МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Системное программирование и компьютерные технологии
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему: Применение самореферентной формулы Таппера для больших объемов данных

Студент М.С. Анпилов
(И.О. Фамилия) (личная подпись)

Руководитель доцент, Н.А. Сосина
(ученая степень, звание, И.О. Фамилия)

Тольятти 2020

АННОТАЦИЯ

Тема выпускной квалификационной работы: «Применение самореферентной формулы Таппера для больших объемов данных»

Работа была выполнена студентом Тольяттинского Государственного Университета, института математики, физики и информационных технологий, группы ПМИп-1601а, Анпиловым Михаилом Сергеевичем.

Выпускная квалификационная работа посвящена реализации алгоритма оптимизации решения и построения «самореферентной» формулы Таппера.

Цель работы – обоснование методов решения неравенства, разработанного Д. Таппером, решение задачи оптимизации.

Объект исследования – задача решения и построения «самореферентной» формулы Таппера.

Предмет исследования – алгоритм решения и построения «самореферентной» формулы Таппера.

Задачи работы:

1) Изучить теоретическую информацию о неравенстве Таппера;

2) Разработать алгоритм для решения задачи формулы Таппера;

3) Выполнить программную реализацию разработанного алгоритма.

Актуальность работы заключается в решении проблемы оптимизации решения математических задач, на примере «саморферентной» формулы Таппера, где метод решения будет использовать алгоритм работы с большими объемами данных.

Результатом работы является программа решения и построения графика неравенства «самореферентной» формулы Таппера.

Бакалаврская работа содержит пояснительную записку объемом 46 страниц, включая 30 иллюстраций, 2 таблиц, список литературы из 16 наименований, включая 10 зарубежных.

ABSTRACT

The title of the graduation work is «Tupper's self-referential formula».

This work is dedicated to the realization and optimization of «Tupper's self-referential formula» problem.

The aim of this work is to explain methods of solving the inequality, made by Jeff Tupper, to find the solution to the methods of problem optimization.

The object of the study is to solve math problem and to construct «Tupper's self-referential formula».

The subject of the study is the algorithm of solving math paradox and constructing «Tupper's self-referential formula».

Relevance of this work lies in solving optimization problem of the math problem, where a solved method and performed software would work with Big Data.

The graduation work is divided by several logically connected chapters. The first chapter tells us about theoretical information of «Tupper's self-referential problem» and other existing methods of solving it. Second chapter tells us about realization of theoretical algorithm of this formula, especially about straight and reverse algorithm: algorithm of drawing the picture by using some constant «k», and reverse algorithm of finding this constant from 106×17 pixel image. Final chapter is about realization of the program, which will perform «Tupper's self-referential formula» and both algorithms. Also, this chapter includes comparisons between other realized, but less optimized, methods.

The result of the graduation work is the developed algorithm and the program that solves the «Tupper's self-referential formula» problem.

The graduation work consists of an explanatory note on 46 pages, introduction, three parts including 30 illustrations, 2 tables, the list of 16 references including 10 foreign sources.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

С появлением компьютеров в математическом мире произошли сильные изменения. Спустя столетия трудов таких математиков, как Рене Декарта, Карла Фридриха Гаусса, Леонарда Эйлера и других, математика стала не только точной наукой, но и наукой о философии. Великие умы стали искать не только объяснения того, как устроен мир, но и эстетику в самой науке. Так, например, в 300 годах до нашей эры Евклид задался вопросом «А какое самое иррациональное число?» и, в поисках ответа на этот не столько научный, сколько эстетический вопрос первым приложил свои руки к открытию так называемого «золотого сечения».

Подобными вопросами задается каждый математик. После ответов на вопросы «как найти площадь окружности?» или «как найти точку пересечения двух прямых», практически любой ученый задается вопросом «а почему окружность?», «а почему параллельные прямые никогда не пересекаются?». В этом и заключается философская сторона математики.

Такой взгляд на мир не только эффективен, но и актуален. Альберт Эйнштейн в 1905 году благодаря философскому взгляду на науку подарил миру физическую концепцию теории относительности, известную всему миру как формулу «$E = m \times c^2$». Пьер-Симон де Лаплас, размышляя о Солнечной системе и ее устройстве, создал фундаментальный труд «Небесная механика».

В 2000 году, Джефф Таппер, размышляя о графике и геометрии открыл формулу, названную в дальнейшем «самореферентной», так как, будучи отображенной на плоскости, создает собственное изображение. Реализовать такой алгоритм сложно, ввиду большого объема данных, которые приходится анализировать.

Реализация данного алгоритма представляет научно-практический интерес. На сегодняшний день существует множество математических

проблем, которые невозможно решить, в связи с большими объемами вычислений. В таких случаях принято использовать мощности вычислительных технологий, однако, из-за иного подхода возникает новые проблемы: проблема оптимизации, то есть, как сконструировать программу так, чтобы проводила меньше вычислений или других математических операций; и проблема эффективности, то есть, как сконструировать программу так, чтобы ее можно было использовать на компьютерах с меньшей вычислительной мощностью, или использовать с большими объемами данных. Одним из примеров такой математической задачи является «самореферентная» формула Таппера. На данный момент существует несколько способов решения этой проблемы, которые, либо занимают много времени для решения, либо требуют большой вычислительной мощности. Таким образом, актуальность бакалаврской работы обусловлена отсутствием оптимизированно алгоритма решения математической задачи, поднятой Джеффом Таппером в 2001 году [1]. На примере ее решения и оптимизации можно будет решать и другие математические задачи, требующие работы с большими объемами данных или задачи, требующие большой вычислительной мощности.

Объект исследования – задача отображения графика «самореферентной» формулы Таппера.

Предмет исследования – алгоритм решения задачи и построения соответствующего графика.

Целью выпускной квалификационной работы является программная реализация решения задачи о «самореферентной» формуле Таппера, с применением методов программирования по работе с большими объемами данных.

Для достижения поставленной цели необходимо решить следующие задачи:

1) изучить теоретический материал о «самореферентной» формуле, изучить другие методы решения проблемы;
2) разработать алгоритм для решения задачи оптимизации, основываясь на примере реализованных ранее методов;
3) выполнить программную реализацию разработанного в предыдущей задаче алгоритма;
4) провести исследование эффективности алгоритма, реализованного в предыдущей задаче, на примере алгоритма, разработанного самим Джеффом Таппером;

Выпускная квалификационная работа состоит из введения, трех глав, заключения, списка используемых источников.

В главе 1 рассматривается общая постановка задачи «самореферентной» формулы Таппера, теоретическая информация о «самореферентности» и работе с большими объемами данных. В главе 2 приводится анализ существующих методов решения и формирование вычислительного алгоритма. В главе 3 разрабатывается программное обеспечение и интерфейс программы, а также проводится сравнительный анализ с существующими методами решения. В заключении представлены результаты и выводы о проделанной работе.

ГЛАВА 1 САМОРЕФЕРЕНТНАЯ ФОРМУЛА ТАППЕРА

1.1 Общая постановка задачи

Термин «самореферентность» означает явление, которое возникает в системах высказываний в тех случаях, когда некое понятие ссылается само на себя. С научной точки зрения, «самореферентность» – когда выражения является и самой функцией, и аргументом этой функции. В математике – «саморферентность» почти всегда означает нарушение предикативности, вызывая логические парадоксы [2]. Причина в том, что объект, указывающий сам на себя во множестве и несущий сам себе оценку, благодаря самому себе – ведет к логическому парадоксу. Это можно увидеть на примере следующего парадокса:

Фраза «я лгу» или «данное утверждение ложно» является примером логического парадокса. Если предположить, что данное утверждение истинно, то, оно ложно, так как утверждает об этом. И наоборот, если утверждение ложно, то оно истинно, так как утверждает об этом. Парадокс лжеца является «самореферентным» парадоксом, указывающим само на себя.

Другим примером «самореферентного» парадокса является парадокс «брадобрея», который звучит так: «Единственный в городе брадобрей бреет всех жителей города, кто не бреется сам. Бреет ли он сам себя?». Парадокс заключается именно в последней части утверждения. Если брадобрей бреет всех, кто не бреет сам себя, то, в случае если он все же себя бреет, нарушается вторая часть утверждения, а если не бреет, то первая. И действительно, брадобрей не может брить самого себя, так как бреет только тех, кто не бреется сам. Но и не брить не может тоже, так как бреет тех, кто не бреется сам.

Однако фраза «почти всегда» означает, что такая ситуация, когда логический парадокс не происходит, возможна. На вопрос «когда именно?» попытался ответить Джефф Таппер.

1.2 Формула Таппера

Путем научных исследований, Джефф Таппер вывел формулу графика функции [1], определенную следующим неравенством:

$$\frac{1}{2} < \left\lfloor mod\left(\left\lfloor \frac{y}{17} \right\rfloor 2^{-17\lfloor x \rfloor - mod(\lfloor y \rfloor, 17)}, 2\right)\right\rfloor, \quad (1)$$

где $\lfloor \cdot \rfloor$ - обозначает целую часть числа (или, округление вниз), а mod – остаток от деления.

Эта формула является примером векторной графики. Путем построения на плоскости, она, на определенном участке, отображает саму себя. Для наглядности предположим, что система координат представлена в виде плоскости, состоящей из квадратов, в дальнейшем – «пикселей». Так, точка (0; 0) – это пиксель с координатами, соответственно (0; 0). И так далее.

Пускай пиксели, удовлетворяющие условию неравенства, будут окрашены, в то время как пиксели, неудовлетворяющие неравенству останутся неокрашенными.

Так же, для удобства, введем константу k и будем рассматривать одно из решений на участке графика [k; k+17] по оси y, и [0; 106] по оси x.

Тогда при условии, что константа k будет равна:

k=960,939,379,918,958,884,971,672,962,127,852,754,715,004,339,660,129,306,651,505,519,271,702,802,395,266,424,689,642,842,174,350,718,121,267,153,782,770,623,355,993,237,280,874,144,307,891,325,963,941,337,723,487,857,735,749,823,926,629,715,517,173,716,995,165,232,890,538,221,612,403,238,855,866,184,013,235,585,136,048,828,693,337,902,491,454,229,288,667,081,096,184,496,091,705,183

,454,067,827,731,551,705,405,381,627,380,967,602,565,625,016,981,482,083,418,783,163,849,115,590,225,610,003,652,351,370,343,874,461,848,378,737,238,198,224,849,863,465,033,159,410,054,974,700,593,138,339,226,497,249,461,751,545,728,366,702,369,745,461,014,655,997,933,798,537,483,143,786,841,806,593,422,227,898,388,722,980,000,748,404,719;

Тогда график такой функции на оговоренном заранее участке будет выглядеть следующим образом:

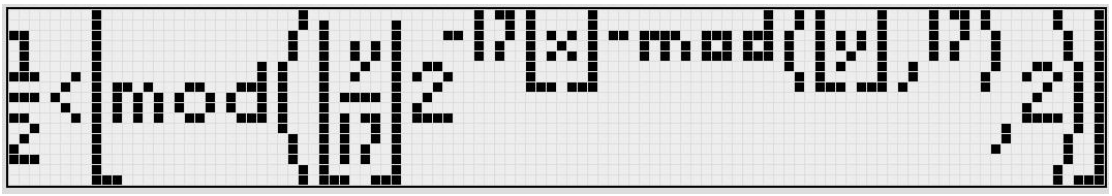

Рисунок 1 - Отображение формулы Таппера при заданном k

Иными словами, формула отобразила сама себя на заранее оговоренном участке графа: $k < y < k+17$ и $0 < x < 106$. На этом примере видно, что «самореферентная» формула Таппера имеет аналогичное применение для декодирования растровых изображений, закодированных в константе k.

Однако – это не единственное изображение, которое можно получить с помощью этой формулы. Более того, можно получить абсолютно любое изображение размером 17×106 пикселей, достаточно лишь задать конкретную константу k.

Иными словами, решением этого неравенства является комбинация закрашенных и не закрашенных пикселей на промежутке графика $[0; +\infty]$ по оси y и $[0; 106]$ по оси x. И, при ограничении участка до 17×106 пикселей, можно получить все возможные комбинации закрашенных и не закрашенных пикселей.

Рассмотрим другой пример. Пусть константа k будет равна:

k=4,858,450,636,189,713,423,582,095,962,494,202,044,581,400,587,983,244,549,483,093,085,061,934,704,708,809,928,450,644,769,865,524,364,849,997,247,024,915,119,110,411,605,739,177,407,856,919,754,326,571,855,442,057,210,445,735,883,681,829,823,754,139,634,338,225,199,452,191,651,284,348,332,905,131,193,199,953,502,413,758,765,239,264,874,613,394,906,870,130,562,295,813,219,481,113,685,339,535,565,290,850,023,875,092,856,892,694,555,974,281,546,386,510,730,049,106,723,058,933,586,052,544,096,664,351,265,349,363,643,957,125,565,695,936,815,184,334,857,605,266,940,161,251,266,951,421,550,539,554,519,153,785,457,525,756,590,740,540,157,929,001,765,967,965,480,064,427,829,131,488,548,259,914,721,248,506,352,686,630,476,300;

Тогда график такой функции на оговорённом заранее участке будет выглядеть следующим образом:

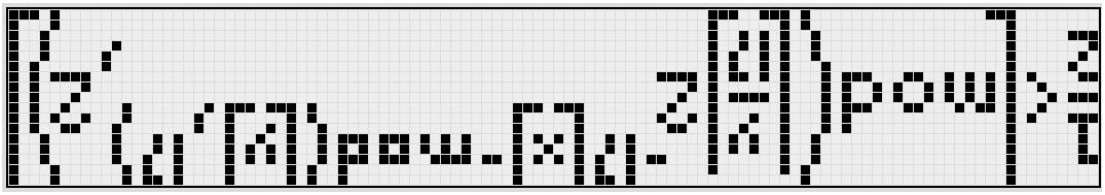

Рисунок 2 – Зеркальное отображение формулы Таппера при заданном k

Так была получена другая комбинация пикселей, отображающая формулу Таппера зеркально отраженную по оси x.

Отсюда следует, что что константа k – простой монохромный растр, используемый как двоичное число помноженное на 17.

1.3 Анализ существующих решений

На данный момент существует несколько реализаций данного уравнения: программа, разработанная самим Джеффом Таппером в 2000 году, и несколько

ее модификаций, реализованных студентами иностранных ВУЗов. Однако, метод построения растровых изображений был изменен в 2001 году, когда за основу была взята научно-исследовательская работа Таппера. Иными словами, открытие «самореферентной» формулы и ее дальнейший анализ, позволил математикам и ученым того времени улучшить механизм построения и отображения растровых и векторных изображений [3]. Из наиболее эффективных на данный момент решений существуют:

1) Адаптированный код Мэтта Паркера со скоростью выполнения программы около 5 минут;
2) Модернизированный код британского математика Джеймса Гриме реализованный на языке программирования «Python» со скоростью выполнения программы около 20-30 секунд.

К сожалению, код Мэтта Паркера или код Джеймса Гриме не доступен на общем ресурсе. Единственным доступным источником являются научные публикации обоих математиков, откуда и были получены данные о времени работы программы [4, 5].

Для решения этой проблемы было принято решение связаться с обоими математики, чтобы получить у них оригиналы программ для дальнейшей работы с ними в ходе выполнения бакалаврской работы.

Мэтт Паркер австралийский преподаватель математики, который, чаще всего, специализируется на «занимательной математике» (англ. Recreational mathematics), то есть, на направлениях в математике, проявляющихся в большей степени в рамках досуга и развлечения, самообразования и популяризации математики. Имея докторскую степень, Паркер написал множество научных статей на тему «развлекательной математики». Ниже представлены примеры его известных работ:

1) «Почти» правильный магический квадрат – существует математическая проблема о том, что невозможно построить

магический квадрат (квадрат размера n×n так, чтобы в каждой диагонали, столбце и строке в сумме было одно и то же число), состоящий только из квадратов чисел. То есть из чисел, например, 9, 36, 256, являющиеся квадратами чисел 3, 6, 16 соответственно. Вторым условиям такого «невозможного» квадрата является то, что числа в нем не должны повторяться. Мэтт Паркер, однако, смог построить такой магический квадрат, размера 3×3, с одним отступлением. В одной из диагоналей сумма чисел не была равна сумме чисел другой диагонали, столбцов или строк. Он назвал эту проблему как «талисман для людей, которые идут на все, но в конечном счете, не справляются». Пример этого квадрата представлен на рисунке 3:

29^2	1^2	47^2
41^2	37^2	1^2
23^2	41^2	29^2

841	1	2209
1681	1369	1
529	1681	841

Рисунок 3 – «почти» правильный магический квадрат

2) Алгоритм победы в игре «Монополия» - с помощью своего коллеги, Ханаана Фрая, Мэтт Паркер разработал алгоритм (стратегию) игры в экономическую игру «Монополия» так, что шанс победы в ней стал равняться более 85%. Иными словами, с использованием методов высшей математики и теории вероятности, Паркер разработал почти идеальную стратегию игры, которая принесет победу в 9 случаях из 10.

В данный момент, Мэтт Паркер преподает в Университете Западной Австралии (англ. The University of Western Australia, UWA), куда и было направлено письмо с просьбой прислать код программы, оптимизирующий решение задачи «самореферентной» формулы Таппера. К сожалению, ввиду того, что этот код был реализован более 10 лет назад, он не сохранился.

Джеймс Гриме, британский математик, оптимизировавший задачу Д. Таппера несколько лет назад, наоборот сохранил код программы. Кандидат математических наук, Джеймс Гриме, имеет свой собственный сайт, куда дублирует все научные публикации и научные выступления, проводимые им с 2012 года. Гриме частый гость таких телеканалов, как BBC News, Sky News, CBBC, BBC Four и других. На своем сайте он предоставляет свободный доступ к информации о своей электронной почте. Туда и было принято решение отправить письмо с просьбой предоставить код программы, для проведения дальнейшего анализа. К счастью, Джеймс прислал код, а также предоставил научные публикации, где более точно представлены скорость работы его программы, а также скорость работы оригинальной программы Джеффа Таппера.

Для проведения сравнительного анализа было принято решение использовать код Джеймса Гриме [6], реализованный на языке программирования «Python», так как именно эта реализация на данный момент является наиболее эффективной.

ГЛАВА 2 РАЗРАБОТКА АЛГОРИТМА

2.1 Анализ вычислительного алгоритма

На основе анализа работ Джеффа Таппера [1] можно сделать вывод о том, что константа k – простой монохромный растр, используемый как двоичное число. Следовательно, если константу k разделить на 17 и результат перевести в двоичную систему счисления, то младший бит будет соответствовать левому нижнему углу. И так каждые 17 бит будут соответствовать столбцу пикселей снизу вверх, тем самым отображая пиксельное изображение размера 17×106, где закрашенный пиксель означает «1» в двоичном числе, а не закрашенный – соответственно «0».

Например, если перевести константу k, соответствующую рисунку 1, то будет получено число начинающееся с «110010101…». Как известно, каждая цифра двоичного числа соответствует пикселю изображения. Однако, размер этого числа будет составлять 1800 бит в этом примере. Но из размера изображения, а именно 17×106, известно, что пикселей должно быть 1802. Тогда, в начало числа нужно дописать недостающее количество байт в виде «0». Таким образом, после проведения элементарных логических операций мы получим:

k_2=00110010101000100001010101011111000010010010100000000000000000000000000000100000000000000010100000000000001000100000000000000000000000011111111111111111111000000000000000010000011110000000000000000100000000000011100000000000000001000000000000111000000000000000000000000000001100000000000000100100000000000010010000000000000011000000000000000000000000000001100000000000001001000000000000100100000000000010010000000000000000000000000000011111100000000000000000000000000001111111000000011100000001110011000000000000011000000000000000001111111111111110100000000

0000000101111101000000000000000101011000001100101001000001000111010
0011000100000000000000001111111111111110000000000000000000000001100
1000000000000000101001000000001001010000000000010001000100000000000
0000100000000000000000000000000000111110000000000000000000000000000
0011001000000000000001110000000000000000000000000111111100000000001
0000000000000000100101000000000000001000000000000100101000000000001
0000000000000000111111100000000000000000000000000000001000000000000
0000100000000000000000000000000000111000000000000000010000000000000
0011100000000000000010000000000000011000000000000000000000000000000
0011100000000000000101000000000000011100000000000000000000000000000
0011100000000000000101000000000000011110000000000000000000000000000
0111100000000000110000110000000000000000000000000111111100000000001
0000000000000000101011000000000000001000000000000100011000000000001
0000000000000000111111100000000000000000000000000100000000000000000
1100000000000000000000000000000000011110000000000000000000000000000
0011001000000000000001110000000000000000000000001100001100010000000
0111100000011000000000000000000000000000000000110010000000000001010
0100000000000100101000001100001000100001100111000000011100100001111
1100000100000000000000000111111111111111111.

Из предыдущих выводов становится ясно, что каждый бит – это информация о том, нужно ли закрашивать конкретный пиксель или нет. Пиксели рассматриваются последовательно, по 17 бит, соответствующие столбцу пикселей на изображении, снизу вверх, слева направо. Графическое представление этого числа и алгоритма представлены на рисунке 4:

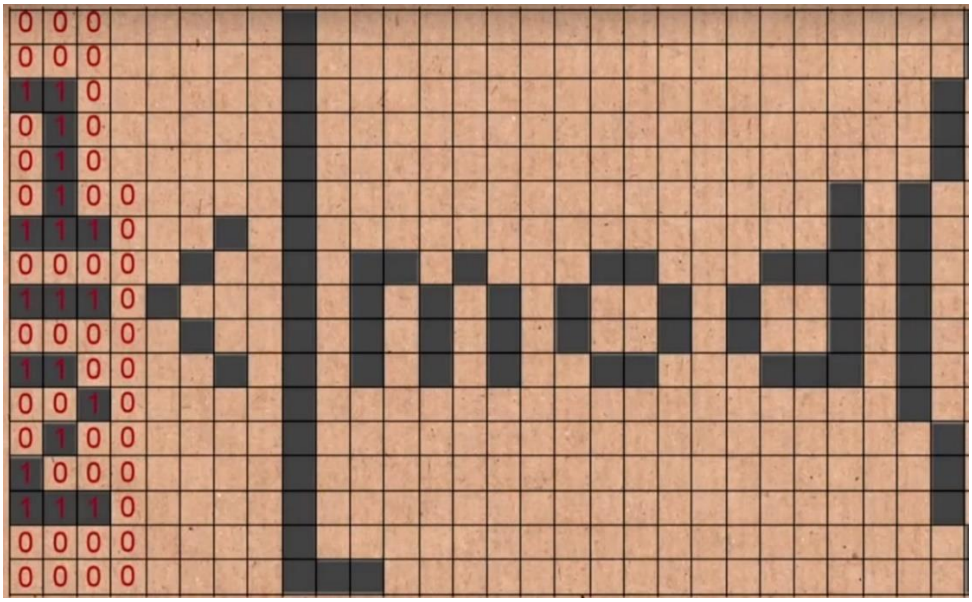

Рисунок 4 – Графическое отображение формулы Таппера

Отсюда следует что алгоритм превращения константы k в изображение следующий:

1) Разделить константу k на 17;
2) Перевести результат в двоичную систему счисления;
3) Если размер числа меньше 1802 бит, дописать недостающие биты в виде «0» слева от числа;
4) Представить полученное число в виде изображения, где каждые следующие 17 бит – столбец пикселей, в котором «1» означает закрашенный пиксель, а «0» – соответственно не закрашенный. Заполнять изображение следует снизу вверх, слева направо.

Аналогичным образом можно найти константу k из изображения:

1) Записать изображение в виде двоичного числа, записывая с крайнего левого угла, двигаясь снизу вверх, слева направо. Не закрашенный пиксель записывать как «0», закрашенный как «1»;
2) Если у полученного числа крайние левые цифры не «1», удалить их до тех пор, пока крайней левой цифрой не окажется «1»;

3) Полученное двоичное число перевести в десятичную систему счисления;
4) Умножить результат на 17.

По окончанию алгоритма будет получено число, обозначенное ранее как константа k. Тогда, используя эту константу, можно найти изображение на графике решения неравенства формулы Таппера, так как это изображение будет находится в пределах [k; k+17] по оси y и [0; 106] по оси x.

2.2 Анализ методов программной реализации

После реализации прямого и обратного алгоритма, можно приступить к реализации программного обеспечения [7], которое будет выполнять две функции: строить изображение размером 17×106 пикселей или представлять это изображение в виде константы, соответствующей одному из решений неравенства формулы Таппера.

Но на примере константы k, соответствующей отображению самой формулы Таппера, становится ясно, что числа могут быть достаточно большими. Однако, большинство языков программирования имеют ограничение на размер числа. Множество целых чисел расположено в диапазоне (-∞; +∞), когда, например, в языке «C++» максимальный тип данных «longlong» позволяет использовать числа, расположенные в диапазоне от «-9,223,372,036,854,775,808» до «9,223,372,036,854,775,808» [8]. График функции, расположенный ниже оси y, не имеет закрашенных пикселей, следовательно искать значение константы k, если оно отрицательно – не имеет смысла. Тогда, можно использовать тип данных «unsinged longlong», расположенный в диапазоне от «0» до «18,446,744,073,709,551,615» [9]. Однако, этого все еще недостаточно, так как оба этих числовых диапазона: «longlong» и «unsinged longlong», значительно меньше константы k в примере.

Следовательно, нужно найти такое решение, для которого не важен размер чисел на входе или на выходе.

Джефф Таппер решил эту проблему в 2000 году следующим образом. Он решил считать константу не как число, а как текст, последовательность букв, с точки зрении информатики, «1» и «0». Следующим шагом, он вручную прописал алгоритм перевода двоичного числа в десятичную систему счисления. Для этого он взял за основу алгоритм, придуманный Уильямом Джорджем Горнером в 1819 году [10]:

1) Разделить число на 2;
2) Результат деления разделить на 2, остаток дописать к остатку первого деления слева;
3) Повторить шаг 2 до тех пор, пока в результате деления не останется «1».

Графическое отображение алгоритма представлено на рисунке 5:

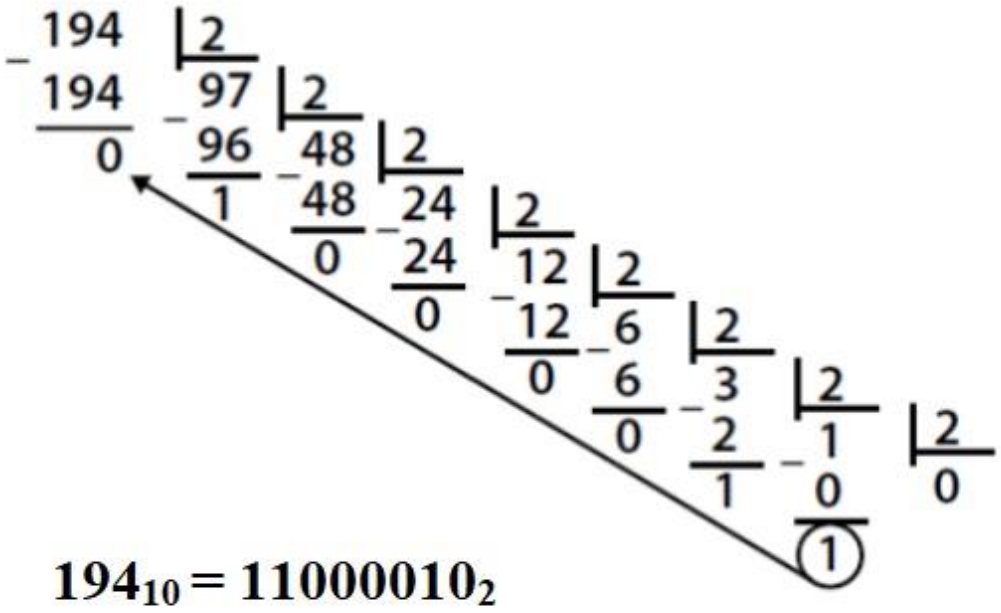

Рисунок 5 – алгоритм Уильяма Горнера

Таким же образом, считая полученное десятичное число за текст, Джефф Таппер умножал его на 17, вручную прописывая алгоритм умножения. Время работы такой программы в 2000 году составляло от 10 до 15 минут, в зависимости от количества закрашенных пикселей. В 2016 году австралийский математик Мэтью Паркер запустил этот код на современных компьютерах. С помощью современных технологий время работы алгоритма получилось уменьшить до 5 минут, что все еще означало сложность алгоритма.

Проблема ограниченности типов данных на языке программирования «C++» была решено на языке «Java» [11, 12]. Принцип работы оказался схожим с идеей Таппера – представление числа в виде текст, но реализация была более эффективной.

Таким образом, было принято использовать язык «Java» для реализации алгоритма для выпускной бакалаврской работы.

ГЛАВА 3 ПРОГРАМНАЯ РЕАЛИЗАЦИЯ И АНАЛИЗ

3.1 Разработка программного обеспечения

Для того, чтобы структурировать полученные результаты был визуально отображен алгоритм нахождения константы k из изображения и наоборот. Результат представлен на рисунках 6 и 7, соответственно:

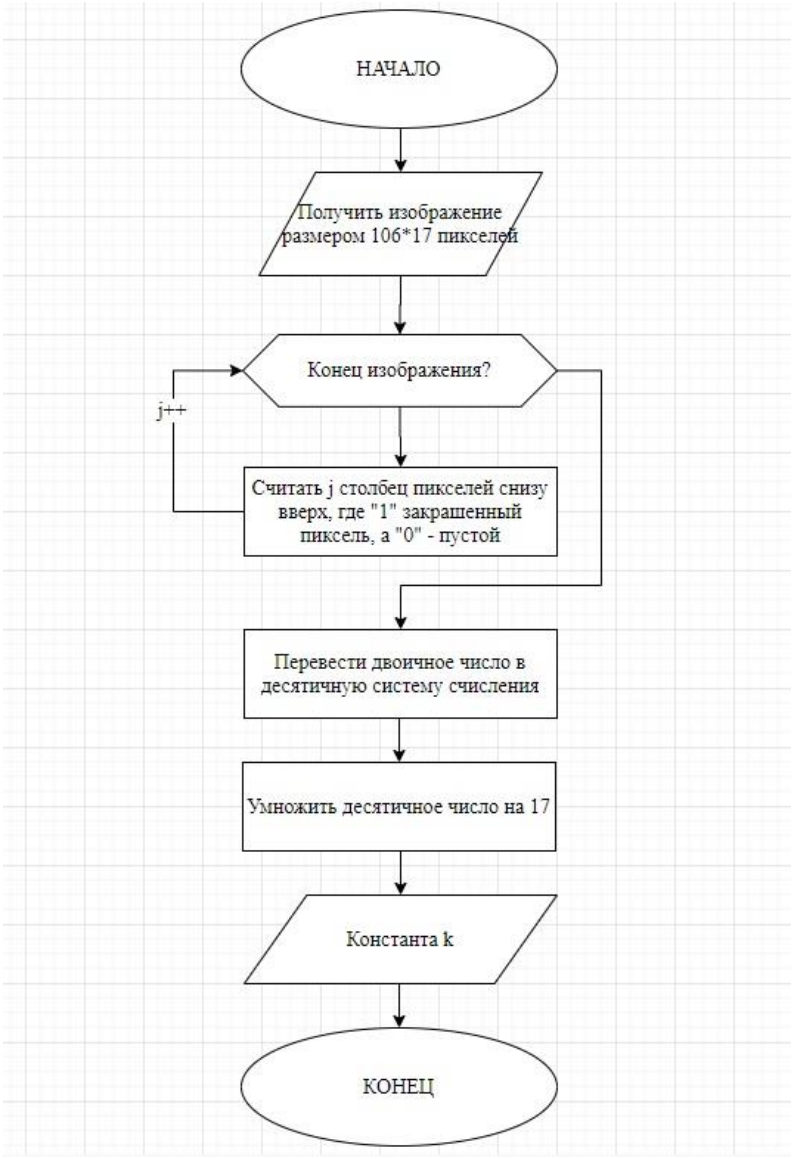

Рисунок 6 – Алгоритм нахождения константы k из изображения

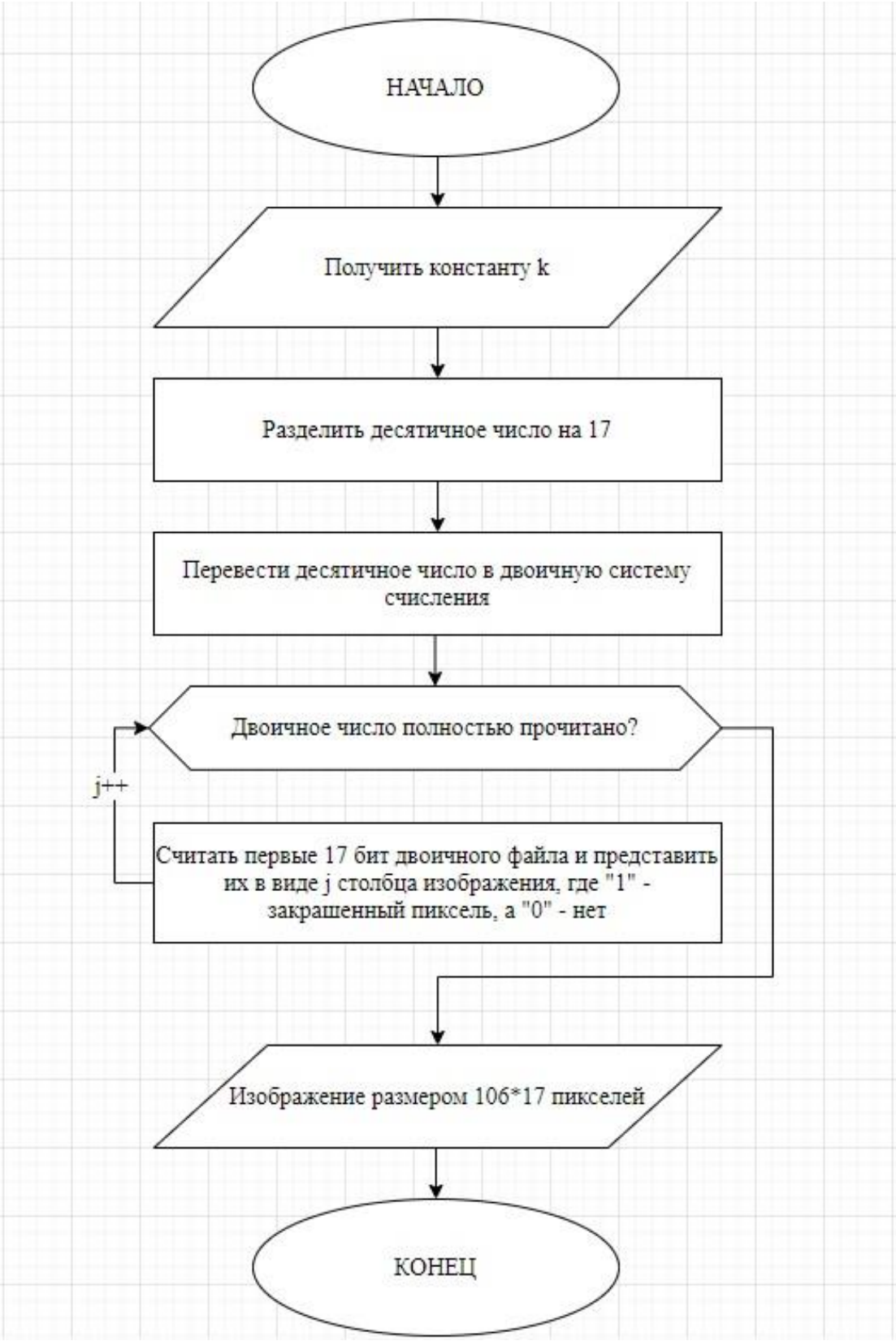

Рисунок 7 – Алгоритм построения изображения по константе k

Для визуального отображения работы программы было принято использовать язык разметки гипертекста HTML [11]. Так, было создано поле «grid-overlay», представляющее из себя сетку с 1802 переключаемыми маркерами. Таким образом была сымитирована возможность закрашивания пикселя. Иными словами, если пиксель закрашен, то будет включен соответствующий маркер в сетке «grid-overlay», возвращающий значение «true», в дальнейшем воспринимаемое программой как «1». Код и пример реализации этого этапа представлен на рисунках 8 и 9, соответственно:

```
▼<div id="grid-overlay">
  ▼<table>
    ▼<tbody>
      ▼<tr id="tr16">
          <td id="td0" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td1" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td2" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td3" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td4" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td5" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td6" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td7" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td8" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td9" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td10" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td11" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td12" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td13" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td14" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td15" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td16" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td17" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td18" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td19" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td20" class="unselected" style="width: 7px; height: 7px;"></td>
          <td id="td21" class="unselected" style="width: 7px; height: 7px;"></td>
```

Рисунок 8 – Фрагмент кода реализации поля «grid-overlay»

Рисунок 9 – Визуальное отображения поля «grid-overlay»

Для начала, было принято решение реализовать прямо алгоритм. То есть, алгоритм нахождения константы k из изображения. Для этого программа должна считывать каждый столбец снизу вверх, слева направо. Так, в зависимости от того, включен ли маркер в поле «grid-overlay» или нет, программа будет записывать в строку значения «1» или «0» соответственно.

Для записи была создана переменная «bitString» типа string. В нее по очереди записывались буквы «0» или «1» в зависимости от состояния маркера в сетке «grid-overlay». Фрагмент кода представлен на рисунке 10:

```
var getDecimalFromMap = function () {
    var bitString = "";
    for (var y = 0; y <= cols; y++) {
        for (var x = 0; x <= rows; x++) {
            var boolBit = $("table #tr" + x + " #td" + y).hasClass('selected') ? 1 : 0;

            bitString += boolBit;
        }
    }
    var decimal = new BigNumber(bitString, 2)

    setBitString(bitString);
    setDecString(decimal.times(17));
}
```

Рисунок 10 – Фрагмент кода реализации алгоритма

Как было сказано ранее, число, полученное в результате алгоритма достаточно большое, поэтому было принято решение использовать библиотеку «BigNumber.min.js», расположенную на общем ресурсе [13]. С помощью этой библиотеки можно производить операции с числами любого размера. С использованием встроенной в эту библиотеку функции «BigNumber (number, k)», где «number» – число, а «k» – это система счисления, этого числа, полученная константа k была переведена из двоичной системы счисления в десятичную и записана в переменную «decimal».

Далее было принято решение создать два дополнительных поля «bitArea» и «decArea», для графического отображения константы k в двоичной и десятичной системах счисления, соответственно. Так же, для удобства отображения, в поле «decArea» была добавлена функция отображения разрядов, представленная в виде маркера «showCommas» [14]. Фрагмент кода представлен на рисунках 11 и 12, соответственно:

```
"

    Двоичный код "
<span id="bitError" class="error"></span>
<textarea id="bitArea" placeholder="Двоичная строка"></textarea>
"

    Десятичный код("
<i>k</i>
") "
<small>
  <input type="checkbox" checked id="showCommas">
  "Показывать разряды?"
</small>
<span id="decError" class="error"></span>
<textarea id="decArea" placeholder="Переменная k"></textarea>
```

Рисунок 11 – Фрагмент кода реализации полей «bitArea» и «decArea»

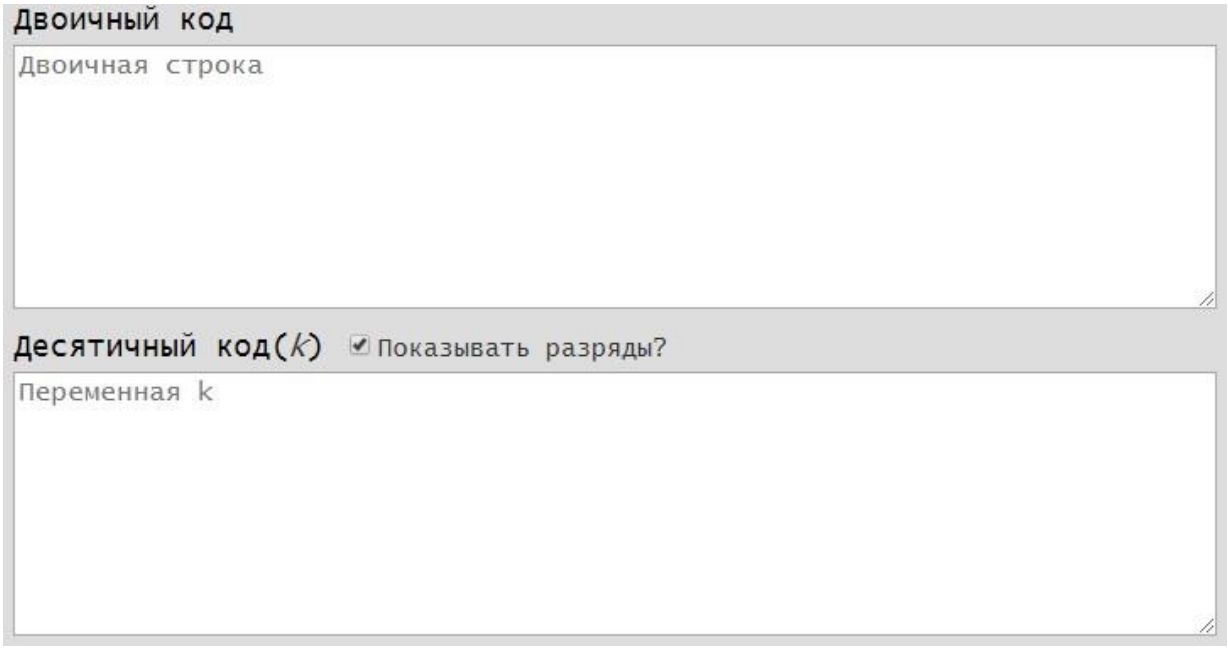

Рисунок 12 – Фрагмент графической реализации полей

Тогда, с помощью функций «setBitString()» и «setDecString()» константа k записывалась в поля «bitArea» и «decArea», соответственно. Функция «bitString()» записывала изначально считанную строку, представленную в двоичной системе счисления. Функция «setDecString()» использовала значение переменной «decimal», использованной ранее. Но так как в алгоритма, описанном в предыдущей главе, число переведенной в десятичную систему счисления должно быть умножено на 17, в функцию передается значения «decimal.times()», что означает переменную, умноженную на число в скобках. В данном случае эта функция будет выглядеть следующим образом: «setDecString(decimal.times(17))» [15]. Фрагмент кода функций «setBitString()» и «setDecString()» представлены на рисунке 13:

```
var setBitString = function(string) {
    if (string.length > gridArea) bitError.text("Thats too big!"); else bitError.text("");
    bitString = string.replace(/0+$/g, "");
    $("#bitArea").val(bitString);
}

var setDecString = function(bigNum) {
    decimal = bigNum;

    var decString = decimal.toFixed();

    convertToWords(decString);

    if ($("#showCommas")[0].checked) {
        decString = decimal.toFormat(3).slice(0, -4)
    }

    $("#decArea").val(decString);
}
```

Рисунок 13 – Фрагмент кода реализации вспомогательных функций

Использовав поле «decArea», был решен первый этап обратного алгоритма, а именно перевода константы k в изображения. Так как для перевода константы, сама константа должна быть использовано, было принято решение использовать то же поле «decArea». Таким образом, программа работала в обоих случаях:

1) При закрашивании пикселей в поле «grid-overlay» выводила константу k в поле «decArea»;
2) При заполнении поля «decArea» произвольной константой k закрашивала соответствующие пиксели в поле «grid-overlay».

Таким образом, поле «decArea» может содержать не только решение прямого алгоритма, то есть, итоговую константу k после работы с изображением, но и использовать введённое в это поле число как константу и строить с помощью нее изображение.

Для выполнения обратного алгоритма, то есть, перевода константы k в изображение, программа считывала введенное число из поля «decArea» и, разделив его на 17, переводила в двоичную систему счисления. На рисунке 14 представлен фрагмент реализации функции «setBitMap»:

```
var setBitMap = function() {
    var i = 0;
    for (var x = 0; x <= cols; x++) {
        for (var y = 0; y <= rows; y++) {
            var tr = $("#tr" + y);
            var td = tr.find("#td" + x);

            var bit = bitString[i];

            if (bit == "0" || !bit) {
                td.addClass('unselected');
                td.removeClass('selected');
            }
            if (bit == "1") {
                td.addClass('selected');
                td.removeClass('unselected');
            }
            i++;
        }
    }
}
```

Рисунок 14 – Фрагмент реализации функции «setBitMap»

С помощью этой функции считывается каждый следующий символ двоичного числа. В случае, когда считываемая цифра имела значение «1», соответствующий ей пиксель закрашивался, то есть, маркер принимал положение «selected». Аналогично, если считываемая цифра имела значение «0», маркер принимал значение «unselected».

3.2 Формирование графического интерфейса

Следующим этапом было принято решение сформировать графический интерфейс. Как было сказано ранее, для этого использовался язык разметки гипертекста HTML [16]. С использованием конструкции «<head>…</head>»

был создан элемент заголовка, куда загружались созданные ранее функции выполненные в виде скриптов с применением языка «JavaScript». А именно:

1) скрипт «index.js» содержащий описанные ранее функции, такие как «setDecString», «setBitString» и так далее;
2) скрипт «bignumber.min.js» содержащий в себе библиотеку работы с большими числами, функционал которой был описан в предыдущем разделе.

Далее были добавлены описанные ранее поля «decArea» и «bitArea», для вывода или считывания десятичных и двоичных чисел соответственно, а также поле «grid-overlay», представляющее из себя поле для отображения или считывания рисунка. Для удобства отображения чисел в поле «decArea» был добавлен маркер «показать разряды», при нажатии на который в десятичное число добавлялись «запятые» для разделения разрядов. Для примера работы программы так же было принято решение создать три кнопки с заранее прописанными значениями:

1) кнопка «Очистить» - очищающая поля «decArea», «bitArea» и «grid-overlay»;
2) кнопка «Самореферентная формула» - отправляющая в программу константу k описанную в предыдущей главе, и отображающую саму формулу неравенства Таппера;
3) кнопка «Перевернутая и зеркально отображенная» - отправляющая в программу такую константу k, чтобы итоговое изображение было идентичным с изображением кнопки «Самореферентная формула», но перевернутое и зеркально отображенное;
4) кнопка «ТГУ» - отправляющая в программу такую константу k, чтобы итоговое изображение выводило на экран аббревиатуру Тольяттинского Государственного Университета.

По окончанию проделанных выше действий была сформирована веб-страница «index.html», содержащая в себе описанные выше решения, а также заголовок «index.css» содержащая в себе информацию о разметке страницы, с использованием языка гипертекстовой разметки. То есть, как и где именно должны находится поля «decArea», «bitArea», «grid-overlay» и кнопки. А также дополнительную графическую информацию.

Таким образом, с использованием созданных ранее функций, можно было сформировать веб-страницы, выполняющую функции пользовательского интерфейса. Более того, такой тип представления программы позволял более удобным образом проводить дальнейший сравнительный анализ [14].

Фрагмент реализации веб страницы с фрагментом работы программ и фрагмент графического интерфейса представлен на рисунках 15, 16 и 17 соответственно:

```
    <link rel="stylesheet" type="text/css" href="index.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js" defer></script>
    <script src="bignumber.min.js" defer></script>
    <script src="numtowords.js" defer></script>
    <script src="index.js" defer></script>
</head>
<body>
    <h1 class="title">Самореферентая формула Таппера</h1>
    <p>Ниже вы можете сделать следующее:</p>
    <p>
        -Выбрать готовые заготовленные изображения<br>
        -Нарисовать в реальном времени собственное изображения<br>
        -Вставить двоичное число или переменную <i>k</i>, с дальнейшим отображением<br />
    </p>

    <div id="presets">
        <button p=0 dec="0">
        Очистить</button>
        <button p=2 dec="960939379918958884971672962127852754715004339660129306651505519271702802395
        Самореферентная формула</button>
```

Рисунок 15 – Фрагмент кода веб страницы «index.html»

```
textarea {
    font-size: 1rem;
    width: 100%;
    height: 150px;
}

#grid {
    position: absolute;
    display: block;
}

#grid-overlay {
    position: relative;
    border: 2px solid black;
    display: inline-block;
}

#presets * {
    font-size: 0.8em;
    margin: 0.3em;
    padding: 0.3em;
}
```

Рисунок 16 – Фрагмент кода страницы разметки «index.css»

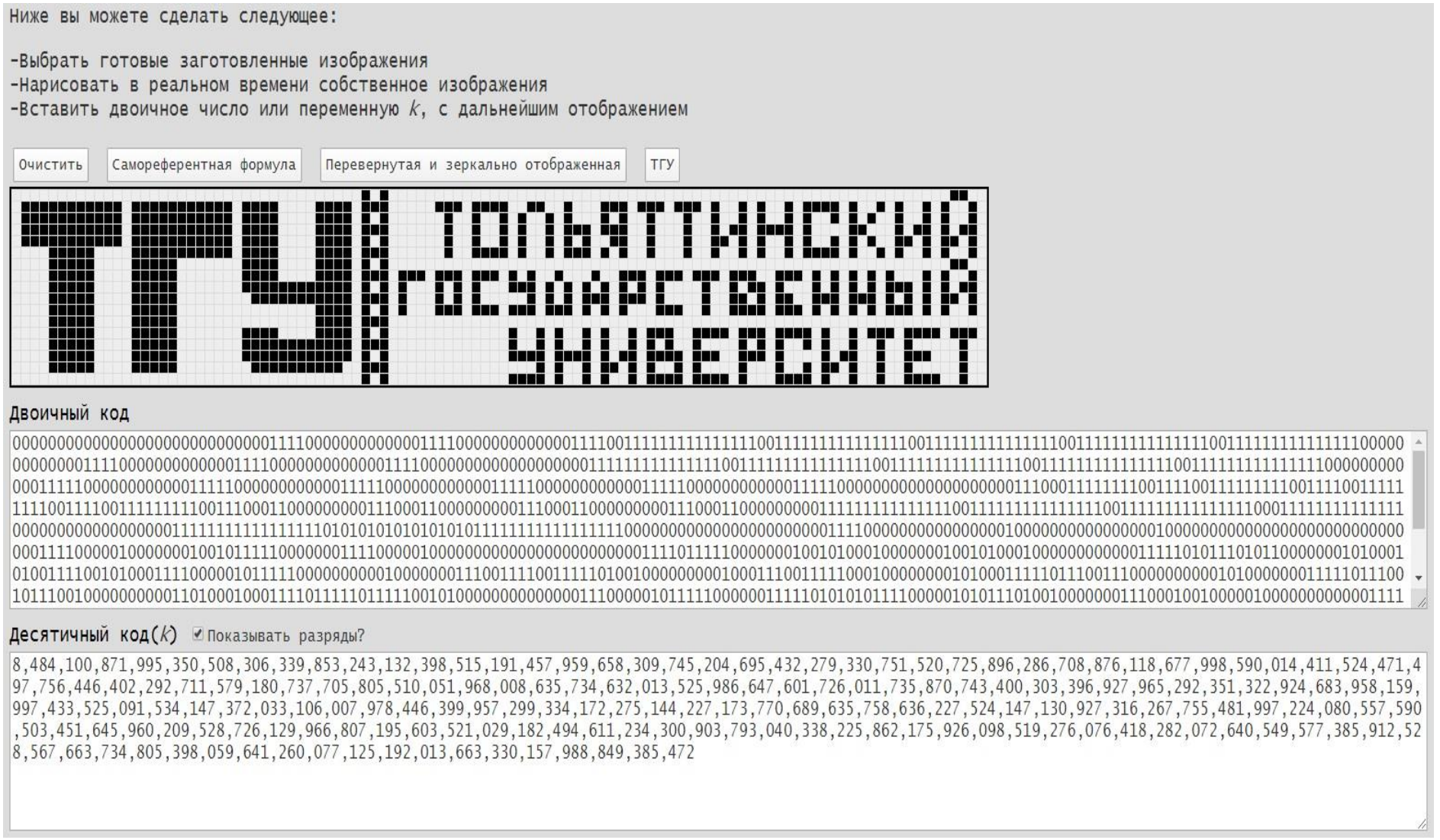

Рисунок 17 – Фрагмент работы программы

Таким образом, было полностью сформировано программное обеспечение. Как было описано ранее, функциональная часть программы была реализована на языке JavaScript, и использовалась в веб странице, реализованной на базе языка гипертекстовой разметки HTML. С помощью готовой программы можно было приступать к проведению сравнительного анализа.

3.3 Проведение сравнительного анализа

Как было описано в предыдущей главе, наиболее эффективной реализацией самореферентной формулы Таппера, является код, написанный математиком Джеймсом Гриме на язык программирования «Python» [10]. К сожалению, сам код недоступен в общественных источниках, поэтому было принято решение обратиться к самому Джеймсу Гриме. К счастью, Джеймс согласился принять участие в проведении научно-исследовательской работы и прислал код, написанный им в 2014 году. Методы, использованные Джеймсом Гриме схожи с методами, описанными в данной бакалаврской работе.

Функция «get_image()» содержит в себе следующие подфункции:

1) «get_k()» - считывает значение константы k с консоли;
2) «from_k_to_bin()» - переводит считанную константу k в двоичную систему счисления;
3) «image()» - рисует изображение используя двоичной число, реализованное в предыдущей подфункции.

Фрагмент реализации функции «get_image()» представлен на рисунке 18:

Рисунок 18 – Фрагмент реализации функции «get_image()»

Функция «generate_image()» наоборот, считывала двоичный код с изображения размером 106×17 пикселей в переменную «byteset». Фрагмент функции представлен на рисунке 19:

```
byteset = ""
for x in range(105, -1, -1):
    for y in range(0, 17):
        byte = str(image.getpixel((x, y)))
        if byte == "255":
            byteset += '1'
        else:
            byteset += '0'

k = int(byteset, 2) * 17

print("Все готово:")
print(k)
```

Рисунок 19 – Фрагмент функции «generate_image()»

На этом этапе все было готово к проведению сравнительного анализа [16]. Было принято решение провести его для трех различных переменных и шести промежуточных значениях, где константа $k_1 = 0$, константа k_2 принимает значение, отображающее формулу Таппера, а константа k_3 принимает максимально возможное значение:

1) константа $k_1 = 0$;
2) константа k_2 умноженная на 0,25;
3) константа k_2 умноженная на 0,5;
4) константа k_2 умноженная на 0,75;
5) константа k_2 принимает значение, отображающее формулу Таппера (это значение приведено в предыдущей главе);
6) к константе k_2 прибавляется разница между константами k_2 и k_3 умноженное на 0,25;
7) к константе k_2 прибавляется разница между константами k_2 и k_3 умноженное на 0,5;
8) к константе k_2 прибавляется разница между константами k_2 и k_3 умноженное на 0,75;
9) константа k_3 принимает максимально допустимое значение.

Аналогично был протестирован обратный алгоритм, так же для трех возможных изображений и шести промежуточных значений:

1) изображение не содержит закрашенных пикселей;
2) изображение содержит четверть формулы Таппера;
3) изображение содержит половину формулы Таппера;
4) изображение содержит три четверти формулы Таппера;
5) изображение содержит «саморферентную» формулу Таппера;
6) изображение на четверть закрашено, а на остальные три четверти содержит оставшуюся часть формулы Таппера;

7) изображение на половину закрашено, а на остальную половину содержит оставшуюся часть формулы Таппера;
8) изображение на три четверти закрашено, а на остальную четверть содержит оставшуюся часть формулы Таппера;
9) изображение содержит максимально возможное количество закрашенных пикселей, а именно 1802 закрашенных пикселя.

Пример этих изображений представлены на рисунках 20, 21, 22, 23, 24, 25, 26, 27 и 28, соответственно:

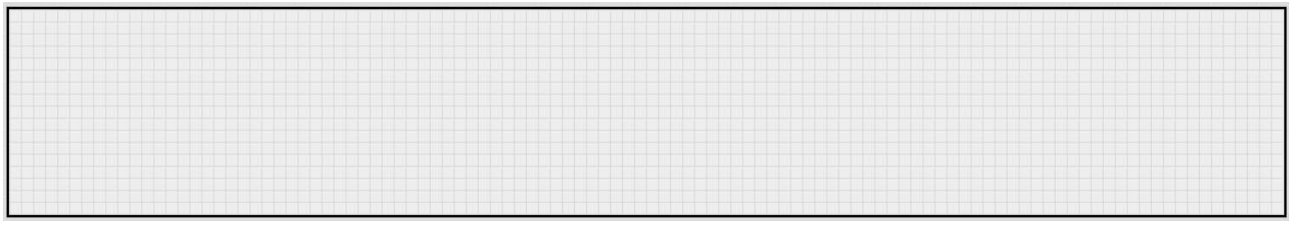

Рисунок 20 – Изображение, не содержащее закрашенных пикселей

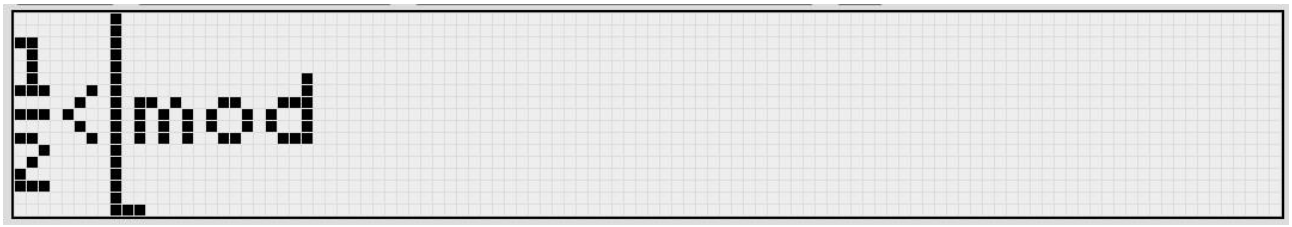

Рисунок 21 – Изображение, содержащее четверть формулы Таппера

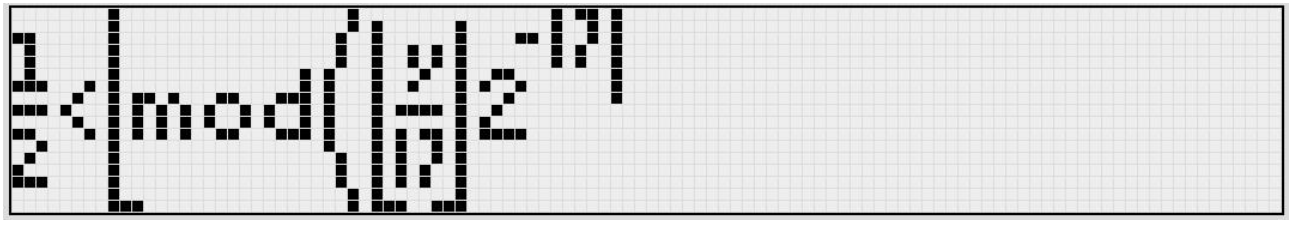

Рисунок 22 – Изображение, содержащее половину формулы Таппера

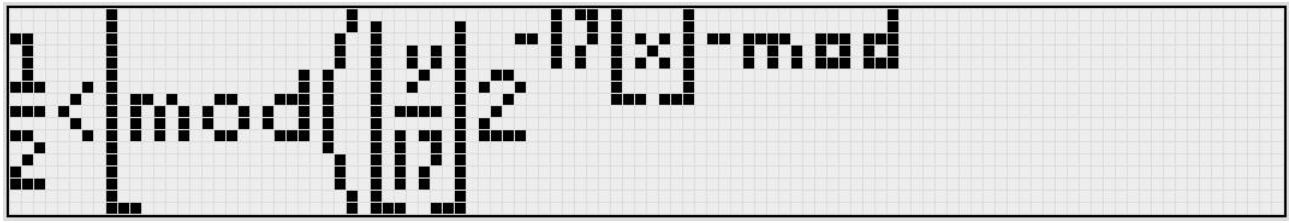

Рисунок 23 – Изображение, содержащее три четверти формулы Таппера

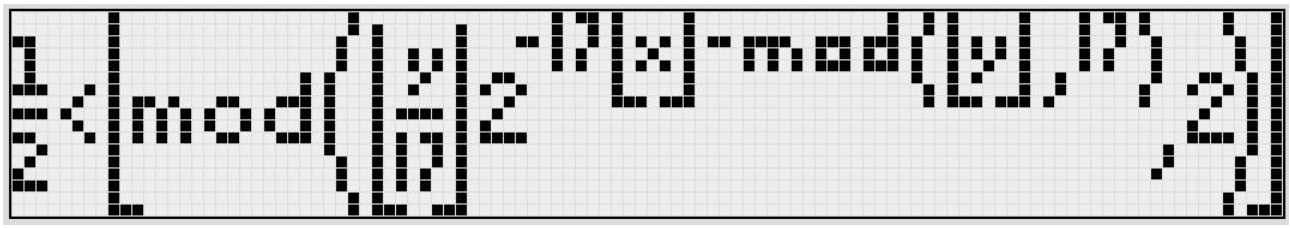

Рисунок 24 – Изображение, содержащее «самореферентную» формулу Таппера

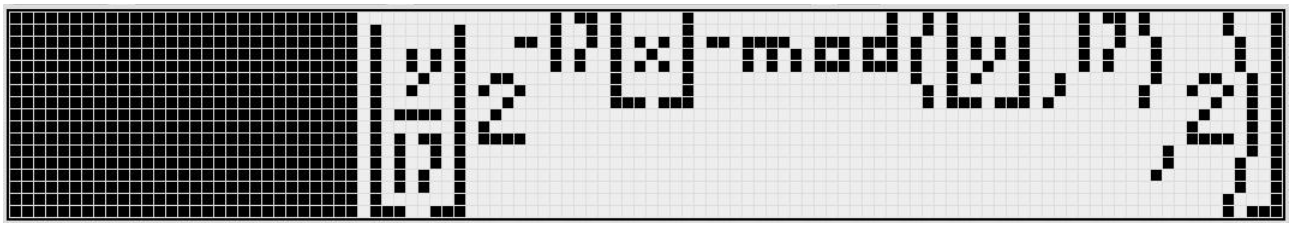

Рисунок 25 – Формула Таппера на четверть закрашено

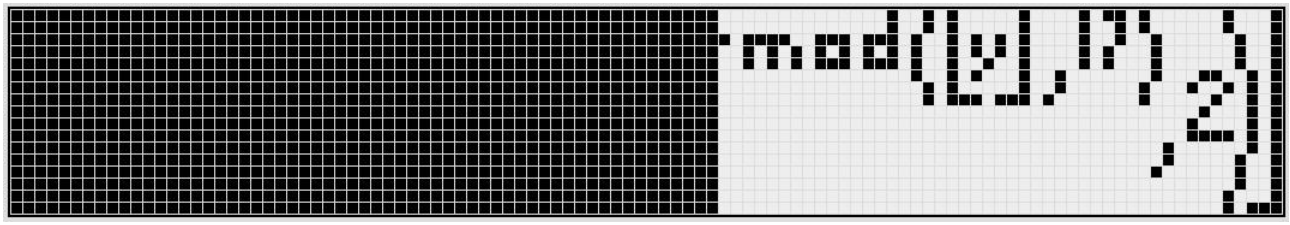

Рисунок 26 – Формула Таппера на половину закрашена

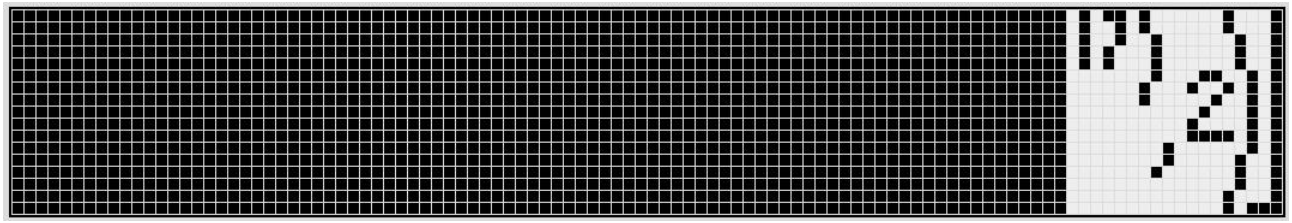

Рисунок 27 – Формула Таппера на три четверти закрашена

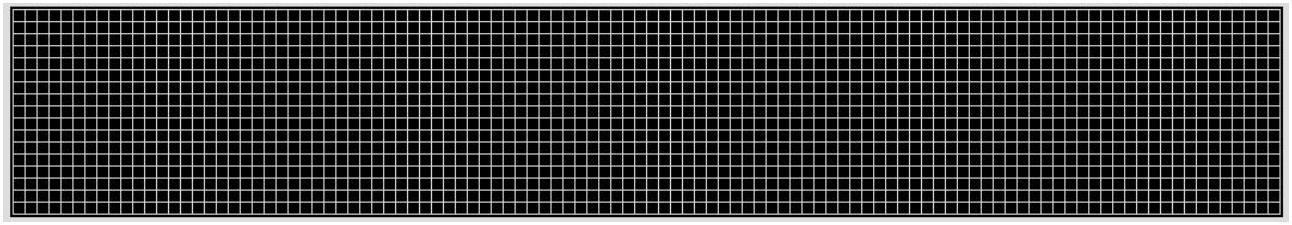

Рисунок 28 – Изображение, содержащее максимально возможное количество закрашенных пикселей

В качестве метода сравнения двух программ была выбрана их скорость. А именно, сравнение времени работы двух программ по каждому из описанных выше пунктов. Результаты скорости работы обоих программы представлены в таблицы 1 и таблице 2, где в столбцах представлена скорость работы алгоритма Д. Гриме и скорость работы алгоритма, реализованного в выпускной квалификационной работе, в секундах, а в строках – все три типа изображения. Для удобства отображения названий в таблице, изображения, представленные на рисунках 20-28 названы как «Picture 1», «Picture 2» и так далее:

Таблица 1 – сравнение работы прямого алгоритма

Тип переменной	Скорость работы программы Д. Гриме (с)	Скорость работы программы в ВКР (с)
$k_1 = 0$	1,98	0,001
$k_2 \times 0{,}25$	4,65	0,348
$k_2 \times 0{,}5$	7,96	0,723
$k_2 \times 0{,}75$	12,77	1,174
$k_2 = 960939 \ldots 719$	14	1,566
$k_2+(k_3-k_2) \times 0{,}25$	16,98	1,873
$k_2+(k_3-k_2) \times 0{,}5$	21,52	2,359
$k_2+(k_3-k_2) \times 0{,}75$	28,34	2,904
$k_3 = 4858487 \ldots 551$	30,12	3,286

Таблица 2 – сравнение работы обратного алгоритма

Тип изображения	Реализация программы Джеймса Гриме	Скорость работы программы в ВКР (с)
Picture 1	0,001	0,001
Picture 2	4,12	0,662
Picture 3	8,47	1,291
Picture 4	13,41	2,126
Picture 5	17,2	2,741
Picture 6	20,98	3,002
Picture 7	24,63	3,399
Picture 8	30,25	3,946
Picture 9	34,84	4,201

Как видно в таблице, в результате получилось оптимизировать алгоритм и добиться поставленной цели. Скорость работы обоих алгоритмов выросла в среднем на 800%, более того, получилось добиться и оптимизации пользовательского интерфейса.

Джеймс Гриме, разрабатывая свой алгоритм, придумал следующий метод: в программу отправлялось уже готовое, сделанное в отдельной программе, изображение размера 106×17 пикселей. Значит, для проведений обратного алгоритма нужно было такое изображение где-то приготовить. Более того, в случае ошибки в составлении изображения, то есть, случайно, изображение будет размером 107×17 пикселей, программа работать не будет.

В случае с реализованным интерфейсом на языке HTML [7], такой ошибки не возникнет, так как изначальное поле для построения ограничено. Более того, не требуется запускать отдельное приложение для составления графика. И, что не менее важно, скорость работы алгоритма в 8 раз быстрее.

С помощью полученных данных в результате проведения эксперимента можно будет построить графическую зависимость скорость работы алгоритмов от их сложности. Таким образом были составлены диаграммы по таблицам с результатами сравнительного анализа. На рисунке 29 представлена диаграмма сравнения скорости работы прямого алгоритма, то есть, алгоритма нахождения изображения из константы k, между программой, реализованной Джеймсом Гриме, и программой, реализованной в процессе выполнения бакалаврской работы. На рисунке 30 представлена диаграмма сравнения скорости работы обратного алгоритма, то есть, алгоритма нахождения константы k из изображения, между программой, реализованной Джеймсом Гриме, и программой, реализованной в процессе выполнения бакалаврской работы:

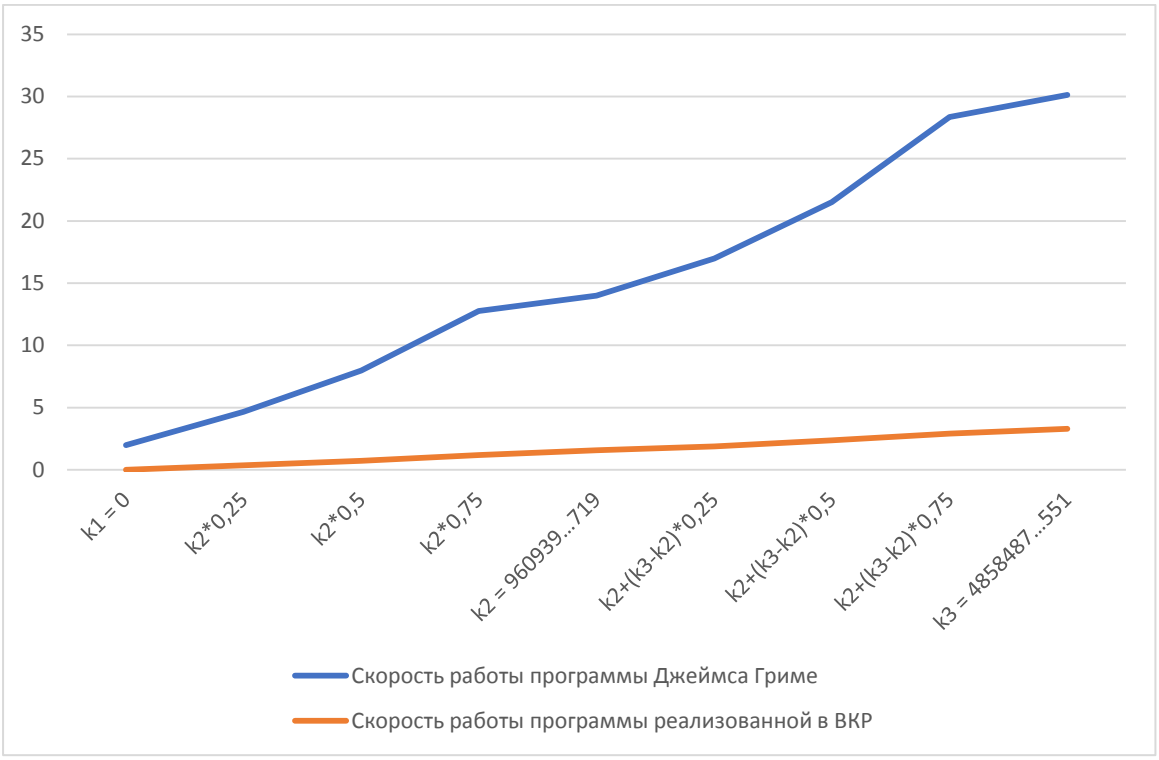

Рисунок 29 – Диаграмма сравнения скорости работы прямых алгоритмов

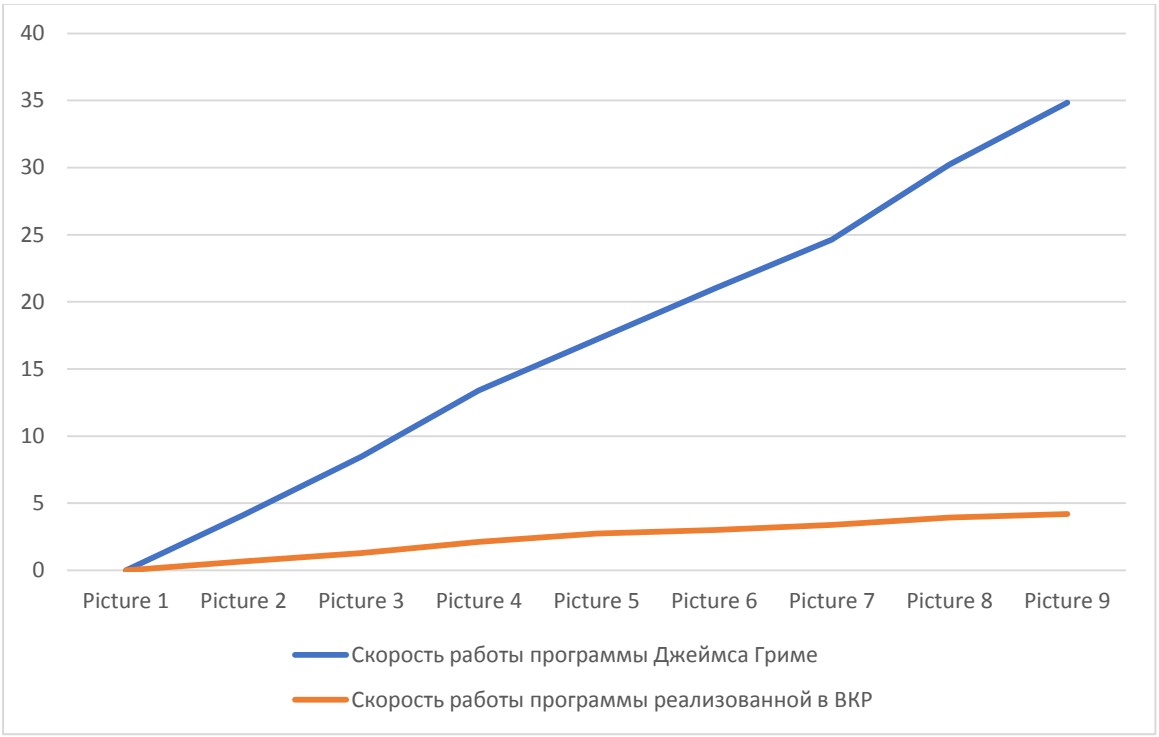

Рисунок 30 – Диаграмма сравнения скорости работы прямых алгоритмов

Как видно на графиках, скорость работы программы напрямую зависит от объема считываемой информации. Так, при полностью закрашенном изображении, программа Джеймса Гриме затрачивает в 2 раза больше времени, чем в случае, когда изображение соответствует «самореферентной» формуле Таппера. И в 8 раз больше времени, чем на то тратит программа, реализованная в бакалаврской работе при одинаковых условиях.

Также на графике видно, что скорость работы обратного алгоритма, то есть нахождения константы k из изображения, в несколько раз больше, скорости работы прямого алгоритма, то есть построения изображения по константе k. Так, например, скорость работы прямого алгоритма реализованного Джеймсом Гриме при максимальном значении константы k составляет 30 секунд, что на 4 секунды быстрее, чем скорость работы обратного алгоритма в тех же условиях. Аналогично, скорость работы прямого алгоритма, реализованного в результате выполнения бакалаврской работы, при обработке максимального значения константы k на 127% быстрее, чем скорость выполнения обратного алгоритма при обработке максимально закрашенного изображения, что соответствует максимальной константе k.

Отсюда следует, что оптимизация решения задачи «самореферентной» формулы Таппера прошла успешно.

ЗАКЛЮЧЕНИЕ

Тема бакалаврской работы была посвящена математической проблеме о «самореферентности», а также проблеме работы с большими объемами данных, на примере работы с большими числами.

В ходе выполнения данной работы был изучен теоретический материал, составленный Джеффом Таппером, сформулировавшим проблему. Был изучены несколько методов решения проблемы:

- Метод Джеффа Таппера;
- Метод Мэтта Паркера;
- Метод Джеймса Гриме.

После проведения анализа методов, для проведения дальнейшего сравнительного анализа был выбран последний метод – метод Джеймса Гриме, так как является наиболее эффективным.

В ходе дальнейшего выполнения работы было разработано программное обеспечение на базе языка программирования JavaScript, с использованием библиотеки по работе с большими объемами данных BigNumber. Дополнительно был реализован графический интерфейс на базе языка разметки HTML, для удобства использования и проведения сравнительного анализа.

В качестве примера, для проведения сравнения, был выбран код, разработанный Джеймсом Гриме. Код, практически без изменений, был дан самим Д. Гриме, реализованный на базе языка программирования Python.

Результат проведения сравнительного анализа показал, что с использованием современных методом программирования, в частности, с использованием методом библиотеки BigNumber, получилось оптимизировать работу программы в 8 раз.

Таким образом, на базе реализованного программного обеспечения в дальнейшем можно будет решать и другие математические проблемы, не

решенные ранее в связи с нехваткой технического ресурса, доступного в то время, или в связи со слишком большими объемами входных значений.

Программа, реализованная в ходе выполнения бакалаврской работы, также может быть рекомендована для обучения и преподавания такого аспекта математики, как «самореферентность». На примере ее работы и на примере трудов Джеффа Таппера, можно обучать студентов высших учебных заведений на тему решения математических парадоксов.

Таким образом, основным результатом процесса проектирования является программная реализация решения задачи «самореферентности», сформулированная Д. Таппером, и ее дальнейшая оптимизация. Результат этой работы может быть рекомендован в различные сферы математики и информационных технологий, в частности, как описано выше: в разделы о «самореферентности» и математических парадоксах, в математике; или в разделы о работе с большими объемами данных, в информационных технологиях.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Tupper, Jeff. «Reliable Two-Dimensional Graphing Methods for Mathematical Formulae with Two Free Variables», [Электронный ресурс]: Режим доступа: http://www.dgp.toronto.edu/people/mooncake/papers;
2. Bailey, D. H.; Borwein, J. M.; Calkin, N. J.; Girgensohn, R.; Luke, D. R.; and Moll, V. H. «Experimental Mathematics in Action». Natick, MA: A. K. Peters, p. 289, 2006;
3. Антонов А.С. Технологии параллельного программирования MPI и OpenMP: Учеб. пособие. Предисл.: В.А.Садовничий. - М.: Издательство Московского университета, 2012.-344 с.;
4. Comba, J.; Stolfi, J. «Affine Arithmetic and its Applications to Computer Graphics. In anais do VI Simp'osio Brasileiro de Compta, c'ao Gr'afica e Processamento de Imagens». (SIBGRAPI '93), p. 9-18, 1993;
5. Weisstein, E. W. «Tupper's Self-Referential Formula from MathWorld-A Wolfram Web», [Электронный ресурс]: Режим доступа: http://mathworld.wolfram.com/TupperSelf-ReferentialFormual.html.;
6. Marc Gregoire. Professional C++. Third Edition. John Wiley & Sons, Inc., 2014, P. 741-781;
7. Шляпкин А.В., Султанов Т.Г., Поведение потоков в среде исполнения Java [Текст] / Шляпкин А.В., Султанов Т. // Информационные технологии в моделировании и управлении: подходы, методы, решения: сб. статей – Тольятти, 2017. – С. 489-493;
8. Gregory R. Andrews. Foundations of Multhithreaded, Parallel and Distributed Programming. First Edition. University of Arizona, 2003, P. 324-477;
9. Лоусон Б., Шарп Р. Изучаем HTML5. Библиотека специалиста: Учеб. пособие. - П.: Издательский дом «Питер», 2012.-304 с.;

10. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования / А.С. Подосельник, А.Б. Ставровский, Г.И. Сингаевская – Киев : Вильямс, 2003. – 512 с.;

11. Карпов В. Е. Введение в распараллеливание алгоритмов и программ: учебное пособие/ В.Е. Карпов- М.: Изд-во МФТИ, 2014. – 272 с.Немнюгин С.А. Средства программирования для многопроцессорных вычислительных систем / С.А. Немнюгин. - СПб.: БХВ, 2013. - 88 с.;

12. Scott Meyers. Effective Modern C++. O'Reilly, 2015.MPI: A Message Passing Interface Standard Version 2.2. [Электронный ресурс]: Режим доступа: http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf;

13. OpenMP Application Program Interface. Examples. Version 4.0.1 February 2014 [Электронный ресурс]: Режим доступа: http://www.openmp.org;

14. Learning PHP, MySQL, JavaScript, CSS & HTML5: A Step-by-Step Guide to Creating Dynamic Websites Третье издание Автор: Робин Никсон Издательство: Питер: 2015;

15. Pro HTML5 with Visual Studio 2015 Автор: Mark Collins Издательство: Apress: 2015;

16. Web Database Application with PHP and MySQL, 2nd Edition By David Lane, Hugh E. Williams. O'Reilly, May 2004. ISBN: 0-596-00543-1.