Федеральное государственное автономное образовательное учреждение
высшего образования «Уральский федеральный университет имени первого
Президента России Б.Н.Ельцина»

На правах рукописи

Меркурьев Олег Андреевич

Эффективные строковые алгоритмы в модели потока данных

05.13.17 — теоретические основы информатики

АВТОРЕФЕРАТ

диссертации на соискание ученой степени
кандидата физико-математических наук

Екатеринбург 2020

Работа выполнена на кафедре алгебры и фундаментальной информатики Института естественных наук и математики федерального государственного автономного образовательного учреждения высшего образования «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»

Научный руководитель: доктор физико-математических наук, профессор **Шур Арсений Михайлович**

Официальные оппоненты:

Колпаков Роман Максимович, доктор физико-математических наук, ФГБОУ ВО «Московский государственный университет имени М. В. Ломоносова», профессор кафедры дискретной математики механико-математического факультета

Хачай Михаил Юрьевич, доктор физико-математических наук, профессор РАН, ФГБУН «Институт математики и механики им. Н. Н. Красовского» Уральского отделения РАН, заведующий отделом математического программирования

Пузынина Светлана Александровна, кандидат физико-математических наук, ФГБОУ ВО «Санкт-Петербургский государственный университет», доцент факультета математики и компьютерных наук

Защита диссертации состоится “09” июля 2020 года в 11:00 часов на заседании диссертационного совета УрФУ 01.01.07, по адресу: 620000, г. Екатеринбург, пр. Ленина 51, к. 248, Зал заседаний диссертационных советов.

С диссертацией можно ознакомиться в библиотеке и на сайте ФГАОУ ВО «Уральский федеральный университет имени первого Президента России Б. Н. Ельцина», https://dissovet2.urfu.ru/mod/data/view.php?d=12&rid=1237.

Автореферат разослан “___” июня 2020 г.

Учёный секретарь
диссертационного совета,
кандидат физ.-мат. наук

Косолобов Д. А.

Общая характеристика работы

Актуальность темы. Строка — один из центральных объектов компьютерных наук. Любая последовательность данных может рассматриваться как строка (последовательность символов). Такое абстрактное представление, не учитывающее структуру обрабатываемых данных, имеет очень широкую область применений — от поиска в базах данных до анализа структуры ДНК и белков. Раздел компьютерных наук, занимающийся алгоритмами обработки строк, называется *стрингология* (англ. stringology от string — строка). Название было предложено в 1985 в работе [19]. Этой динамически развивающейся в настоящее время области посвящен ряд монографий [13, 15, 24, 42] и специализированных конференций с высоким уровнем цитируемости: Combinatorial Pattern Matching (CPM), String Processing and Information Retrieval (SPIRE) и др. Задачи, рассматриваемые в стрингологии, можно сгруппировать в несколько кластеров: поиск по образцу (pattern matching), индексирование данных, сжатие данных и алгоритмы на сжатых представлениях, а также поиск регулярных структур, таких как повторы, палиндромы, периодические фрагменты и различные их обобщения. Последнему кластеру принадлежат задачи, рассматриваемые в диссертации.

Одной из активно исследуемых областей являются строковые алгоритмы в модели потока данных. В этой модели элементы последовательности данных поступают один за одним и не могут быть сохранены в связи с малым объёмом доступной памяти. Первые работы, рассматривающие подобные ограничения, появились в 1970-е годы [39]. При этом понятие модели потока данных было сформулировано в работе [1], авторы которой получили в 2005 г. премию Гёделя за основополагающие исследования в области потоковых алгоритмов. Интересной особенностью модели потока данных является то, что в ней зачастую удаётся получить нижние оценки на вычислительную сложность задач, в первую очередь на необходимый объём памяти.

Первый значительный результат о строковых алгоритмах в модели потока данных был получен в [40]. В последнее десятилетие модель потока данных является заметным трендом в стрингологии. В настоящей работе рассматриваются строковые задачи, связанные с поиском регулярных структур в потоке данных.

Степень разработанности темы исследования. В одной из первых работ в потоковой модели [39] Мунро и Патерсон рассмотрели задачи сортировки потока и вычисления k-ой порядковой статистики, а также медианы как важного частного случая. Среди прочих результатов в статье показывается, что для вычисления медианы детерминированному алгоритму требуется линейная память, при этом представлен рандомизированный алгоритм, вычисляющий медиану с высокой вероятностью и использующий память объёмом $\mathcal{O}(\sqrt{n})$. Вообще, необходимость рандомизированных алгоритмов является характерной чертой модели потока

данных.

В ключевой в области работе Алона, Матиаса и Сегеди [1] рассматривалась задача вычисления k-х частотных моментов потока, где k-й частотный момент равен сумме k-х степеней частот встретившихся в потоке символов. В частности, 0-й частотный момент — это количество различных символов, 1-й — это длина потока, 2-й — это квадрат евклидовой нормы вектора частот. В статье показано, что для приближения k-х частотных моментов при $k \geq 6$ требуется линейная память. А вот 0-й, 1-й и 2-й моменты могут быть вычислены с логарифмической. При этом 0-й и 2-й моменты могут быть вычислены только приближённо и только с использованием рандомизированных алгоритмов. Рандомизированному алгоритму, вычисляющему точное значение, или детерминированному алгоритму, решающему задачу приближённо, уже понадобится линейная память.

Для задачи приближения количества различных элементов в потоке Бар-Йосеф и др. [5] представили позже более эффективный рандомизированный алгоритм.

Другие интересные результаты, полученные в модели потока данных, относятся к задачам на графах. В этом случае элементы потока рассматриваются как новые или удаляемые рёбра графа. Например, Кейн и др. [26] рассматривали задачу приближённого подсчёта числа вхождений фиксированного подграфа константного размера в большой граф, заданный потоком. Обзор задач на графах в модели потока данных представлен в работе Макгрегора [36].

Изучение алгоритмов на строках в модели потока данных началось со статьи Пората и Пората [40], в которой рассматривалась задача поиска по образцу в модели потока данных. В ней был представлен алгоритм, который предварительно обрабатывает образец, а затем, используя $\mathcal{O}(\log m)$ памяти (где m это длина образца), обрабатывает каждый символ текста за $\mathcal{O}(\log m)$, и с высокой вероятностью находит все вхождения образца в текст. И уже через два года Галил и Бреслауэр [8] представили алгоритм реального времени, то есть обрабатывающий каждый символ потока за $\mathcal{O}(1)$, который решает задачу поиска по образцу в полностью потоковой постановке задачи (во входном потоке сначала записан образец и после него через разделитель записан текст) с использованием $\mathcal{O}(\log m)$ памяти.

Более сложные задачи поиска также рассматриваются в модели потока данных. Один из примеров — это задача поиска с k ошибками. В этой задаче позиция считается вхождением образца, если расстояние Хэмминга между образцом и соответствующей подстрокой текста не превосходит k. Первый результат по этой задаче был представлен в статье [40]. После серии улучшений Клиффорд, Кочумака и Порат [12] представили алгоритм для полностью потоковой версии задачи, использующий память объёмом $\mathcal{O}(k \cdot \mathrm{poly}(\log n))$ и обрабатывающий один символ потока за $\mathcal{O}(\sqrt{k} \cdot \mathrm{poly}(\log n))$. Сходными задачами, рассмотренными

в потоковой модели, являются поиск многих образцов [10], приближенный поиск образца [11] и параметризованный поиск образца [25].

Кроме того, в потоковой модели рассматривались задачи поиска регулярных структур в строках. Так, отправной точкой для работы над данной диссертацией послужила статья Беренбринк и др. [6], авторы которой представили потоковые алгоритмы для приближенного поиска палиндромов в строке.

Цели и задачи диссертации. Целью диссертации является анализ вычислительной сложности поиска регулярных структур в строках в модели потока данных. Для достижения этой цели в диссертации рассмотрены следующие комбинаторные задачи:

- Задача LPS (длиннейшая палиндромная подстрока, longest palindromic substring): дан поток S, найти длину и начальную позицию самой длинной подстроки в S, являющейся палиндромом.
- Задача LRS (длиннейший повтор, longest repeating substring): дан поток S, найти длину наибольшей строки w, которая имеет больше одного вхождения в S, и начальные позиции её двух вхождений.
- Задача LRRS (длиннейший обратный повтор, longest repeating reversed substring): дан поток S, найти длину наибольшей строки w, такой что w и $\overleftarrow{w}$ имеют вхождения в S, и начальные позиции вхождений w и $\overleftarrow{w}$.
- Задача Runs (максимальные периодические подстроки): дан поток S, найти все максимальные периодические подстроки в S.

Научная новизна. Все представленные в диссертации результаты являются новыми.

Теоретическая и практическая значимость работы. Диссертация является теоретической работой; ее результаты могут применяться для дальнейших исследований в области строковых алгоритмов, для разработки алгоритмов обработки реальных потоков данных, а также для чтения специальных курсов по алгоритмам и структурам данных.

Методы исследования. В работе используются методы теории алгоритмов, теории сложности, разнообразные структуры данных.

Исследование вычислительных возможностей модели состоит из двух частей: поиск нижних оценок, то есть нахождение ограничений при которых задача не имеет решения, и поиск верхних оценок, то есть построение эффективных алгоритмов. Задача может считаться полностью исследованной, когда верхняя и нижняя оценки совпадают, то есть для каждого возможного набора ограничений или приведён удовлетворяющий им алгоритм, или доказано его отсутствие.

Доказательство нижних границ в работе основано на принципе Яо. Для построения эффективных алгоритмов в работе используются хэши Карпа–Рабина, а также различные структуры данных, в том числе словари, суффиксные деревья, бинарные индексные структуры [2] и динамическое взвешенное дерево предков [31]. При доказательстве корректности и вычислительной сложности алгоритмов активно используется комбинаторика слов.

Положения, выносимые на защиту.

- Алгоритмы типа Монте-Карло приближенного решения задачи LPS и доказательства их корректности и вычислительной сложности (алгоритмы A, M и M'; теоремы 2.3.1, 2.3.2 и 2.3.3).
- Теорема о нижних оценках для точных и приближенных алгоритмов типа Лас-Вегас для задачи LRS (теорема 3.6.1) и теорема о нижней оценки для алгоритма типа Монте-Карло для приближенного решения задачи LRS с аддитивной погрешностью (теорема 3.6.2).
- Алгоритмы типа Монте-Карло приближенного решения задач LRS и LRRS и доказательства их корректности и вычислительной сложности (алгоритмы ARR, FastARR и FastAR; теоремы 3.4.1, 3.4.2 и 3.5.2).
- Теорема о нижней оценке для точного потокового алгоритма, решающего задачу midSquare (теорема 4.2.1).
- Алгоритм типа Монте-Карло решения приближенной задачи approxRuns и доказательство его корректности и вычислительной сложности (алгоритм R; теорема 4.1.1).

Степень достоверности и апробация результатов. Все представленные в диссертации результаты снабжены строгими математическими доказательствами, обеспечивающими их достоверность.

Все представленные результаты опубликованы в статьях автора [22, 23, 37, 38] (в соавторстве с Арсением Михайловичем Шуром, две из них в соавторстве с Павлом Гавриховским и Пшемыславом Узнаньским) в журнале Алгоритмика (Algorithmica, Springer) и в сборниках следующих международных конференций: 27-й ежегодный симпозиум по комбинаторному поиску образцов (CPM 2016, Тель-Авив, Израиль), 30-й ежегодный симпозиум по комбинаторному поиску образцов (CPM 2019, Пиза, Италия), 26-й международный симпозиум по обработке строк и извлечению информации (SPIRE 2019, Сеговия, Испания). Все названные сборники вышли в сериях Leibniz International Proceedings in Informatics (LIPIcs), изд-во Schloss Dagstuhl – Leibniz-Zentrum für Informatik, и Lecture Notes in Computer Science (LNCS), изд-во Springer; они удовлетворяют достаточному условию ВАК для опубликования результатов диссертации. Кроме указанных

конференций, результаты также докладывались на конференции «Проблемы теоретической информатики 2019» (ВШЭ совместно с ПОМИ РАН, Москва), конкурсе студенческих работ по теоретической информатике и дискретной математике им. Алана Тьюринга (СПбАУ РАН, Санкт-Петербург, 2016, 1 место) и на семинаре «Алгебраические системы» (УрФУ, рук. Шеврин Л. Н. 2019–2020). В следующем разделе после краткого изложения каждого результата указывается, в какой именно статье он опубликован.

Личный вклад автора. Из работ [22, 23] в диссертацию включены алгоритмы A, M, M' и теоремы об их корректности и вычислительной сложности, принадлежащие автору. А.М. Шуру в этих статьях принадлежит алгоритм E, не вошедший в диссертацию, а П. Гавриховскому и П. Узнаньскому — доказательство нижних границ. В статьях [37, 38] основные результаты принадлежат автору (А.М. Шуру принадлежат постановка задач и оптимизация некоторых доказательств).

Объем и структура диссертационной работы. Диссертация состоит из введения, трёх глав, заключения, списка литературы, списка иллюстраций и списка таблиц. Объём диссертации составляет 75 страниц. Библиографический список содержит 59 наименований.

Основное содержание работы

Глава 1 (Введение) содержит предварительные сведения. В ней дается обзор исследований по тематике диссертации и формулируются основные задачи, решаемые в диссертации. Далее дается обзор диссертации по главам, сводка ее основных результатов, сведения об их апробации и список публикаций автора по теме диссертации.

Приведём определения, необходимые для дальнейшего изложения.

Строки. *Строка* длины n — это отображение $w : \{1, \dots, n\} \to \Sigma$, где Σ — это некоторое конечное множество, называемое *алфавитом*. Элементы алфавита Σ называются символами. Длина строки w обозначается через $|w|$. В работе мы ограничиваемся рассмотрением целочисленных алфавитов полиномиального размера от длины строки[1], то есть $\Sigma = \{1, 2, \dots, \sigma\}$, и при этом $\sigma \in \mathcal{O}(\mathrm{poly}(n))$. Через $w[1], w[2], \dots, w[n]$ обозначаются 1-й, 2-й, …, n-й символы строки w соответственно. Для произвольных целых чисел i и j, таких что $1 \le i \le j \le n$, обозначим $w[i..j] = w[i]w[i+1]\dots w[j]$; в частности, $w = w[1..n]$. Строка $\overleftarrow{w} = w[n]\dots w[2]w[1]$ называется строкой, *обратной* к w.

Строка $p = w[i..j]$ называется *подстрокой* строки w. В этом случае у строки p

[1]Такой тип алфавита используется в строковых алгоритмах наиболее часто; полиномиальное ограничение выполняется для большинства практических видов данных и позволяет реализовывать некоторые алгоритмы быстрее, чем в общем случае (например, сортировку массива за линейное время).

есть *вхождение* в w в позиции i. Подстроки вида $w[1..j]$ называются *префиксами* строки, а подстроки вида $w[i..n]$ называются *суффиксами*.

Палиндромы. *Палиндромом* называется строка, совпадающая с обратной к себе ($w = \overleftarrow{w}$). Ясно, что если строка w является палиндромом и $|w| > 2$, то строка $w[2..|w|-1]$ также является палиндромом. Рассматривая подстроки w, являющиеся палиндромами, будем говорить, что $w[i..j]$ и $w[h..k]$ имеют один и тот же *центр*, если $i + j = h + k$. Так для заданного центра можно определить наибольший палиндром, и тогда все меньшие подстроки, имеющие тот же центр, будут палиндромами.

Повторы. *Повтор* в строке w — это пара равных подстрок $w[i..i+l-1] = w[j..j+l-1]$, где $i < j$. Такой повтор будем обозначать тройкой (i, j, l). *Обратный повтор* — это пара подстрок, обратных друг к другу, т.е. $w[i..i+l-1] = \overleftarrow{w[j..j+l-1]}$, где $i \le j$. Такой обратный повтор также будем индексировать тройкой (i, j, l). Заметим, что палиндром $w[i..i+l-1]$ является обратным повтором (i, i, l).

Периодические подстроки. У строки w есть *период* $p \ge 1$, если $w[i] = w[i+p]$ для всех подходящих i ($1 \le i \le |w|-p$). Особое значение имеет *минимальный период*. Характеристика периодичности строки, *экспонента*, определяется как отношение длины строки к минимальному периоду. Строка называется *периодической*, если её экспонента не меньше 2.

Из теоремы Файна–Вильфа [16] следует, что при конкатенации периодической строки и одного символа (слева или справа), строка либо остаётся периодической с тем же периодом, либо перестаёт быть периодической. Таким образом естественно определяются максимальные по включению периодические подстроки.

Алгоритмы. Мы используем в алгоритмах стандартный набор операций, как, например, в языке C. Таким образом, за исключением представления входных данных в виде потока (см. ниже), используется обычная RAM-модель вычислений. В главе 4 отдельно оговаривается использование операций для работы со словарями. Память измеряется в машинных словах, имеющих размер $\mathcal{O}(\log n)$ бит для входа длины n.

Приближённый алгоритм для задачи максимизации имеет *аддитивную погрешность* E (соотв., *мультипликативную погрешность* ε) если он находит решение со стоимостью не менее $OPT - E$ (соотв., $\frac{OPT}{1+\varepsilon}$), где OPT — стоимость оптимального решения; здесь E и ε могут быть функциями от размера входа.

Вероятностные алгоритмы. В рассматриваемых задачах будет показываться неприменимость детерминированных алгоритмов. Вместо этого будут использоваться алгоритмы, совершающие случайный выбор в процессе работы. Такие алгоритмы называются *рандомизированными*, и существует два основных типа рандомизированных алгоритмов:

- *алгоритм типа Монте-Карло* возвращает корректный ответ с высокой

вероятностью (больше чем $1 - 1/n$) и имеет детерминированные время работы и объём используемой памяти;

- *алгоритм типа Лас-Вегас* всегда возвращает корректный ответ, но его время работы и используемая память на входах размера n являются случайными величинами.

Мы будем использовать в первую очередь алгоритмы типа Монте-Карло. При этом для алгоритмов типа Лас-Вегас в рассматриваемых задачах будут доказываться линейные нижние оценки на необходимую память.

Хэш Карпа–Рабина. Хэш-функции, известные как *хэш Карпа–Рабина* [28], используются в большинстве работ по строковым алгоритмам в модели потока данных. В диссертации все алгоритмы также используют хэши Карпа–Рабина. Использование хэшей позволяет заменить сравнение строк сравнением их хэшей, которое требует константного времени. В работе используются *фреймы* строк, которые являются кортежами из хэш-значения и $\mathcal{O}(1)$ вспомогательных числовых значений. Известно [8], что, зная фреймы любых двух строк из A, B, AB, можно вычислить фрейм третьей строки за время $\mathcal{O}(1)$.

Для доказательства нижних оценок на вычислительную сложность задач мы используем принцип Яо [44]. С его помощью можно получать оценки на ресурсы, необходимые рандомизированному алгоритму.

Теорема (Принцип Яо). Пусть $\mathcal{X}$ — множество входов некоторой задачи, $\mathcal{A}$ — множество всех детерминированных алгоритмов, решающих её, и $c(a, x) \geq 0$ — стоимость запуска алгоритма $a \in \mathcal{A}$ на $x \in \mathcal{X}$.

Пусть p — распределение вероятностей над $\mathcal{A}$, и пусть A обозначает алгоритм, выбранный случайно в соответствии с распределением p. Пусть q — распределение вероятностей над $\mathcal{X}$, а X обозначает вход, выбранный случайно в соответствии с распределением q. Тогда $\max_{x \in \mathcal{X}} \mathbf{E}[c(A, x)] \geq \min_{a \in \mathcal{A}} \mathbf{E}[c(a, X)]$.

Модель потока данных. Мы рассматриваем модель потока данных: входная строка $S[1..n]$ (называемая *потоком*) читается слева направо, по одному символу, и не может быть сохранена в памяти, потому что доступная алгоритму память является сублинейной функцией от n. Мы называем алгоритмы, работающие в приведенной модели, *потоковыми*. Потоковый алгоритм обязательно является онлайн-алгоритмом, т.е. его работу можно представить циклом

1: **for** $i = 1$ to n **do**
2: прочитать $S[i]$; обработать $S[1..i]$; выдать ответ для $S[1..i]$

Итерацией потокового алгоритма называется одна итерация этого внешнего цикла.

Чаще всего потоковые алгоритмы являются рандомизированными типа Монте-Карло и приближенными; для других типов алгоритмов удается доказать линейные нижние оценки на используемую память, что исключает их из модели потока данных. Все приводимые в диссертации алгоритмы являются приближенными рандомизированными алгоритмами типа Монте-Карло.

Самыми важными характеристиками вычислительной сложности потоковых алгоритмов являются объём используемой памяти и время обновления (максимальное время, требуемое для выполнения одной итерации); общее время работы алгоритма менее важно. Отличительной чертой приближенных алгоритмов является то, что используемые алгоритмом ресурсы (в особенности память) обычно зависят от погрешности; эта зависимость формирует «трейд-офф» между точностью алгоритма и потребляемыми им ресурсами.

Глава 2 (Палиндромы в потоках). Палиндромы — одна из базовых регулярных структур в тексте, и они хорошо изучены в классической модели вычислений. Так линейный алгоритм, вычисляющий все палиндромы в строке, был представлен в [35], а в [17] показано, как преобразовать его в алгоритм реального времени. Другие результаты, ставшие классическими, были доказаны в [3, 20, 29]. Активное изучение задач, связанных с палиндромами, не прекращается; например, в статье [41] была приведена структура данных, являющаяся эффективным представлением множества палиндромов в строке. Другие свежие результаты включают [7, 9, 34]. Одной из причин активного изучения палиндромов является практическая значимость в вычислительной биологии «инволютивных» палиндромов, см., например, [27]. В модели потока данных задачи поиска палиндромов были впервые рассмотрены в [6].

В главе 2 мы рассматриваем задачу LPS о наибольшем палиндроме в модели потока данных (результаты опубликованы в [22, 23]). В этих работах соавторами доказано, что точных потоковых алгоритмов и приближенных рандомизированных потоковых алгоритмов типа Лас-Вегас для этой задачи не существует, а для приближенных рандомизированных алгоритмов типа Монте-Карло доказаны нижние оценки на используемую память. Эти оценки равны $\Omega(M \log \min\{|\Sigma|, M\})$ бит памяти, где $M = n/E$ для приближения ответа с аддитивной погрешностью E и $M = \log n / \log(1 + \varepsilon)$ для приближения ответа с мультипликативной погрешностью $(1 + \varepsilon)$. Точные формулировки теорем приводятся в начале главы.

Далее излагается основное содержание главы: алгоритмы типа Монте-Карло, решающие задачу LPS максимально эффективно и по времени (алгоритмы работают в реальном времени), и по памяти (используемая память совпадает с нижними оценками с точностью до логарифмического множителя, а для широких диапазонов задействованных параметров совпадает точно).

Все алгоритмы в этом разделе используют общую схему. Внешний цикл вы-

полняет $n = |S|$ итераций; на i-й итерации обрабатывается символ $S[i]$. Каждый алгоритм вычисляет фреймы всех префиксов S, а хранит только некоторые из них, основываясь на объёме доступной памяти. После чтения $S[i]$, каждый алгоритм проверяет некоторые суффиксы строки $S[1..i]$ — являются ли они палиндромами длины большей, чем наибольший найденный ранее палиндром, и обновляет наибольший палиндром соответствующим образом. Проверка, является ли строка палиндромом, производится за время $\mathcal{O}(1)$ сравнением хэша от самой строки и от обратной к ней строки. Какие именно суффиксы доступны для проверки, зависит от хранимых фреймов; каждая проверка занимает $\mathcal{O}(1)$ времени. При помощи комбинаторных лемм доказывается, что на каждой итерации достаточно проверить лишь константное число суффиксов.

Теорема 2.3.1. Существует потоковый алгоритм реального времени типа Монте-Карло, решающий задачу $\mathsf{LPS}(S)$ с *аддитивной* погрешностью $E = E(n)$ и использующий $\mathcal{O}(n/E)$ машинных слов памяти, где $n = |S|$.

Для доказательства теоремы 2.3.1 построен Алгоритм А, работающий для любых значений $E \in [1, n]$. Заметим, что используемая память точно совпадает с нижней оценкой при разумных допущениях $|\Sigma| > n^{0.01}, E < n^{0.99}$.

Алгоритм хранит фреймы для префиксов с фиксированным шагом и на каждой итерации проверяет не более двух суффиксов, которые могут обновить ответ (см рис.1).

Рис. 1: Поиск палиндрома, который может обновить ответ. Квадраты обозначают позиции j, такие что фрейм префикса $S[1..j-1]$ сохранён; скобки показывают подстроки, которые могут быть проверены на палиндромность. Только подстроки-«кандидаты» могут быть палиндромами длины $> answer.len$.

Теорема 2.3.2. Существует потоковый алгоритм реального времени типа Монте-Карло, решающий задачу $\mathsf{LPS}(S)$ с *мультипликативной* погрешностью $\varepsilon = \varepsilon(n) \in (0, 1]$ и использующий $\mathcal{O}\big(\frac{\log(n\varepsilon)}{\varepsilon}\big)$ машинных слов памяти, где $n = |S|$.

Для доказательства теоремы 2.3.2 построен Алгоритм М. Используемая им память совпадает с нижней оценкой с точностью до логарифмического множителя в худшем случае (если ε это константа и $|\Sigma| = \mathcal{O}(\log n)$); при $\varepsilon < n^{-0.01}, |\Sigma| > n^{0.01}$ совпадение точное.

Основное отличие в построении алгоритма с мультипликативной погрешностью от алгоритма с аддитивной погрешностью заключается в том, что здесь фрейм *каждого* префикса сохраняется, но затем после некоторого числа итераций удаляется. Количество итераций, которое фрейм $I(i)$ хранится, определяется

функцией *времени жизни* $\mathsf{ttl}(i)$. Эта функция отвечает за корректность алгоритма и за объём используемой памяти; она основана на двоичном представлении числа i и подобрана так, что на каждой итерации удаляется не более одного фрейма. При этом конфигурация сохраненных фреймов соответствует рис. 2.

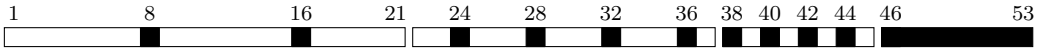

Рис. 2: Сохранённые фреймы после итерации $i = 53$ (при ε близком к 1). Чёрные квадраты обозначают числа j, такие что фреймы префиксов $S[1..j-1]$ сохранены в текущий момент.

При такой стратегии хранения фреймов удается доказать, что на каждой итерации для обновления ответа достаточно проверить не более трех доступных суффиксов на палиндромность.

Теорема 2.3.3. Существует потоковый алгоритм реального времени типа Монте-Карло, решающий задачу $\mathsf{LPS}(S)$ с *мультипликативной* погрешностью $\varepsilon = \varepsilon(n) \in (1, n]$ и использующий $\mathcal{O}\big(\frac{\log n}{\log(1+\varepsilon)}\big)$ машинных слов памяти, где $n = |S|$.

Для доказательства теоремы 2.3.3 построен Алгоритм M'. Используемая им память совпадает с нижней оценкой с точностью до логарифмического множителя. Алгоритм является усложненной модификацией алгоритма M, в которой, в частности, все двоичные представления заменены на представления в системе счисления с основанием, пропорциональным ε.

Глава 3 (Повторы и обратные повторы в потоках). В главе 3 речь идёт о тесно связанных задачах LRS и LRRS. Поиск наибольшей повторяющейся подстроки в классической модели вычислений является известным приложением суффиксных деревьев, упомянутым в пионерской статье Вейнера о них [43]; задача LRRS в классической модели решается аналогично.

Мы рассматриваем задачи LRS и LRRS в модели потока данных; результаты опубликованы в статье [37]. Мы приводим по два эффективных приближенных алгоритма типа Монте-Карло для решения каждой из задач и подкрепляем эти алгоритмы нижними оценками.

Основная сложность задач LRS и LRRS заключается в том, что они не локальны: два вхождения могут быть разделены, например, $\Omega(n)$ символами. Для того, чтобы не хранить весь входной поток, мы вычисляем хэш от подстрок (*блоков*) фиксированного размера, рассматриваем хэши как новые символы (мы называем их *хэш-буквами*) и ищем повторяющиеся строки из хэш-букв (мы называем их *сжатыми повторами*, а их прообразы — *сжимаемыми повторами*); см. рис. 3.

При этом мы храним только один след и суффиксное дерево, построенное по нему алгоритмом Вейнера. Для каждого из остальных следов хранится единственное значение — позиция в суффиксном дереве, соответствующая наибольшему суффиксу следа, имеющему вхождение в суффиксное дерево. Прямолинейная реализация этих идей даёт Алгоритм ARR, доказывающий следующую теорему.

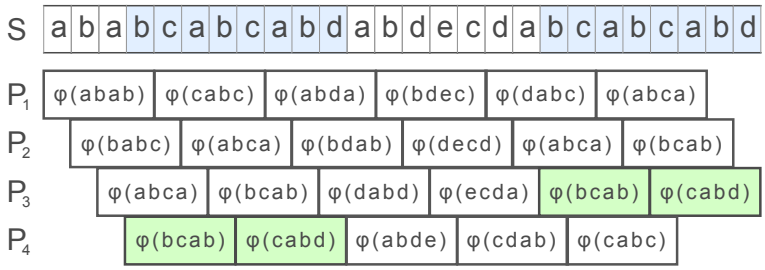

Рис. 3: Следы (строки из хэш-букв; $\phi(w)$ обозначает хэш Карпа-Рабина строки w) с разными сдвигами (позициями первого блока) и сжатые повторы. Сжимаемый повтор $(4, 19, 8)$ и его сжатый повтор $(3, 1, 5, 2)$ (выделено цветом).

Теорема 3.4.1. Существует потоковый алгоритм типа Монте-Карло, решающий задачу LRRS с заданной аддитивной погрешностью $E = \mathcal{O}(n^{0.99})$, использующий $\mathcal{O}(\frac{n}{E} + E)$ машинных слов памяти, работающий за $\mathcal{O}(n)$ времени суммарно и имеющий время обновления $\mathcal{O}(\frac{n}{E})$.

Далее мы используем некоторые сложные структуры данных [31] и модифицированный алгоритм построения суффиксного дерева [2], а также применяем технику отложенных операций [18], получая в результате Алгоритм FastARR. Он имеет логарифмическое время обновления и тем самым доказывает следующую теорему.

Теорема 3.4.2. Существует потоковый алгоритм типа Монте-Карло, решающий задачу LRRS с заданной аддитивной погрешностью $E = \mathcal{O}(n^{0.99})$, использующий $\mathcal{O}(\frac{n}{E} + E)$ машинных слов памяти, работающий за $\mathcal{O}(n + \frac{n}{E} \log n)$ времени суммарно и имеющий время обновления $\mathcal{O}(\log n)$.

Почти дословная переформулировка Алгоритма ARR для задачи LRS дает аналог теоремы 3.4.1 (**Теорема 3.5.1**). В то же время для получения логарифмического обновления в задаче LRS требуется глубокая модификация алгоритма FastARR. Полученный Алгоритм FastAR доказывает следующую теорему.

Теорема 3.5.2. Существует потоковый алгоритм типа Монте-Карло, решающий задачу LRS с заданной аддитивной погрешностью $E = \mathcal{O}(n^{0.99})$, использующий $\mathcal{O}(\frac{n}{E} + E)$ машинных слов памяти, работающий за $\mathcal{O}(n + \frac{n}{E} \log n)$ времени суммарно и имеющий время обновления $\mathcal{O}(\log n)$.

В последней части главы применяется принцип Яо, чтобы доказать нижние оценки для задачи LRS, с заданной длиной входа n и алфавитом Σ; мы используем обозначение $\mathsf{LRS}_{|\Sigma|}[n]$. Сначала мы доказываем, что любой алгоритм типа Лас-Вегас, решающий LRS с заданной аддитивной погрешностью, требует как минимум линейной памяти, а значит, не является потоковым.

Теорема 3.6.1. Пусть A — рандомизированный онлайн-алгоритм типа Лас-Вегас, решающий задачу $\mathsf{LRS}_{|\Sigma|}[n]$ с аддитивной ошибкой $E \leq 0.49n$ и использующий $s(n)$ бит памяти. Тогда $\mathbf{E}[s(n)] = \Omega(n \log |\Sigma|)$.

Для алгоритмов типа Монте-Карло с аддитивной погрешностью мы сначала доказываем простую техническую лемму, затем вспомогательную грубую оценку и, наконец, точную оценку, используя сведение к точным алгоритмам типа Монте-Карло.

Теорема 3.6.2. Любой рандомизированный онлайн-алгоритм типа Монте-Карло, решающий задачу $\mathsf{LRS}_{|\Sigma|}[n]$ с аддитивной погрешностью $E \leq 0.49n$, с вероятностью $1 - \frac{1}{n}$ использует $\Omega(\frac{n}{E} \log \min\{|\Sigma|, \frac{n}{E}\})$ бит памяти.

Точно такие же оценки для задачи LRRS следуют из нижних оценок для задачи LPS, полученных Гавриховским и Узнаньским [22, 23].

Глава 4 (Максимальные периодические подстроки в потоках). В этой главе рассмотрена задача Runs о максимальных периодических подстроках. Такие подстроки хорошо изучены и в алгоритмической, и в комбинаторной постановках. В частности, гипотеза Колпакова–Кучерова [30] о том, что их количество не превосходит длину строки, доказана в [4]. Известны эффективные алгоритмы для нахождения всех максимальных периодических подстрок как в случае полиномиального алфавита [30], так и в случае общего алфавита [14, 21, 32, 33]. Задача поиска всех подстрок, обладающих определённой регулярной структурой, в некотором смысле нехарактерна для потоковой модели. Тем не менее, нам удалось ее приближенно решить.

В начале главы при помощи принципа Яо показано, что потоковый алгоритм, в том числе рандомизированный алгоритм типа Монте-Карло, не может точно найти (и даже посчитать) все квадраты в строке, а следовательно, и все максимальные периодические подстроки. Для этого используется вспомогательная задача о поиске длиннейшего квадрата в центре строки.

Теорема 4.2.1. Существует константа γ, такая что любой алгоритм, точно решающий задачу $\mathsf{midSquare}(\Sigma, n)$ с вероятностью не меньше $1 - \frac{1}{n}$, использует не меньше $\gamma n \log \sigma$ бит памяти.

Таким образом, для задачи Runs в модели потока данных необходимо вначале сформулировать подходящую приближенную задачу. Эта задача ($\mathsf{approxRuns}$) сформулирована нами следующим образом: *дан поток S и параметр допустимой погрешности ε, требуется (1) для каждой максимальной периодической подстроки в S с экспонентой $\beta \geq 2 + \varepsilon$ выдать одну подстроку с тем же периодом и экспонентой не меньше $\beta - \varepsilon$; (2) каждой максимальной периодической подстроки с экспонентой меньше $2 + \varepsilon$ выдать одну или ноль подстрок с тем же периодом и экспонентой не меньше 2.*

Основной результат главы — это рандомизированный алгоритм типа Монте-Карло (алгоритм R), решающий задачу approxRuns и доказывающий следующую теорему.

Теорема 4.1.1. Существует потоковый алгоритм типа Монте-Карло, решающий задачу approxRuns, использующий $\mathcal{O}(\frac{\log^2 n}{\varepsilon})$ машинных слов памяти и имеющий время обновления[2] $\mathcal{O}(\log n)$.

Вся информация, хранимая алгоритмом R, привязана к *чекпойнтам*, которые образуют подмножество всех позиций. Каждая позиция k становится чекпойнтом на k-й итерации и «живёт» в течение $\mathsf{ttl}(k)$ итераций (используется функция времени жизни, введенная в главе 1 для задачи LPS).

Алгоритм R работает с подстроками длины 2^j, $j = 0, \ldots, \lfloor \log n \rfloor$. Такие подстроки, имеющие вхождение в чекпойнте, мы называем *j-блоками*. Ключевую роль играют периодические строки, покрытые парами вхождений двух перекрывающихся (или касающихся) j-блоков; мы называем такие строки *видимыми* (см рис. 4). Ключевую роль играет утверждение о том, что алгоритм, который находит все видимые периодические подстроки вида $S[h..i]$ во время i-й итерации, может решить задачу Runs.

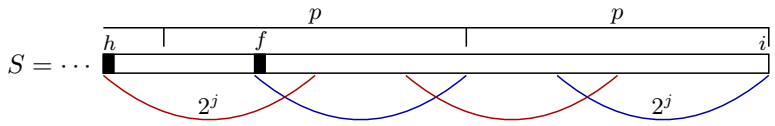

Рис. 4: Видимая периодическая строка покрыта перекрывающимися вхождениями двух j-блоков.

Для повышения эффективности работы Алгоритм R использует шесть словарей (хэш-таблиц) и ряд нетривиальных комбинаторных свойств строк.

Заключение

Итоги выполненного исследования. В диссертации рассмотрены задачи поиска основных регулярных структур в строках в модели потока данных.

Все представленные в диссертации алгоритмы являются рандомизированными алгоритмами типа Монте-Карло, т.е. решают задачи с высокой вероятностью, используя детерминированный объём памяти за детерминированное время. При этом доказано, что другие типы алгоритмов для решения поставленных задач требуют как минимум линейных затрат памяти, а значит, непригодны для потоковой модели.

Для задачи LPS о наибольшем палиндроме мы представили алгоритмы реального времени для приближенного решения с аддитивной и мультипликативной

[2]Мы расширили множество элементарных операций операциями со словарём (вставка, удаление, поиск). Оптимальный выбор словаря зависит от ε и обсуждается в главе 4 в замечании 4.4.1

погрешностью. Объём памяти, используемый этими алгоритмами, асимптотически точно совпадает с нижними оценками при большинстве значений параметров задачи. Таким образом, вопрос о вычислительной сложности задачи LPS в модели потока данных полностью решён. Отдельно можно отметить введённую нами функцию времени жизни (ttl), которая показала себя удобным инструментом и для решения задачи approxRuns о периодических подстроках.

Мы показали, что задачи поиска наибольшего повтора (LRS) и наибольшего обратного повтора (LRRS) в модели потока данных могут быть решены только с использованием приближенных алгоритмов типа Монте-Карло, а также доказали для таких алгоритмов нижнюю оценку в $\Omega(\frac{n}{E}\log\min\{|\Sigma|, \frac{n}{E}\})$ бит памяти для приближения ответа с аддитивной погрешностью E. Для решения обеих задач представлены алгоритмы типа Монте-Карло, близкие к реальному времени, использующие $\mathcal{O}(E+\frac{n}{E})$ слов памяти, где E — аддитивная ошибка приближения. Используемая память точно совпадает с нижними оценками при условиях $E = \mathcal{O}(\sqrt{n})$ и $|\Sigma| = \Omega(n^{0.01})$. Алгоритмы основаны на суффиксных деревьях, что весьма необычно для потоковой модели.

Мы доказали, что в потоковой модели не существует алгоритма, точно решающего задачу Runs о поиске всех максимальных периодических подстрок и сформулировали приближенную версию этой задачи (approxRuns). Для приближенной задачи мы представили алгоритм типа Монте-Карло, близкий к реальному времени и использующий всего $\mathcal{O}(\frac{\log^2 n}{\varepsilon})$ слов памяти, где $\varepsilon \in (0, \frac{1}{2}]$ — параметр ошибки приближения максимальных периодических подстрок.

Перспективы дальнейшей разработки темы. В задачах поиска наибольшего повтора (LRS) и наибольшего обратного повтора (LRRS) в модели потока данных открытыми остаются вопросы о нижней оценке на необходимую память для приближения с мультипликативной ошибкой, а также с аддитивной ошибкой $E = \Omega(\sqrt{n})$. Кроме того, открыт вопрос о потоковом алгоритме с мультипликативной ошибкой.

В задаче приближенного поиска всех максимальных периодических подстрок в потоках открытым остаётся вопрос о нижней оценке на необходимую память.

Список работ, опубликованных автором по теме диссертации

Статьи, опубликованные в рецензируемых научных журналах и изданиях, определенных ВАК и Аттестационным советом УрФУ:

1. Tight Tradeoffs for Real-Time Approximation of Longest Palindromes in Streams / P. Gawrychowski, O. Merkurev, A. M. Shur, P. Uznanski // Proceedings CPM 2016.—Vol. 54 of LIPIcs.—Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. — P. 18:1–18:13, 0.81п.л/0.37п.л. (Scopus)

2. Tight tradeoffs for real-time approximation of longest palindromes in streams / P. Gawrychowski, O. Merkurev, A. M. Shur, Przemyslaw Uznanski //Algorithmica. — 2019. — Vol. 81, no. 9. — P. 3630–3654, 1.56п.л/0.65п.л. (Scopus, WoS)

3. Merkurev O., Shur A. M. Searching Long Repeats in Streams // Proceedings CPM 2019.—Vol. 128 of LIPIcs.—Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. — P. 31:1–31:14, 0.87п.л/0.65п.л. (Scopus)

4. Merkurev O., Shur A. M. Searching Runs in Streams // International Symposium on String Processing and Information Retrieval / Vol. 11811 of LNCS. — Springer. — 2019. — P. 203–220, 1.13п.л/0.85п.л. (Scopus)

Список литературы

[1] Alon N., Matias Y., Szegedy M. The space complexity of approximating the frequency moments // Journal of Computer and system sciences. — 1999. — Vol. 58, no. 1. — P. 137–147.

[2] Towards real-time suffix tree construction / A. Amir, T. Kopelowitz, M. Lewenstein, N. Lewenstein // International Symposium on String Processing and Information Retrieval / Springer. — 2005. — P. 67–78.

[3] Apostolico A., Breslauer D., Galil Z. Parallel detection of all palindromes in a string // Theoret. Comput. Sci. — 1995. — Vol. 141. — P. 163–173.

[4] The "Runs" Theorem / H. Bannai, T. I, S. Inenaga et al. // SIAM J. Comput. — 2017. — Vol. 46, no. 5. — P. 1501–1514.

[5] Counting distinct elements in a data stream / Z. Bar-Yossef, TS Jayram, R. Kumar et al. // International Workshop on Randomization and Approximation Techniques in Computer Science / Springer. — 2002. — P. 1–10.

[6] Palindrome recognition in the streaming model / P. Berenbrink, F. Ergün, F. Mallmann-Trenn, E. Sadeqi Azer // STACS 2014. — Vol. 25 of LIPIcs. — Germany : Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl Publishing, 2014. — P. 149–161.

[7] Palindromic length in linear time / K. Borozdin, D. Kosolobov, M. Rubinchik, A. M. Shur // 28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017) / Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. — 2017.

[8] Breslauer D., Galil Z. Real-time streaming string-matching // Combinatorial Pattern Matching. — Vol. 6661 of LNCS. — Berlin : Springer, 2011. — P. 162–172.

[9] Computing a longest common palindromic subsequence / S. R. Chowdhury, Md Hasan, S. Iqbal et al. // Fundamenta Informaticae. — 2014. — Vol. 129, no. 4. — P. 329–340.

[10] Dictionary Matching in a Stream / R. Clifford, A. Fontaine, E. Porat et al. // ESA 2015. — Vol. 9294 of LNCS. — Springer, 2015. — P. 361–372.

[11] The k-mismatch problem revisited / R. Clifford, A. Fontaine, E. Porat et al. // SODA 2016. — SIAM, 2016. — P. 2039–2052.

[12] Clifford R., Kociumaka T., Porat E. The streaming k-mismatch problem // Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms / SIAM. — 2019. — P. 1106–1125.

[13] Crochemore M., Hancart C., Lecroq T. Algorithms on strings. — Cambridge University Press, 2007.

[14] Near-Optimal Computation of Runs over General Alphabet via Non-Crossing LCE Queries / M. Crochemore, C. S. Iliopoulos, T. Kociumaka et al. // String Processing and Information Retrieval - 23rd International Symposium, SPIRE 2016. — Vol. 9954 of Lecture Notes in Computer Science. — 2016. — P. 22–34.

[15] Crochemore M., Rytter W. Jewels of stringology: text algorithms. — World Scientific, 2002.

[16] Fine N. J., Wilf H. S. Uniqueness theorems for periodic functions // Proc. Amer. Math. Soc. — 1965. — Vol. 16. — P. 109–114.

[17] Galil Z. Real-time algorithms for string-matching and palindrome recognition // Proc. 8th annual ACM symposium on Theory of computing (STOC'76). — New York, USA : ACM, 1976. — P. 161–173.

[18] Galil Z. Real-time algorithms for string-matching and palindrome recognition // Proc. 8th annual ACM symposium on Theory of computing (STOC'76). — New York, USA : ACM, 1976. — P. 161–173.

[19] Galil Z. Open problems in stringology // Combinatorial Algorithms on Words. — Springer, 1985. — P. 1–8.

[20] Galil Z., Seiferas J. A Linear-Time On-Line Recognition Algorithm for "Palstar" // J. ACM. — 1978. — Vol. 25. — P. 102–111.

[21] Faster Longest Common Extension Queries in Strings over General Alphabets / P. Gawrychowski, T. Kociumaka, W. Rytter, T. Walen // 27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016. — Vol. 54 of LIPIcs. — Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. — P. 5:1–5:13.

[22] Tight Tradeoffs for Real-Time Approximation of Longest Palindromes in Streams / P. Gawrychowski, O. Merkurev, A. M. Shur, P. Uznanski // Proceedings CPM 2016. — Vol. 54 of LIPIcs. — Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. — P. 18:1–18:13.

[23] Tight tradeoffs for real-time approximation of longest palindromes in streams / P. Gawrychowski, O. Merkurev, A. M. Shur, P. Uznanski // Algorithmica. — 2019. — Vol. 81, no. 9. — P. 3630–3654.

[24] Gusfield D. Algorithms on Strings, Trees and Sequences. Computer Science and Computational Biology. — Cambridge University Press, 1997.

[25] Jalsenius M., Porat B., Sach B. Parameterized Matching in the Streaming Model // STACS 2013. — Vol. 20 of LIPIcs. — Dagstuhl Publishing, 2013. — P. 400–411.

[26] Counting arbitrary subgraphs in data streams / D. M Kane, K. Mehlhorn, T. Sauerwald, H. Sun // International Colloquium on Automata, Languages, and Programming / Springer. — 2012. — P. 598–609.

[27] Kari L., Mahalingam K. Watson–Crick palindromes in DNA computing // Natural Computing. — 2010. — Vol. 9, no. 2. — P. 297–316.

[28] Karp R., Rabin M. Efficient randomized pattern matching algorithms // IBM Journal of Research and Development. — 1987. — Vol. 31. — P. 249–260.

[29] Knuth D. E., Morris J., Pratt V. Fast pattern matching in strings // SIAM J. Comput. — 1977. — Vol. 6. — P. 323–350.

[30] Kolpakov R., Kucherov G. Finding maximal repetitions in a word in linear time // Proc. 40th Ann. Symp. Found. Comput. Sci. — IEEE Press, 1999. — P. 596–604.

[31] Kopelowitz T., Lewenstein M. Dynamic weighted ancestors // Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms / Society for Industrial and Applied Mathematics. — 2007. — P. 565–574.

[32] Kosolobov D. Lempel-Ziv Factorization May Be Harder Than Computing All Runs // 32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany. — Vol. 30 of LIPIcs. — Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. — P. 582–593.

[33] Kosolobov D. Computing runs on a general alphabet // Inf. Process. Lett. — 2016. — Vol. 116, no. 3. — P. 241–244.

[34] Kosolobov D., Rubinchik M., Shur A. M. Pal^k is linear recognizable online // Proc. 41th Int. Conf. on Theory and Practice of Computer Science (SOFSEM 2015). — Vol. 8939 of LNCS. — Springer, 2015. — P. 289–301.

[35] Manacher G. A new linear-time on-line algorithm finding the smallest initial palindrome of a string // J. ACM. — 1975. — Vol. 22, no. 3. — P. 346–351.

[36] McGregor A. Graph stream algorithms: a survey // ACM SIGMOD Record. — 2014. — Vol. 43, no. 1. — P. 9–20.

[37] Merkurev O., Shur A. M. Searching Long Repeats in Streams // Proceedings CPM 2019. — Vol. 128 of LIPIcs. — Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. — P. 31:1–31:14.

[38] Merkurev O., Shur A. M. Searching Runs in Streams // International Symposium on String Processing and Information Retrieval / Springer. — Vol. 11811 of LNCS. — 2019. — P. 203–220.

[39] Munro J. I., Paterson M. S. Selection and sorting with limited storage // Theoretical computer science. — 1980. — Vol. 12, no. 3. — P. 315–323.

[40] Porat B., Porat E. Exact and approximate pattern matching in the streaming model // FOCS 2009. — IEEE Computer Society, 2009. — P. 315–323.

[41] Rubinchik M., Shur A. M. EERTREE: An Efficient Data Structure for Processing Palindromes in Strings // Combinatorial Algorithms - 26th International Workshop, IWOCA 2015, Revised Selected Papers. — Vol. 9538 of LNCS. — Springer, 2016. — P. 321–333.

[42] Smyth B., Smyth W. Computing patterns in strings. — Pearson Education, 2003.

[43] Weiner P. Linear pattern matching algorithms // Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on / IEEE. — 1973. — P. 1–11.

[44] Yao A. C.-C. Probabilistic Computations: Toward a Unified Measure of Complexity (Extended Abstract) // FOCS 1977. — IEEE Computer Society, 1977. — P. 222–227.