

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
УРАЛЬСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
имени первого Президента России Б. Н. Ельцина

ИНСТИТУТ ЕСТЕСТВЕННЫХ НАУК И МАТЕМАТИКИ

Кафедра алгебры и фундаментальной информатики

Модуль архивации JPEG файлов

Специальность 10.05.01 «Компьютерная безопасность»

Допустить к защите

Зав. Кафедрой:

д.ф.-м.н., профессор

М. В. Волков

Дипломная работа

Кулакова Владислава

Сергеевича

Нормоконтролер:

д.п.н., профессор

А. Г. Гейн

Научный руководитель:

к.ф.-м.н., доцент

Д. С. Ананичев

Екатеринбург

2021

РЕФЕРАТ

КУЛАКОВ В. С. МОДУЛЬ АРХИВАЦИИ JPEG ФАЙЛОВ, дипломная работа: стр. 27, библиография 13 названий, таблица 4, иллюстрация 5.

Ключевые слова: формат JPEG, цветовое пространство, дискретное косинусное преобразование, квантование матриц, код Голomba, код Хаффмана, архивация квантованных матриц.

Объект исследования: квантованные матрицы, генерируемые алгоритмом сжатия JPEG

Цель: разработка алгоритма архивации квантованных матриц в JPEG файлах, сравнение и анализ с другими алгоритмами архивации.

В работе выполняются следующие задачи:

- Изучить алгоритм получения квантованных матриц в формате JPEG;
- Описать различные эвристики процесса архивации квантованных матриц;
- Описать алгоритм хранения матриц, приведённый в формате JPEG;
- Описать и разработать алгоритм, позволяющий архивировать содержимое квантованных матриц;
- Оценить степень сжатия разработанного алгоритма относительно других известных программ архивации данных.

Результат: разработан алгоритм архивации, который позволяет уменьшить размер группы файлов JPEG, тем самым снизить нагрузку на память устройства хранения.

СОДЕРЖАНИЕ

РЕФЕРАТ.....	2
СОДЕРЖАНИЕ	3
ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ	4
ВВЕДЕНИЕ.....	6
ГЛАВА 1. ПОЛУЧЕНИЕ КВАНТОВАННЫХ МАТРИЦ В JPEG	8
1.1. Цветовое пространство.....	8
1.2. Дискретное косинусное преобразование.....	9
1.2. Квантование матриц	9
ГЛАВА 2. ХРАНЕНИЕ КВАНТОВАННЫХ МАТРИЦ.....	11
2.1. Зигзаг преобразование.....	11
2.2. Обрезка нулей.....	12
2.3. Экспоненциальный код Голомба.....	12
2.4. Серии нулей	14
2.5. Алгоритм Хаффмана.....	14
2.6. DC, AC коэффициенты	15
2.7. Алгоритм кодирования квантованных матриц в JPEG	16
ГЛАВА 3. АЛГОРИТМ АРХИВАЦИИ JPEG ФАЙЛОВ	18
3.1. Описание алгоритма архивации	18
3.2. Реализация алгоритма.....	19
3.2.1. Сборка частот и генерация кодов Хаффмана.....	19
3.2.2. Сжатие файлов и восстановление	21
3.2. Анализ эффективности	21
ЗАКЛЮЧЕНИЕ.....	24
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	26

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

JPEG (Joint Photographic Experts Group) - цифровой растровый формат представления изображений с потерями, разработанный для хранения фотографий.

RGB (Red, Green, Blue) - цветовое пространство, в котором каждый оттенок представлен в виде трёх компонент красного, зелёного, синего.

YCbCr - цветовое пространство, в котором три компоненты: Y (яркость), Color blue (доля синего), Color red (доля красного), представлены относительно зелёного цвета.

ДКП матрица - матрица, полученная из исходной путём применения к ней дискретного косинусного преобразования.

Квантованная матрица - ДКП матрица, прошедшая процедуру квантования (каждый коэффициент разделён на соответствующий в матрице квантования).

DC – крайний левый коэффициент в квантованной матрице (является основным и назван как постоянный ток (Direct Current)).

AC – все остальные не нулевые коэффициенты в квантованной матрице, переменный ток (Alternating Current).

END – маркер конца последовательности значащих коэффициентов.

MCU – (Minimum Coded Unit) минимальный кодируемый блок информации в формате JPEG состоит из матрицы 8x8 пикселей или нескольких матриц 8x8 коэффициентов по одной на каждую компоненту цветового пространства.

Категория – значение группы чисел по таблице формата JPEG (также их длина в двоичном представлении), по которому определяется, как считывать идущее следом число.

Зигзаг преобразование – преобразование двумерной матрицы 8x8 в одномерный массив из 64 коэффициентов.

Экспоненциальный код Голомба – универсальный код, позволяющий представить любые неотрицательные целые числа в виде бинарной последовательности.

Код Хаффмана – основанный на частотах встречаемости символов оптимальный префиксный код. (каждый символ кодируется уникальной бинарной последовательностью, декодируемой с её начала)

WinRAR – популярный архиватор компьютерных данных использующий алгоритмы rar и zip для сжатия файлов.

ВВЕДЕНИЕ

Важным аспектом информационной безопасности является сохранение целостности и доступности данных. В современном мире количество информации хранимой и передаваемой между людьми увеличивается очень быстро. Каждый год появляются сотни фильмов, тысячи музыкальных произведений, миллионы фотографий и множество других источников информации, хранимых на цифровых носителях. Число пользователей социальных сетей увеличилось за год на 13% и насчитывает порядка 4.2 миллиарда активных пользователей [1], которые каждый день пересылают, публикуют, сохраняют различную цифровую информацию. В период пандемии из-за перехода на удалённый режим количество трафика в Европе возросло, настолько, что пришлось занижать качество передаваемой информации [2]. В дополнение появляются новые законодательные акты, которые обязывают компании хранить информацию, передаваемую по их информационным каналам (сроком на 3 года в Пакете Яровой [3]). Всё это повышает нагрузку на устройства передачи и хранения информации.

Для обеспечения целостности данных также часто используется резервное копирование информации, но хранить информацию в исходном виде достаточно накладно, именно поэтому очевидна актуальность использования различных алгоритмов архивации чтобы хранимая и передаваемая информация занимала как можно меньше места.

Большинство алгоритмов архивации данных созданы для преобразования любых файлов будь то видео, изображение, музыка текст. Они используют набор различных алгоритмов для достижения максимального сжатия. Некоторые из них более специфичны и используют разные алгоритмы для различных типов информации в соответствии с их статистической значимостью, например цепочки повторяющихся символов хорошо сжимаются алгоритмом RLE, а в изображениях или видео файлах

лучше себя показывают коды Хаффмана. Существуют различные форматы сжатия с потерями, например такие как JPEG, где часть не значимой для человека информации убирается из результата.

В работе представлен ещё один подход, основанный на анализе структуры конкретных файлов. Так на олимпиаде NSUCRYPTO-2020 была представлена задача архивации квантованных матриц, генерируемых алгоритмом JPEG [4]. Как структура этих матриц, так и способ их получения являются объектами для анализа и генерации алгоритмов позволяющих уменьшать размер JPEG файла.

Цель работы: описать алгоритм и реализовать модуль способный архивировать JPEG файлы на основе его внутренней структуры

Для это был сформулирован следующий ряд задач для достижения цели:

- 1) Изучить структуру формата JPEG и процесс получения квантованных матриц.
- 2) Описать алгоритм хранения матриц в формате JPEG.
- 3) Представить вариант алгоритма архивации JPEG файла на основе его структуры.
- 4) Реализовать данный алгоритм в виде модуля архивации.
- 5) Сравнить его возможности с другими известными архиваторами (например WinRAR)

ГЛАВА 1. ПОЛУЧЕНИЕ КВАНТОВАННЫХ МАТРИЦ В JPEG

1.1. Цветовое пространство

Для создания формата хранения любого графического файла в первую очередь надо определить, как будут представляться цвета. Для этого определяется цветовое пространство. Самое распространённое цветовое пространство, используемое в компьютерной технике это **RGB** (Red, Green, Blue). В нём каждый цвет разбивается на 3 составляющих: красный цвет (Red), зелёный цвет (Green) и синий (Blue). Однако в JPEG принято использовать другое пространство **YCbCr**, которое было разработано для телевидения. Так как человеческий глаз наиболее восприимчив к зелёному цвету, он берётся за точку отсчёта. Y – определяет степень яркости, Cb (Color blue) – доля синего цвета, Cr (Color red) – доля красного.

Теперь пусть у нас есть сырое изображение в цветах RGB и на каждую компоненту отведён 1 байт информации 256 значений, то для преобразования в формат YCbCr нужно применить формулу [5]:

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix} \quad (1.1.1)$$

Если же мы хотим восстановить цвета, то используем обратную формулу:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.772 & 0 \end{pmatrix} * \begin{pmatrix} Y \\ Cb - 128 \\ Cr - 128 \end{pmatrix} \quad (1.1.2)$$

Однако стандарт JPEG не запрещает использование других цветовых пространств в том числе возможно использование и RGB, встречается СМΥК, но в основном преобладает YCbCr, поэтому ориентируемся на него.

1.2. Дискретное косинусное преобразование

Если увеличить JPEG изображение, то можно увидеть характерные квадраты. Это связано с тем, что картинка разбивается на матрицы 8x8 пикселей и дальнейшее преобразование будет производиться с ними. Каждый пиксель представляем в виде трёх числовых компонент в цветовом пространстве YCbCr. Поэтому каждый блок 8x8 пикселей представляют в виде трёх матриц на каждую компоненту. Эти три матрицы называют **MCU** (Minimal Coded Unit) или минимальный объект кодирования. Далее происходит процесс дискретного косинусного преобразования (ДКП) матриц, где x - номер строки, а y - номер столбца исходной матрицы, u - номер строки и v - номер столбца в результирующей соответственно [6].

$$F(u, v) = \frac{c(u, v)}{4} * \sum_{x=0}^7 \sum_{y=0}^7 \left(f(x, y) \times \cos\left(\frac{2*x+1}{16} u\pi\right) \times \cos\left(\frac{2*y+1}{16} v\pi\right) \right) \quad (1.2.1)$$

$$c(u, v) = \begin{cases} \left(\frac{1}{2}\right), & u = 0 \text{ и } v = 0 \\ 1, & u \neq 0 \text{ или } v \neq 0 \end{cases}$$

$$f(x, y) = \frac{1}{4} * \sum_{u=0}^7 \sum_{v=0}^7 \left(c(u, v) \times f(x, y) \times \cos\left(\frac{2*x+1}{16} u\pi\right) \times \cos\left(\frac{2*y+1}{16} v\pi\right) \right) \quad (1.2.2)$$

Применением формулы (1.2.1) преобразует матрицу 8x8 в ДКП матрицу такого же размера, но при этом в левом верхнем углу матрицы собираются низкочастотные коэффициенты, а в правом нижнем высокочастотные. Обратная формула (1.2.2) позволяет восстановить коэффициенты в YCbCr.

1.2. Квантование матриц

Именно на этапе квантования происходят потери части информации изображения. Человек плохо воспринимает высокочастотную информацию, поэтому слегка изменив значения в правом нижнем углу матрицы, картинка

почти не изменится. В свою очередь левый верхний коэффициент сосредотачивает в себе основную часть низкочастотной информации, и его потеря приведёт к значительным изменениям в изображении. На основании этого факта создатели формата JPEG предложили использовать матрицу квантования, например вот такую, описанную в стандарте [7]:

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Каждый коэффициент в ДКП матрице делится на соответствующий в матрице квантования, генерируя новую квантованную матрицу. В Квантованной матрице низкочастотные коэффициенты остаются значащими, а высокочастотные, как правило, обращаются в ноль. Пример получения квантованной матрицы:

$$\begin{bmatrix} -414 & -33 & -59 & 36 & 57 & -51 & -16 & -12 \\ 6 & -36 & 40 & 19 & 27 & 1 & -5 & 3 \\ -47 & 13 & 81 & -35 & -50 & 20 & 8 & -18 \\ -50 & 21 & 35 & -20 & 2 & 34 & 39 & 12 \\ 8 & -3 & 8 & -5 & -30 & -15 & 40 & 36 \\ -7 & 17 & -16 & 7 & -7 & 16 & 4 & 6 \\ 20 & -29 & -1 & -27 & -1 & 8 & -43 & -21 \\ 18 & 26 & -11 & -45 & 31 & 47 & -36 & 4 \end{bmatrix} \bigg/ \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} =$$

$$= \begin{bmatrix} -25 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 1 & -3 & 3 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Для каждой компоненты цветового пространства в JPEG могут храниться отдельные матрицы квантования.

ГЛАВА 2. ХРАНЕНИЕ КВАНТОВАННЫХ МАТРИЦ

2.1. Зигзаг преобразование

Для решения задачи компактного хранения квантованных матриц большинство алгоритмов начинают с зигзаг преобразования. Квантованная матрица 8x8 преобразуется в одномерный массив из 64 элементов путём обхода по диагоналям, начиная с левого верхнего угла, где собраны значащие коэффициенты в соответствии с рисунком 2.1.1 [4].

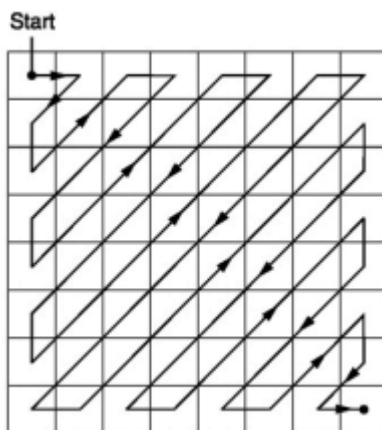


Рисунок 2.1.1 Последовательность зигзаг преобразования

Таким образом слева в массиве находятся низкочастотные элементы матрицы со значащими коэффициентами, а справа последовательность высоких частот, квантованная в нули.

$[-25, -3, 1, -3, -3, -6, 2, 3, 1, -4, 0, 1, 5, 1, 2, -1, 1, -1, 2, 0, 0, 0, 0, 0, -1, -1, 0, 0, \dots]$

Возможен и зеркальный вариант обхода матрицы, эквивалентный её транспонированию, таким образом можно увеличить количество нулей в конце последовательности, если последний значащий коэффициент находится ближе к краю своей диагонали.

$[-25, 1, -3, -6, -3, -3, -4, 1, 3, 2, 2, 1, 5, 1, 0, 0, 0, 2, -1, 1, -1, 0, 0, -1, -1, 0, 0, 0, \dots]$

2.2. Обрезка нулей

Для уменьшения размера полученного из матрицы массива можно обрезать его нулевой хвост. Чтобы это сделать есть 2 подхода:

1) Вставка количества значащих коэффициентов с начала массива или же количество нулей в конце массива. (В примере 26 значащих)

26: [-25, -3, 1, -3, -3, -6, 2, 3, 1, -4, 0, 1, 5, 1, 2, -1, 1, -1, 2, 0, 0, 0, 0, 0, -1, -1]

2) Вставка маркера конца значащих элементов массива (*END*).
Остальные элементы вплоть до 64 будут нули.

[-25, -3, 1, -3, -3, -6, 2, 3, 1, -4, 0, 1, 5, 1, 2, -1, 1, -1, 2, 0, 0, 0, 0, 0, -1, -1, *END*]

Так как в худшем случае количество значащих элементов ($2^6 = 64$) требует 6 бит информации, то маркер (*END*) должен занимать меньше, чтобы было выгодно использовать его.

2.3. Экспоненциальный код Голомба

Один из самых простых способов эффективного кодирования чисел является экспоненциальный код Голомба при равновероятном условии появления чисел. Каждый коэффициент представляется в виде трёх частей [4]:

1) Последовательность единиц равная длине модуля числа в двоичном представлении, завершающаяся нулём.

$$|-25_{10}| = 25_{10} = 11001_2 \Rightarrow 111110$$

2) Знак числа, если 0, то минус, иначе плюс.

$$-25 \Rightarrow 0$$

3) Двоичное представление модуля числа без ведущей единицы.

$$|-25_{10}| = 25_{10} = 11001_2 \Rightarrow 1001$$

Так как 0 не имеет знаков и ведущей единицы, то будем использовать, не используемый код 0 и коды чисел будут выглядеть так:

$$-25 \Rightarrow 111110\ 0\ 1001$$

$$1 \Rightarrow 10\ 1$$

$$-1 \Rightarrow 10\ 0$$

Однако было замечено, что коэффициенты распределены неравномерно, поэтому код был недостаточно эффективен. В JPEG используется подобная структура для преобразования чисел называемая категории. Каждое число в JPEG представляется в виде пары (категория, код в категории) по таблице 2.3.1 [7]:

Таблица 2.3.1 Соответствие чисел и категорий

Значения	Категория	Код
0	0	-
-1, 1	1	0, 1
-3, -2, 2, 3	2	00, 01, 10, 11
-7,-6,-5,-4,4,5,6,7	3	000,001,010,011,100,101,110,111
⋮	⋮	⋮
-32767...-16384, 16384,,32767	15	...

Категория — это длина двоичного представления чисел, для положительных чисел, код в категории эквивалентен их двоичному представлению, для отрицательных он инвертирован.

$$5 \Rightarrow 5_{10} = 101_2 \Rightarrow 101$$

$$-5 \Rightarrow 5_{10} = 101_2 \Rightarrow 010$$

Но сами категории записаны не количеством единиц, а кодами Хаффмана.

2.4. Серии нулей

Создатели формата JPEG так же заметили, что часто встречается ситуация, когда группа значимых коэффициентов возникает в массиве после достаточно больших групп нулей, например может возникнуть не нуль в правом нижнем углу матрицы, то есть где-то в конце массива. Это проблема ведь тогда обрезка нулей не особо помогает. Было принято решение дополнительно кодировать длины нулей. Сейчас каждый коэффициент массива кодируются парой (категория, код в категории), теперь расширим его до тройки (количество нулей до символа, категория, код в категории), будем называть (нули, категория, код).

$$[2,0,0, \mathbf{0}, \mathbf{0}, \mathbf{0}, -1] \Rightarrow (0, 2, 10), (5, 1, 0)$$

Всего категории ограничены количеством от 0 до 15, это 16 категорий, 4 бита, для выравнивания на байт (8 бит), так же решили ограничить количество нулей. Если нулей больше 16, то они кодируются в несколько троек (15,0,-) тройка означает 16 нулей, максимальная серия. Также очень удобно используется тройка, (0,0,-) как маркер **END** конец значащих элементов массива.

2.5. Алгоритм Хаффмана

В коде Голомба длины кодируются условно в единичной системе счисления, что очень накладно и коэффициенты большего размера в матрице могут встречаться чаще, чем меньшего, что и привело к использованию категорий. Эти категории в алгоритме JPEG кодируются алгоритмом Хаффмана (однако в отличие от кода Голомба это требует предварительного создания таблицы частот). В общем случае кодируется пара (нули, категория), код в категории уже закодирован.

Кодирование Хаффмана строит оптимальный префиксный код на основе частоты встречаемости символов [10]. Строится дерево, где жадным алгоритмом на каждом выбирается 2 реже встречаемых узла и объединяются в один, каждому узлу присваивается код 0 или 1, а объединённый узел добавляется в следующий шаг с суммарным весом. Потом по дереву Хаффмана (см. рисунок 2.4.1) восстанавливаются коды от результирующего узла (корня) к листьям обратно.

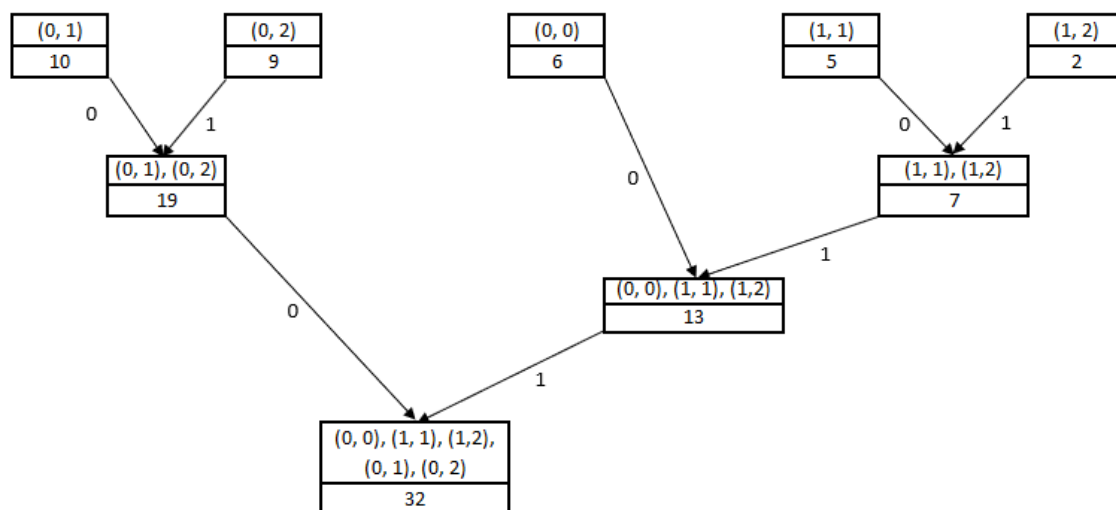


Рисунок 2.4.1 Дерево Хаффмана

В каждом JPEG файле хранится информация о используемых кодах Хаффмана в формате (длина кода, символы). Восстановление кодов начинается с кода 0...0 наименьшей длины [8].

2.6. DC, AC коэффициенты

В квантованных матрицах самый верхний левый коэффициент обычно очень большой по сравнению с остальными, в нём сконцентрирована основная информация изображения, его называют DC коэффициентом или постоянным током (Direct Current). Остальные коэффициенты называют AC или переменным током (Alternating Current). Так как коэффициенты существенно различны и у первого не может быть нулей предшественников, то было принято решение его категорию кодировать отдельно. Также чтобы

избежать достаточно больших кодов символов и при условии, что на фотографиях мало резких переходов, так что все DC коэффициенты рядом близки по значению, вместо самих коэффициентов хранится разница между текущим и предыдущим. Эта разница, как правило, такая же маленькая, как и AC коэффициенты. В итоге в JPEG файле хранят, как правило, по 2 кода Хаффмана на каждую компоненту, так как значение компонент тоже имеют разницу. На деле же разницу в основном имеет только компонента яркости (Y), поэтому Cb и Cr имеют одинаковые коды.

2.7. Алгоритм кодирования квантованных матриц в JPEG

Сформируем результирующий алгоритм преобразования квантованных матриц в алгоритме JPEG:

- 1) Применяя Зигзаг преобразование преобразуем квантованную матрицу в массив.
- 2) Отделяем первый DC коэффициент массива. Считаем разницу с предыдущим DC или с нулём, если матрица первая.
- 3) Определяем категорию разницы и записываем её код Хаффмана DC.
- 4) Записываем разницу по коду в категории.
- 5) Обрезаем нули в конце массива и добавляем маркер **END** = (0,0,-) - тройка, если их меньше 63 (64-ый DC уже отделили).
- 6) Сворачиваем коэффициенты массива в тройку (нули, категория, код), получается массив троек.
- 7) Для каждой тройки записываем код Хаффмана AC пары (нули, категория).
- 8) Записываем код AC коэффициента по категории, последний из тройки.
- 9) Переходим к следующей тройке пока они не закончатся.

Такой алгоритм сжимает ~20% лучше кода Голомба, однако требует подготовки кодов Хаффмана. Обычно готовится 4 кода Хаффмана:

- 1) DC код для коэффициентов яркости (Y).
- 2) AC код для яркости (Y).
- 3) DC код для доли красного и синего (Cb и Cr).
- 4) AC код для Cb и Cr.

ГЛАВА 3. АЛГОРИТМ АРХИВАЦИИ JPEG ФАЙЛОВ

3.1. Описание алгоритма архивации

По описанному формату JPEG видно, что в каждом файле JPEG зашита последовательность квантованных матриц, которые имеют сходную структуру. Так как эти таблицы генерируются кодами Хаффмана, они плохо сжимаются многими стандартными алгоритмами архивации так как в их основе тоже лежат коды Хаффмана. Но если бы у них была возможность кодировать исходные матрицы, то возможно их качество сжатия увеличилось. Представим возможный алгоритм архивации основанный на JPEG кодирование:

- 1) Декодируем матрицы каждого JPEG файла.
- 2) Уберём коды Хаффмана сгенерированный для файлов.
- 3) Соберём частоты встречаемых категорий для DC и AC коэффициентов матрицы для каждой компоненты цветового пространства по всем файлам.
- 4) Построим коды Хаффмана для этих частот.
- 5) Закодируем квантованные матрицы каждого файла используя сгенерированные коды Хаффмана.
- 6) Сохраним их в файлы и отдельно файл с кодами Хаффмана.

Такой алгоритм можно представить себе как большой файл JPEG. Однако для каждого JPEG файла есть ограничения на его размер и свои матрицы квантования поэтому показать его не представляется возможным. Алгоритм же просто вырезает из файлов матрицы Хаффмана и перекодирует информацию в соответствии со своими. Для восстановления исходных файлов мы снова декодируем матрицы по нашим кодам и генерируем новые по частотам для каждого файла, восстанавливая структуру. Файлы могут не совпасть ведь для кодов Хаффмана одинаковой длины нет строгого порядка

символов, однако сами матрицы, а следовательно, и изображение будет восстанавливаться однозначно.

3.2. Реализация алгоритма

Алгоритм был реализован на 2 языках в виде маленьких приложений C++, каждое из которых выполняет свою подзадачу и скрипт сжатия *compress.py* на языке Python3, аналогично скрипт *decompress.py* производит обратный процесс, восстанавливая исходные файлы JPEG. Оба скрипта принимают на вход папку с файлами. Далее файлы jpeg стандартного формата фильтруются, в стандарте JPEG описаны реже используемые форматы с другой структурой, цветовыми пространствами и даже без потерь, к которым нужен другой подход, поэтому обрабатываются только файлы стандартной структуры (Baseline).

3.2.1. Сборка частот и генерация кодов Хаффмана

За сборку частот отвечает программа *jpegstats.exe*, написанная на языке C++. Принимает на вход JPEG файл (*[file].jpg*) и сохраняет в новый файл, (*[file].jpg.stats*) представленный на рисунке 3.2.1.1 частоты категорий по компонентам и типу.

```

1 Y·DC
2 {0: 49, 1: 111, 2: 177, 3: 304, 4: 404, 5: 491, 6: 480, 7: 278, 8: 175,
9: 31}
3 Y·AC
4 {0: 2494, 1: 10883, 2: 7889, 3: 5369, 4: 3543, 5: 2049, 6: 985, 7: 412,
8: 101, 9: 9, 17: 3188, 18: 1170, 19: 397, 20: 159, 21: 40, 22: 9, 23: 7,
24: 1, 33: 1467, 34: 299, 35: 48, 36: 6, 37: 3, 39: 1, 49: 828, 50: 110,
51: 18, 52: 2, 65: 625, 66: 64, 67: 4, 68: 2, 81: 443, 82: 31, 83: 2, 97:
263, 98: 20, 113: 164, 114: 8, 115: 1, 129: 92, 130: 4, 145: 76, 146: 8,
161: 49, 162: 2, 177: 30, 178: 3, 193: 22, 209: 15, 225: 12, 240: 8, 241:
7}
5 Cb·DC
6 {0: 336, 1: 439, 2: 538, 3: 510, 4: 432, 5: 186, 6: 56, 7: 3}
7 Cb·AC
8 {0: 2500, 1: 2865, 2: 1559, 3: 645, 4: 145, 5: 24, 6: 1, 17: 1032, 18:
266, 19: 61, 20: 4, 21: 3, 33: 483, 34: 42, 35: 5, 36: 1, 49: 276, 50:
13, 65: 119, 66: 5, 81: 47, 97: 41, 113: 11, 129: 4, 145: 11, 161: 3,
162: 1, 177: 7, 209: 6, 240: 2}
9 Cr·DC
10 {0: 622, 1: 570, 2: 598, 3: 367, 4: 190, 5: 115, 6: 35, 7: 3}
11 Cr·AC
12 {0: 2500, 1: 2078, 2: 772, 3: 351, 4: 113, 5: 18, 6: 2, 17: 794, 18: 174,
19: 30, 20: 4, 33: 317, 34: 17, 35: 3, 49: 215, 50: 14, 51: 1, 65: 101,
66: 2, 81: 57, 82: 1, 97: 47, 113: 22, 129: 4, 145: 6, 161: 8, 177: 2,
193: 4, 209: 6, 240: 2, 241: 2}
13

```

Рисунок 3.2.1.1 Частоты категорий в матрицах

Сгенерированные частоты основной скрипт *compress.py* считывает и объединяет в единый набор для каждой компоненты и типа. Затем используя алгоритм Хаффмана собирает дерево Хаффмана и генерирует файл с такой же структурой хранения, как и в алгоритме Хаффмана (см. Рисунок 3.2.1.2).

```

1 16
2 Y·DC
3 0 1 4 3 1 1 1 1 0 0 0 0 0 0 0
4 4 3 5 6 7 1 2 8 0 9 10 11
5 Y·AC
6 0 2 2 1 3 2 4 3 6 3 3 5 5 3 3 85
7 1 2 3 17 4 0 18 33 5 49 6 19 34 65 7 81 97 8 20 50 113 129 145 35 161 240 21 66 177 9 82
193 209 225 22 36 51 98 241 23 114 130 10 67 146 24 25 26 37 38 39 40 41 42 52 53 54 55
56 57 58 68 69 70 71 72 73 83 84 85 86 87 88 89 99 100 101 102 103 104 105 115 116 117
118 119 120 121 131 132 133 134 135 136 147 148 149 150 151 152 162 163 164 165 166 167
168 178 179 180 181 182 183 194 195 196 197 198 210 211 212 213 214 215 226 227 228 229
230 242
8 Cb·DC
9 0 3 1 1 1 1 1 1 1 1 0 0 0 0 0
10 0 2 3 1 4 5 6 7 8 9 10 11
11 Cb·AC
12 0 2 2 1 3 2 3 6 4 3 6 2 4 4 5 37
13 0 1 2 17 33 3 18 49 4 65 5 81 97 6 19 34 50 113 240 129 145 161 177 7 20 193 21 35 66 209
225 241 51 82 8 22 36 178 52 98 114 146 23 37 67 130 162 9 10 24 25 26 38 39 40 53 54 55
56 68 69 70 71 83 84 85 86 99 100 115 116 131 132 147 148 163 179 194 195 210 211 226 227
242
14 Cr·DC
15 1 0 3 1 1 1 1 1 1 1 0 0 0 0 0
16 0 1 2 3 4 5 6 7 8 9 10 11
17 Cr·AC
18 0 2 2 1 3 2 3 6 4 3 6 3 2 3 8 35
19 0 1 2 17 33 3 18 49 4 65 5 81 97 6 19 34 50 113 240 7 129 145 161 20 177 193 21 35 66 209
225 241 8 51 82 22 178 36 98 130 23 37 52 67 114 146 179 194 9 24 25 38 39 40 53 54 55 56
68 69 70 71 72 83 84 85 99 100 115 116 131 132 147 148 162 163 180 195 196 210 211 226 242
--

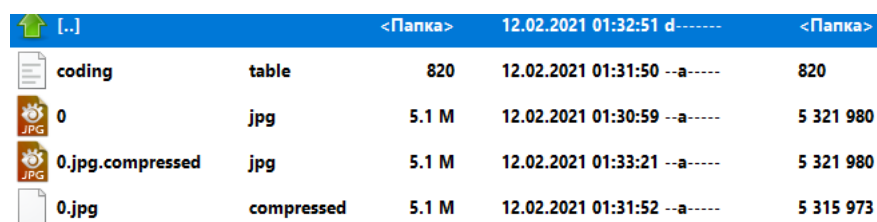
```

Рисунок 3.2.1.2 Последовательность кодов Хаффмана

3.2.2. Сжатие файлов и восстановление

Программа *jpegcompress.exe* написана на языке C++ и занимается перекодированием исходных файлов используя файл с кодами Хаффмана полученный ранее. На вход получает соответственно оба файла. И собирает файл JPEG по алгоритму из прошлой главы. Сохраняет новый файл в *[file].jpg.compressed*.

Обратную процедуру производит программа *jpegdecompress.exe*, написанная на C++. На основе файла с кодами и сжатого файла она восстанавливает квантованные матрицы, собирает новую статистику для них, генерирует коды по этой статистике и восстанавливает исходный JPEG файл (см. Рисунок 3.2.2.1).



[...]		<Папка>		12.02.2021 01:32:51 d-----		<Папка>	
coding	table	820	12.02.2021 01:31:50 --a----	820			
0	jpg	5.1 M	12.02.2021 01:30:59 --a----	5 321 980			
0.jpg.compressed	jpg	5.1 M	12.02.2021 01:33:21 --a----	5 321 980			
0.jpg	compressed	5.1 M	12.02.2021 01:31:52 --a----	5 315 973			

Рисунок 3.2.2.1 Исходный, сжатый, восстановленный файлы

3.2. Анализ эффективности

На рисунке 3.2.2.1 приведён пример отдельно сжатого файла, со степенью сжатия ~ 0.998 , что меньше 1%, тем не менее архиватор WinRAR [9] вообще не может сжать отдельно этот файл в формате *rar*, формат *zip* тоже сжимает слабее (см. таблицу 3.2.1).

Таблица 3.2.1 Сравнение алгоритма с встроенными в WinRAR

Тест (описание)	Размер файлов (Б-байт)	Алгоритм <i>compress.py</i>	WinRAR rar (32 Б) обычный	WinRAR rar (1024 Б) максимальный	WinRAR zip
0 (один файл)	5 321 980	5 315 973 (0.9989)	5 322 131	5 322 131	5 316 599
1 (много разных файлов)	362 227 195	352 042 512 (0.9719)	357 456 809	357 452 689	358 807 3 93
2 (15 одного типа)	1 254 765	1 251 105 (0.997)	1 255 457	1 255 457	1 252 307
3 (набор фото одного устройства)	3 339 347 585	3 316 346 2 18 (0.9931)	3 334 708 038	3 334 705 146	3 329 845 467
5 (серия из 4 фото)	13 395 043	13 072 305 (0.9759)	13 218 699	13 218 638	13 218 668

Замечу, что помимо вырезания таблицы Хаффмана, алгоритм так же не объединяет компоненты синего и красного цвета, как это обычно делает jpeg, что иногда может улучшить сжатие.

Очевидно, что алгоритм теряет свою актуальность при использовании малого количества файлов, так как напрямую зависит от количества кодов Хаффмана, которые он заменяет. Ещё эффективность алгоритма зависит от размера файлов, чем они меньше, тем большую часть файла занимает код Хаффмана.

В общем случае сжимает на произвольном большом наборе файлов до 1-3%, на маленьком наборе, меньше 0.3%, однако стандартные алгоритмы программы WinRAR иногда вообще не производят сжатия. Стоит пояснить, что рассматриваются файлы с кодами Хаффмана оптимальной длины, так как в общем случае JPEG не регламентирует какой тип кодов использовать и некоторые программы генерируют или используют коды, не основываясь на

частотах. Поэтому для файлов с неоптимальными кодами алгоритм сжимает видоизменённые до оптимальных кодов файлы, и восстанавливает не в точности исходные, а видоизменённые, но, так как сами матрицы одинаковые во всех трёх состояниях, (исходном, сжатом и восстановленном) то изображение тоже выглядит идентично. И потерь данных изображения нет.

Таблица 3.2.2 Дополнительное сжатие алгоритмом zip

Тест (описание)	Размер файлов (Б- байт)	Алгоритм <i>compress.py</i> сжатый zip
0 (один файл)	5 321 980	5 311 298 (0.998)
1 (много разных файлов)	362 227 195	348 812 823 (0.963)
2 (15 одного типа)	1 254 765	1 249 709 (0.996)
3 (набор фото одного типа)	3 339 347 585	3 311 383 122 (0.9916)
5 (серия из 4 фото)	13 395 043	12 972 457 (0.9685)

В таблице 3.2.2 показано, что zip архивирование способно улучшить степень сжатия предварительно архивированных *compress.py*. Таким образом алгоритм архивации JPEG файлов можно использовать, как предварительное сжатие, для других алгоритмов, чтобы уменьшить результирующий архив.

Исходный код проекта выложен на сервисе *github.com* в сети Интернет и доступен по ссылке <https://github.com/KVSrep/diplom>. В файле *Readme.md* содержится описание возможностей программы и пример использования.

ЗАКЛЮЧЕНИЕ

В рамках данной работы был описан и реализован алгоритм архивации JPEG файлов в виде нескольких программ, объединённых скриптами. Эти программы могут быть использованы отдельно для анализа внутренних структур файлов и генерации своих кодов Хаффмана, которыми впоследствии можно будет закодировать JPEG файлы. Также есть возможность собрать достаточно большую коллекцию файлов JPEG, чтобы сгенерировать код, которым впоследствии можно будет архивировать поток файлов.

Данный алгоритм очень сильно зависит от частот категорий в сжимаемых файлах, однако при использовании серийной съёмки фотографий или в студиях фотографов, где получаются визуально схожие изображения будут генерироваться и схожие частоты, что улучшит сжимающие возможности алгоритма.

Следует отметить, что в некоторой фототехнике для оптимизации скорости получения изображения уже существуют заранее сгенерированные коды Хаффмана, что позволяет получить существенный прирост при архивации таких изображений, так как эти коды не всегда оптимальны для частных случаев, а программа делает оптимальный код именно для набора.

Также в работе проведено сравнение алгоритма с популярным архиватором WinRAR и было показано, что сжатие JPEG информации без анализа структуры достаточно трудоёмкий процесс, и в некоторых случаях алгоритм *rar* генерирует архив большего размера, чем исходный набор.

Показана возможность использования алгоритма вместе с другими алгоритмами архивации, так например архивация zip файлов, предварительно архивированных *compress.py* ещё больше уменьшает размер файла из-за изменённой структуры кодов.

Результат работы показывает, что на основе подобного подхода возможно создавать более качественные алгоритмы архивации, основываясь на структуре внутреннего формата самого файла. Например, можно изменить способ кодирования или добавить новых эвристик в уже имеющийся алгоритм описанный в JPEG, и получить ещё более компактное представления квантованных матриц.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1) Сергеева Юлия. Вся статистика интернета и соцсетей на 2021 год – цифры и тренды в мире и в России [Электронный ресурс]
Режим доступа: <https://www.web-canape.ru/business/vsya-statistika-interneta-i-socsetej-na-2021-god-cifry-i-trendy-v-mire-i-v-rossii>
[дата обращения: 02.02.2021]
- 2) Иван Черноусов. Удаленная работа загрузила Сеть [Электронный ресурс] // Российская газета - Федеральный выпуск № 61(8115)
Режим доступа: <https://rg.ru/2020/03/22/evropejskie-vlasti-poprosili-videoservisy-ogranichit-kachestvo-transliacij.html>
[дата обращения: 02.12.2020]
- 3) Федеральный закон № 374-ФЗ от 06 июля 2016 г.: принят Государственной Думой 24 июня 2016 г. // Собрание законодательства Российской Федерации 2016 – №28 – Ст. 4558
- 4) Problem 5. “JPEG Encoding” [Электронный ресурс] // NSUCRYPTO 2020
Режим доступа: https://nsucrypto.nsu.ru/media/Olympiads/2020/Round_2/Tasks/20-problems-second-round-5-advtg.pdf
[дата обращения: 20.11.2020]
- 5) Eric Hamilton. JPEG File Interchange Format [Электронный ресурс]// C-Cube Microsystems
Режим доступа: <https://www.w3.org/Graphics/JPEG/jfif3.pdf>
[дата обращения: 25.11.2020]
- 6) Cristi Cuturicu. CRYX's note about the JPEG decoding algorithm [Электронный ресурс] // © 1999 Cristi Cuturicu
Режим доступа: <https://www.opennet.ru/docs/formats/jpeg.txt>
[дата обращения: 20.11.2020]

- 7) TERMINAL EQUIPMENT AND PROTOCOLS FOR TELEMATIC SERVICES [Электронный ресурс] // © CCITT Rec. T.81 (1992 E)
Режим доступа: <https://www.w3.org/Graphics/JPEG/itu-t81.pdf>
[дата обращения: 02.12.2020]
- 8) Lawrence L. Larmore and Daniel S. Hirschberg A Fast Algorithm for Optimal Length-Limited Huffman Codes [Электронный ресурс] // Dan Hirschberg - Computer Science Department University of California
Режим доступа: <https://www.ics.uci.edu/~dan/pubs/LenLimHuff.pdf>
[дата обращения: 20.12.2020]
- 9) WinRAR Documentation [Электронный ресурс]
Режим доступа: <https://documentation.help/WinRAR/>
[дата обращения: 20.12.2020]
- 10) Huffman D.A., A Method for the construction of minimum redundancy codes // Proc. Inst. Radio Engineers 40 (1952) pp. 1098-1101.
- 11) Yasoob Khalid, Understanding and Decoding a JPEG Image using Python [Электронный ресурс]
Режим доступа: <https://yasoob.me/posts/understanding-and-writing-jpeg-decoder-in-python/>
[дата обращения: 22.11.2020]
- 12) G. K. Wallace, The JPEG still picture compression standard // IEEE Trans. Consumer Electronics, vol. 38, no. 1, pp. xviii-xxxiv, Feb. 1992.
- 13) Белов А. С., Апухтин Р. В., Романов С. Г. Методы сжатия сигналов телевизионного изображения с внутрикадровым кодированием [Электронный ресурс]
Режим доступа: <https://docplayer.ru/26275623-Diskretno-kosinusnoe-preobrazovanie-i-szhatie-izobrazheniy.html>
[дата обращения: 18.12.2020]