Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

«Поиск абелевых строк наибольшей длины»

Автор: Збань Илья Константинович _____
Направление подготовки (специальность): 01.03.02 Прикладная математика и информатика
Квалификация: Бакалавр
Руководитель: Аксёнов В.Е., магистр _____

К защите допустить
Зав. кафедрой Васильев В.Н., докт. техн. наук, проф. _____

«__» _____ 20__ г.

Санкт-Петербург, 2017 г.

Студент Збань И.К. **Группа** М3439 **Кафедра** компьютерных технологий **Факультет** информационных технологий и программирования

Направленность (профиль), специализация Математические модели и алгоритмы разработки программного обеспечения

Квалификационная работа выполнена с оценкой _____

Дата защиты «20» июня 2015 г.

Секретарь ГЭК *Павлова О.Н.* Принято: «___» _____ 20___ г.

Листов хранения _____

Демонстрационных материалов/Чертежей хранения _____

Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

УТВЕРЖДАЮ
Зав. каф. компьютерных технологий
докт. техн. наук, проф.
_____ Васильев В.Н.
«___» _____ 20___ г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Студент Збань И.К. **Группа** M3439 **Кафедра** компьютерных технологий **Факультет** информационных технологий и программирования **Руководитель** Аксёнов Виталий Евгеньевич, магистр, аспирант Университета ИТМО/INRIA PARIS

1 Наименование темы: Поиск абелевых строк наибольшей длины

Направление подготовки (специальность): 01.03.02 Прикладная математика и информатика

Направленность (профиль): Математические модели и алгоритмы разработки программного обеспечения

Квалификация: Бакалавр

2 Срок сдачи студентом законченной работы: «31» мая 2017 г.

3 Техническое задание и исходные данные к работе.

Улучшить существующие алгоритмы поиска наибольшей Абелевой подстроки, проанализировать существующие решения

4 Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов)

Тестирование на практике оптимального на данный момент алгоритма поиска абелевых подквадратов и работа над алгоритмами поиска НОАП

5 Перечень графического материала (с указанием обязательного материала)

Не предусмотрено

6 Исходные материалы и пособия

Опубликованные за последние годы публикации об абелевых строках

7 Календарный план

№№ пп.	Наименование этапов выпускной квалификационной работы	Срок выполнения этапов работы	Отметка о выполнении, подпись руков.
1	Изучение предметной области	09.2016	
2	Проверка известных теоретических методов на компьютере	11.2016	
3	Постановка конкретных задач	12.2016	
4	Разработка алгоритмов для решения поставленных задач	03.2017	
5	Написать пояснительную записку	05.2017	

8 Дата выдачи задания: «01» сентября 2016 г.

Руководитель _____

Задание принял к исполнению _____ «01» сентября 2016 г.

Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Студент: Збань Илья Константинович
Наименование темы работы: Поиск абелевых строк наибольшей длины
Наименование организации, где выполнена работа: Университет ИТМО

ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

1 Цель исследования: Получить новый алгоритм поиска НОАП, улучшающий существующие результаты
2 Задачи, решаемые в работе: Тестирование существующих алгоритмов, теоретическая и практическая оценка матожидания НОАП случайных строк, решение задачи о поиске НОАП в общем случае
3 Число источников, использованных при составлении обзора: _____
4 Полное число источников, использованных в работе: 0
5 В том числе источников по годам

Отечественных			Иностранных		
Последние 5 лет	От 5 до 10 лет	Более 10 лет	Последние 5 лет	От 5 до 10 лет	Более 10 лет

6 Использование информационных ресурсов Internet: _____
7 Использование современных пакетов компьютерных программ и технологий: c++, python, gnuplot, git, etc
8 Краткая характеристика полученных результатов: Главным итогом работы является новый алгоритм поиска НОАП, на данный момент являющийся оптимальным по затратам времени и памяти
9 Гранты, полученные при выполнении работы: Грантов при выполнении работы получено не было
10 Наличие публикаций и выступлений на конференциях по теме работы: Публикаций и выступлений на конференциях по теме выпускной работы не было

Выпусник: Збань И.К. _____

Руководитель: Аксёнов В.Е. _____

«___» _____ 20___ г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В последнее время тема Абелевой эквивалентности строк стала часто появляться в научных работах. Первые определения, такие как вектор числа встреч каждого символа в подстроке, называемый вектор Парея, и первые сформулированные задачи на эту тему, такие как нахождение подстроки с заданным вектором Парея в строке, называемой *jumbled indexing*, были предложены ещё в 60-х годах прошлого века. Но только около двадцати лет назад стали появляться первые результаты в этой области. С тех пор результатов с каждым годом становилось всё больше, и, в конце концов, Абелева эквивалентность выделилась в самостоятельную подобласть.

Задачи, связанные с Абелевой эквивалентностью, встречаются в широком классе областей. Например, в биоинформатике, представленная выше задача *jumbled indexing* находит применение в бионформатике при решении задач *mass spectrometry* и *gene clusters*. Кроме того, Абелево совпадение слов, иначе, совпадение векторов Парея, может применяться как критерий эвристического фильтра для поиска шаблона в тексте, а также поиска шаблона с ошибками.

В главе 1 мы вводим основные определения, используемые в работе, и представляем известные на сегодняшний день результаты.

В главе 2 мы предложены новые алгоритмы для решения задачи о поиске числа Абелевых подквадратов и задачи о нахождении наибольшей общей Абелевой подстроки. В этой же главе мы приводим теоретические оценки к поставленным задачам.

В главе 3 мы приводим результаты практических запусков алгоритмов, предложенных в главе 2.

ГЛАВА 1. ОБЗОР

1.1. Используемые определения

Введем набор определений, которые будут использоваться по ходу работы.

Определение 1. Алфавит $\Sigma = \{c_1, c_2, \ldots, c_\sigma\}$ — конечное множество символов. Количество символов алфавита $|\Sigma| = \sigma$.

Определение 2. $|s|_{c_1}$ — количество символов c_1 в строке s.

Определение 3. $\mathcal{P}(s) = (|s|_{c_1}, |s|_{c_2}, \ldots, |s|_{c_\sigma})$ — вектор Парея, вектор частот символов строки s.

Определение 4. $a \equiv b$, если $\mathcal{P}(a) = \mathcal{P}(b)$ — Абелева эквивалентность двух строк. Две строки Абелево эквивалентны, если существует перестановка, переводящая одну из строк в другую.

Определение 5. Строка s является Абелевым квадратом, если существуют две Абелево эквивалентные строки $a \equiv b$, что $s = ab$.

Определение 6. Абелев подквадрат строки s — подстрока строки s, являющаяся Абелевым квадратом.

Определение 7. `w.h.p.` — *with high probability*, решение, с большой вероятностью работающее за такое время.

Определение 8. Будем говорить, что алгоритм работает за $\langle \mathcal{O}(f(n)), \mathcal{O}(g(n)) \rangle$, если он работает, используя $\mathcal{O}(f(n))$ времени и $\mathcal{O}(g(n))$ памяти.

Определение 9. *НОАП* (*LCAF*) — наибольшая общая Абелева подстрока (longest common Abelian factor).

Задача 1. $3SUM^+$: дано три множества целых чисел A, B, C, нужно найти три числа $a \in A, b \in B, c \in C$ такие, что $a + b = c$.

Задача 2. Поиск НОАП: даны две строки a, b над алфавитом Σ, нужно найти такие подстроки x строки a и подстроку y строки b, что $x \equiv y$, а длина подстрок максимальна.

Задача 3. Число Абелевых подквадратов: дана строка $s = s_0 s_1 \ldots s_{n-1}$ над алфавитом мощности 2, $\Sigma = \{c_0, c_1\}$. Нужно найти количество различных ее подстрок $s_{i \ldots j}$, являющихся Абелевыми квадратами.

Задача 4. *Jumbled indexing* — задача проверки, содержит ли данный текст T подстроку, Абелево эквивалентную шаблону S.

1.2. Предыдущие результаты

1.2.1. Наибольшая общая Абелева подстрока

По постановке задача о поиске наибольшей общей Абелевой подстроке очень похожа на поиск наидлиннейшей общей подстроки — очень важную задачу, исследовавшуюся в 70-е года прошлого века. В частности, суффиксное дерево было разработано в процессе работы над несколькими задачами, одной из которых являлся линейный поиск наидлиннейшей общей подстроки.

Задача поиска наибольшей общей Абелевой подстроки ($LCAS$) была сформулирована в 2013 году на семинаре StringMasters. Там она была поставлена как открытая задача.

После этого в 2015 году A.Attabi et al в [**1**] предложили два алгоритма решения этой задачи.

Первый алгоритм решает задачу поиска НОАП за $\langle \mathcal{O}(n^2\sigma), \mathcal{O}(n\sigma)\rangle$. В этом алгоритме автор предлагает для каждой длины посчитать для обеих строк векторы Парея для всех подстрок за $\langle \mathcal{O}(n\sigma), \mathcal{O}(n\sigma)\rangle$, отсортировать их сортировкой подсчетом, и после этого двумя указателями проверить, есть ли две Абелево эквивалентные подстроки.

Второй алгоритм решает задачу поиска НОАП для строки на бинарном алфавите. Он использует два факта: для фиксированной длины строки достаточно сравнивать лишь количество одного из символов c_1, и что если у строки есть подстрока длины l с x символов c_1, и подстрока длины l с y символов c_1, то найдется и подстрока с z символов c_1 для любого $x \leqslant z \leqslant y$. Используя эти факты, в алгоритме для каждой длины l у обеих строк считается максимальное и минимальное количество символов c_1, которые могут быть у подстроки длины l. Если эти отрезки для длины l у строк пересекаются, то l — кандидат на НОАП. На алгоритм подсчета этих отрезков за $\mathcal{O}(n^2/\log n)$ авторы ссылаются как на уже известный.

В 2016 году на конференции SPIRE S.Grabowski et al [**4**] улучшили алгоритм 2015 года, уменьшив требование памяти до $\mathcal{O}(n)$, и алгоритм, решающий задачу для случая алфавитов большого размера, работающий за $(\mathcal{O}(n^2 \log^2 n \log^* n), \mathcal{O}(n \log^2 n))$ времени и памяти.

Первый результат, улучшение памяти алгорита A.Attabi et al основано на двух оптимизациях: во-первых, во время сортировки не хранится все векторы Парея, а на каждой из σ итераций количество вхождений очередно-

го символа вычисляется заново. Во-вторых, для того, чтобы все еще можно было сравнивать векторы Парея, явно хранятся векторы Парея для всех позиций, кратных σ, т.е. для $s_0 \dots s_{l-1}$, $s_\sigma \dots s_{\sigma+l-1}$, и так далее. Для сравнения векторов Парея двух произвольных подстрок, нужно взять ближайший посчитанный вектор Парея, изменить в нем $\mathcal{O}(\sigma)$ элементов, так что сравнение все еще работает за такое же время.

Сравнительное времени работы этих алгоритмов можно увидеть в таблице 1.

Таблица 1 – Существующие детерменированные алгоритмы поиска НОАП

Год	Авторы	Время	Память
2015	A. Alattabi et al	$\mathcal{O}(n^2\sigma)$	$\mathcal{O}(n\sigma)$
2016	S. Grabowski et al	$\mathcal{O}(n^2\sigma)$	$\mathcal{O}(n)$
2016	S. Grabowski et al	$\mathcal{O}(n^2 \log^2 n \log^* n)$	$\mathcal{O}(n \log^2 n)$
2017	Данная работа	$\mathcal{O}(n^2 \log \sigma)$	$\mathcal{O}(n)$

Кроме того, нельзя не упомянуть о недетерменированных версиях решения этой задачи. Я не нашел публикаций, посвященных этому алгоритму, так что назовем это фольклором.

Определим полиномиальный хеш последовательности $s = s_0 s_1 \dots s_{|s|-1}$ как $h(s) = (\sum_{i=0}^{|s|-1} s_i \cdot p^i) \mod m$ для каких-то параметров p и m.

Фиксировав длину l, можно скользящим окном посчитать полиномиальный хеш последовательности $\mathcal{P}(s_i \dots s_{i+l-1})$ для всех подстрок длины l.

Посчитав полиномиальные хеши всех подстрок обеих строк, можно проверить, есть ли совпадающие, используя хешмап. В случае совпадения хешей нужно проверить подстроки на Абелеву эквивалентность отдельно, поскольку совпадение полиномиальных хешей еще не гарантирует эквивалентность строк, но ложное совпадение очень маловероятно.

Оценить матожидание времени работы такого алгоритма довольно сложно, но на практике он работает очень быстро, и есть все основания полагать, что можно доказать его временную оценку в $\mathcal{O}(n^2)$. На этот алгоритм была предложена задача на интернет-олимпиаду для школьников в 2015 году [**6**].

1.2.2. Наибольший Абелев подквадрат и количество Абелевых подквадратов строки

Задачи о нахождении наидлиннейшего Абелево подквадрата и их количества были поставлены в 2016 году, с указанием метода, опубликованным в том же году, позволяющего решать некоторые задачи, связанные с Абелевой эквивалетностью, за субквадратичное время [**5**].

Так, авторы этой статьи предложили алгоритмы поиска длиннейшего/кратчайшего Абелевого подквадрата за $\mathcal{O}(n^2/\log^2 n)$, поиска кратчайшего Абелевого периода за $\mathcal{O}(n^2/\sqrt{\log n})$, поиска Абелевых бордеров строки за $\mathcal{O}(n^2/\log^2 n)$ и поиска всех Абелевых покрытий строки за $\mathcal{O}(n^2/\log n)$.

Так же недавно был опубликован новый алгоритм для решения задачи *3-SUM* используя методы аддитивной комбинаторики, решающий частный случай задачи значительно быстрее, чем за квадратичное время — за $\mathcal{O}(n^{1.86})$. Так же они показали, как отвечать на запросы *histogram queries* за $\mathcal{O}(1)$ после предподсчета за $\mathcal{O}(n^{1.86})$.

ГЛАВА 2. ТЕОРЕТИЧЕСКИЕ ИССЛЕДОВАНИЯ

2.1. Абелевы квадраты

2.1.1. Обзор алгоритма решения 3SUM+

Кратко рассмотрим алгоритм, предложенный в [2].

В этой статье предложено решение задачи $3SUM^+$ для монотонных линейно ограниченных множеств. Оно основано на использовании *BSG-теоремы* и быстрого преобразования фурье для выделения набора пар подмножеств с относительно небольшой декартовой суммой, достаточно плотно покрывающих полную декартову сумму $A + B$.

Предложенный алгоритм является довольно большим продвижением в изучении задачи $3SUM^+$, являясь первым строго субквадратичным алгоритмом для задач, основанных на ограниченной монотонной (min,+) свертке.

В алгоритме есть большое пространство для дальнейшего исследования. Так, авторами поставлена задача для улучшения детерменированной версии алгоритма с целью от избавления от перемножения матриц с предположением, что можно так же использовать преобразование фурье. Кроме того, поиск применений мощных методов аддитивной комбинаторики в задачах дискретной математики выглядит очень перспективной областью исследования.

2.1.2. Поиск числа Абелевых подквадратов

2.1.2.1. Сведение к 3SUM+

Задачу о поиске числа Абелевых подквадратов бинарной можно свести к задаче 3SUM+.

Пусть дана строка $s = s_0 s_1 \dots s_{n-1}$. Рассмотрим следующие множества:

$$A = \{(cnt_a(i), cnt_b(i)) \mid 0 \leqslant i \leqslant n\}, \tag{1}$$

$$B = \{(cnt_a(i), cnt_b(i)) \mid 0 \leqslant i \leqslant n\}, \tag{2}$$

$$C = \{(2 \cdot cnt_a(i), 2 \cdot cnt_b(i)) \mid 0 \leqslant i \leqslant n\}, \tag{3}$$

где $cnt_x(i)$ — количество символов типа c на префиксе строки s длины i. Мотивация для этого сведения в том, что мы хотим, чтобы

по обоим символам количество вхождений этого символа в первой половине строки было равно количеству вхождений во второй. Если рассматривать подстроку $[i; j)$, середина которой $\frac{k=i+j}{2}$, то должно быть выполнено $cnt_a(k) - cnt_a(i) = cnt_a(j) - cnt_a(k)$ и $cnt_b(k) - cnt_b(i) = cnt_b(j) - cnt_b(k)$, или $cnt_x(i) + cnt_x(j) = 2cnt_x(k)$. Поскольку $cnt_a(i) + cnt_b(i) = i$, становится понятно, что число абелевых подквадратов можно найти по формуле

$$(\#3SUM^+(A, B, C) - (n + 1))/2, \tag{4}$$

где $n + 1$ приходится вычитать, потому что нам неинтересны решения длины 0, а делить на два, потому что каждая подстрока будет посчитана дважды, с $i < j$ и $i > j$.

Поскольку лучшее известное решение задачи $3SUM^+$ для монотонных ограниченных $\mathcal{O}(n)$, работает за $\mathcal{O}(n^{1.86})$, а сведение работает за линию, получаем такое же решение задачи о количестве Абелевых подквадратов.

На самом деле, предполагая, что средний компьютер выполняет порядка 10^9 операций в секунду, и предположив, что мы будем проверять решение на ограничениях порядка $n = 10^5$, асимптотическое ускорение в $n^{0.14}$ переводя в числа ускоряет всего в $10^{5^{0.14}} \approx 5$ раз, что может оказаться незаметным, а вспоминая о большой константе алгоритма и нескольких логарифмах можно предположить о его практической неэффективности. Все же, представляет интерес для изучения работа алгоритма на специфичных тестах или в среднем случае, вдруг он работает достаточно быстро с какой-то стороны.

2.2. Наибольшая общая абелева подстрока

2.2.1. Общий алгоритм

Отметим, что нас интересует детерменированный алгоритм решения задачи. Известно несколько недетерменированных решений, на практике работающих достаточно быстро, но они не являются темой исследования данной работы.

Будем подходить к лучшему решению по шагам от самого простого, на каждом шаге оптимизируя какую-то часть алгоритма, для лучшего понимания.

2.2.1.1. $\langle \mathcal{O}(n^2 \log \sigma), \mathcal{O}(n \log \sigma) \rangle$ w.h.p

Будем перебирать длину l и проверять, есть ли общая абелева подстрока длины l. План: построить $\mathcal{P}(t)$ для всех подстрок t длины l строк a и b, потом понять, есть ли две строки с одинаковым $\mathcal{P}(t)$.

Будем строить векторы $P(t)$ для всех подстрок длины l строк a и b по очереди, переходя от одной подстроки к следующей. Для этого нужно уметь удалять первый символ текущей строки и дописывать в конец новый символ. Расширим алфавит на один символ, добавив разделитель \$, нигде ранее не встречающийся, и будем идти скользящим окном длины l по строке $a\$b$.

Хранить векторы $P(t)$ будем в персистентном массиве, реализованном на персистентном дереве отрезков. Для того, чтобы перейти к следующей подстроке, нужно уменьшить значение в одной ячейке на 1, и увеличить значение в другой ячейке на 1.

Для того, чтобы научиться сравнивать на равенство две вершины дерева, соответствующие двум векторам $P(t)$, будем при построении считать некоторое число $h(v)$ — класс эквивалентности вершины. Этот класс эквивалентности будет соответствовать набору значений на подотрезке, соответствующему этой вершине — две вершины в одном классе эквивалентности, если они соответствуют одному и тому же подотрезку символов, и количество вхождений каждого символа у них одинаково.

Будем поддерживать хешмап, в котором для пары чисел $\langle h_1, h_2 \rangle$ хранится класс эквивалетности пары этих чисел, если она уже встречалась. Чтобы посчитать хеш для листа, проверим класс эквивалентности у пары $\langle -pos, val \rangle$, где pos — номер символа, соответствующего этому листу, а val — значение, записанное в этой вершине. Такая пара, с отрицательным $-pos$, выбирается для того, чтобы избежать коллизии со внутренними вершинами, характеристиками которых являются пары неотрицательных чисел — пара уже посчитанных классов эквивалентности сыновей $\langle h(v_l), h(v_r) \rangle$. Когда нам нужно узнать хеш пары $\langle h_1, h_2 \rangle$, смотрим в хешмап: если там есть элемент с таким ключом, то соответствующий класс эквивалентности уже посчитан, иначе кладем туда новый элемент с таким ключом и значением, равным размеру хешмапа. Значения всех хешей таким образом будут принимать значения от 0 до $MapSize - 1$.

Таким образом, после подсчета класса эквивалентности каждой вершины, для всех подстрок длины l первой и второй строки можно выписать их классы эквивалентности, и нужно проверить, есть ли в двух массивах одинаковое число. Поскольку все значения имеют порядок $\mathcal{O}(n \log \sigma)$, это можно сделать используя сортировку подсчетом.

Время работы — n итераций по l, и $\mathcal{O}(n \log \sigma)$ операций для каждой длины: каждая вершина дерева отрезков создается за $\mathcal{O}(1)$ w.h.p. используя хешмап. Расходуемая память $\mathcal{O}(n \log \sigma)$ на хранение дерева отрезков и хешмапа.

2.2.1.2. $\langle \mathcal{O}(n^2 \log \sigma), \mathcal{O}(n^2) \rangle$ deterministic

Посмотрим внимательнее на персистентное дерево отрезков из предыдущего решения. Это ациклический ориентированный граф, в котором каждая вершина имеет свой уровень (глубину) от 1 до $\log \sigma$, при чем на каждой глубине по $\mathcal{O}(n)$ вершин.

Будем считать классы эквивалентности всех вершин, поднимаясь по уровням от листьев к корням, используя один хешмап размера $\mathcal{O}(n^2)$, который умеем очищать за $\mathcal{O}(1)$. Под хешмапом здесь и далее я подразумеваю просто массив на $\mathcal{O}(n^2)$ элементов с пометкой последнего изменения для возможности обнуления за $\mathcal{O}(1)$.

В этом решении класс эквивалентности будет будет иметь не сквозную нумерацию среди всех вершин дерева, как в предыдущем решении, а иметь отдельную нумерацию для каждого уровня вершин дерева.

База: посчитать классы эквивалентности листьев. Класс эквивалентности листа, как и в прошлом пункте, $h(\langle -pos, val \rangle)$, где и pos, и val принимают значения порядка $\mathcal{O}(n)$. Поэтому можно пройти по всем листам в дереве, и посчитать классы, обращаясь к хешмапу напрямую и спрашивая, был ли уже такой же лист, и какой у него класс эквивалентности. Для того, чтобы обойти все листы за их количество, при построении дерева можно в каждый лист складывать ссылку на новый лист, который появляется в следующей версии дерева отрезков.

Переход: посчитан класс эквивалентности всех вершин более глубокого уровня. Обратим внимание, что поскольку на каждой глубине $\mathcal{O}(n)$ вершин, классы этих вершин так же будут принимать значения $\mathcal{O}(n)$. Поэтому мы можем очистить хешмап и точно так же, как и для листьев, считать значение

класса, к которому относится вершина, проверяя, была ли уже такая пара $\langle h(v_l), h(v_r)\rangle$.

Таким образом, мы построили дерево и посчитали хеши всех вершин за $\langle n^2 \log \sigma, n^2\rangle$ полностью детерменированно.

2.2.1.3. $\langle \mathcal{O}(n^2 \log \sigma), \mathcal{O}(n)\rangle$ deterministic

Начнем с того, что на хранение дерева отрезков у нас сейчас уходит $n \log \sigma$ памяти, это много. Чтобы уменьшить потребление памяти, можно использовать технику **limited node copying**. Краткое введение, которое будет необходимо для дальнейшего понимания алгоритма:

Вместо того, чтобы после пересоздания очередного листа пересоздавать весь путь до корня, будем хранить в каждой вершине дополнительный указатель, изначально нулевой. При изменении значения в листе будем подниматься по предкам, пока у предка дополнительный указатель уже занят, и создавать в этом случае новую вершину. Когда мы стоим в вершине и знаем, что один из ее сыновей был изменен, а дополнительный указатель еще не занят, просто установим этот дополнительный указатель на новую версию этого сына и подпишем текущим глобальным временем. После такого изменения все еще несложно обратиться к какой-то версии дерева отрезков: нужно просто при переходе к сыновьям при выборе, куда спускаться, посмотреть, не нужно ли идти по дополнительному указателю.

Можно доказать, что таким образом построенное дерево занимает $\mathcal{O}(n)$ памяти, используя амортизационный анализ, но не будем об этом.

Подсчет классов эквивалентности для листьев и внутренних вершин в этом решении отличается.

База: подсчет классов для листьев. Будем считать классы для листьев группами, для каждой позиции все листья, соответствующие этой позиции в массиве, вместе. Будем поддерживать счетчик ch — первый еще не использованный номер класса эквивалентности. Фиксировав, какую позицию мы сейчас обрабатываем, просто обойдем все листья с этой позицией (для этого можно хранить в каждом листе ссылку на предыдущий лист этой позиции), и листу со значением val присвоим хеш $ch + val$. после чего увеличим ch на $maxValue_{pos} + 1$, где $maxValue_{pos}$ — наибольшее значение, которое было в ячейке pos в одной из версий дерева отрезков. Мы знаем $maxValue_{pos}$ для

каждой позиции, и можно заметить, что количество классов эквивалентности на этом уровне $\mathcal{O}(n)$, поскольку $\sum_{pos=0}^{\sigma} maxValue_{pos} = \mathcal{O}(n)$.

Переход: посчитали классы для всех (даже больше, чем для всех вершин, об этом далее) вершин на предыдущем уровне. Кроме класса вершины мы записываем не только эту вершину, а еще и все ее копии во все времена, которые мы не создавали явно в дереве отрезков (при переходе на послеследующий уровень их можно безопасно удалить, чтобы сохранить линейную память). Чтобы получить для вершины список всех времен, когда она должна была бы существовать без сжатия, нужно просто взять список всех времен всех трех ее сыновей и смерджить, это делается за линейное время, поскольку они уже отсортированы.

Следующее, что нужно сделать — сгруппировать все вершины с одинаковым $h(v_l)$ в одну группу, и обработать их вместе, чтобы назначить им соответствующие классы эквивалентности. Пусть у очередной вершины (для каждого варианта глобального времени, которое в ней интересно) классы сыновей $h(v_l)$ и $h(v_r)$. Запишем в вектор с номером $h(v_l)$ напоминание: нужно посчитать $h(\langle h_1, h_2 \rangle)$ и записать его в текущую вершину v. После того, как сделали это для всех вершин текущего уровня, можно перебирать $h(v_l)$, очищать хешмап размера $\mathcal{O}(n)$, и перебирать соответствующее ему $h(v_r)$, назначая вершинам текущего уровня соответствующие классы. Как обычно, если $h(v_r)$ есть в хешмапе, достаем оттуда посчитанный класс, иначе сопоставляем ему новый.

После того, как классы эквивалентности всех вершин посчитаны, можно освобождать память с предыдущего уровня и переходить к следующему. В конце получим посчитанные классы для всех корней.

Используемое время так и осталось $\mathcal{O}(n^2 \log \sigma)$, а вот требуемая память стала всего $\mathcal{O}(n)$.

Остается открытым вопрос существования более быстрых детерменированных алгоритмов, работающих за $o(n^2 \log \sigma)$, в частности, $\mathcal{O}(n^2)$. К существованию такого алгоритма есть такие предпосылки, как недетерменированные алгоритмы, работающие за $\langle \mathcal{O}(n^2), \mathcal{O}(n) \rangle$.

2.2.2. Случай бинарных строк

Отдельный интерес представляет случай $\sigma = 2$. Есть известный алгоритм, описанный, например, в [1], работающий за время $O(n^2 / \log n)$.

В этой же статье рассмотрена задача матожидания длины НОАП двух случайных строк длины n и сделано предположение 5.1 о том, что $LCAF_{avg} \geqslant n - O(\log n)$. Это преположение выглядит слишком смелым, рассмотрим эту задачу подробнее.

Теорема 10. Для любой функции $f(n) = o(n)$ верно что для двух случайных бинарных строк длины n: $LCAF_{avg} < n - f(n)$.

Доказательство.

Лемма: для любой функции $f(n) = o(n)$ с фиксированной вероятностью $P > 0$, не зависящей от n, верно $LCAF_{avg} < n - f(n)$.

Обратим внимание, что центральная подстрока длины $n - 2f(n)$, полученная отрезанием суффикса и префикса длины $f(n)$, является подстрокой любой строки длины $n - f(n)$ этой же строки.

Рассмотрим задачу как задачу случайного блуждания: пусть $x_i = A_i - B_i$, где A, B — наши случайные строки. От стандартной задачи случайного блуждания она отличается тем, что кроме переходов $|x_{i+1} - x_i| = 1$ разрешены переходы $x_{i+1} = x_i$. Абелево равенство двух подстрок A и B эквивалентно тому, что подпуть блуждания x возвращается в свое начало, $x_r = x_l$.

Наше случайное блуждание имеет следующие вероятности:

Δx	$p(\Delta x)$
-1	0.25
0	0.5
1	0.25

$P(n, k)$ — вероятность после n испытаний получить сумму k. Можно заметить, что $P(n, k) = C(2n, n + k) \cdot 2^{-2n}$. И действительно, если посмотреть на геометрический смысл этого распределения, то $P(n,k)$ — вероятность за $2n$ равновероятных шагов вправо или вверх дойти до диагонали $x + y = 2n$ и остановиться на диагонали $y - x = k$, что сходится с формулой $P(n,k) = P(n-1,k-1) + 2P(n-1,k) + 2P(n-1,k+1)$.

При больших n будем приближать наше биномиальное распределение нормальным, $P(n, k) = \sqrt{n}N(0,1)$

Вспомним о правиле трех сигм:

Поскольку у нормального распределения $\sqrt{n}N(0,1)$ среднеквадратичное отклонение $\sigma = \sqrt{n}$, по правилу трех сигм можно сказать, что у цен-

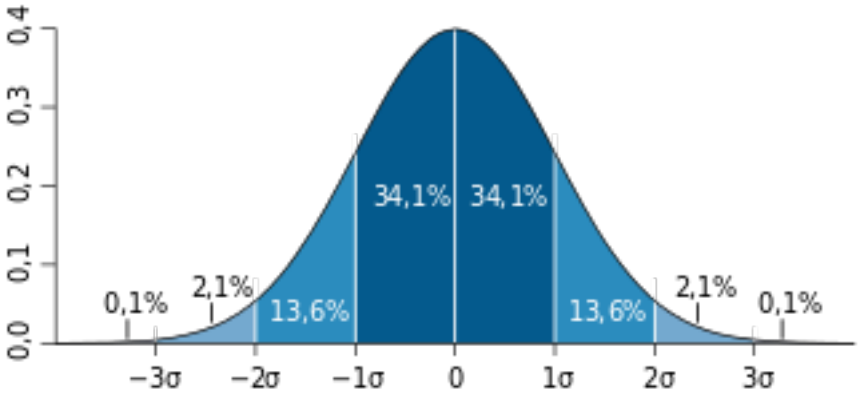

Рисунок 1 – правило трех сигм [3]

трального пути длины $n - 2f(n)$ вероятность остановиться в промежутке $[-3\sqrt{n-2f(n)}; 3\sqrt{n-2f(n)}]$ около 0.9973, а поскольку $f(n) = o(n)$, вероятность того, что изменение координаты окажется вне промежутка $[-3\sqrt{n}; 3\sqrt{n}]$, хотя бы 0.0026.

Для того, чтобы получить отрезок длины $n - f(n)$ с нулевой суммой, нужно взять какой-то суффикс префикса длины $f(n)$ и какой-то префикс суффикса длины $f(n)$. Покажем, что с достаточной вероятностью мы не сможем приблизиться к нулю за $f(n)$ шагов:

Скажем, что наше блуждание сейчас будет обычным случайным, с двумя переходами $+1$ и -1. Этот переход лишь делает оценку строже, т.к. можно продлить все испытания, в которых был переход по 0 до ровно k ненулевых переходов и прийти к случайному блужданию, при чем максимум модуля отклонения на префиксе мог только увеличиться.

Поскольку $2f(n) = o(n)$, докажем более сильное условие. За n шагов с вероятностью $C > 0$ случайное блуждание не попадет в область с координатой меньше $-\sqrt{n}$.

Переформулируем эту задачу как задачу о разорении: игрок имеет $\sqrt{n}$ денег и играет n раундов против бесконечно богатого казино, и нужно найти вероятность разорения игрока. В [7] и в [8] можно найти следующую формулу: вероятность проигрыша игрока со стартовым капиталом a за n раундов равна $y_{a,n} = 1 - \frac{2}{\sqrt{\pi}} \int_0^t e^{-u^2} du + \Delta$, где Δ — малый остаточный член, а $t = \frac{a}{\sqrt{2(n+\frac{2}{3})}}$.

В нашем случае, $a = \sqrt{n}$, $t \approx \frac{1}{\sqrt{2}}$, и вероятность проигрыша в пределе равна $1 - \frac{2}{\sqrt{\pi}} 1 \int_0^{\frac{1}{\sqrt{2}}} e^{-u^2} du \approx 0.395$.

Таким образом, с вероятностью хотя бы 0.0026 центральный подпуть будет иметь отклонение от нуля хотя бы в $3\sqrt{n}$, и с вероятностью хотя бы 0.605^2 и префикс, и суффикс, который мы допишем к этой строке, будут иметь отклонение не больше, чем на $\sqrt{n}$, то есть, с вероятностью $P \geqslant 0.026 \cdot 0.605^2$ у двух случайных строк наибольшая абелева подстрока будет меньше, чем $n - f(n)$.

Вернемся к доказательству теоремы. Будем доказывать ее от противного — пусть есть $f(n) = o(n)$ такое, что $LCAF_{avg} \geqslant n - f(n)$.

Оценим $LCAF_{avg}$. По лемме, с вероятностью $P > 0$ $LCAF$ будет не больше, чем $g(n) = \sqrt{nf(n)}$. Тогда

$LCAF_{avg} \leqslant P(n-g(n))+(1-P)n = n-Pg(n) < n-f(n)$, т.к. $f = o(g)$. Противоречие.

Кроме того, докажем грубую оценку снизу:

Теорема 11. Для двух случайных бинарных строк длины n: $LCAF_{avg} \geqslant 0.05n$.

Доказательство. Снова приблизим наше случайное блуждание нормальным распределением и воспользуемся правилом трех сигм.

С вероятностью $2 \cdot 0.34$ за первые $n/2$ шагов мы остановимся в зоне $[-\sigma; \sigma]$. После этого, с вероятностью хотя бы $0.136 + 0.021 + 0.001 \geqslant 0.15$ мы за следующие $n/2$ шагов пройдем в другую сторону хотя бы σ шагов, обязательно перейдя через точку старта. Таким образом, с вероятностью хотя бы $2 \cdot 0.34 \cdot 0.15$ НОАП будет хотя бы $n/2$, или $LCAF_{avg} \geqslant 0.05$.

Итого, получаем, что матожидание НОАП у двух случайных бинарных строк сверху и снизу ограничено линейными функциями, но более точная оценка ее поведения остается нерешенной задачей.

ГЛАВА 3. ПРАКТИЧЕСКИЕ РЕЗУЛЬТАТЫ

3.1. Наибольшая общая Абелева подстрока

3.1.1. Случай бинарного алфавита

Первое, что хочется сделать — посмотреть, как себя ведет на практике матожидание наибольшей общей Абелевой подстроки двух случайных бинарных строк.

Я выполнил 10^4 запусков поиска НОАП для различных значений n до 10^4. Такого количества запусков оказалось вполне достаточно, чтобы среднее значение НОАП стабилизировалось. Полученный результат можно увидеть на рисунке 2.

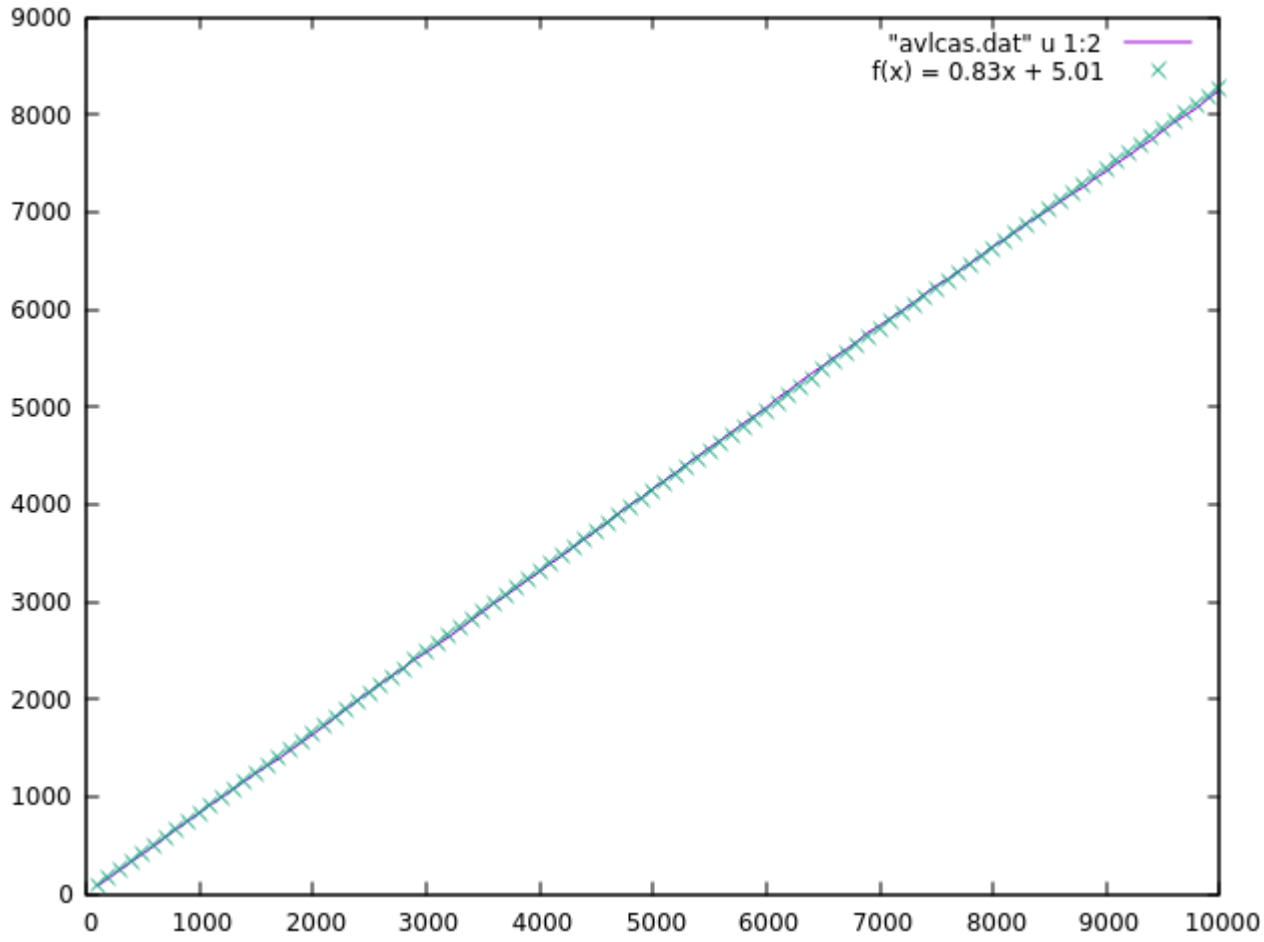

Рисунок 2 – зависимость матожидания НОАП от длин строк

Видно, что функция ведет себя очень точно как прямая $y = 0.83x$, что подтверждает полученные теоретические линейные оценки как сверху, так и снизу.

3.2. Количество Абелевых подквадратов

Реализованный алгоритм решения $3SUM^+$ был протестирован на строке из одинаковых символов, и на наборе пар случайных бинарных строк. На практике он показал себя достаточно плохо, в несколько раз проигрывая наивному решению за $\mathcal{O}(n^2)$.

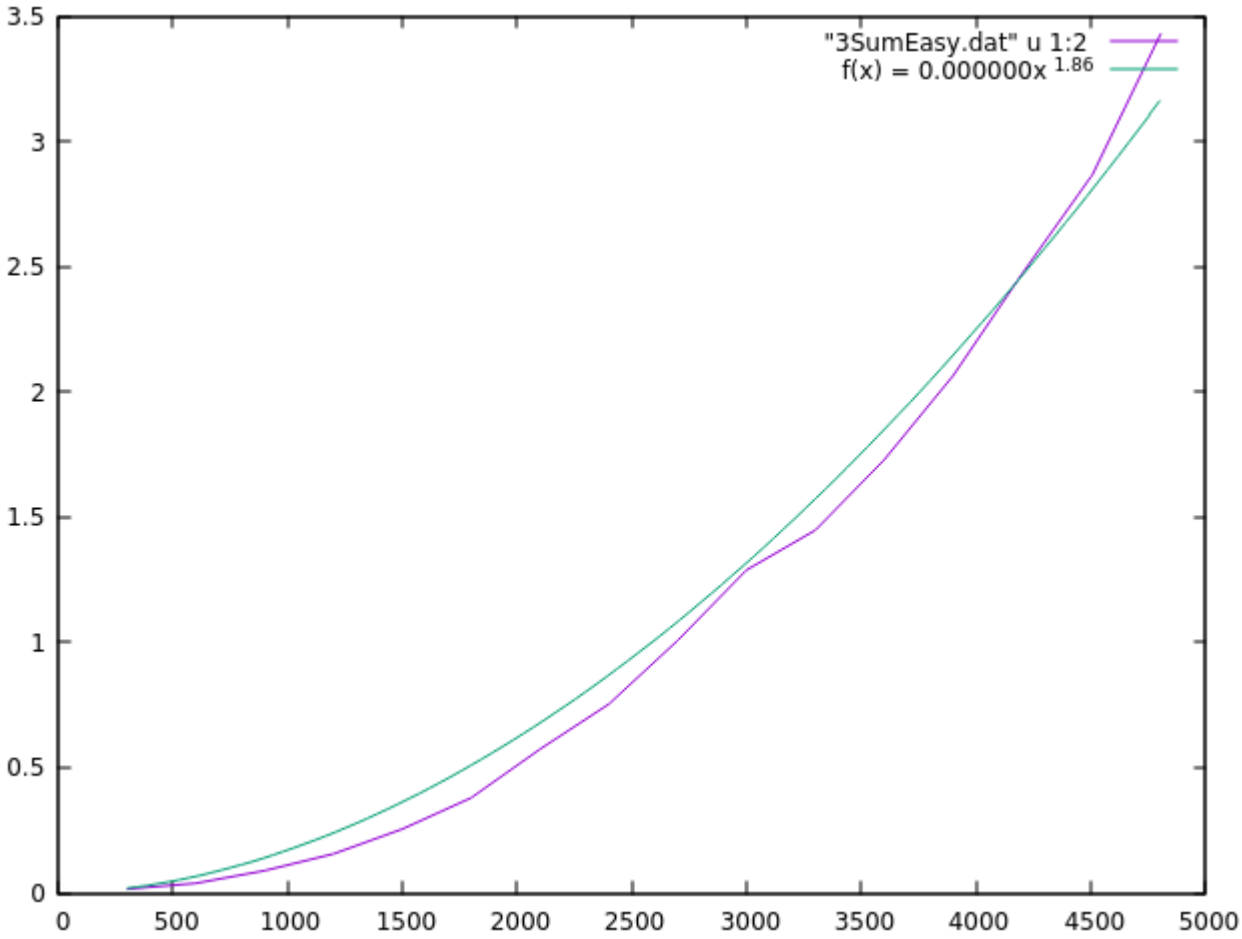

Рисунок 3 – зависимость средней времени работы на унарной строке от ее длины

На рисунке 3 можно увидеть зависимость времени работы решения в секундах от n — длины строк, данных на вход, на тесте со строками из одинаковых символов. График действительно довольно похож на $n^{1.86}$, но из-за нескольких логарифмов в асимптотике растет несколько быстрее.

На рисунке 4 можно увидеть зависимость времени работы решения в секундах от n — длины строк, данных на вход, на тесте со случайными строками из двух различных символов.

Время работы алгоритма довольно сильно меняется как от запуска к запуску из-за разных тестов, так и от различных значений n, так как различные

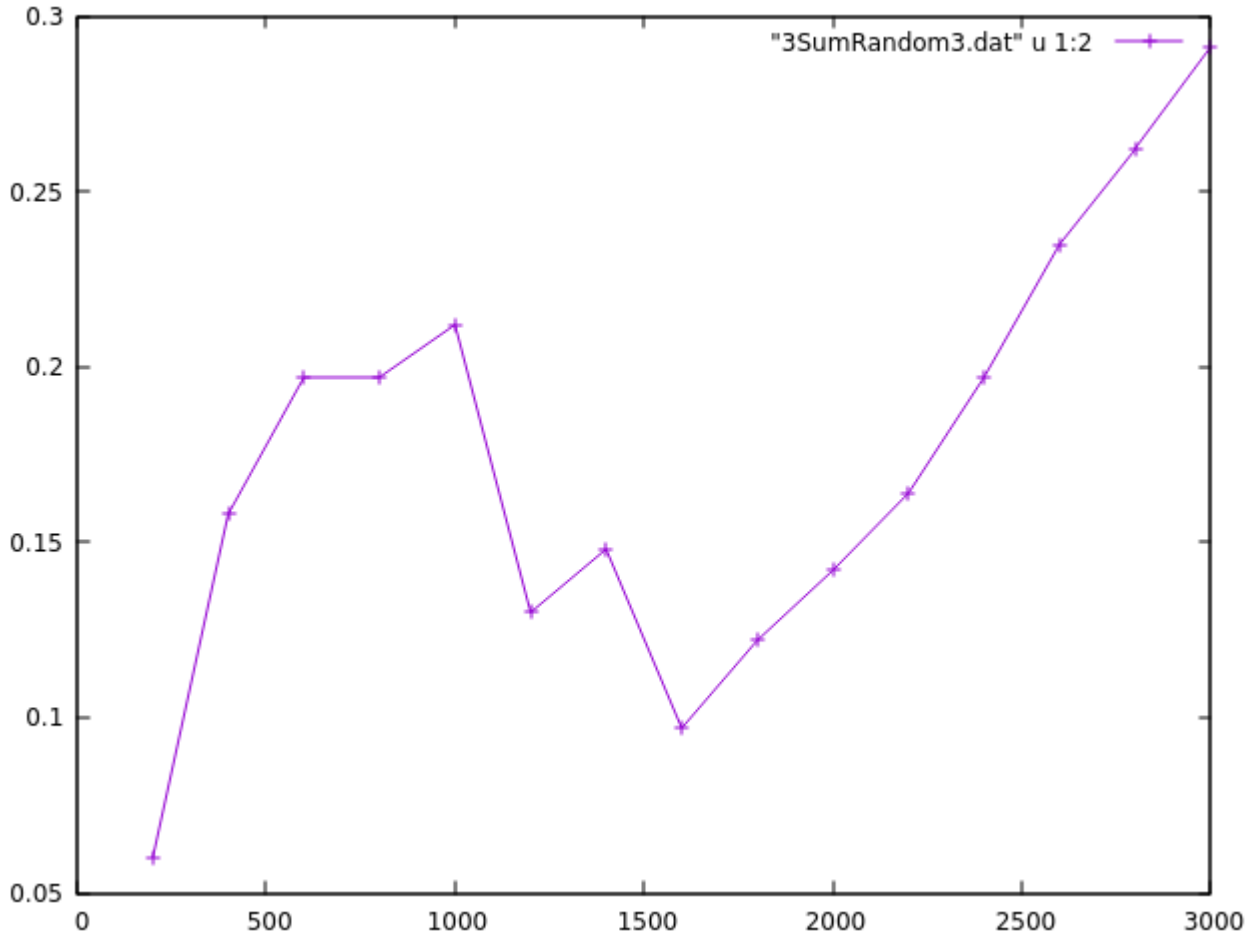

Рисунок 4 – зависимость средней времени работы на случайной строке от ее длины

ветки программы выполняются с разными вероятностями и работают разное время — не приходится удивляться некоторому увеличению производительности при увеличении n.

ЗАКЛЮЧЕНИЕ

В рамках данной работы я свел задачу о числе Абелевых подквадратов к монотонному и линейно ограниченному случаю задачи $3SUM^+$. У этой задачи задачи недавно было опубликовано неожиданное асимптотически очень хорошее решение. Я реализовал это решение, и пришел к выводу, что простая его реализация не очень эффективна, проигрывая в несколько раз простым алгоритмам с более плохой асимптотикой, но хорошей константой. Несмотря на это, заметен потенциал предложенного метода, и он вероятно может быть улучшен до производительности, действительно применимой на практике.

Кроме того, в этой работе я провел исследование задачи о НОАП. Был разработан новый алгоритм, существенно улучшающий известные результаты для задачи в общей формулировке без ограничений на размер алфавита. По сравнению с предыдущим лучшим результатом для больших алфавитов, алгоритм не только асимптотически выигрывает по затрачиваемому времени и памяти, но и более прост для понимания. Несмотря на хороший результат, остается открытым вопрос поиска более быстрых детерменированных алгоритмов.

Был проведен анализ поведения НОАП для случайных бинарных строк — вопрос, исследовавшийся ранее. Экспериментально показана линейная зависимость матожидания НОАП от длины строк, и теоретически обоснованы линейные оценки сверху и снизу. Поиск точных оценок поставлен нерешенной задачей.