УДК 004.92:004.451.53:511.33

В. О. Хамков

ПРОСТЫЕ ЧИСЛА В ФОРМЕ РОЗОВЫХ СЕРДЕЧЕК

Целью данной работы является создание метода генерации простых чисел, которые при их сохранении в файловой системе компьютера становятся графическими файлами с изображениями розового сердечка. Статья посвящена особенностям различных графических форматов, способам хранить большие числа в памяти компьютера, статистическим свойствам распределения простых чисел и вероятностным тестам простоты.

Введение. Простые числа – это одна из самых древних и фундаментальных тем в математике. Они изучаются уже несколько тысяч лет, но до сих пор человечеству не удавалось найти такое простое число, сохраняя которое в файл с расширением графического формата, мы получим графический файл с изображением розового сердечка на чёрном фоне. Программный поиск именно такого простого числа описывается в данной работе.

Хранение простых чисел в файловой системе компьютера. Как известно, вся информация в компьютерах хранится в виде последовательности бит (нулей и единиц), а в файловых системах размер файла измеряется в байтах, то есть в группках по 8 бит. Поэтому для хранения какого-либо числа в файле необходимо это число записать в двоичном или шестнадцатеричном виде, разделить на байты (группки по 8 бит) и сохранить эти байты в файл. Например, число 682312 в шестнадцатеричном виде можно записать как A6948, разбить на три байта в порядке от младшего разряда к старшему (причины такого направления будут описаны ниже): 48, 69, 0A, и сохранить их в файл с расширением txt. В результате мы получим текстовый файл в кодировке UTF-8, который содержит одну единственную строчку «Hi».

Проблема порядка байт: с большого конца или с малого. В современной технике числа представлены в виде последовательности байт, и порядок, в котором эти байты соотносятся с разрядами числа, имеет значение. В большинстве языков числа читаются слева направо, но в электронной технике зачастую числа хранятся в обратном порядке. Таким образом, число (в шестнадцатеричном виде) 12345ABC обычно хранится в памяти компьютера не в порядке 12, 34, 5A, BC, а в обратном ему порядке [1, с. 97]. В контексте данной работы порядок важен, так как от него зависит какой байт файла будет соответствовать младшему разряду простого числа. У популярных графических форматов, таких как PNG, JPEG и SVG, спецификацией зафиксированы специальные последовательности байт, которые должны стоять в начале [2, с. 12] и в конце [2, с. 18] файла с изображением. Например, JPEG файл всегда начинается с байтов FF, D8, FF, а заканчивается байтами FF, D9. Нетрудно заметить, что конкретно для JPEG-файла соответствующее ему число не делится на 2 независимо от выбранного порядка байт, но с PNG ситуация несколько иная: там последний байт файла должен быть 82. К сожалению, 82 делится на два, поэтому, выбирая порядок байт от старшего к младшему, мы сделаем невозможным создание такого простого числа, которое будет корректным PNG-файлом.

В этой работе было решено использовать порядок байт от младшего к старшему, то есть число 12345ABC будет сохранено в файл как последовательность байт BC, 5A, 34, 12. Но методология, описываемая в этой работе, применима и в случае использования порядка байт от старшего к младшему, но в таком случае станет невозможным использование формата PNG из-за чётности первого байта его сигнатуры [2, с. 12].

О рыхлости множества сердечек и простых чисел. Согласно теореме о распределении простых чисел [3, с. 5], у случайно выбранного числа длиной в некоторое количество цифр вероятность оказаться простым обратно пропорциональна количеству цифр в нём. В частности, для случайного файла размером в один килобайт вероятность того, что он случайно окажется простым числом, равна примерно 0,0001. А какой-нибудь простой формулы для генерации простого числа человечество ещё не придумало. С сердечками ситуация ещё хуже, так как и в PNG, и в JPEG присутствуют (и проверяются просмотрщиками) различные контрольные суммы частей файла [2, с. 12], поэтому у случайно выбранного числа практически нет шансов случайно оказаться корректным PNG или JPEG файлом. Поэтому оптимальнее всего разбить генерацию сердечек на следующие подзадачи:

1 Генерация множества различных корректных файлов, которые хранят одну и ту же картинку сердечка, но отличаются байтами. При этом алгоритм генерации должен так работать, чтобы максимизировать вероятность того, что файл случайно окажется записью простого числа.

2 Быстрый поиск простого числа среди этих файлов, т. е. проверка на простоту огромных чисел.

Для решения второй подзадачи обычно используются вероятностные тесты, которые с некоторой вероятностью могут дать неправильный ответ [4, с. 1012]. Для нахождения теста, наиболее подходящего целям данной работы, были реализованы следующие вероятностные тесты: тест Миллера-Рабина, тест Лемера и тест Ферма. Результаты измерения времени работы этих тестов приведены в таблице 1. В результате сравнения этих реализаций на случайных данных выяснилось, что оптимальнее всего тот алгоритм, который реализован кем-то другим, а именно тест Бейли-Померанца-Селфриджа-Уогстаффа, реализованный авторами библиотеки sympy. Этот тест тоже является вероятностным, поэтому теоретически существует такое составное число, которое проходит этот тест, но человечеству ещё не удалось найти такое число, потому далее полагается, что этот тест никогда не ошибётся на числах-сердечках.

Таблица 1 – Сравнение времени выполнения тестов простоты

Тест	Огромное число-сердечко	21-е число Мерсенна
Миллера-Рабина	5 секунд	9 секунд
Лемера	46 секунд	91 секунда
Ферма	3 секунды	4 секунды
БПСУ	2 секунды	2 секунды

Графические форматы: PNG, JPEG и SVG. В работе присутствует только описание использования формата PNG для хранения картинки с сердечком, но идея алгоритма та же и для форматов JPEG и SVG. Файл формата PNG из себя представляет список именованных массивов байт, называемых чанками [2, с. 12]. Также для каждого чанка хранится контрольная сумма его содержимого в формате CRC. Эти чанки бывают публичными (определёнными официальной спецификацией) и приватными (придуманными кем-нибудь для персонального использования), критическими (нужны для показывания картинки) и вспомогательными (обычно игнорируются просмотрщиками изображений), зависимыми и независимыми от критических чанков. Все эти свойства чанка закодированы в его названии и должны учитываться просмотрщиками и редакторами изображений [2, с. 14]. Согласно спецификации формата PNG, редакторы изображений имеют право вставлять какие-нибудь свои приватные чанки, но должны указывать вышеуказанные семантические свойства для того, чтобы просмотрщики изображений смогли корректно их обрабатывать.

Согласно спецификации, для корректного отображения картинки в файле обязательно должны присутствовать следующие критические чанки [2, с. 15]: IHDR (информация о ширине, высоте и других свойствах картинки), IDAT (пиксели изображения), IEND (без полезной информации). Существует очень мало возможностей что-либо поменять внутри этих чанков без изменения отображаемого изображения. Закодировать одно и то же изображение в тысячи разных файлов не получится, если использовать только эти чанки. После прочтения спецификации данного формата было принято решение, что наиболее идеологически верным решением в такой ситуации является создание нового приватного чанка, строго следуя спецификации формата PNG. Этому чанку следует указать такие семантические свойства, при которых просмотрщики будут игнорировать его содержимое. Тогда можно будет генерировать различные файлы с одинаковыми картинками, подставляя разные данные в этот чанк.

Генерация вспомогательного приватного чанка. Для изменения файла без изменения соответствующей ему картинки был разработан чанк под названием igNr. В названии не просто так прыгает регистр у букв: дело в том, что, согласно спецификации, семантические свойства чанка задаются регистрами букв его названия [2, с. 14]. В частности, маленький регистр буквы i означает то, что чанк не является критическим, а маленький регистр буквы g означает то, что чанк является приватным. Согласно спецификации, чанк с такими семантическими параметрами будет игнорироваться просмотрщиками изображений, а это именно то, что нужно для достижения наших благородных целей.

Максимизация вероятности числа оказаться простым. Для минимизации срока производства сердечка следует генерировать содержимое чанка igNr так, чтобы максимизировать вероятность того, что весь файл окажется простым числом. Для этого следует вспомнить, что простые числа не делятся на другие простые числа, поэтому наше большое число с кучей цифр не может делиться на 2, 3, 5 и другие маленькие простые числа. Поэтому перед перебором содержимого фиктивного чанка следует по свойствам делимости на какие-нибудь из маленьких простых чисел определить, какое содержимое чанка имеет смысл проверять. Например, по свойствам делимости на число 5 известно, что число x делится на 5 тогда и только тогда, когда на 5 делится его сумма цифр в 256-ричной записи. Говоря другими словами, если сумма байт файла делится на 5, то и число, соответствующее всему файлу, тоже делится на 5. Поэтому можно заранее посчитать сумму всех байт картинки, кроме нашего чанка, и откинуть те варианты содержимого чанка, которые в сумме с остальными байтами файла будут делиться на 5. Таким образом, воспользовавшись свойством делимости на 5 для пары байтов, можно достигнуть уменьшения количества потенциально простых чисел на 20 %, и мы получаем соответствующее ускорение в поиске простого числа. Аналогичные оптимизации следует произвести и для других простых чисел, помимо 5. Эти оптимизации ускоряют процесс поиска простых чисел в несколько раз, что в совокупности с быстрой проверкой на простоту позволяет находить число-сердечко в разумные сроки.

Таким образом, задача генерации простого числа-сердечка была успешно решена по следующему алгоритму действий:

1) Нарисовать картинку с сердечком и максимально компактно сжать её в чанк IDAT.

2) Подготовить чанки IHDR и IEND для будущего файла с картинкой.

3) По свойствам делимости посчитать необходимые суммы для нахождения подходящего содержимого чанка igNr.

4) Используя найденные суммы, составить список вариантов содержимого чанка igNr, которые максимизируют вероятность того, что содержимое файла окажется простым числом.

5) Составить варианты возможных простых чисел, используя разные варианты содержимого чанка igNr.

6) Тестом Бейли-Померанца-Селфриджа-Уогстаффа найти то из них, которое настоящее, а не просто притворяется простым.

7) Сохранить найденное число в файл с соответствующим расширением.

Заключение. Эволюция вычислительной техники, графических форматов, алгоритмов сжатия, а также труды великих учёных на протяжении тысяч лет в области теории чисел и теории информации наконец-то позволили создать то, что ранее казалось невозможным: розовое сердечко, которое ещё и простое число. В результате проделанной работы были найдены простые числа, которые являются графическими файлами в форматах PNG, SVG и JPEG (рисунок 1). Найденные простые числа, записанные в десятичной системе счисления, публично доступны на веб-странице https://phearts.github.io/.

Рисунок 1 – Найденные простые числа в форматах PNG, SVG и JPEG

Литература

1 Таненбаум, Э. С. Архитектура компьютера / Э. С. Таненбаум, Т. Остин. – Изд. 6-е. – Санкт-Петербург : Питер, 2013. – 816 с.

2 Randers-Pehrson, G. Portable Network Graphics Specification, Version 1.2 / G. Randers-Pehrson. – MIT, 1999. – 92 p.

3 Балазар, М. Асимптотический закон распределения простых чисел / М. Балазар. – Москва : МЦНМО, 2013. – 64 с.

4 Алгоритмы: построение и анализ / Т. Х. Кормен [и др.]. – Изд. 3-е. – Москва : Вильямс, 2013. – 1328 с.

УДК 004.52

Я. А. Шаповалов

СОЗДАНИЕ УМНОГО АССИСТЕНТА НА ЯЗЫКЕ PYTHON

Статья посвящена вопросам создания умного ассистента на языке программирования Python. В настоящее время помощник знает только английский язык. Ассистент может осуществлять задачи открытия веб-сайтов, отправку писем по e-mail, запуск системных приложений, воспроизведение песни, изменение обоев на рабочем столе компьютера, поиск информации в Wikipedia, сообщать текущее время и последние новости из новостной ленты google и реализовывать приветствие/завершение.