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** M3439 **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 стало появляться множество работ на тему Абелевой эквивалентности строк. Первые определения и сформулированные задачи были предложены в 60-х годах прошлого века, но только около двадцати лет назад начали появляться исследования на эту тему, и с тех пор наука шагнула далеко вперед в этом направлении.

Задачи данного типа встречаются в широком классе областей. Так, *jumbled indexing* находит применение в бионформатике, при решении задач *mass spectrometry* и *gene clusters*. Кроме того, Абелево совпадение слов — хороший критерий эвристического фильтра для *exact matching* и поиска с ошибками.

В главе 1 будут рассмотрены основные определения, используемые в работе, и известные на сегодняшний день результаты.

В главе 2 будут предложены новые алгоритмы и оценки на задачи по данной теме.

В главе 3 будут приведены практические результаты предложенных алгоритмов.

ГЛАВА 1. ОБЗОР

1.1. Используемые определения

Определение 1. $P(s)$ — вектор Парея, вектор частот символов строки s.

Определение 2. $a \equiv b$, если $P(a) = P(b)$ — Абелева эквивалентность двух строк. Две строки Абелево эквивалентны, если существует перестановка, переводящая одну из строк в другую.

Определение 3. s — Абелев квадрат, если $s = ab$, где $a \equiv b$.

Определение 4. Абелев подквадрат строки s — подстрока строки s, являющаяся Абелевым квадратом.

Определение 5. `w.h.p.` — *with high probability*, решение, с большой вероятностью работающее за такое время.

Определение 6. 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 7. *НОАП* (*LCAF*) — наибольшая общая Абелева подстрока (longest common Abelian factor).

Задача 1. $3SUM^+$: дано три множества 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 $a, b \in \Sigma^n$. Нужно найти наидлиннейшую строку x такую, что $xs_a \equiv a, xs_b \equiv b$ для некоторых s_a, s_b.

Задача 3. Число Абелевых подквадратов: дана строка $s = s_0 s_1 \dots s_{n-1}$ из символов двух типов, a и b. Нужно найти количество различных ее подстрок $s_{i \dots j}$, 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 конференции StringMasters. Там она была поставлена как открытая задача.

После этого в 2015 году A.Attabi в [**1**] предложил субквадратичный алгоритм для случая $\sigma = 2$, работающий за $\mathcal{O}(n^2/\log n)$ времени, и решение общего случая за $\mathcal{O}(n^2\sigma)$ времени и $\mathcal{O}(n\sigma)$ памяти.

В 2016 году на конференции SPIRE [**4**] был улучшен 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.

Сравнительное времени работы этих алгоритмов можно увидеть в таблице 1.

Таблица 1 – Существующие алгоритмы поиска НОАП

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

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 находить большое количество Абелевых характеристик за $\mathcal{O}(n^2/\log^2 n)$ [**5**].

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 строка . Рассмотрим следующие множества:

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

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

$$C = \{(2cnt_a(i), 2cnt_b(i)) | 0 \leqslant i \leqslant n\}, \quad (3)$$

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 $P(t)$ для всех t таких, что $|t| = l$, и $a = xty$ для некоторых $x, y \in \Sigma^*$. После этого проверить, есть ли такая строка r, что $|r| = l, b = xty, P(r) = P(t)$ для некоторого 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.

Хранить векторы $P(t)$ будем в персистентном массиве, реализованном на персистентном дереве отрезков. Для того, чтобы перейти к следующей строке, 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. 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 $\langle -pos, val \rangle$, а для внутренней вершины уже точно посчитаны хеши 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 хеша 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 обнуления для возможности

База: посчитать классы эквивалентности листьев. Класс эквивалентности листа — $h(\langle -pos, val\rangle)$, 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 $max(pos)+1$. Мы знаем $max(pos)$ для каждой позиции, и можно заметить, что будут листья со значением в этой позиции от 0 до $max(pos)$, и количество классов эквивалентности на этом уровне $\mathcal{O}(n)$.

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. 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 идти по $h(v_l)$, очищать хешмап размера $\mathcal{O}(n)$, и перебирать соответствующее ему $h(v_r)$. Как обычно, если оно 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 $\mathcal{O}(n)$.

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 8. 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 вероятностью $P > 0$ 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:

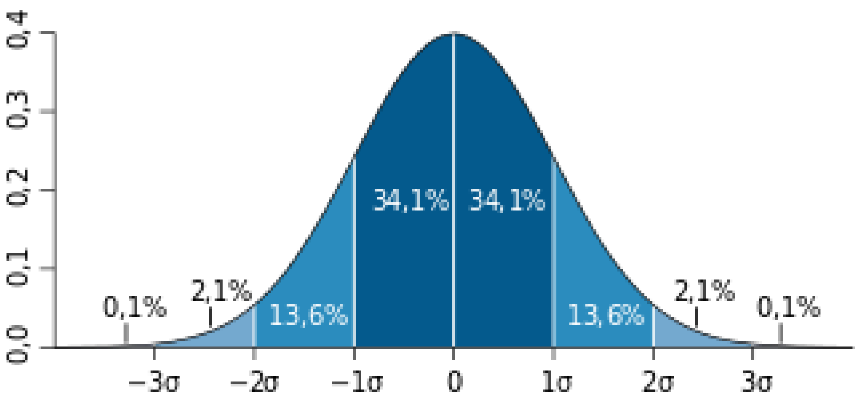

Рисунок 1 – правило трех сигм [**3**]

По правилу трех сигм можно сказать, что у центрального 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.

Есть известный факт, что в случайном блуждании если мы находимся в 0, и заканчиваем, когда попадем в точку с координатой a или $-b$ $(0 < a, b)$, то матожидание шагов до этого события ab. Воспользуемся этим: матожидание количества шагов до момента, когда мы попадем первый раз в точку $-\sqrt{n}$ или $\sqrt{n}$, равно $(\sqrt{n})^2 = n$. Поскольку матожидание равно n, значит существует вероятность $C > 0$, с которой весь наш путь из n шагов будет в полосе $(-\sqrt{n}, \sqrt{n})$.

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 C^2 и префикс, и суффикс, который мы допишем к этой строке, будут иметь отклонение не больше, чем на $\sqrt{n}$, то есть, с вероятностью $P \geqslant 0.026C^2$ 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 9. 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.

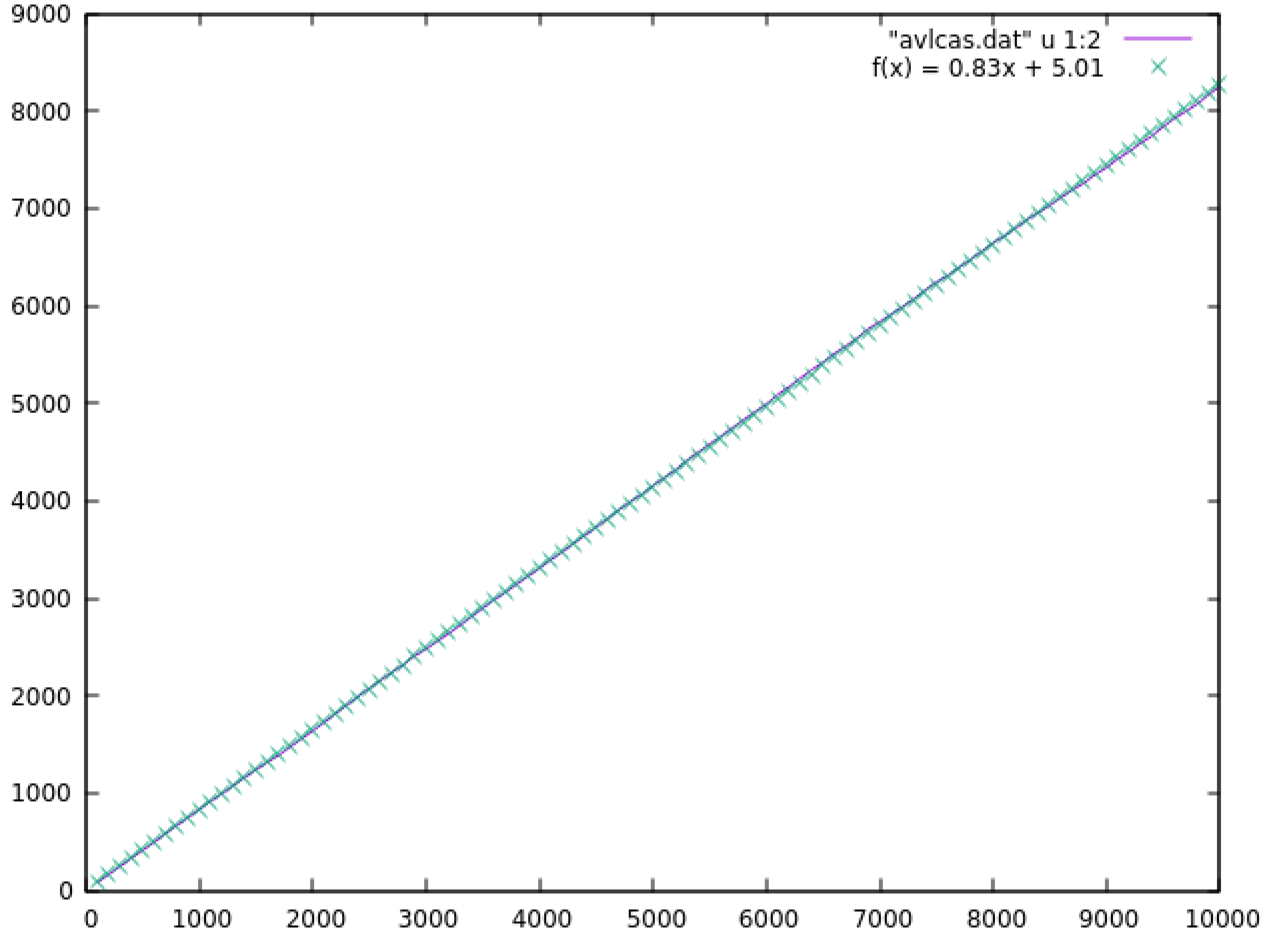

Рисунок 2 – зависимость матожидания НОАП от длин строк

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 $\mathcal{O}(n^2)$.

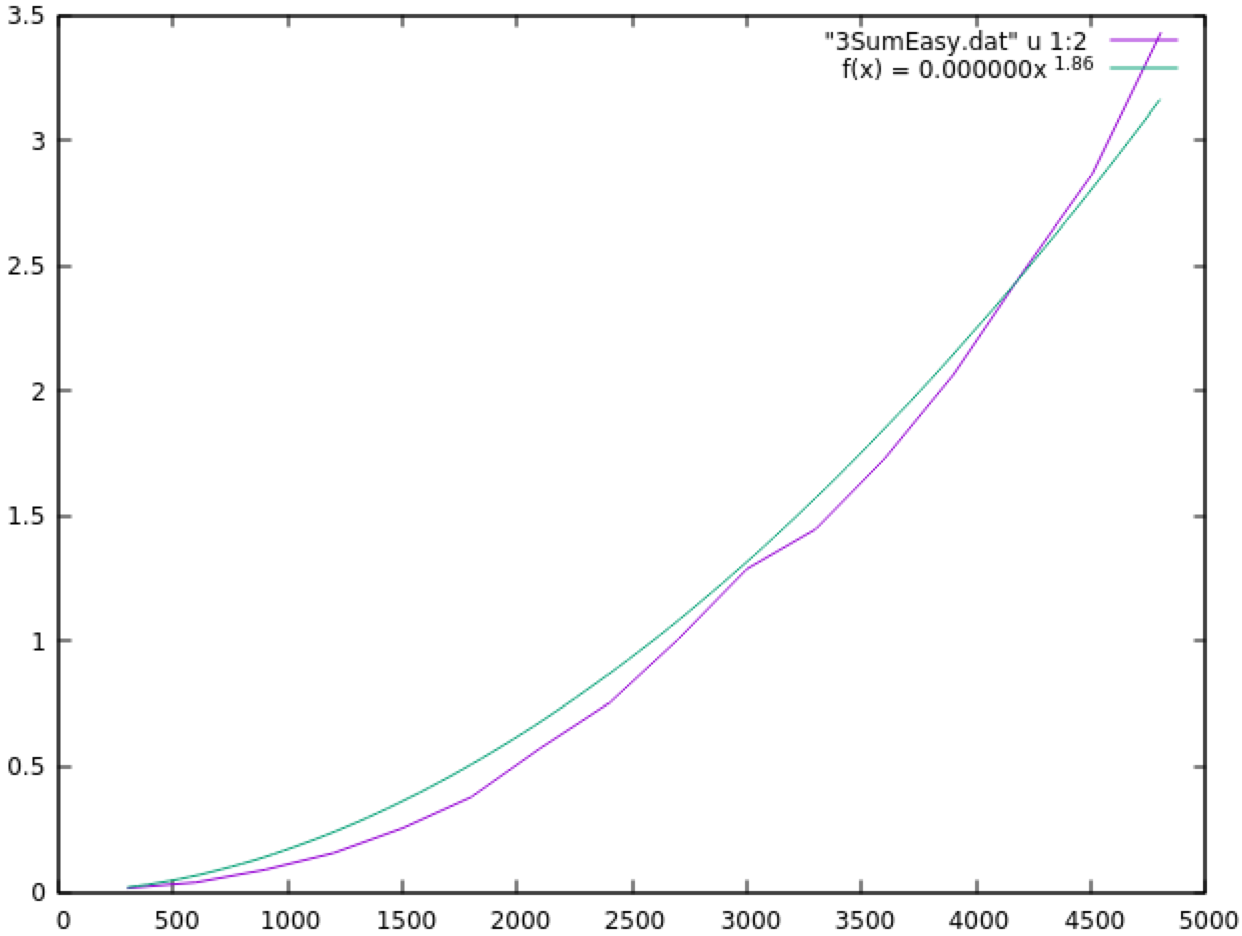

Рисунок 3 – зависимость средней времени работы на унарной строке от ее длины

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

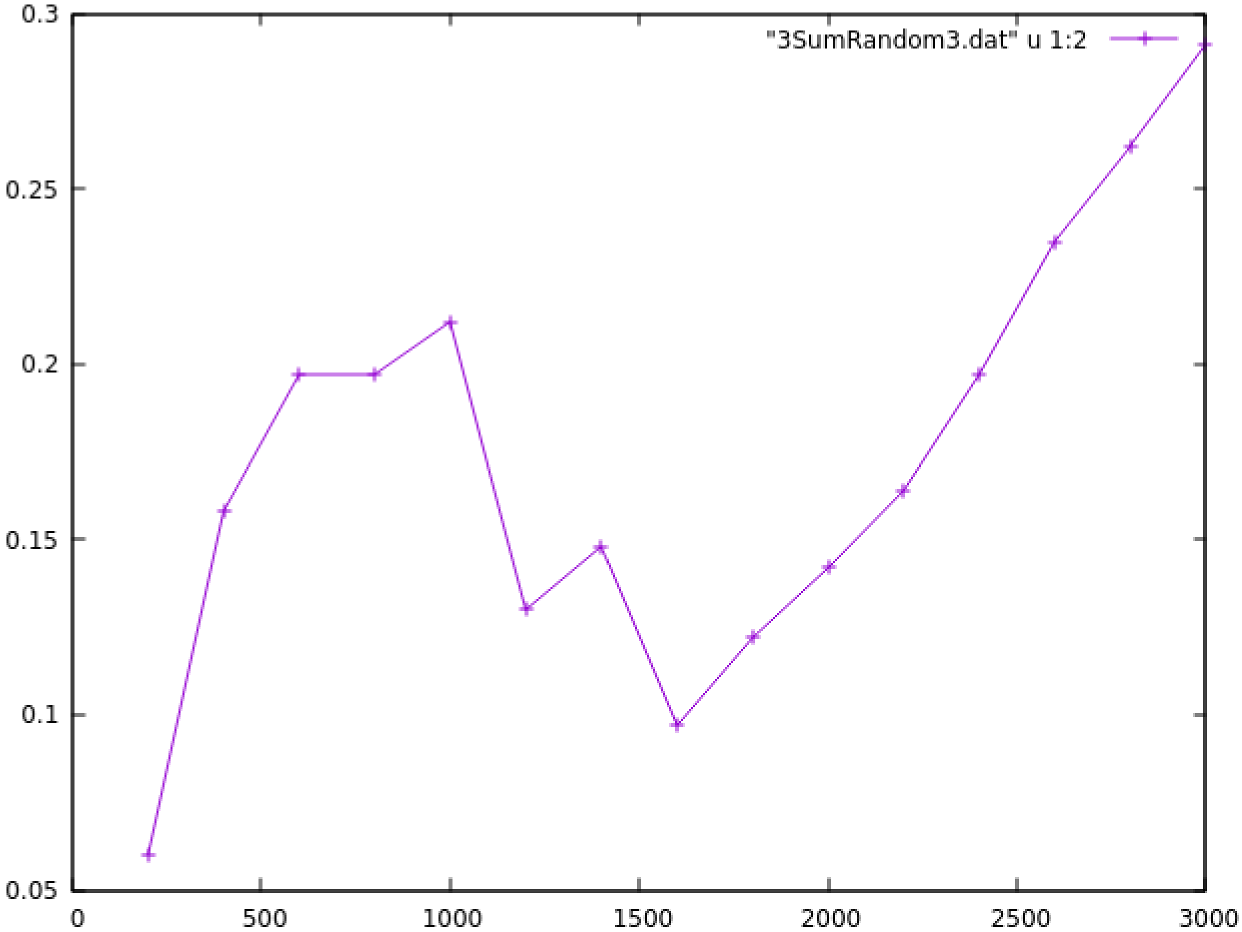

Рисунок 4 – зависимость средней времени работы на случайной строке от ее длины

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

ЗАКЛЮЧЕНИЕ

Я протестировал асимптотически оптимальный алгоритм решения монотонного 3SUM и выяснил, что он медленный.

А еще придумал хорошие оценки для binary LCAS и улучшил оптимальный алгоритм для LCAS в общем случае.