МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ имени М. В. ЛОМОНОСОВА

На правах рукописи

Стариковская Татьяна Андреевна

Эффективные алгоритмы для некоторых задач обработки слов

Специальность 01.01.06 — математическая логика, алгебра и теория чисел

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата физико-математических наук

Москва 2013

Работа выполнена на кафедре математической логики и теории алгоритмов Механико-математического факультета Московского государственного университета имени М. В. Ломоносова.

Научный руководитель:	академик РАН, профессор Семёнов Алексей Львович
Официальные оппоненты:	Ройтберг Михаил Абрамович, доктор физико-математических наук (Институт математических проблем биологии РАН, зав. лабораторией прикладной математики)
	Горбунов Константин Юрьевич, кандидат физико-математических наук (Институт проблем передачи информации имени А.А. Харкевича РАН, старший научный сотрудник)
Ведущая организация:	ФГБОУ ВПО «Санкт-Петербургский государственный университет»

Защита диссертации состоится 22 марта 2013 г. в 16 часов 45 минут на заседании диссертационного совета Д 501.001.84 при Московском государственном университете имени М. В. Ломоносова по адресу: Российская Федерация, 119991, Москва, ГСП-1, Ленинские горы, д. 1, МГУ имени М. В. Ломоносова, Механико-математический факультет, аудитория 14-08.

С диссертацией можно ознакомиться в Фундаментальной библиотеке Московского государственного университета имени М. В. Ломоносова (Ломоносовский проспект 27, сектор А, 8-й этаж).

Автореферат разослан 22 февраля 2013 года.

Учёный секретарь диссертационного
совета Д 501.001.84 при МГУ,
доктор физико-математических наук,
профессор

Иванов Александр Олегович

Общая характеристика работы

Диссертация посвящена построению эффективных алгоритмов для некоторых задач обработки слов. В диссертации исследуются различные варианты задачи о поиске образца и задачи о наибольших общих подсловах. Кроме того, рассматривается задача о построении разложения Лемпеля — Зива.

Актуальность темы. Задача о поиске образца является одной из основных задач обработки слов. Постановка этой задачи звучит следующим образом: перечислить начальные позиции всех вхождений слова P (образца) длины $|P|$ в слово T длины n. Вхождение слова P в слово T — это подслово слова T, совпадающее с P.

Первый алгоритм с временем работы $O(|P| + n)$ был предложен в 1970 г. Джеймсом Моррисом и Воном Праттом[1]. Алгоритм Морриса и Пратта использует $O(|P|)$ ячеек памяти. Алгоритм Морриса и Пратта сперва получает на вход образец P и обрабатывает его, после чего для любого слова T длины n алгоритм может вычислить все вхождения P в T за $O(n)$ времени.

С 1970 г. было разработано более 80 алгоритмов, решающих задачу о поиске образца в случае, когда образец известен заранее[2]. Классическими идеями алгоритмов решения задачи о поиске образца в этом случае являются применение сравнений букв, применение автоматов, применение операций над двоичными векторами и применение фильтрации.

Не менее интересным случаем этой задачи является такой, когда заранее известно слово T. Для решения задачи в этом случае сперва строится так называемый текстовый индекс слова T, который в компактном виде хранит информацию о всех подсловах T. После того, как индекс построен, алгоритм может эффективно вычислять вхождения P в T для любого образца P.

Основными текстовыми индексами являются суффиксные деревья, определенные Питером Вайнером в 1973 г.[3], и суффиксные массивы, определение которых дали Джин Майерс и Уди Манбер в 1990 г.[4]. На рис. 1

[1] J. H. Morris, V. R. Pratt. A linear pattern-matching algorithm. *Technical report 40*. University of California, Berkley, 1970.

[2] S. Faro, T. Lecroq. The exact string matching problem: a comprehensive experimental evaluation. *Computing Research Repository*, 2010.

[3] P. Weiner. Linear pattern matching algorithms. *Proceedings of the 14th Annual Symposium on Foundations of Computer Science*. New York, NY: IEEE Computer Society, 1973. — P. 1–11.

[4] U. Manber, G. Myers. Suffix arrays: a new method for on-line string searches. *Proceedings of the 1st ACM-SIAM Symposium on Discrete algorithms,* ed. by D.S. Johnson.

Индекс	Память	Время построения	Время работы
Суфф. дерево	$O(n)$	$O(n)$[3] [5] [6] [7]	$O(\|P\| + output)$
Суфф. массив	$O(n)$	$O(n)$[8] [9]	$O(\|P\| + \log n + output)$

Рис. 1: Время работы алгоритмов, вычисляющих все вхождения образца P в текст T длины n, где *output* — количество вхождений.

приведены вычислительные сложности алгоритмов вычисления вхождений образца при помощи суффиксных деревьев и суффиксных массивов.

Заметим, что для хранения слова T длины n над алфавитом Σ размера σ достаточно всего $O(n \log \sigma)$ бит (в случае, когда слово хранится в сжатом виде, и того меньше), а для хранения суффиксного дерева или суффиксного массива необходимо $O(n \log n)$ бит памяти. Так как память, используемая алгоритмами, по-прежнему является существенным ограничением для практических приложений, то в этом направлении ведутся активные исследования (см. обзор [10]).

В диссертации вводится новый текстовый индекс, основой которого выступает так называемое r-разреженное суффиксное дерево[11], где r — произвольно задаваемый параметр. В качестве вычислительной модели рассматривается РАМ-машина с размером ячейки памяти w. Доказывается, что в этой модели вычислений указанный текстовый индекс занимает $O(\frac{nw}{r})$ бит памяти и позволяет найти начальные пози-

Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1990. — P. 319–327.

[5] E.M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM,* Vol. **23**, № 2. New York, NY, USA: ACM, 1976. — P. 262–272.

[6] M. Farach. Optimal suffix tree construction with large alphabets. *Proceedings of the 38th Annual Symposium on Foundations of Computer Science.* New York, NY: IEEE Computer Society, 1997. — P. 137–143.

[7] E. Ukkonen. Constructing suffix trees on-line in linear time. *Proceedings of the 12th World Computer Congress on Algorithms*, Vol. **1**. Amsterdam, The Netherlands: North-Holland Publishing Co., 1992. — P. 484–492.

[8] J. Kärkkäinen, P. Sanders. Simple linear work suffix array construction. *Proceedings of the 30th International Colloquium on Automata, Languages and Programming,* ed. by J.C.M. Baeten, J.K. Lenstra, J. Parrow, G.J. Woeginger. *Lecture Notes in Computer Science,* Vol. **2719**. Berlin etc.: Springer, 2003. — P. 943–955.

[9] S.J. Puglisi, W.F. Smyth, A.H. Turpin. A taxonomy of suffix array construction algorithms. *ACM Comput. Surv.,* Vol. **39**, № 2. New York, NY, USA: ACM, 2007.

[10] G. Navarro, V. Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.,* Vol. **39**, № 1. New York, NY, USA: ACM, 2007.

[11] J. Kärkkäinen, E. Ukkonen. Sparse suffix trees. *Proceedings of the 2nd Annual International Computing and Combinatorics Conference,* ed. by J.-Y. Cai, C. K. Wong. *Lecture Notes in Computer Science*, Vol. **1090**. Berlin etc.: Springer, 1996. — P. 219–230.

ции вхождений образца P длины $|P| \geq r$ в текст T длины n за время $O(|P| \cdot \max\{1, \frac{r \log \sigma}{w}\} + \max\{output, r\} \cdot \log n / \log \log n)$, где $output$ — количество вхождений P в T. В частности, при $r = O(\frac{w}{\log \sigma})$ предложенный текстовый индекс занимает $O(n \log \sigma)$ бит памяти (меньше, чем суффиксное дерево или суффиксный массив) и позволяет перечислить начальные позиции вхождений образца P длины $|P| \geq r$ в текст T длины n за время $O(|P| + \max\{output, \frac{w}{\log \sigma}\} \cdot \log n / \log \log n)$.

Рассмотрим обобщение задачи о поиске образца. Пусть дано множество слов $S = \{T_1, T_2, \ldots, T_m\}$. Задача состоит в построении текстового индекса для этого множества слов, используя который мы бы могли эффективно перечислить все вхождения образца P в слово $T_\ell \in S$ или найти количество вхождений образца P в слово $T_\ell \in S$.

Как мы уже говорили, первая подзадача может быть решена на суффиксном дереве для слова T_ℓ за $O(|P| + output)$ времени, для решения второй подзадачи потребуется $O(|P|)$ времени.

В диссертации изучается специальный случай этой задачи — задача о перекрестном поиске образца, когда образец является подсловом одного из слов $T_1, T_2, \ldots, T_m$. В этом случае для определенной выше задачи можно предъявить решение с линейной памятью и временем запроса либо не зависящим от длины образца, либо зависящим слабо (как двойной логарифм от длины). Также в диссертации описывается динамический текстовый индекс для указанной задачи, который можно эффективно изменять при добавлении слова в множество S.

Следующей задачей, которая рассматривается в данной работе, является задача о вычислении разложения Лемпеля — Зива. Разложение Лемпеля — Зива слова T — это разбиение $T = f_1 f_2 \ldots f_z$, где подслово f_i, $1 \leq i \leq z$, является либо буквой, не встречавшейся до этого, либо самым длинным начальным отрезком слова $f_i \ldots f_z$, входящим в $f_1 f_2 \ldots f_i$ хотя бы дважды. Подслова f_i называются факторами разложения Лемпеля — Зива[14] [15].

Разложение Лемпеля — Зива имеет множество приложений, например, сжатие данных (разложение Лемпеля — Зива используется в таких архиваторах как gzip, WinZip, и PKZIP). Кроме того, разложение Лемпеля —

[14] M. Crochemore. Transducers and repetitions. *Theor. Comput. Sci.,* Vol. **45**, № 1. Essex, UK: Elsevier Science Publishers Ltd., 1986. — P. 63–68.

[15] J. Ziv, A. Lempel. A universal algorithm for sequential data compression. *Transactions on Information Theory*, Vol. **23**, № 3. New York, NY: IEEE Computer Society Press, 1977. — P. 337–343.

Зива является основой нескольких алгоритмов[16] [17] и текстовых индексов. Поэтому, разработка эффективных алгоритмов построения разложения Лемпеля — Зива является важной и актуальной задачей.

Пусть T — слово длины n над алфавитом размера σ. Известно несколько алгоритмов, вычисляющих разложение Лемпеля — Зива с использованием $O(n \log n)$ бит памяти. Основой этих алгоритмов служат суффиксные деревья[18], суффиксные автоматы[14] и суффиксные массивы[19] [20] [21] [22] [23] [24].

Тем не менее, известны только два алгоритма, использующие $O(n \log \sigma)$ бит памяти[25] [26]. Идеи алгоритмов похожи (в частности, оба алгоритма используют FM-индекс[27] и сжатый суффиксный массив). Ал-

[16] R. Kolpakov, G. Kucherov. Finding maximal repetitions in a word in linear time. *Proceedings of the 40th Symposium on Foundations of Computer Science*. New York, NY: IEEE Computer Society, 1999. — P. 596–604.

[17] D. Gusfield, J. Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *J. Comput. Syst. Sci.,* Vol. **69**, № 4. Orlando, FL, USA: Academic Press, Inc., 2004. — P. 525–546.

[18] M. Rodeh, V.R. Pratt, S. Even. Linear algorithm for data compression via string matching. *J. ACM,* Vol. **28**, № 1. New York, NY, USA: ACM, 1981. — P. 16–24.

[19] M. I. Abouelhoda, S. Kurtz, E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays.*J. of Discrete Algorithms*, Vol. **2**, № 1. Amsterdam, The Netherlands: Elsevier Science Publishers B. V., 2004. — P. 53–86.

[20] G. Chen, S.J. Puglisi, W.F. Smyth. Lempel — Ziv factorization using less time & space. *Mathematics in Computer Science,* Vol. **1**, № 4. Birkhauser Basel, 2008. — P. 605–623.

[21] M. Crochemore, L. Ilie. Computing longest previous factor in linear time and applications. *Inf. Process. Lett.,* Vol. **106**, № 2. Amsterdam, The Netherlands: Elsevier North-Holland, Inc., 2008. — P. 75–80.

[22] M. Crochemore, L. Ilie, C.S. Iliopoulos, M. Kubica, W. Rytter, T. Walen. LPF computation revisited. Proceedings of the 2nd International Workshop on Combinatorial Algorithms, ed. by J. Fiala, J. Kratochvíl, M. Miller. *Lecture Notes in Computer Science,* Vol. **5874**. Berlin etc.: Springer, 2009. — P. 158–169.

[23] M. Crochemore, L. Ilie, W.F. Smyth. A simple algorithm for computing the Lempel — Ziv factorization. *Proceedings of the 18th Data Compression Conference*. New York, NY: IEEE Computer Society, 2008. — P. 482–488.

[24] E. Ohlebusch, S. Gog. Lempel — Ziv factorization revisited. *Proceedings of the 22nd annual conference on Combinatorial Pattern Matching,* ed. by R. Giancarlo, G. Manzini. *Lecture Notes in Computer Science,* Vol. **6661**. Berlin etc.: Springer, 2011. — P. 15–26.

[25] D. Okanohara, K. Sadakane, Kunihiko. An online algorithm for finding the longest previous factors. *Proceedings of the 16th Annual European Symposium on Algorithms,* ed. by D. Halperin, K. Mehlhorn. *Lecture Notes in Computer Science,* Vol. **5193**. Berlin etc.: Springer, 2008. — P. 696–707.

[26] E. Ohlebusch, S. Gog. Lempel-Ziv factorization revisited. *Proceedings of the 22nd annual conference on Combinatorial Pattern Matching,* ed. by R. Giancarlo, G. Manzini. *Lecture Notes in Computer Science,* Vol. **6661**. Berlin etc.: Springer, 2011. — P. 15–26.

[27] P. Ferragina, G. Manzini. Opportunistic Data Structures with Applications. *Proceedings of the 31rd Annual Symposium on Foundations of Computer Science*. New York, NY: IEEE Computer Society, 2000, — P. 390–398.

горитм, предложенный Энно Охлебушем и Саймоном Гогом в 2011 г., предполагает, что слово T известно заранее, время его работы составляет $O(n)$[26].

Время работы алгоритма[25], разработанного Дайсуке Оканохара и Кунихико Садаканом в 2008 г., довольно большое, $O(n \log^3 n)$. Его достоинство по сравнению с алгоритмом[26] состоит в том, что он вычисляет разложение Лемпеля — Зива онлайн, т.е. одновременно с чтением слова T. Рассмотрим факторы $f_1, f_2, \ldots, f_i$ разложения Лемпеля — Зива слова T. Разложение Лемпеля — Зива слова Ta, где a — буква, содержит либо i, либо $i+1$ фактор: в первом случае это факторы $f_1, f_2, \ldots, f_{i-1}, f'_i$, где фактор $f'_i = f_i a$; а во втором случае — $f_1, f_2, \ldots, f_i, f_{i+1}$, где $f_{i+1} = a$. Алгоритм[25] читает T слева направо и после прочтения каждой новой буквы обновляет разложение Лемпеля — Зива, т.е. или увеличивает длину последнего фактора на единицу, или добавляет новый фактор.

Для многих практических приложений, работающих с большими объемами данных, было бы естественно разрешить обновлять разложение Лемпеля — Зива не так часто, например, только после прочтения $r > 1$ букв, с целью уменьшения времени работы алгоритма. К сожалению, прямолинейное применение этой идеи к алгоритму[25] не позволяет получить более быстрый алгоритм.

В диссертации был разработан новый алгоритм с памятью $O(n \log \sigma)$ бит, который достигает разумного компромисса между временем работы и частотой обновления разложения Лемпеля — Зива. Алгоритм обновляет разложение Лемпеля — Зива слова T каждые $\frac{\log_\sigma n}{4}$ букв. Время работы алгоритма равно $O(n \log^2 n)$.

Также в диссертации рассматриваются две задачи о максимальных и минимальных общих подсловах. Предположим, что дано множество слов $S = \{T_1, T_2, \ldots, T_m\}$ суммарной длины n. Максимальным общим подсловом для заданного d будем называть слово W, входящее в, по меньшей мере, d различных слов множества S, такое, что любое слово Wa, где a — произвольная буква алфавита, встречается в менее чем d словах множества S. Минимальные общие подслова для заданного d определяются аналогично: слово W называется минимальным общим подсловом для d и S, если W встречается в не более чем d словах множества S, а любой его начальный отрезок — в более чем d словах.

Предположим, что даны образец P и число $d \leq m$. Первая задача состоит в вычислении максимальных общих подслов для d, а вторая задача — в вычислении минимальных общих подслов для d. В обеих задачах нас будут интересовать только те подслова, которые начинаются с образца P (образец может быть и пустым словом в том числе).

В качестве примера рассмотрим слова $T_1 = ababa$, $T_2 = aabbba$, $T_3 = bbabcb$. Максимальные общие подслова для $d = 2$ (и пустого образца P) — это ab, bab и bba. Заметим, что ab входит в три слова, но любое из слов aba, abb, abc входит только в одно из слов T_1, T_2, T_3. Минимальные общие подслова для $P = b$ и $d = 2$ — это bab и bb.

Решения определенных выше задач используют обобщенное суффиксное дерево для слов $T_1, T_2, \ldots, T_m$. Для первой из задач мы предлагаем решение с оптимальным временем работы $O(|P| + output)$. Для второй задачи мы предлагаем решение со временем работы $O(|P| + \log\log n + output)$, сводя задачу к известной задаче вычислительной геометрии. В каждой из оценок времени *output* обозначает количество слов, которые будут выданы алгоритмом. Алгоритмы не выписывают слова явно, так как это могло бы занять слишком много времени, а представляют их в некотором компактном виде.

Последняя задача состоит в вычислении неточного наибольшего общего подслова двух слов. Определим d-неточное наибольшее общее подслово слов T_1 и T_2 как наибольшее по длине подслово слова T_1, для которого существует подслово слова T_2, отличающееся от него в не более чем d буквах. Существует несколько подходов к решению задачи о вычислении d-неточного наибольшего общего подслова двух слов, одним из которых является динамическое программирование. С его помощью можно построить алгоритм, использующий время и память, пропорциональные произведению длин слов T_1 и T_2[12].

В диссертации приводится алгоритм для нахождения 1-неточного наибольшего общего подслова слов T_1 и T_2 с временем работы $O(|T_1| \cdot |T_2|)$. Предположим, что длина T_2 существенно больше длины T_1. Описываемый алгоритм читает слово T_2, большее по длине, несколько раз, но только слева направо, начиная с первой буквы. Помимо памяти, требующейся для хранения слов, алгоритм дополнительно использует $O(|T_1|)$ памяти.

Условие на чтение слова только слева направо, начиная с первой буквы, позволяет уменьшить время работы на практике, если слово T_1 хранится в РАМ-памяти компьютера, а T_2 — на жестком диске.

Цель работы. Построение текстового индекса на основе разреженного суффиксного дерева. Построение эффективного алгоритма для различных вариантов задачи о перекрестном поиске образца. Разработка онлайн алгоритма построения разложения Лемпеля-Зива, использующего линейную память. Исследование задачи об 1-неточном наибольшем общем подслове. Построение эффективных алгоритмов для решения задачи о максимальных и минимальных общих подсловах для заданного d.

Научная новизна. Результаты диссертации являются новыми и получены автором диссертации самостоятельно. Основные результаты диссертации состоят в следующем:

1. Определен новый текстовый индекс, на основе которого разработан эффективный по памяти алгоритм поиска вхождений образца в слово.
2. Разработан эффективный алгоритм решения задачи о перекрестном поиске образца.
3. Разработан новый алгоритм вычисления разложения Лемпеля — Зива, вычисляющий разложение почти одновременно с чтением слова и использующий небольшой объем памяти.
4. Разработан алгоритм, эффективно вычисляющий 1-неточное наибольшее общее подслово двух слов.
5. Сформулированы задачи о максимальных и минимальных общих подсловах для заданного d и предложены эффективные алгоритмы их решения.

Методы исследования. В работе применяются методы из области алгоритмов обработки слов и вычислительной геометрии.

Теоретическая и практическая ценность. Работа имеет теоретический характер. Результаты, полученные в диссертационной работе, имеют приложения в области алгоритмов обработки слов, в биоинформатике, обработке текстов, распознавании речи, компьютерном зрении.

Апробация диссертации. Результаты диссертации докладывались на следующих семинарах и конференциях:

- на международной конференции «Сжатие, передача и обработка данных» (Compression, Communications and Processing 2011), Палинуро, Италия, 21—24 июня 2011 года;
- на международной конференции «Комбинаторные алгоритмы для решения задачи о поиске образца» (Combinatorial Pattern Matching 2012), Хельсинки, Финляндия, 3—5 июля 2012 года;
- на международной конференции «Математические основы информатики» (Mathematical Foundations of Computer Science 2012), Братислава, Словакия, 27—31 августа 2012 года;
- на международной конференции «Обработка слов и информационный поиск» (String Processing and Information Retrieval 2012), Картахена, Колумбия, 21—25 октября 2012 года;

- в 2011 г. на семинаре «Алгоритмические вопросы алгебры и логики» под руководством академика РАН С.И. Адяна, Москва, Россия;
- в 2008—2011 гг. на Колмогоровском семинаре по сложности вычислений и сложности определений под руководством д.ф.-м.н. Н.К. Верещагина, к.ф.-м.н. А.Е. Ромащенко, академика РАН А.Л. Семёнова, к.ф.-м.н. А.Х. Шеня, Москва, Россия;
- в 2011—2012 гг. на семинаре «Комбинаторная оптимизация и теория алгоритмов» под руководством к.ф.-м.н. М.А. Бабенко, Москва, Россия.

Публикации. Основные результаты диссертации опубликованы в пяти работах [1]—[5], из них пять — в периодических изданиях из перечня ВАК.

Структура диссертации. Работа состоит из введения, четырех глав, содержащих 22 раздела, и списка литературы. Библиография содержит 67 наименований. Текст диссертации изложен на 94 страницах.

Содержание работы

Глава 1 является вспомогательной. В ней вводятся необходимые для дальнейшего изложения понятия и формулируются ранее известные результаты.

В **разделе 1.1** вводятся понятия алфавита, слова и лексикографического порядка на словах. Под словом понимается конечная упорядоченная последовательность элементов конечного непустого множества, называемого алфавитом. Элементы этого множества называются буквами.

Буквы слова T длины n обозначаются через $T[1], T[2], \ldots, T[n]$. Подслово слова T, начинающееся в позиции i и заканчивающееся в позиции j (включая позиции i и j) обозначается через $T[i..j]$. Индексы, равные 1 или n, опускаются. Слово $T[..j]$ называется префиксом слова T, а слово $T[i..]$ — суффиксом слова T.

Предполагается, что алфавит является упорядоченным множеством, и любые две буквы можно сравнить за $O(1)$ времени. Порядок, заданный на буквах, может быть естественным образом продолжен на множество слов. Будем говорить, что слово T_1 лексикографически меньше слова T_2, если выполняется одно из следующих условий:

1) существует число $1 \leq i < \min\{|T_1|, |T_2|\}$ такое, что $T_1[1..i] = T_2[1..i]$, а $T_1[i+1] < T_2[i+1]$,

2) T_1 является префиксом T_2.

В **разделе 1.2** дается определение суффиксного дерева[1]. Суффиксное дерево слова T длины n — это ориентированное дерево с корнем, имеющее ровно n листьев, занумерованных от 1 до n. Каждая внутренняя вершина, отличная от корня, имеет не менее двух детей, а каждое ребро помечено непустым подсловом слова $T\$$, где $\$$ — буква, не входящая в слово T. Никакие два ребра, выходящих из одной и той же вершины, не могут иметь меток, начинающихся с одного и того же символа. Наконец, если идти вдоль пути от корня до листа с номером i и читать метки вслух, то будет произнесен в точности суффикс $T[i..]\$$ слова $T\$$. В **разделе 1.2.1** объясняется, как решать задачу о поиске образца с помощью суффиксного дерева. В **разделе 1.2.2** перечисляются основные алгоритмы построения суффиксного дерева и приводится описание алгоритма Укконена построения суффиксного дерева.

В **разделе 1.3** дается определение суффиксного массива. Пусть T — произвольное слово длины n. Рассмотрим его суффиксы $T[1..], T[2..], \ldots, T[n]$ и упорядочим их лексикографически. Суффиксный массив SA слова T — это массив длины n, в ячейке $SA[i]$ которого записана начальная позиция i-ого в лексикографическом порядке суффикса слова T, $1 \leq i \leq n$. Очевидно, SA однозначно определяется по T.

В **разделе 1.4** объясняется, как понятия суффиксного дерева и суффиксного массива обобщаются на случай множества слов.

Глава 2 посвящена задаче о поиске образца. В **разделе 2.1.1** описывается новый текстовый индекс. Основой текстового индекса выступает так называемое r-разреженное суффиксное дерево. Фиксируем произвольное число r и разобьем слово T длины n на блоки по r букв. Будем хранить в суффиксном дереве только те суффиксы, которые начинаются на границах блоков. В этом случае у суффиксного дерева будет не более n/r листьев, и, тем самым, $O(n/r)$ вершин. Такое определение r-разреженного суффиксного дерева впервые было рассмотрено в работе[11].

r-разреженное суффиксное дерево позволяет легко находить вхождения образца, начинающиеся на границах блоков. Рассмотрим образец P, где $|P| \geq r$. Для того, чтобы найти вхождения P в T, используется следующая идея. Сперва при помощи суффиксного дерева вычисляются вхождения суффиксов $P[1..], P[2..], \ldots, P[r..]$, начинающиеся на границах блоков в T. Затем вычисляются вхождения $P[1..k]$, $k = 1..r-1$, заканчивающиеся на границах блоков. Наконец, вычисляются границы блоков, которые являются одновременно последней позицией вхождения $P[1..k]$

и первой позицией вхождения $P[k+1..]$ для одного и того же k. Очевидно, эти границы соответствуют вхождениям P в T. В **разделе 2.1.2** приводится подробное описание этого алгоритма и доказывается

Теорема 3. *Поиск всех вхождений образца P длины $|P| \geq r$ в слово T требует $O(|P| \cdot \max\{1, \frac{r \log \sigma}{w}\} + \max\{output, r\} \cdot \log n / \log \log n)$ времени и $O(\frac{n}{r})$ памяти, где $output$ — количество вхождений.*

В **разделе 2.1.3** описывается алгоритм построения r-разреженного суффиксного дерева, а также доказывается верхняя оценка на время его работы.

Теорема 4. *r-разреженное суффиксное дерево для слова T длины n может быть построено за $O(nr)$ времени при использовании $O(\frac{n}{r})$ памяти.*

В **разделе 2.2** изучается задача о перекрестном поиске образца. В **разделе 2.2.1** приводится постановка задачи о взвешенных предках, которая используется далее для построения эффективных алгоритмов. В **разделе 2.2.2** доказываются

Теорема 5. *Пусть дано множество слов $S = \{T_1, T_2, \ldots, T_m\}$ суммарной длины n. Для любых $1 \leq k, \ell \leq m$ и $1 \leq i \leq j \leq |T_k|$, количество вхождений $T_k[i..j]$ в T_ℓ может быть вычислено за $O(t + \log\log m)$ времени, где $t = \min\{\sqrt{\log output / \log\log output}, \log\log(j-i+1)\}$, а $output$ обозначает количество вхождений. Структуры данных, которые используются алгоритмом, занимают $O(n)$ памяти и могут быть построены за $O(n)$ времени.*

Теорема 6. *Пусть дано множество слов $S = \{T_1, T_2, \ldots, T_m\}$ суммарной длины n. Все вхождения $T_k[i..j]$ в T_ℓ могут быть перечислены за $O(\log\log m + output)$ времени, где $output$ — количество вхождений. Используемая структура данных занимает $O(n)$ памяти и может быть построена за $O(n)$ времени.*

В **разделе 2.2.3** описывается динамический текстовый индекс для задачи о перекрестном поиске образца. А именно, доказывается следующая теорема:

Теорема 7. *Пусть дано множество слов $S = \{T_1, T_2, \ldots, T_m\}$ суммарной длины n. Существует динамическая структура данных, позволяющая вычислять количество вхождений $T_k[i..j]$ в T_ℓ за $O(\log n)$ времени, а перечислять вхождения — за $O(\log n + output)$ времени, где $output$ — количество вхождений. Эта структура данных занимает $O(n)$ памяти. Время ее обновления при добавлении нового слова в множество S составляет $O(\log n)$ на букву.*

В **Главе 3** изучается задача построения разложения Лемпеля — Зива слова T над алфавитом размера σ. Разложение Лемпеля — Зива слова T — это разбиение $T = f_1 f_2 \dots f_z$, где подслово f_i, $1 \le i \le z$, является либо буквой, не встречавшейся до этого, либо самым длинным префиксом $f_i \dots f_z$, входящим в $f_1 f_2 \dots f_i$ хотя бы дважды[14 15]. Подслова f_i называются факторами разложения Лемпеля — Зива. Предъявленный алгоритм вычисляет разложение Лемпеля — Зива одновременно с чтением слова T. А именно, он читает T слева направо и после прочтения очередных $\frac{\log_\sigma n}{4}$ букв обновляет разложение Лемпеля — Зива.

В **разделе 3.1** дается описание используемых структур данных и алгоритма. В **разделе 3.2** объясняются детали реализации структур данных. В **разделе 3.3** доказываются оценки времени работы алгоритма и объема используемой им памяти:

Теорема 9. *Предъявленный алгоритм вычисляет разложение Лемпеля — Зива слова T длины n за $O(n \log^2 n)$ времени, используя при этом $O(n \log \sigma)$ бит памяти.*

Глава 4 посвящена задачам об общих подсловах множества слов. Определим d-неточное наибольшее общее подслово слов T_1 и T_2 как наибольшее по длине подслово слова T_1, для которого существует подслово слова T_2, отличающееся от него в не более чем d буквах. В разделе 4.1 рассматривается задача о вычислении 1-неточного наибольшего общего подслова слов T_1 и T_2. Доказывается следующая теорема:

Теорема 10. *1-неточное наибольшее общее подслово слов T_1 и T_2 длин n_1 и n_2 соответственно ($n_2 \ge n_1$) может быть вычислено за время $O(n_1 n_2)$ при использовании $O(n_1)$ памяти (не считая памяти, требуемой для хранения самих слов). Алгоритм читает слово T_2 только слева направо, начиная с первой буквы.*

Пусть W_1 — 1-неточное наибольшее общее подслово T_1 и T_2, а W_2 — подслово T_2, которое отличается от W_1 в не более чем одной букве. Обозначим совпадающие части слов W_1 и W_2 через W' и W''. Пусть позиции несовпадающих букв в словах W_1 и W_2 — это позиция i в T_1 и позиция j в T_2. Очевидно, W' — наибольший общий суффикс префиксов слов $T_1[..i-1]$ и $T_2[..j-1]$, а W'' — наибольший общий префикс суффиксов слов $T_1[i+1..]$ и $T_2[j+1..]$. В **разделе 4.1.2** описывается процедура, вычисляющая наибольшие общие префиксы слова $T_1[i..]$ и слов $T_2[j..]$, $j = 1, 2, \dots, n_2$. В **разделе 4.1.3** описывается процедура, вычисляющая наибольшие общие суффиксы слова $T_1[i..]$ и слов $T_2[j..]$, $j = 1, 2, \dots, n_2$. В **разделе 4.1.1** объясняется, как указанные процедуры используются для вычисления W_1.

Раздел 4.2 посвящен задаче о минимальных и максимальных общих подсловах. Предположим, что дано множество слов $S = \{T_1, T_2, \ldots, T_m\}$ суммарной длины n. Максимальным общим подсловом для заданного d будем называть слово W, входящее в, по меньшей мере, d различных слов множества S, такое, что любое слово Wa, где a — произвольная буква алфавита, встречается в менее чем d словах множества S. Минимальные общие подслова для заданного d определяются аналогично: слово W называется минимальным общим подсловом для d и S, если W встречается в не более чем d словах множества S, а любой его префикс — в более чем d словах.

Пусть дан образец P и число $d \leq m$. В разделе 4.2.1 рассматривается задача вычисления максимальных общих подслов для заданного d, являющихся расширениями образца P вправо, т.е., начинающихся с образца P.

Теорема 11. *Пусть дано множество слов $S = \{T_1, T_2, \ldots, T_m\}$ суммарной длины n. Для любого образца P и числа d, максимальные общие подслова для d, являющиеся расширениями образца P вправо, могут быть перечислены за $O(|P| + output)$ времени, где $output$ — количество таких продолжений. Используемые структуры данных занимают $O(n)$ памяти и могут быть построены за $O(n)$ времени, где n — суммарная длина слов в множестве S.*

Раздел 4.2.2 посвящен задаче о поиске минимальных общих подслов для заданного d, являющихся расширениями образца P вправо. Эта задача может быть сведена к известной задаче вычислительной геометрии — задаче о пересечении перпендикулярных отрезков.

Теорема 12. *Пусть дано множество слов $S = \{T_1, T_2, \ldots, T_m\}$ суммарной длины n. Для любого образца P и числа d, минимальные общие подслова для d, являющиеся расширениями образца P вправо, могут быть перечислены за $O(|P| + \log\log n + output)$ времени. Используемая структура данных занимает $O(n)$ памяти и может быть построена за $O(n \log n)$ времени.*

Автор выражает глубокую благодарность своему научному руководителю академику РАН Алексею Львовичу Семёнову, а также академику РАН Сергею Ивановичу Адяну, к.ф.-м.н. Максиму Александровичу Бабенко, к.ф.-м.н. Андрею Альбертовичу Мучнику (1958 — 2007) за постановку задач и многочисленные обсуждения работы. Автор благодарен всем сотрудникам кафедры математической логики и теории алгоритмов за создание творческой доброжелательной атмосферы.

Работы автора по теме диссертации

[1] М. А. Бабенко, Т. А. Стариковская. Вычисление длиннейшей общей подстроки с одной ошибкой. *Пробл. передачи информ.,* Том 47, № 1 (2011), 33–39.

[2] R. Kolpakov, G. Kucherov, T. Starikovskaya. Pattern Matching on Sparse Suffix Trees. *Proceedings of the 1st International Conference on Data Compression, Communications and Processing.* New York, NY: IEEE Computer Society Press, 2011. — P. 92–97.

[3] G. Kucherov, Y. Nekrich, T. Starikovskaya. Computing discriminating and generic subwords. *Proceedings of the 19th International Symposium on String Processing and Information Retrieval,* ed. by E. Chávez, N. Ziviani. *Lecture Notes in Computer Science,* Vol. **7608**. Berlin etc.: Springer, 2012. — P. 307–317.

[4] G. Kucherov, Y. Nekrich, T. Starikovskaya. Cross-document pattern matching. *Proceedings of the 23rd Symposium on Combinatorial Pattern Matching,* ed. by J. Kärkkäinen, J. Stoye. *Lecture Notes in Computer Science,* Vol. **7354**. Berlin etc.: Springer, 2012. — P. 196–207.

[5] T.A. Starikovskaya. Computing Lempel-Ziv factorization online. *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science,* ed. by V. Sassone, P. Widmayer. *Lecture Notes in Computer Science,* Vol. **7464**. Berlin etc.: Springer, 2012. — P. 789-799.

В [1] М.А. Бабенко принадлежит постановка задачи, автору диссертации принадлежат описание алгоритма и доказательство оценок времени работы и используемой памяти.

В [2] автору принадлежат описания и доказательства оценок сложности для процедуры LeftSearch и алгоритма построения разреженного суффиксного дерева.

В работе [3] автору принадлежат доказательства теорем 1 и 2.

В совместной работе [4] автору диссертации принадлежит постановка задачи, а также теоремы 2, 3 и 4.