

ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ _____
ПРЕПОДАВАТЕЛЬ

старший преподаватель		Т.А. Суетина
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

РАБОТА С ОБЪЕКТАМИ В ФОРМАТЕ BMP

по курсу: Техника аудиовизуальных средств информации

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	4329		Д.С. Шаповалова
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

1. Цель работы:

Получить теоретические знания по методам и алгоритмам сжатия текстовой информации и практические навыки реализации этих алгоритмов и методов.

2. Задание:

Для изображения в формате BMP с глубиной цвета 24 или 32 без сжатия выполнить:

1. Считать заголовок файла.
2. Описать все используемые структуры, переменные и методы класса.
3. Продемонстрировать результаты изменения глубины цвета.
4. Привести листинг кода программы с комментариями.

Выбранный вариант задания – 5: Исходная глубина – 32, Новая глубина – 4.

Исходное изображение представлено на рисунке 1:



Рисунок 1.1 – Исходное изображение

3. Ход работы:

Для начала опишем используемые структуры, переменные и методы класса:

Структуры

BITMAPFILEHEADER

Заголовок BMP-файла (14 байт).

Переменные:

Type — сигнатура "BM"

Size — размер файла

Reserved1 / Reserved2 — не используются

OffsetBits — смещение начала пиксельных данных

Методы:

from_bytes(data) — создать структуру из 14 байт

to_bytes() — собрать 14 байт для записи в файл

BITMAPINFOHEADER

Информация об изображении (40 байт).

Переменные:

Size — размер структуры (40)

Width, Height

Planes — всегда 1

BitCount — глубина цвета

Compression — тип сжатия (0 = без сжатия)

SizeImage — размер пиксельных данных

XPelsPerMeter, YPelsPerMeter — разрешение

ColorUsed — кол-во цветов палитры

ColorImportant — важные цвета

Методы:

from_bytes(data) — создать из 40 байт

to_bytes() — собрать обратно

RGBTRIPLE

Один пиксель/элемент палитры.

Переменные:

Blue, Green, Red

Reserved — для палитры BMP (4-й байт)

Методы:

from_bytes(data) — 3 или 4 байта BGR/BGRX

to_bytes(include_reserved) — обратно в BGR или BGRX

Класс BMPImage

Переменные объекта:

file_header — BITMAPFILEHEADER

info_header — BITMAPINFOHEADER

palette — список RGBTRIPLE

pixels — двумерный массив пикселей (RGBTRIPLE)

Методы класса BMPImage

load_image(filename)

Загружает BMP:

1. читает два заголовка
2. проверяет формат
3. вычисляет размеры строки
4. считывает пиксели (учитывая выравнивание и ориентацию)
5. change_color_depth(new_depth)
6. Создаёт новое изображение с глубиной 1, 4 или 8 бит:
7. копирует заголовки
8. создаёт палитру
9. пересчитывает R,G,B → grayscale
10. находит индекс палитры
11. упаковывает пиксели
12. пересчитывает размеры

_create_palette(bit_depth)

Создаёт палитру:

2 цвета (1 бит)

16 серых (4 бит)

256 серых (8 бит)

_rgb_to_gray(r, g, b)

Считает яркость по формуле:

$$0.299 R + 0.597 G + 0.114 B$$

_find_palette_index(gray, bit_depth)

Возвращает номер цвета в палитре для 1/4/8 бит.

_pack_pixels(bit_depth, width, height, top_down)

Упаковывает индексы пикселей в байты:

Для 4 бит: два пикселя в один байт

Добавляет выравнивание строк

`write_image(filename)`

Записывает:

1. BITMAPFILEHEADER
2. BITMAPINFOHEADER
3. палитру
4. упакованные данные пикселей

1. Загрузка файлов и чтение заголовков:

Процесс чтения BMP файла осуществляется в несколько этапов:

Чтение заголовков: программа последовательно считывает 14 байтов для BITMAPFILEHEADER и 40 байтов для BITMAPINFOHEADER.

Парсинг структур: данные заголовков преобразуются из бинарного формата в структурированное представление с использованием модуля struct, учитывая направление хранения строк (снизу вверх) – порядок байтов, характерный для формата BMP.

Позиционирование в потоке: с помощью значения OffsetBits из BITMAPFILEHEADER осуществляется переход к началу растровых данных.

Чтение пикселей: для 32-битных изображений считывается по 4 байта на пиксель (Blue, Green, Red, Alpha), при этом учитывается выравнивание строк по 4-х байтной границе, как того требует спецификация формата.

Информация о считанных заголовках изображения представлена на рисунке 2.1:

```
--- Чтение BITMAPFILEHEADER (14 байт) ---
Парсинг BITMAPFILEHEADER:
Сигнатура: b'BM' (корректная)
Размер файла: 2160056 байт
Зарезервированные поля: 0, 0
Смещение до данных: 54 байт

--- Чтение BITMAPINFOHEADER (40 байт) ---
Парсинг BITMAPINFOHEADER:
Размер структуры: 40 байт
Размеры: 900 x 600
Плоскости: 1
Битов на пиксель: 32
Сжатие: 0 (BI_RGB)
Размер изображения: 2160002 байт
Используемых цветов: 0

--- ИНФОРМАЦИЯ ОБ ИЗОБРАЖЕНИИ ---
Размер изображения: 900 x 600 пикселей
Глубина цвета: 32 бит
Размер файла: 2160056 байт
Смещение до данных: 54 байт
Сжатие: нет
```

Рисунок 2.1 – Чтение заголовков изображения

2. Преобразование цветов в градации серого

Применяется формула:

$$\text{Gray} = 0.299 \cdot R + 0.597 \cdot G + 0.114 \cdot B, \quad (1)$$

Учитывается разная чувствительность человеческого глаза к цветам – зелёный виден больше, красный – средне, а синий – наименее. Каждый пиксель преобразуется в одно

значение яркости, описывающее его серый тон. Таким образом, трёхканальное RGB изображение сводится к одной компоненте яркости.

На рисунках 2.1 – 2.2 представлен алгоритм преобразования и пример расчёта некоторых пикселей:

```

--- ПРЕОБРАЗОВАНИЕ ПИКСЕЛЕЙ ---
Алгоритм преобразования:
1. RGB → Grayscale:  $Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$ 
2. Квантование до 16 уровней
3. Упаковка в 4-битные индексы

```

Рисунок 2.1 – Алгоритм преобразования

Примеры преобразования (5 случайных пикселей):

Координаты	R	G	B	→ Gray	→ Индекс	Формула
(466,350)	2	37	45	→ 27.82	→ 2	$\text{round}(27.8/17) = 2$
(640,142)	11	127	136	→ 94.61	→ 6	$\text{round}(94.6/17) = 6$
(890,240)	249	234	215	→ 238.66	→ 15	$\text{gray} \geq 238 \rightarrow 15$
(302, 77)	249	233	213	→ 237.83	→ 14	$\text{round}(237.8/17) = 14$
(96,535)	250	235	216	→ 239.67	→ 15	$\text{gray} \geq 238 \rightarrow 15$

Рисунок 2.2 – Пример преобразования некоторых пикселей

3. Создание палитры 4-х битного изображения

В BMP глубина цвета 4 бита означает, что каждый пиксель кодируется одним из 16 возможных индексов, соответствующих определённым цветам в палитре: 16 оттенков серого – от чёрного (0,0,0) до белого (255,255,255), с шагом яркости 17 единиц (0x11).

Каждый цвет записывается в структуре(классе) RGBTRIPLE, содержащей три байта: компоненты Blue, Green, Red. В результате формируется таблица из 16 записей, которая будет добавлена в BMP-файл перед областью пикселей.

На рисунке 3 представлена палитра из 16 градаций серого:

```

--- СОЗДАНИЕ ПАЛИТРЫ ---
ColorUsed: 0 → 16
Палитра (16 цветов):

```

Индекс	R	G	B	HEX	Уровень серого
0	0	0	0	0x00	0
1	17	17	17	0x11	17
2	34	34	34	0x22	34
3	51	51	51	0x33	51
4	68	68	68	0x44	68
5	85	85	85	0x55	85
6	102	102	102	0x66	102
7	119	119	119	0x77	119
8	136	136	136	0x88	136
9	153	153	153	0x99	153
10	170	170	170	0xAA	170
11	187	187	187	0xBB	187
12	204	204	204	0xCC	204
13	221	221	221	0xDD	221
14	238	238	238	0xEE	238
15	255	255	255	0xFF	255

Рисунок 3 – Палитра из 16 градаций серого

4. Квантование и определение индексов палитры

Для каждого исходного пикселя определяется его новый цвет в палитре, ближайший по значению серости к рассчитанной серости исходного пикселя. Эта операция аналогична квантованию: значения яркости (от 0 до 255) округляются до ближайшего уровня из 16 возможных. Если значение яркости меньше 238 индекс вычисляется делением на 17 и округлением, соблюдая диапазон от 0 до 14. Если больше, то берётся максимальный индекс = 15 (самый яркий уровень).

На рисунке 4 представлен пример результатов квантования для некоторых пикселей:

РЕЗУЛЬТАТЫ КВАНТОВАНИЯ (образцы):		
Координаты	Gray → Индекс	Алгоритм
----- ----- -----		
(9,484)	240.27 → 15	gray >= 238 → idx = 15
(199,583)	239.07 → 15	gray >= 238 → idx = 15
(252,102)	236.34 → 14	round(236.3 / 17) = 14
(579,353)	31.97 → 2	round(32.0 / 17) = 2
(168, 3)	238.96 → 15	gray >= 238 → idx = 15
(869, 41)	240.27 → 15	gray >= 238 → idx = 15
(680,168)	78.56 → 5	round(78.6 / 17) = 5
(874,198)	238.66 → 15	gray >= 238 → idx = 15

Рисунок 4 – Результаты квантования

5. Упаковка данных

После этапов преобразования цветов производится упаковка индексов палитры в байты. В нашем случае, 4 бита – это половина байта, значит в одном байте хранятся сразу два пикселя. Старшие 4 бита содержат индекс первого пикселя, младшие – второго. В случае нечётного количества пикселей последний байт дополняется нулями.

Затем каждая строка изображения выравнивается по границе в 4 байта, в соответствии со спецификацией BMP => в конце строки может добавляться от 0 до 3 дополнительных байт заполнения.

Все байты формируются в единый поток, который будет записан в выходной файл.

На рисунке 5 представлены первые 48 байт выходного файла:

```
УПАКОВАННЫЕ ДАННЫЕ (_prepared_pixel_bytes):
Общий размер: 271200 байт
Первые 48 байт:
0000: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0010: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0020: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

Рисунок 5 – Упакованные данные

6. Обновление заголовков

Далее создаются новые заголовки файла и изображения. В них пересчитываются параметры: размер изображения (SizeImage), смещение до пикселей (OffsetBits, зависящее от длины палитры), общий размер файла (Size).

На рисунке 6 представлены новые заголовки для нашего файла:

```
=====
BITMAPFILEHEADER:
  Type: 0x4D42
  Size: 271318 байт
  Reserved1: 0
  Reserved2: 0
  OffsetBits: 118 байт

BITMAPINFOHEADER:
  Size: 40 байт
  Width: 900
  Height: 600
  Planes: 1
  BitCount: 4
  Compression: 0
  SizeImage: 271200
  XPelsPerMeter: 0
  YPelsPerMeter: 0
  ColorUsed: 16
  ColorImportant: 0
```

Рисунок 6 – Новые заголовки

В конечном итоге формируется BMP файл из: новых заголовков, палитры 16 оттенков серого, упакованных данных пикселей.

Результат представлен на рисунке 7:



Рисунок 7 – Результат преобразования глубины цвета 32 бит в 4 бита.

На рисунке 8 продемонстрированы итоги конвертации:

```
ИТОГИ КОНВЕРТАЦИИ
=====
СРАВНЕНИЕ РАЗМЕРОВ:
Исходный файл: 2160056 байт (2109.4 КБ)
Результирующий файл: 271318 байт (265.0 КБ)
Коэффициент сжатия: 7.96x
Уменьшение размера: 87.4%

ОБЪЯСНЕНИЕ РЕЗУЛЬТАТА:
- 32-bit: 4 байта на пиксель → 1000×1000 = 4,000,000 байт данных
- 4-bit: 0.5 байта на пиксель → 1000×1000 = 500,000 байт данных
- Палитра: 16 цветов × 4 байта = 64 байта
- Заголовки: 14 + 40 = 54 байта
- Теоретическое сжатие: 8x, фактическое: 7.96x
```

Рисунок 8 – Итоги конвертации

По итогам можем наблюдать уменьшение объема файла в примерно 8 раз, что объясняется затратами памяти на сохранение 1 пикселя. При глубине цвета 32 бит затрачивается 4 байта на пиксель, при глубине цвета в 4 бита затрачивается 0,5 байта на пиксель.

4. Вывод:

В ходе выполнения лабораторной работы были изучены внутренняя структура и особенности формата BMP, включая организацию заголовков, механизм хранения палитровых изображений и требования к выравниванию строк. Практически реализован алгоритм понижения глубины цвета с 32 до 4 бит с использованием палитры из 16 градаций серого.

Преобразование выполнено в соответствии с методическими указаниями: применена стандартная формула расчёта яркости, сформирована корректная палитра, реализована упаковка 4-битных индексов и учтены все особенности формата (смещения, выравнивание, размеры). Анализ показал, что объём файла уменьшается в среднем в 6–8 раз при сохранении приемлемого визуального качества за счёт достаточного количества градаций серого.

Таким образом, цель работы достигнута: получены как теоретические знания, так и практические навыки работы с растровыми данными на уровне байтовой структуры.

ПРИЛОЖЕНИЕ А

Листинг Программы

```
import struct
import os
import math
from dataclasses import dataclass
from typing import List, Tuple
import random

# Структуры BMP заголовков

@dataclass
class BITMAPFILEHEADER:
    Type: int = 0x4D42 # 'BM'
    Size: int = 0
    Reserved1: int = 0
    Reserved2: int = 0
    OffsetBits: int = 54

    @classmethod
    def from_bytes(cls, data: bytes) -> "BITMAPFILEHEADER":
        t, size, r1, r2, off = struct.unpack("<HHHHI", data)
        return cls(Type=t, Size=size, Reserved1=r1, Reserved2=r2,
OffsetBits=off)

    def to_bytes(self) -> bytes:
        return struct.pack("<HHHHI", self.Type, self.Size, self.Reserved1,
self.Reserved2, self.OffsetBits)

@dataclass
class BITMAPINFOHEADER:
    Size: int = 40
    Width: int = 0
    Height: int = 0
    Planes: int = 1
    BitCount: int = 0
    Compression: int = 0
    SizeImage: int = 0
    XPelsPerMeter: int = 0
    YPelsPerMeter: int = 0
    ColorUsed: int = 0
    ColorImportant: int = 0

    @classmethod
    def from_bytes(cls, data: bytes) -> "BITMAPINFOHEADER":
        vals = struct.unpack("<IiiHHIiIII", data)
        return cls(*vals)

    def to_bytes(self) -> bytes:
        return struct.pack("<IiiHHIiIII", self.Size, self.Width,
self.Height,
                                self.Planes, self.BitCount, self.Compression,
self.SizeImage,
                                self.XPelsPerMeter, self.YPelsPerMeter,
self.ColorUsed, self.ColorImportant)

@dataclass
class RGBTRIPLE:
    """
```

```

Структура для представления RGB пикселя
Используется как для хранения цветowych данных пикселей изображения (3
байта),
так и для элементов палитры (4 байта с Reserved полем)
"""
Blue: int = 0
Green: int = 0
Red: int = 0
Reserved: int = 0 # Дополнительное поле для совместимости с палитрой BMP

@classmethod
def from_bytes(cls, data: bytes) -> "RGBTRIPLE":
    """Создает RGBTRIPLE из данных (BGR или BGRX порядок) """
    if len(data) == 3:
        b, g, r = struct.unpack("<BBB", data)
        return cls(Blue=b, Green=g, Red=r, Reserved=0)
    elif len(data) == 4:
        b, g, r, reserved = struct.unpack("<BBBB", data)
        return cls(Blue=b, Green=g, Red=r, Reserved=reserved)
    else:
        raise ValueError("Данные должны быть 3 или 4 байта")

def to_bytes(self, include_reserved: bool = True) -> bytes:
    """Конвертирует RGBTRIPLE в байты данных (BGR или BGRX порядок) """
    if include_reserved:
        return struct.pack("<BBBB", self.Blue, self.Green, self.Red,
self.Reserved)
    else:
        return struct.pack("<BBB", self.Blue, self.Green, self.Red)

# Класс для работы с BMP изображениями
class BMPImage:
    def __init__(self) -> None:
        self.file_header = BITMAPFILEHEADER()
        self.info_header = BITMAPINFOHEADER()
        self.palette: List[RGBTRIPLE] = []
        self.pixels: List[List[RGBTRIPLE]] = []

    def load_image(self, filename: str) -> None:
        """Загрузка BMP файла с подробным выводом"""
        print("=" * 60)
        print("ШАГ 1: ЗАГРУЗКА ИСХОДНОГО ФАЙЛА")
        print("=" * 60)

        with open(filename, "rb") as f:
            # Чтение BITMAPFILEHEADER
            print("\n--- ЧТЕНИЕ BITMAPFILEHEADER (14 байт) ---")
            fh_data = f.read(14)
            if len(fh_data) != 14:
                raise ValueError("Файл слишком короткий для
BITMAPFILEHEADER")
            self.file_header = BITMAPFILEHEADER.from_bytes(fh_data)
            self._print_file_header("ИСХОДНЫЙ BITMAPFILEHEADER")

            # Чтение BITMAPINFOHEADER
            print("\n--- ЧТЕНИЕ BITMAPINFOHEADER (40 байт) ---")
            ih_data = f.read(40)
            if len(ih_data) != 40:
                raise ValueError("Файл слишком короткий для
BITMAPINFOHEADER")
            self.info_header = BITMAPINFOHEADER.from_bytes(ih_data)
            self._print_info_header("ИСХОДНЫЙ BITMAPINFOHEADER")

```

```

# Проверки
if self.file_header.Type != 0x4D42:
    raise ValueError("Не BMP файл")
if self.info_header.Compression != 0:
    raise ValueError("Поддерживаются только несжатые BMP")
if self.info_header.BitCount not in (24, 32):
    raise ValueError("Поддерживаются только 24-bit и 32-bit BMP")

width = self.info_header.Width
height = self.info_header.Height
top_down = False
if height < 0:
    top_down = True
    height = -height

bytes_per_pixel = self.info_header.BitCount // 8
row_size = ((width * bytes_per_pixel * 8 + 31) // 32) * 4

print(f"\n--- ПАРАМЕТРЫ ИЗОБРАЖЕНИЯ ---")
print(f"Размер: {width} × {height} пикселей")
print(f"Бит на пиксель: {self.info_header.BitCount}")
print(f"Байт на пиксель: {bytes_per_pixel}")
print(f"Размер строки с выравниванием: {row_size} байт")
print(f"Ориентация: {'сверху-вниз' if top_down else 'снизу-вверх'}")

# Переход к данным изображения
print(f"\nПереход к позиции {self.file_header.OffsetBits} (начало растровых данных)")
f.seek(self.file_header.OffsetBits)

# Чтение пикселей
print("\n--- ЧТЕНИЕ ПИКСЕЛЬНЫХ ДАННЫХ ---")
self.pixels = [[RGBTRIPLE() for _ in range(width)] for __ in range(height)]

total_pixels = width * height
pixels_read = 0

for row_idx in range(height):
    if row_idx % 50 == 0 or row_idx == height - 1:
        progress = (row_idx * width) / total_pixels * 100
        print(f"    Чтение строки {row_idx + 1}/{height} ({progress:.1f}%)...")

    row = f.read(row_size)
    if len(row) < row_size:
        raise ValueError("Файл поврежден")

    y = row_idx if top_down else (height - 1 - row_idx)

    for x in range(width):
        off = x * bytes_per_pixel
        b = row[off]
        g = row[off + 1]
        r = row[off + 2]
        self.pixels[y][x] = RGBTRIPLE(Red=r, Green=g, Blue=b)
        pixels_read += 1

print(f"Прочитано пикселей: {pixels_read}")

def change_color_depth(self, new_depth: int) -> "BMPImage":

```

```

    """Изменение глубины цвета с подробным выводом"""
    print("\n" + "=" * 60)
    print(f"ШАГ 2: КОНВЕРТАЦИЯ {self.info_header.BitCount} → {new_depth}
БИТ")
    print("=" * 60)

    if new_depth not in (1, 4, 8):
        raise NotImplementedError(f"Поддерживаются глубины: 1, 4, 8 бит")

    out = BMPImage()

    # Копирование и модификация заголовков
    print("\n--- МОДИФИКАЦИЯ ЗАГОЛОВКОВ ---")
    out.info_header.Width = self.info_header.Width
    out.info_header.Height = self.info_header.Height
    out.info_header.Planes = 1
    out.info_header.BitCount = new_depth
    out.info_header.Compression = 0

    width = self.info_header.Width
    height = self.info_header.Height
    top_down = height < 0
    if top_down:
        height = -height

    print(f"    BitCount: {self.info_header.BitCount} → {new_depth}")

    # Создание палитры
    print("\n--- СОЗДАНИЕ ПАЛИТРЫ ---")
    out.palette = self._create_palette(new_depth)
    out.info_header.ColorUsed = len(out.palette)
    print(f"    ColorUsed: 0 → {len(out.palette)}")
    self._print_palette(out.palette, new_depth)

    # Преобразование пикселей
    print("\n--- ПРЕОБРАЗОВАНИЕ ПИКСЕЛЕЙ ---")
    print("Алгоритм преобразования:")
    print("    1. RGB → Grayscale:  $Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$ ")
    print("    2. Квантование до 16 уровней")
    print("    3. Упаковка в 4-битные индексы")

    # Показываем примеры преобразования
    self._show_conversion_examples(new_depth, 5)

    pixel_bytes = self._pack_pixels(new_depth, width, height, top_down)

    # Расчет новых размеров
    print("\n--- РАСЧЕТ НОВЫХ РАЗМЕРОВ ---")
    out.info_header.SizeImage = len(pixel_bytes)
    palette_size = len(out.palette) * 4
    out.file_header.OffsetBits = 14 + out.info_header.Size + palette_size
    out.file_header.Size = out.file_header.OffsetBits +
out.info_header.SizeImage

    print(f"    SizeImage: {self.info_header.SizeImage} →
{out.info_header.SizeImage}")
    print(f"    OffsetBits: {self.file_header.OffsetBits} →
{out.file_header.OffsetBits}")
    print(f"    File Size: {self.file_header.Size} →
{out.file_header.Size}")

    out._prepared_pixel_bytes = pixel_bytes

```

```

# Вывод новых заголовков
print("\n" + "=" * 60)
print("ИЗМЕНЕННЫЕ ЗАГОЛОВКИ")
print("=" * 60)
out._print_file_header("ОБНОВЛЕННЫЙ BITMAPFILEHEADER")
out._print_info_header("ОБНОВЛЕННЫЙ BITMAPINFOHEADER")

return out

def _create_palette(self, bit_depth: int) -> List[RGBTRIPLE]:
    palette = []

    if bit_depth == 1:
        palette.append(RGBTRIPLE(Blue=0, Green=0, Red=0, Reserved=0))
        palette.append(RGBTRIPLE(Blue=255, Green=255, Red=255,
Reserved=0))

    elif bit_depth == 4:
        for i in range(16):
            if i == 15:
                gray = 0xFF
            else:
                gray = i * 0x11
            palette.append(RGBTRIPLE(Blue=gray, Green=gray, Red=gray,
Reserved=0))

    elif bit_depth == 8:
        for i in range(256):
            palette.append(RGBTRIPLE(Blue=i, Green=i, Red=i, Reserved=0))

    return palette

def _rgb_to_gray(self, r: int, g: int, b: int) -> float:
    # Формула из методички: R8 = G8 = B8 = 0.299*R24 + 0.597*G24 +
0.114*B24
    return 0.299 * r + 0.597 * g + 0.114 * b

def _find_palette_index(self, gray: float, bit_depth: int) -> int:
    if bit_depth == 1:
        return 0 if gray < 128 else 1

    elif bit_depth == 4:
        if gray >= 238:
            return 15
        else:
            idx = int(round(gray / 17))
            return max(0, min(14, idx))

    elif bit_depth == 8:
        return max(0, min(255, int(round(gray))))

    else:
        raise ValueError(f"Неподдерживаемая глубина: {bit_depth}")

def _pack_pixels(self, bit_depth: int, width: int, height: int, top_down:
bool) -> bytearray:
    if bit_depth == 4:
        bytes_per_row = (width + 1) // 2
    else:
        bytes_per_row = width // (8 // bit_depth)

    padded_row_size = ((bytes_per_row + 3) // 4) * 4
    pixel_bytes = bytearray()

```

```

print(f"\n--- УПАКОВКА ПИКСЕЛЕЙ ---")
print(f"Байт на строку (без выравнивания): {bytes_per_row}")
print(f"Размер строки с выравниванием: {padded_row_size} байт")

for row_idx in range(height):
    if row_idx % 50 == 0 or row_idx == height - 1:
        progress = (row_idx * width) / (width * height) * 100
        print(f"    Обработка строки {row_idx + 1}/{height}
({progress:.1f}%)...")

    y = row_idx if top_down else (height - 1 - row_idx)

    indices = []
    for x in range(width):
        pixel = self.pixels[y][x]
        gray = self._rgb_to_gray(pixel.Red, pixel.Green, pixel.Blue)
        idx = self._find_palette_index(gray, bit_depth)
        indices.append(idx)

    row_bytes = bytearray()

    if bit_depth == 4:
        for i in range(0, width, 2):
            hi = indices[i]
            lo = indices[i + 1] if i + 1 < width else 0
            byte = ((hi & 0xF) << 4) | (lo & 0xF)
            row_bytes.append(byte)

    padding = padded_row_size - len(row_bytes)
    if padding > 0:
        row_bytes.extend(b"\x00" * padding)

    pixel_bytes.extend(row_bytes)

print(f"Общий размер упакованных данных: {len(pixel_bytes)} байт")
return pixel_bytes

def write_image(self, filename: str) -> None:
    """Сохранение BMP файла"""
    print(f"\n--- СОХРАНЕНИЕ РЕЗУЛЬТАТА ---")
    print(f"Запись в файл: {filename}")

    with open(filename, "wb") as f:
        f.write(self.file_header.to_bytes())
        f.write(self.info_header.to_bytes())

        for c in self.palette:
            f.write(c.to_bytes())

        if hasattr(self, "_prepared_pixel_bytes"):
            f.write(self._prepared_pixel_bytes)
            print(f"Записано байт данных:
{len(self._prepared_pixel_bytes)}")
        else:
            raise RuntimeError("Данные не подготовлены для записи")

def _print_file_header(self, title: str):
    """Вывод информации о BITMAPFILEHEADER"""
    print(f"\n{title}:")
    print(f"    Type: 0x{self.file_header.Type:04X} ({'BM' if
self.file_header.Type == 0x4D42 else 'неизвестно'})")
    print(f"    Size: {self.file_header.Size} байт ({self.file_header.Size

```

```

/ 1024:.1f} KB)")
    print(f"    Reserved1: {self.file_header.Reserved1}")
    print(f"    Reserved2: {self.file_header.Reserved2}")
    print(f"    OffsetBits: {self.file_header.OffsetBits} байт")

    def _print_info_header(self, title: str):
        """Вывод информации о BITMAPINFOHEADER"""
        print(f"\n{title}:")
        print(f"    Size: {self.info_header.Size} байт")
        print(f"    Width: {self.info_header.Width} пикселей")
        print(f"    Height: {self.info_header.Height} пикселей")
        print(f"    Planes: {self.info_header.Planes}")
        print(f"    BitCount: {self.info_header.BitCount} бит на пиксель")
        print(f"    Compression: {self.info_header.Compression}")
        print(f"    SizeImage: {self.info_header.SizeImage} байт")
        print(f"    XPelsPerMeter: {self.info_header.XPelsPerMeter}")
        print(f"    YPelsPerMeter: {self.info_header.YPelsPerMeter}")
        print(f"    ColorUsed: {self.info_header.ColorUsed}")
        print(f"    ColorImportant: {self.info_header.ColorImportant}")

    def _print_palette(self, palette: List[RGBTRIPLE], bit_depth: int):
        print(f"Палитра ({len(palette)} цветов):")
        if bit_depth == 4:
            print("    Индекс | R    G    B    | HEX      | Уровень серого")
            print("    -----|-----|-----|-----")
            for i, color in enumerate(palette):
                gray_level = i * 17
                print(
                    f"        {i:2d} | {color.Red:3d} {color.Green:3d} {color.Blue:3d} | 0x{color.Red:02X} | {gray_level:3d}"
                )

    def _show_conversion_examples(self, bit_depth: int, sample_size: int = 5):
        print(f"\nПримеры преобразования ({sample_size} случайных пикселей):")
        print("    Координаты | R    G    B    → Gray      → Индекс | Формула")
        print("    -----|-----|-----|-----")

        width = self.info_header.Width
        height = abs(self.info_header.Height)

        for i in range(min(sample_size, width * height)):
            x = random.randint(0, width - 1)
            y = random.randint(0, height - 1)
            pixel = self.pixels[y][x]
            r, g, b = pixel.Red, pixel.Green, pixel.Blue
            gray = self._rgb_to_gray(r, g, b)
            idx = self._find_palette_index(gray, bit_depth)

            if bit_depth == 4:
                formula = f"round({gray:.1f}/17) = {idx}" if gray < 238 else "gray≥238 → 15"
            else:
                formula = "N/A"

            print(f"    ({x:3d},{y:3d}) | {r:3d} {g:3d} {b:3d} → {gray:6.2f} → {idx:2d} | {formula}")

    def analyze_file_structure(filename: str):
        print(f"\n{'=' * 50}")
        print(f"АНАЛИЗ СТРУКТУРЫ ФАЙЛА: {os.path.basename(filename)}")

```

```

print(f"{'=' * 50}")

file_size = os.path.getsize(filename)
print(f"Общий размер файла: {file_size} байт ({file_size / 1024:.1f}
КБ)")

with open(filename, 'rb') as f:
    # BITMAPFILEHEADER
    fh_data = f.read(14)
    fh = BITMAPFILEHEADER.from_bytes(fh_data)

    # BITMAPINFOHEADER
    ih_data = f.read(40)
    ih = BITMAPINFOHEADER.from_bytes(ih_data)

    # Вычисления размеров
    print(f"\n=== ВЫЧИСЛЕНИЯ ===")
    width = ih.Width
    height = abs(ih.Height)

    if ih.BitCount == 32:
        bytes_per_pixel = 4
        row_size = ((width * bytes_per_pixel + 3) // 4) * 4
        theoretical_data_size = row_size * height
        theoretical_total_size = 14 + 40 + theoretical_data_size

    elif ih.BitCount == 4:
        bytes_per_row = (width + 1) // 2
        padded_row_size = ((bytes_per_row + 3) // 4) * 4
        theoretical_data_size = padded_row_size * height
        palette_size = ih.ColorUsed * 4
        theoretical_total_size = 14 + 40 + palette_size +
theoretical_data_size

        print(f"Байт на строку (без выравнивания): {bytes_per_row}")
        print(f"Размер строки с выравниванием: {padded_row_size} байт")
        print(f"Размер палитры: {palette_size} байт")

    print(f"Теоретический размер данных изображения:
{theoretical_data_size} байт")
    print(f"Теоретический общий размер файла: {theoretical_total_size}
байт")

    if file_size == theoretical_total_size:
        print("Размеры файла соответствуют расчетам")
    else:
        print(f"Разница: {file_size - theoretical_total_size} байт")

def main():

    # Укажи путь к своему 32-битному BMP файлу
    input_file = "input_32bit.bmp"
    output_file = "output_4bit.bmp"

    if not os.path.exists(input_file):
        print(f"Файл {input_file} не найден")
        return

    try:
        # Загрузка исходного изображения
        original_image = BMPImage()
        original_image.load_image(input_file)

```

```

# Анализ исходной структуры
analyze_file_structure(input_file)

# Конвертация в 4 бита
converted_image = original_image.change_color_depth(4)

# Сохранение результата
converted_image.write_image(output_file)

# Анализ результата
analyze_file_structure(output_file)

# Сравнение размеров
original_size = os.path.getsize(input_file)
result_size = os.path.getsize(output_file)
compression_ratio = original_size / result_size
size_reduction = (1 - result_size / original_size) * 100

print(f"\n{'=' * 60}")
print("ИТОГИ КОНВЕРТАЦИИ")
print(f"{'=' * 60}")
print("СРАВНЕНИЕ РАЗМЕРОВ:")
print(f"Исходный файл: {original_size} байт ({original_size / 1024:.1f} КБ)")
print(f"Результирующий файл: {result_size} байт ({result_size / 1024:.1f} КБ)")
print(f"Коэффициент сжатия: {compression_ratio:.2f}x")
print(f"Уменьшение размера: {size_reduction:.1f}%")

print(f"\nОБЪЯСНЕНИЕ РЕЗУЛЬТАТА:")
print("- 32-bit: 4 байта на пиксель → 1000×1000 = 4,000,000 байт данных")
print("- 4-bit: 0.5 байта на пиксель → 1000×1000 = 500,000 байт данных")
print("- Палитра: 16 цветов × 4 байта = 64 байта")
print("- Заголовки: 14 + 40 = 54 байта")
print(f"- Теоретическое уменьшение объема: 8x, фактическое: {compression_ratio:.2f}x")

except Exception as e:
    print(f"\n ошибка: {e}")
    import traceback
    traceback.print_exc()

if __name__ == "__main__":
    main()

```