битов памяти и позволяющей за константное время производить следующую операцию: для заданной вершины p и заданного символа a проверить, существует ли в кардинальном дереве ближайший потомок вершины p (смежная с p вершина, находящаяся на следующем уровне кардинального дерева), отмеченный символом a, и найти этот ближайший потомок, если он существует.*

Для эффективного решения проблем 2 и 3 мы используем древовидную структуру данных, которая получается из дерева участия склеиванием всех вершин, соответствующих одному и тому же буквенному составу. Нетрудно заметить, что данная структура, называемая *трай-деревом буквенных составов*, эквивалентна дереву обратного поиска, используемому в [5]. После выполнения процедуры вычисления имён всех буквенных составов из $\mathcal{F}(S)$ трай-дерево буквенных составов строки S может быть получено из $\mathrm{PT}(S)$ за время $O(|\mathcal{L}_C(S)|)$. Отметим, что имеется взаимно-однозначное соответствие между буквенными

составами из $\mathcal{F}(S)$ и вершинами трай-дерева буквенных составов строки S. Мы используем фактически два различных способа представления трай-дерева буквенных составов. Первым способом представления является *функция обратного поиска*, ставящая в соответствие каждому буквенному составу f из $\mathcal{F}(S)$ символ, являющийся в трай-дереве буквенных составов S меткой последнего ребра на пути из корня в вершину, соответствующую f. Пользуясь леммой 1, за время $O(|\mathcal{F}(S)|$ мы можем найти хеш-функцию h_X, такую что значения $h_X(f)$ для всех буквенных составов f из $\mathcal{F}(S)$ являются различными, и реализовать функцию обратного поиска как функцию, определённую на значениях $h_X(f)$ для буквенных составов f. Из результатов [11] вытекает, что данная реализация может быть представлена с использованием всего лишь $O\Big(|\mathcal{F}(S)|\log\sigma\big(1+o(1)\big)\Big)$ битов памяти так, что для любого значения $h_X(f)$, где $f \in \mathcal{F}(S)$, значение функции обратного поиска для f может быть выявлено за константное время (если $f \notin \mathcal{F}(S)$, то выявленное для f значение может быть любым). Второй способ представления трай-дерева буквенных составов, который называется *нисходящим преставлением*, является представлением в виде кардинального дерева, предложенным в лемме 2. Согласно лемме 2 данное представление трай-дерева буквенных составов S требует всего лишь $|\mathcal{F}(S)|\big(\log\sigma+\log e+o(1)\big)$ битов памяти и позволяет для любой строки U найти в этом трай-дереве за время $O(|U|)$ такую вершину, что U является строчкой символов, последовательно приписанных рёбрам, содержащимся в пути из корня трай-дерева в эту вершину, если такая вершина существует.

В [3] предложено следующее решение для проблемы 2. Пусть f — заданное подмножество алфавита Σ. Сначала мы вычисляем за время $O(|f|)$ значение $h_X(f)$ для f. На втором шаге работы алгоритма, исходя из значения $h_X(f)$, пользуясь реализацией функции обратного поиска, мы вычисляем за время $O(|f|)$ единственно возможную строку U, которая может быть строчкой символов, последовательно приписанных рёбрам трай-дерева буквенных составов, содержащимся в пути из корня в вершину, соответствующую единственному буквенному составу f' из $\mathcal{F}(S)$, такому что $h_X(f') = h_X(f)$, если f' существует. На третьем шаге мы сравниваем f с множеством всех символов, содержащихся в U. Это может быть сделано либо за время $O(k|f|)$ с использованием $O(\sigma^{1/k}\log\sigma)$ битов дополнительной памяти, где k — любое натуральное число, либо за среднее ожидаемое время $O(|f|)$ с использованием всего лишь $O(|f|\log\sigma)$ битов дополнительной памяти. Если f не является множеством всех символов, содержащихся в U, то мы можем сделать вывод, что $f \notin \mathcal{F}(S)$. В противном случае на последнем шаге мы производим в трай-дереве буквенных составов поиск такой вершины, что U является строкой символов, последовательно приписанных рёбрам, содержащимся в пути из корня в данную вершину (используя нисходящее представление трай-дерева буквенных составов, это можно сделать за время $O(|f|)$). Если такая вершина существует, мы получаем, что $f \in \mathcal{F}(S)$ (заметим, что в этом случае f является буквенным составом, соответствующим данной вершине), в противном случае $f \notin \mathcal{F}(S)$.

Общее время работы предложенного алгоритма решения проблемы 2 составляет $O(|f|)$.

Чтобы решить проблему 3, мы дополнительно присоединяем к каждой вершине нисходящего представления трай-дерева буквенных составов список всех максимальных ареалов для буквенного состава, соответствующего этой вершине. Тогда после нахождения на последнем шаге работы алгоритма вершины, соответствующей заданному буквенному составу f, мы можем выявить все максимальные ареалы для f за время $O(K)$, где K — число этих максимальных ареалов. Учитывая другие различные усиления оценок сложности решения рассматриваемых алгоритмов, предложенные в [3], результаты этой работы можно резюмировать следующим образом.

Проблема 1 может быть решена

— за время
$$O(n + |\mathcal{L}_C(S)| \log \sigma)$$
с использованием
$$O(|\mathcal{L}(S)| + |\mathcal{F}(S)| \log \sigma \log n)$$
битов памяти;

— за время
$$O(|\mathcal{L}(S)| \log \sigma)$$
с использованием
$$O(|\mathcal{F}(S)| \log \sigma \log n)$$
битов памяти;

— за среднее ожидаемое время
$$O(|\mathcal{L}(S)|)$$
с использованием
$$O(|\mathcal{F}(S)| \log n)$$
битов памяти при условии экстремально малой вероятности ошибки.

С использованием $|\mathcal{F}(S)|(2 \log \sigma + \log e)\big(1 + o(1)\big)$ битов дополнительной памяти проблема 2 может быть решена

— за время $O(|f|)$ при наличии $O(\sigma^{1/\varepsilon} \log n)$ битов рабочей памяти для любого положительного целого ε;

— за среднее ожидаемое время $O(|f|)$ при наличии $O(|f| \log n)$ битов рабочей памяти.

Оценки времени решения для проблемы 3 совпадают с вышеприведёнными оценками для проблемы 2, сложенными (с точностью до порядка) с размером K выходных данных.

2. Двумерные буквенные составы

Пусть M — двумерный массив над алфавитом Σ

$$\begin{array}{ccccc} a_{m1} & a_{m2} & \dots & a_{m(n-1)} & a_{mn} \\ a_{(m-1)1} & a_{(m-1)2} & \dots & a_{(m-1)(n-1)} & a_{(m-1)n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{21} & a_{22} & \dots & a_{2(n-1)} & a_{2n} \\ a_{11} & a_{12} & \dots & a_{1(n-1)} & a_{1n} \end{array}$$

где, без ограничения общности, $m \leqslant n$. Буквенным составом $f(M)$ массива M является множество $\{a \in \Sigma \mid$ найдутся i, j, такие что $a_{i,j} = a\} \subseteq \Sigma$ всех различных букв, содержащихся в M.

Пусть $1 \leqslant i' \leqslant i'' \leqslant m$ и $1 \leqslant j' \leqslant j'' \leqslant n$. Тогда через $\langle i', i''; j', j''\rangle$ мы обозначаем двумерный массив

$$\begin{array}{ccccc} a_{i''j'} & a_{i''(j'+1)} & \dots & a_{i''j''} \\ \vdots & \vdots & \ddots & \vdots \\ a_{(i'+1)j'} & a_{(i'+1)(j'+1)} & \dots & a_{(i'+1)j''} \\ a_{i'j'} & a_{i'(j'+1)} & \dots & a_{i'j''} \end{array},$$

который называется *прямоугольником* в M. Буквенный состав $f_M\langle i', i''; j', j''\rangle$ — это множество $f(\langle i', i''; j', j''\rangle)$ всех различных букв, содержащихся в прямоугольнике $\langle i', i''; j', j''\rangle$, при этом прямоугольник $\langle i', i''; j', j''\rangle$ называется *ареалом* буквенного состава $f_M\langle i', i''; j', j''\rangle$. Через $\mathcal{F}_{\mathrm{R}}(M)$ мы обозначаем множество

$$\{f \subseteq \Sigma \mid \text{найдутся } i',\ i'',\ j',\ j'', \text{ такие что } f = f_M\langle i', i''; j', j''\rangle\}$$

всех различных буквенных составов прямоугольников в M.

Прямоугольник $\langle i', i''; j', j''\rangle$ в M называется *максимальным ареалом* буквенного состава $f = f_M\langle i', i''; j', j''\rangle$, если этот прямоугольник не содержится в прямоугольнике большего размера с тем же самым буквенным составом. Примеры максимальных ареалов приведены на рис. 7.

Множество всех максимальных ареалов в M обозначается через $\mathcal{L}_{\mathrm{R}}(M)$. Можно показать, что $|\mathcal{F}_{\mathrm{R}}(M)| \leqslant |\mathcal{L}_{\mathrm{R}}(M)| \leqslant nm^2\sigma$.

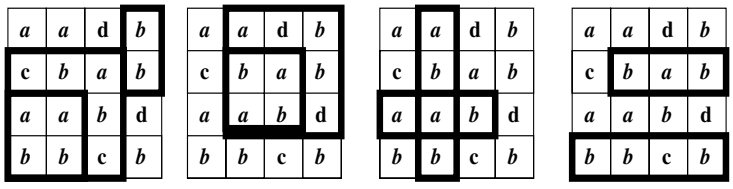

Рис. 7. Примеры максимальных ареалов (обведены жирными рамками) в двумерный массивах

Прямоугольник $\langle i', i''; j', j'' \rangle$ называется *квадратом*, если $i'' - i' = j'' - j'$. Через $\mathcal{F}_{\mathrm{S}}(M)$ мы обозначаем множество

$$\{f \subseteq \Sigma \mid \text{найдётся квадрат } \langle i', i''; j', j'' \rangle, \text{ такой что } f = f_M\langle i', i''; j', j'' \rangle\}$$

всех различных буквенных составов для квадратов в M. Квадрат в M называется *квадратным максимальным ареалом*, если он не содержится в квадрате большего размера с тем же самым буквенным составом. Через $\mathcal{L}_{\mathrm{S}}(M)$ мы обозначаем множество всех квадратных максимальных ареалов в M. Можно показать, что $|\mathcal{F}_{\mathrm{S}}(M)| \leqslant |\mathcal{L}_{\mathrm{S}}(M)| \leqslant nm\sigma$.

Заметим, что проблемы 1—3, рассматриваемые для факторов строк, могут также естественным образом быть рассмотрены как для прямоугольников, так и для квадратов в двумерный массивах. Проблемы 1—3, рассматриваемые для прямоугольников и квадратов в двумерных массивах, изучались в недавней работе [4], в которой были получены следующие результаты (K — размер выходных данных).

— Множество $\mathcal{F}_{\mathrm{R}}(M)$ может быть вычислено за время

$$O\left(nm^2\sigma \log\left(\frac{|\mathcal{L}_{\mathrm{R}}(M)|}{(nm^2)} + 2\right)\right)$$

или за среднее ожидаемое время $O(nm^2\sigma)$ с вероятностью ошибки, убывающей с полиномиальной скоростью.

— Множество $\mathcal{F}_{\mathrm{S}}(M)$ может быть вычислено за время

$$O\left(nm\sigma \log\left(\frac{|\mathcal{L}_{\mathrm{S}}(M)|}{nm} + 2\right)\right)$$

или за среднее ожидаемое время $O(nm\sigma)$ с вероятностью ошибки, убывающей с полиномиальной скоростью.

— Проблема 2 (проблема 3) для прямоугольников может быть решена за время $O(|f| + \log\log n)$ ($O(|f| + \log\log n + K)$) с использованием $O(nm \log n + |\mathcal{F}_{\mathrm{R}}(M)|)$ ($O(nm \log n + |\mathcal{L}_{\mathrm{R}}(M)|)$) битов дополнительной памяти.

— Проблема 2 (проблема 3) для квадратов может быть решена за время $O(|f| + \log\log n)$ ($O(|f| + \log\log n + K)$) с использованием $O(nm \log n + |\mathcal{F}_{\mathrm{S}}(M)|)$ ($O(nm \log n + |\mathcal{L}_{\mathrm{S}}(M)|)$) битов дополнительной памяти.

Работа частично поддержана Российским фондом фундаментальных исследований (грант 14-01-00598).

Литература

[1] Abouelhoda M. I., Kurtz S., Ohlenbusch E. Replacing suffix trees with enhanced suffix arrays // J. Discrete Algorithms. — 2004. — Vol. 2, no. 1. — P. 53—86.

[2] Amir A., Apostolico A., Landau G. M., Satta G. Efficient text fingerprinting via Parikh mapping // J. Discrete Algorithms. — 2003. — Vol. 1, no. 5-6. — P. 409—421.

[3] Belazzougui D., Kolpakov R., Raffinot M. Various improvements to text fingerprinting // J. Discrete Algorithms. — 2013. — Vol. 22. — P. 1—18.

[4] Belazzougui D., Kolpakov R., Raffinot M. Indexing and querying color sets of images // Theor. Comput. Sci. — 2016. — Vol. 647. — P. 74—84.

[5] Chan C.-Y., Yu H.-I., Hon W.-K., Wang B.-F. Faster query algorithms for the text fingerprinting problem // Inf. Comput. — 2011. — Vol. 209, no. 7. — P. 1057—1069.

[6] Didier G., Schmidt T., Stoye J., Tsur D. Character sets of strings // J. Discrete Algorithms. — 2007. — Vol. 5, no. 2. — P. 330—340.

[7] Kolpakov R., Raffinot M. New algorithms for text fingerprinting // Combinatorial Pattern Matching. 17th Annual Symp., CPM 2006, Barcelona, Spain, July 5—7, 2006. Proceedings. — Berlin: Springer, 2006. — (Lect. Notes Comput. Sci.; Vol. 4009). — P. 342—353.

[8] Kolpakov R., Raffinot M. New algorithms for text fingerprinting // J. Discrete Algorithms. — 2008. — Vol. 6, no. 2. — P. 243—255.

[9] Kolpakov R., Raffinot M. Faster text fingerprinting // Proc. 15th Int. Symp. on String Processing and Information Retrieval. — Berlin: Springer, 2009. — (Lect. Notes Comput. Sci.; Vol. 5280). — P. 15—26.

[10] McCreight E. M. A space-economical suffix tree construction algorithm // J. Algorithms. — 1976. — Vol. 23, no. 2. — P. 262—272.

[11] Porat E. An optimal bloom filter replacement based on matrix solving // CSR '09. Proc. Fourth Int. Comput. Sci. Symp. in Russia on Comput. Sci. — Theory and Applications. — Berlin: Springer, 2009. — (Lect. Notes Comput. Sci.; Vol. 5675). — P. 263—273.

[12] Raman R., Raman V., Rao Satti S. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets // ACM Trans. Algorithms. — 2007. — Vol. 3, no. 4. — Art. no. 43.

[13] Ukkonen E. Constructing suffix trees on-line in linear time // Proc. IFIP 12th World Comput. Congr. on Algorithms, Software, Architecture — Information Processing '92. Vol. 1. — Amsterdam: North-Holland, 1992. — P. 484—492.

[14] Weiner P. Linear pattern matching algorithm // SWAT '73 Proc. 14th Annual Symp. on Switching and Automata Theory. — Washington: IEEE Comput. Soc., 1973. — P. 1—11.