

JPEG Decoder

Кодирование JPEG (step=1)

Для реализации декодера нужно сначала понять, как устроено кодирование

- На вход подаётся изображение в R'G'B' (RGB с гамма коррекцией)
- R'G'B' преобразуется в Y'C_bC_r
 - Y' отвечает за яркость
 - C_b и C_r -- за хроматическое сэмплирование
- Разбиваем картинку на блоки 8x8
- Если не делится нацело -- дублируем последние столбец/строку

Кодирование JPEG (step=2)

Chroma downsampling

Хроматические данные менее важны, поэтому можно ими частично пренебречь

Например, заменив каждый блок 2x2 в Cb и Cr на среднее в этом блоке

Есть 4 режима сабсемплинга:

- 4:4:4 -- без сжатия
- 4:2:2 horizontal -- усредняем каждый блок 1x2
- 4:2:2 vertical -- усредняем каждый блок 2x1
- 4:2:0 -- усредняем каждый блок 2x2

Кодирование JPEG (step=3)

Отнимаем от каждого элемента каждого блока 128

Вычисляем Discrete Cosine Transformation (DCT) над новыми значениями

$$G(u, v) = \frac{1}{4} \alpha(u) \alpha(v) \sum_{x=0}^7 \sum_{y=0}^7 g_{x,y} \cos\left(\frac{(2x+1)u\pi}{16}\right) \cos\left(\frac{(2y+1)v\pi}{16}\right),$$

$$\alpha(u) = \frac{1}{\sqrt{2}}, \text{ если } u = 0, \text{ иначе } 1$$

Кодирование JPEG (step=3)

- Элемент с индексами (0, 0) -- DC коэффициент
- Остальные -- AC коэффициенты
- Чем ближе элементы к левому верхнему углу, тем они важнее для восприятия (низкие частоты)

Кодирование JPEG (step=4)

Каждый блок 8x8 после предыдущего шага:

- Делим на таблицу квантования
- Округляем до целого
- Элементы ближе к правому нижнему углу будут нулями (не все, конечно)

a. Low compression

1	1	1	1	1	2	3	4
1	1	1	1	1	2	3	4
1	1	1	1	2	2	2	4
1	1	1	1	2	2	4	8
1	1	2	2	2	2	4	8
2	2	2	2	2	4	8	8
2	2	2	4	4	8	8	16
4	4	4	4	8	8	16	16

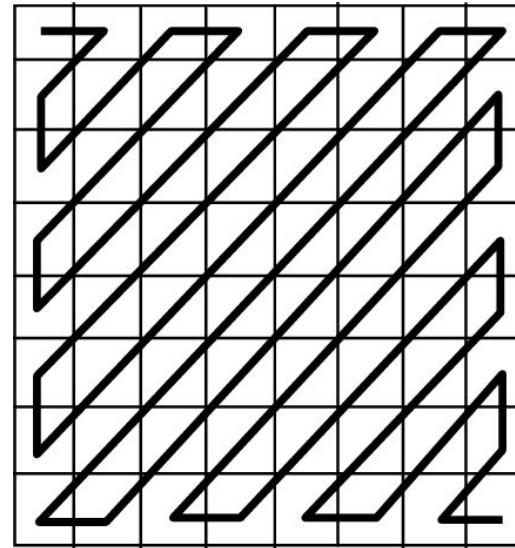
b. High compression

1	2	4	8	16	32	64	128
2	4	4	8	16	32	64	128
4	4	8	16	32	64	128	128
8	8	16	32	64	128	128	256
16	16	32	64	128	128	256	256
32	32	64	128	128	256	256	256
64	64	128	128	256	256	256	256
128	128	128	256	256	256	256	256

Кодирование JPEG (step=5)

Каждый блок 8x8 после предыдущего шага:

- Выписываем матрицу в одномерный вектор в zig-zag порядке



Кодирование JPEG (step=6)

Воспользуемся тем, что в конце вектора много нулей.

Для каждого коэффициента вектора записываем:

- Число нулей до самого коэффициента
- Сам коэффициент
- $(0, 0) ==$ остальное заполнено нулями
- Пример: $(4, 0, 0, 3, 6, 0, 5, 0, \dots, 0) == (0, 4), (2, 3), (0, 6), (1, 5), (0, 0)$

Кодирование JPEG (step=7)

Список пар после предыдущего шага:

- Кодируется и сжимается кодом Хаффмана
- Код общий для всех блоков этой компоненты

Кодирование JPEG (step=8)

Для закодированных блоков:

- Вводится понятие MCU -- блок реального исходного изображения
- Например, при 4:2:0 (усреднении по квадратам 2x2) размер MCU 16x16, он содержит 4 блока Y и по 1 блоку Cb и Cr
- В итоговый файл в порядке сверху вниз, слева направо пишутся MCU
- Внутри самого MCU в том же порядке пишутся сначала блоки Y, затем Cb и Cr

$$Y_{0,0}, Y_{0,1}, Y_{1,0}, Y_{1,1}, Cb_{0,0}, Cr_{0,0} \dots$$

Кодирование JPEG

1. R'G'B' -> Y'CbCr
2. Downsampling
3. Делим на блоки 8x8
4. Нормируем (-128)
5. Discrete Cosine Transformation
6. Делим на таблицу квантизации
7. Выписываем zig-zag-ом
8. Преобразуем в пары (число нулей до коэф, коэф)
9. Кодируем Хаффманом
10. Выписываем по MCU

Декодирование JPEG

1. R'G'B' -> Y'CbCr
2. Downsampling
3. Делим на блоки 8x8
4. Нормируем (-128)
5. Discrete Cosine Transformation
6. Делим на таблицу квантизации
7. Выписываем zig-zag-ом
8. Преобразуем в пары (число нулей до коэф, коэф)
9. Кодируем Хаффманом
10. Выписываем по MCU



Декодирование JPEG

1. R'G'B' -> Y'CbCr
2. Downsampling — какой режим?
3. Делим на блоки 8x8
4. Нормируем (-128)
5. Discrete Cosine Transformation
6. Делим на таблицу квантизации — какая таблица?
7. Выписываем zig-zag-ом
8. Преобразуем в пары (число нулей до коэф, коэф)
9. Кодируем Хаффманом — какой код?
10. Выписываем по MCU



Raw JPEG

```
00000000 ff d8 ff fe 00 04 3a 29 ff db 00 43 00 a0 6e 78
00000010 8c 78 64 a0 8c 82 8c b4 aa a0 be f0 ff ff f0 dc
00000020 dc f0 ff ff
00000030 ff ff
00000040 ff db 00
00000050 43 01 aa b4 b4 f0 d2 f0 ff ff ff ff ff ff ff ff ff
00000060 ff ff
*
00000090 ff ff ff c0 00 11 08 00 10 00 10 03 01 22 00 02
00000a0 11 01 03 11 01 ff c4 00 15 00 01 01 00 00 00 00 00
00000b0 00 00 00 00 00 00 00 00 00 00 03 02 ff c4 00 1a
00000c0 10 01 00 02 03 01 00 00 00 00 00 00 00 00 00 00
00000d0 00 01 00 12 02 11 31 21 ff c4 00 15 01 01 01 00
00000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 ff
00000f0 c4 00 16 11 01 01 01 00 00 00 00 00 00 00 00 00
0000100 00 00 00 00 11 00 01 ff da 00 0c 03 01 00 02 11
0000110 03 11 00 3f 00 ae e7 61 f2 1b d5 22 85 5d 04 3c
0000120 82 c8 48 b1 dc bf ff d9
0000128
```

Структура

- Закодированный JPEG можно поделить на несколько секций
- Каждая из них отвечает за свой кусок информации
- структура: [маркер][длина][контент]
 - Маркер представляет собой два записанных подряд байта
 - Сначала идёт 0xFF, а потом особый байт для каждой секции
 - После маркера записана длина самой секции (кроме секции SOS)
 - Длина учитывает размер себя
 - После длины идёт содержимое секции
- Порядок секций не специфицирован

Маркеры

Маркеры и размер

SOI, EOI

- SOI [0xFFD8] -- маркер начала
- EOI [0xFFD9] -- маркер конца

Если кого-то из них нет на месте нужно
бросать исключение

```
00000000 ff d8 ff fe 00 04 3a 29 ff db 00 43 00 a0 6e 78
0000010 8c 78 64 a0 8c 82 8c b4 aa a0 be f0 ff ff f0 dc
0000020 dc f0 ff ff
0000030 ff ff
0000040 ff db 00
0000050 43 01 aa b4 b4 f0 d2 f0 ff ff ff ff ff ff ff ff ff ff
0000060 ff ff
*
0000090 ff ff ff c0 00 11 08 00 10 00 10 03 01 22 00 02
00000a0 11 01 03 11 01 ff c4 00 15 00 01 01 00 00 00 00
00000b0 00 00 00 00 00 00 00 00 00 00 03 02 ff c4 00 1a
00000c0 10 01 00 02 03 01 00 00 00 00 00 00 00 00 00 00 00
00000d0 00 01 00 12 02 11 31 21 ff c4 00 15 01 01 01 00
00000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 ff
00000f0 c4 00 16 11 01 01 01 00 00 00 00 00 00 00 00 00 00
0000100 00 00 00 00 11 00 01 ff da 00 0c 03 01 00 02 11
0000110 03 11 00 3f 00 ae e7 61 f2 1b d5 22 85 5d 04 3c
0000120 82 c8 48 b1 dc bf ff d9
0000128
```

COM, APP0

- COM [0xFFFF] -- маркер секции с комментарием
 - APPn [0xFFE0-0xFFEF] -- маркер application-specific данных (например, Photoshop)
 -

Ваша задача прочитать и сохранить
комментарий, APPn скипнуть

3a 29 = :)

DQT [0xFFDB]

Этот маркер отвечает за начало описания таблицы квантования

[00 43] Длина: 0x43 = 67 байт

[0_] Длина значений в таблице: 0 (0 — 1 байт, 1 — 2 байта)

[_0] Идентификатор таблицы: 0

DQT [0xFFDB]

[00 43] Длина: 0x43 = 67 байт

[0_] Длина значений в таблице: 0 (0 — 1 байт, 1 — 2 байта)

[_0] Идентификатор таблицы: 0

Выписываем зигзагом

160 110 100 160 240 255 255 255

120 120 140 190 255 255 255 255

140 130 160 240 255 255 255 255

$$\mathbf{DQT}_0 = \begin{matrix} 140 & 170 & 220 & 255 & 255 & 255 & 255 & 255 \\ 180 & 220 & 255 & 255 & 255 & 255 & 255 & 255 \end{matrix}$$

180 220 255 255 255 255 255 255
240 255 255 255 255 255 255 255

240 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255

255 255 255 255 255 255 255 255

233 233 233 233 233 233 233 233

SOFO [0xFFC0]

Хранит мета информацию об изображении в целом:

- [08] Precision: 8 бит
- [00 10] Высота рисунка: 16
- [00 10] Ширина рисунка: 16
- [03] Количество каналов: 3

И о каждом из каналов:

- [01] Идентификатор: 1
- [2_] Горизонтальное прореживание (H1): 2
- [_2] Вертикальное прореживание (V1): 2
- [00] Идентификатор таблицы квантования: 0

...

```
0000000 ff d8 ff fe 00 04 3a 29 ff db 00 43 00 a0 6e 78
0000010 8c 78 64 a0 8c 82 8c b4 aa a0 be f0 ff ff f0 dc
0000020 dc f0 ff ff
0000030 ff ff
0000040 ff db 00
0000050 43 01 aa b4 b4 f0 d2 f0 ff ff
0000060 ff ff
*
0000090 ff ff ff c0 00 11 08 00 10 00 10 03 01 22 00 02
00000a0 11 01 03 11 01 ff c4 00 15 00 01 01 00 00 00 00
00000b0 00 00 00 00 00 00 00 00 00 00 03 02 ff c4 00 1a
00000c0 10 01 00 02 03 01 00 00 00 00 00 00 00 00 00 00
00000d0 00 01 00 12 02 11 31 21 ff c4 00 15 01 01 01 00
00000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 ff
00000f0 c4 00 16 11 01 01 01 00 00 00 00 00 00 00 00 00
0000100 00 00 00 11 00 01 ff da 00 0c 03 01 00 02 11
0000110 03 11 00 3f 00 ae e7 61 f2 1b d5 22 85 5d 04 3c
0000120 82 c8 48 b1 dc bf ff d9
0000128
```

DHT [0xFFC4]

Таблица Хаффмана

[x_] Класс: x (0 — таблица DC коэффициентов, 1 — таблица AC коэффициентов).

[i] Идентификатор таблицы: i

[00]

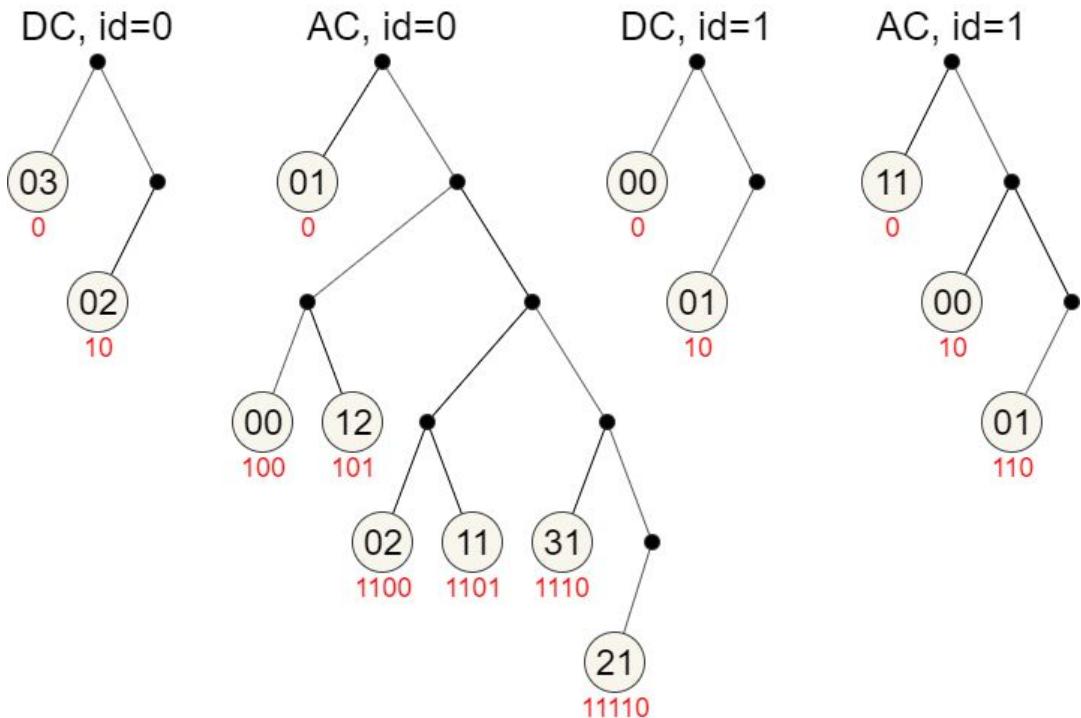
Длина кода Хаффмана: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Значения: [03, 02]

• • •

Дерево Хаффмана

- Пытаемся добавить значение в левую ветвь
- Если она занята, то идем в правую; если и она занята, то возвращаемся на уровень выше, и пробуем оттуда
- Остановиться на уровне равном длине кода
- Левым ветвям соответствует значение 0, правым — 1



SOS [0xFFDA]

Мета информация про закодированные данные:

- [03] Количество каналов. У нас 3, по одному на Y, Cb, Cr.
- [01] Идентификатор канала: 1 (Y)
- [0_] Идентификатор таблицы Хаффмана для DC коэффициентов: 0
- [_0] Идентификатор таблицы Хаффмана для AC коэффициентов: 0
- ...
- [00 3F 00] для progressive режима

```
0000000 ff d8 ff fe 00 04 3a 29 ff db 00 43 00 a0 6e 78
0000010 8c 78 64 a0 8c 82 8c b4 aa a0 be f0 ff f0 dc
0000020 dc f0 ff ff
0000030 ff ff
0000040 ff db 00
0000050 43 01 aa b4 b4 f0 d2 f0 ff ff
0000060 ff ff
*
0000090 ff ff ff c0 00 11 08 00 10 00 10 03 01 22 00 02
00000a0 11 01 03 11 01 ff c4 00 15 00 01 01 00 00 00 00
00000b0 00 00 00 00 00 00 00 00 00 00 03 02 ff c4 00 1a
00000c0 10 01 00 02 03 01 00 00 00 00 00 00 00 00 00 00
00000d0 00 01 00 12 02 11 31 21 ff c4 00 15 01 01 01 00
00000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000f0 c4 00 16 11 01 01 01 00 00 00 00 00 00 00 00 00
000100 00 00 00 00 11 00 01 ff da 00 0c 03 01 00 02 11
000110 03 11 00 3f 00 ae e7 61 f2 1b d5 22 85 5d 04 3c
000120 82 c8 48 b1 dc bf ff d9
000128
```

SOS [0xFFDA]

Сразу после этого идут сами данные.

Рассматриваем как последовательность бит:

10|10|1110|1|1100|11|101|10|0|0|1|11110|0|100 ...

! FF 00 – экранизация байта FF

Сами коэффициенты закодированы минимальным числом бит.

Например, для 13 это 4 бита: 1101. У отрицательных коэффициентов при кодировании берется модуль и их запись инвертируется, 0010 надо будет прочесть как -13.

```
00000000 ff d8 ff fe 00 04 3a 29 ff db 00 43 00 a0 6e 78
00000010 8c 78 64 a0 8c 82 8c b4 aa a0 be f0 ff f0 dc
00000020 dc f0 ff ff
00000030 ff ff
00000040 ff db 00
00000050 43 01 aa b4 b4 f0 d2 f0 ff ff ff ff ff ff ff ff ff
00000060 ff ff
*
00000090 ff ff ff c0 00 11 08 00 10 00 10 03 01 22 00 02
00000a0 11 01 03 11 01 ff c4 00 15 00 01 01 00 00 00 00 00
00000b0 00 00 00 00 00 00 00 00 00 00 03 02 ff c4 00 1a
00000c0 10 01 00 02 03 01 00 00 00 00 00 00 00 00 00 00
00000d0 00 01 00 12 02 11 31 21 ff c4 00 15 01 01 01 01 00
00000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 ff
00000f0 c4 00 16 11 01 01 01 00 00 00 00 00 00 00 00 00
0000100 00 00 00 00 11 00 01 ff da 00 0c 03 01 00 02 11
0000110 03 11 00 3f 00 ae e7 61 f2 1b d5 22 85 5d 04 3c
0000120 82 c8 48 b1 dc bf ff d9
0000128
```

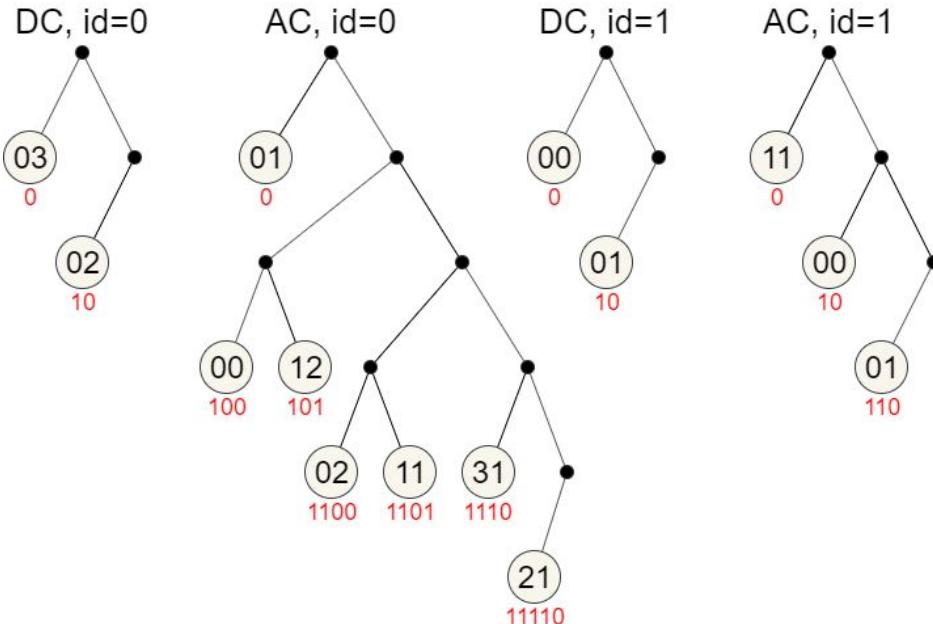
SOS [0xFFDA]

Сразу после этого идут сами данные.

Рассматриваем как последовательность бит:

10|10|1110|1|1100|11|101|10|0|0|0|1|11110|0|100 ...

- 10 = (по Хаффману) 0x02 — длина коэффициента (в битах)
- 10 = DC коэффициент
- 1110 = (по Хаффману) 0x31: 3 нуля перед коэффициентом, 1 его длина
- 1 = AC коэффициент
- 1100 = (по Хаффману) 0x02: 0 нулей перед коэффициентом, 2 его длина
- 11 = AC коэффициент
- ...
- 100 = (по Хаффману) 0x00: Терминальная точка, остальные AC коэффициенты нулевые



2 0 0 0 1 3 0 2 -1 1 0 0 -1 0 0 0 ... 0

SOS [0xFFDA]

Выписываем значения по порядку zig-zag - ом

2 0 0 0 1 3 0 2 -1 1 0 0 -1 0 0 0 ... 0



$$Y_0 = \begin{bmatrix} 2 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Аналогично читаем еще 3 таблицы для Y и по одной для Cb и Cr

SOS [0xFFDA]

Аналогично читаем еще 3 таблицы для Y:

[-4 1 1 1 0 0 0 0 0]	[5 -1 1 0 0 0 0 0 0]	[-4 2 2 1 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0]	[-1 -2 -1 0 0 0 0 0 0]	[-1 0 -1 0 0 0 0 0 0]
[0 -1 0 0 0 0 0 0 0]	[0 -1 0 0 0 0 0 0 0]	[-1 -1 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0]	[-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 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 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 0 0 0 0 0 0 0 0]	[0 0 0 0 0 0 0 0 0]

На самом деле коэффициент (0, 0) это разность настоящего DC от предыдущего, поэтому:

[-2 1 1 1 0 0 0 0 0]	[3 -1 1 0 0 0 0 0 0]	[-1 2 2 1 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0]	[-1 -2 -1 0 0 0 0 0 0]	[-1 0 -1 0 0 0 0 0 0]
[0 -1 0 0 0 0 0 0 0]	[0 -1 0 0 0 0 0 0 0]	[-1 -1 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0]	[-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 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 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 0 0 0 0 0 0 0 0]	[0 0 0 0 0 0 0 0 0]

Дальше читаем по одной матрицу Cb и Cr. Прочитали один блок MCU.

Декодирование JPEG

1. R'G'B' -> Y'CbCr
2. Делим на блоки 8x8
3. Downsampling
4. Нормируем (-128)
5. Discrete Cosine Transformation
6. Делим на таблицу квантизации
7. Выписываем zig-zag-ом
8. Преобразуем в пары (число нулей до коэф, коэф)
9. Кодируем Хаффманом
10. Выписываем по MCU

← you are here



Декодирование JPEG

Дальше:

1. Умножаем на таблицу квантизации
2. Обратное дискретное косинусное преобразование
3. Переносим в диапазон [0, 255]
4. Растворяем обратно если нужно
5. $Y'CbCr \rightarrow R'G'B'$

Формула оценки

Название	Стоимость
huffman	2
fftw	1.5
baseline	2.5
faster	1.5
fuzz	≤ 2.5
progressive (бонус)	2.5

BitReader

- JPEG декодер работает с бинарным файлом
- Работать с данными придётся побитово
- Заведем специальный класс BitReader
- В конструктор передаем ссылку/указатель на `std::istream`
- `std::istream` позволяет читать только побайтово
- Заводим "буффер" для бит
- Метод чтения может принимать размер (в битах)
- JPEG использует big-endian -- научитесь собирать двух-байтовое слово

Советы

- Постарайтесь продумать архитектуру
 - Не вкладывайте классы друг в друга
 - Каждому обработчику нужен какой-то контекст
 - Попробуйте выстроить пайплайн
- Активно пользуйтесь инструментами
 - Логгирование ([glog](#))
 - Стэктрейс ([Boost.Stacktrace](#))
 - Для просмотра внутренностей JPEG есть [JPEGsnoop](#)