Декодирвание jpg (достаточное для шифрования)

Пусть у нас есть изображение:

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
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 FF FF FF FF FF FF FF FF FF FF FF FF
00000030   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000040   FF FF FF FF FF FF FF FF FF FF FF FF FF FF DB 00
00000050   43 01 AA B4 B4 F0 D2 F0 FF FF FF FF FF FF FF FF
00000060   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000070   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000080   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000090   FF FF FF C0 00 11 08 00 10 00 10 03 01 22 00 02
000000A0   11 01 03 11 01 FF C4 00 15 00 01 01 00 00 00 00
000000B0   00 00 00 00 00 00 00 00 00 00 03 02 FF C4 00 1A
000000C0   10 01 00 02 03 01 00 00 00 00 00 00 00 00 00 00
000000D0   00 01 00 12 02 11 31 21 FF C4 00 15 01 01 01 00
000000E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 FF
000000F0   C4 00 16 11 01 01 01 00 00 00 00 00 00 00 00 00
00000100   00 00 00 00 11 00 01 FF DA 00 0C 03 01 00 02 11
00000110   03 11 00 3F 00 AE E7 61 F2 1B D5 22 85 5D 04 3C
00000120   82 C8 48 B1 DC BF FF D9
```

Находим первый маркер FFC4 и далее с каждым маркером FFC4 проделываем одинаковые действия. Т.е. нашли:

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
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 FF FF FF FF FF FF FF FF FF FF FF FF
00000030   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000040   FF FF FF FF FF FF FF FF FF FF FF FF FF FF DB 00
00000050   43 01 AA B4 B4 F0 D2 F0 FF FF FF FF FF FF FF FF
00000060   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000070   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000080   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000090   FF FF FF C0 00 11 08 00 10 00 10 03 01 22 00 02
000000A0   11 01 03 11 01 FF C4 00 15 00 01 01 00 00 00 00
000000B0   00 00 00 00 00 00 00 00 00 00 03 02 FF C4 00 1A
000000C0   10 01 00 02 03 01 00 00 00 00 00 00 00 00 00 00
000000D0   00 01 00 12 02 11 31 21 FF C4 00 15 01 01 01 00
000000E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 FF
000000F0   C4 00 16 11 01 01 01 00 00 00 00 00 00 00 00 00
00000100   00 00 00 00 11 00 01 FF DA 00 0C 03 01 00 02 11
00000110   03 11 00 3F 00 AE E7 61 F2 1B D5 22 85 5D 04 3C
00000120   82 C8 48 B1 DC BF FF D9
```

Разберем подробно на первом маркере:

FF C4 00 15 00 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 03 02

FF C4 – начало секции

00 15 – длина в 16-ричной системе счисления, т.е. $15_{16} = 21_{10}$ байт

00 : [0_] – класс (возможно 2 значения: 0 - таблица DC, 1 - таблица AC)

[_0] – id таблицы (в данном случае 0)

т.е. 00 – DC с id = 0

Далее устанавливаем соответствие между длиной кодом Хаффмана и количеством кодом такой длины:

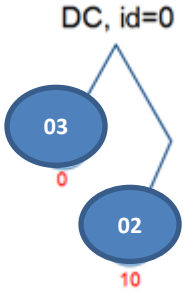
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
01	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00

где 1, 2, …, 16 означают длину кода Хаффмана, а значение ниже – количество кодов такой длины (например, 1 – 01: 1 код длины 1, 2 – 02: 1 код длины 2 и т.д.)

Построим дерево Хаффмана.

Значения добавляем в том порядке, в каком указаны в таблице. Алгоритм: в каком бы узле мы ни находились, всегда пытаемся добавить значение в левую ветвь. А если она занята, то в правую. А если и там нет места, то возвращаемся на уровень выше, и пробуем оттуда. Остановиться нужно на уровне, равном длине кода. Левым ветвям соответствует значение 0, правым — 1.

Замечание: Не нужно каждый раз начинать с вершины. Добавили значение — вернитесь на уровень выше. Правая ветвь существует? Если да, идите опять вверх. Если нет — создайте правую ветвь и перейдите туда. Затем, с этого места, начинайте поиск для добавления следующего значения.



В построенном дереве в узлах дерева находятся значения 03 02(в порядке следования в секции), а красным подписаны сами коды Хаффмана.

Вторая секция FF C4:

FF C4 00 1A 10 01 00 02 03 01 00 00 00 00 00 00 00 00 00 00 01 00 12 02 11 31 21

00 1A – длина, 26 байт

10 – AC с id = 0

Третья секция:

FF C4 00 15 01 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01

00 15 – длина, 21 байт

01 - DC с id = 1

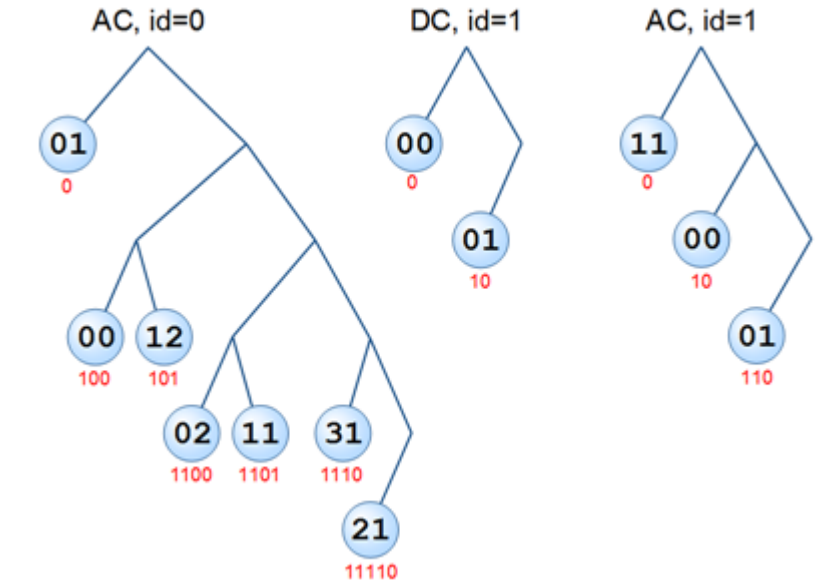
Четвертая секция:

FF C4 00 16 11 01 01 01 00 00 00 00 00 00 00 00 00 00 00 00 11 00 01

00 16 – длина, 22 байта

11 - AC с id = 1

Полученные деревья:



Далее находим маркер [FF DA], в нем хранится секция закодированного изображения:

FF DA 00 0C 03 01 00 02 11 03 11 00 3F 00

FF DA – начало секции

00 0C – длина заголовочной части, 12 байт

03 – количество компонент сканирования (у нас модель Y, Cb, Cr, т.е. 3)

Далее разбор по компонентам:

01 00:

01 – номер компоненты

0_ - id DC

-0 - id AC

т.е. у первой компоненты DC с id = 0 и AC с id = 0

02 11: у второй компоненты DC с id = 1 и AC с id = 1

03 11: у третьей компоненты DC с id = 1 и AC с id = 1

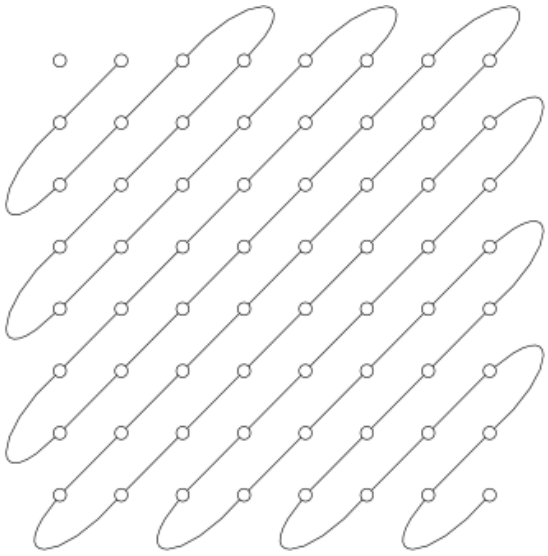
00 3F 00 – нам для кодирования не важно (кому интересно, можно посмотреть спецификацию).

Далее заголовочная часть заканчивается, и начинаются закодированные данные. Действуем так:

1. Переводим в двоичную систему счисления (дополняем нулями спереди до 8 бит, если нужно)
2. Находим DC коэффициент:
   2.1. Читаем последовательность битов. После каждого бита двигаемся по дереву Хаффмана по ветки 0 или 1 в зависимости от прочитанного бита. Останавливаемся, если оказались в конечном узле.
   2.2. Берем значение узла. Если оно равно 0, то записываем в таблицу и переходим к чтению других коэффициентов. Если не ноль, то значение – длина коэффициента в битах, т.е. читаем следующие [количество битов] бита, это и будет коэффициент.
   2.3. Если первая цифра в значении = 1, то ничего не меняем, просто переводим в 10-ую систему счисления. Если 0, то побитово инвертируем и добавляем знак минус, т.е.: $01_2 \rightarrow 10_2 = -2_{10}$
3. Нахождение коэффициентов AC:
   3.1. Аналогично п. 2.1
   3.2. Берем значение узла. Если оно равно 0, то оставшиеся значения матрицы заполняем нулями. Иначе: читаем байт ab следующим образом: a – количество нулей для добавления в матрицу, b – длина коэффициента в битах.
   3.3. Аналогично п. 2.3

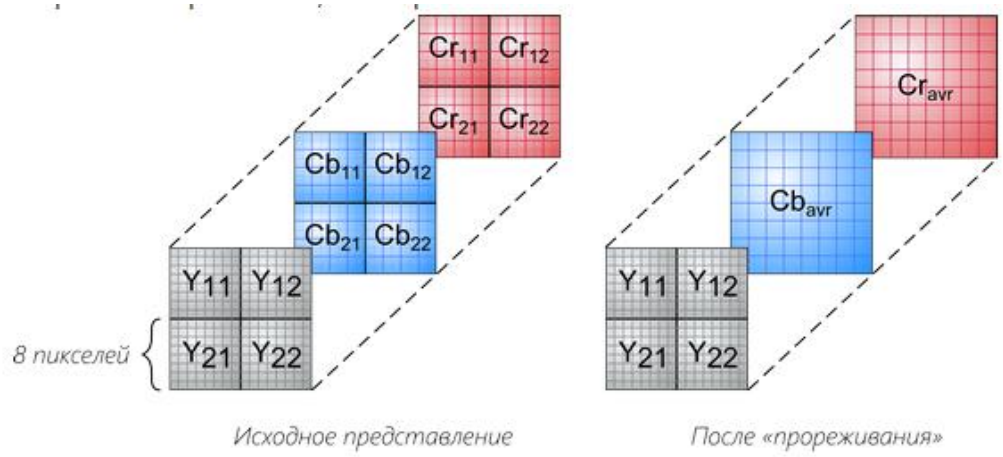
Важно: DC – коэффициенты – это не сами коэффициенты, а их разность между коэффициентами предыдущей таблицы того же канала.

Заполнение матрицы идет по схеме «zig-zag order»:



Zig-zag order

При сжатии jpg мы действуем таким образом (для модели Y, Cb, Cr):



Исходное представление *После «прореживания»*

Т.е. каналы Cb и Cr прореживают.

Укладываются они в таком порядке:

В таком порядке блоки разных каналов кодируются в JPEG

Пример:

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
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 FF FF FF FF FF FF FF FF FF FF FF FF
00000030   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000040   FF FF FF FF FF FF FF FF FF FF FF FF FF FF DB 00
00000050   43 01 AA B4 B4 F0 D2 F0 FF FF FF FF FF FF FF FF
00000060   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000070   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000080   FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000090   FF FF FF C0 00 11 08 00 10 00 10 03 01 22 00 02
000000A0   11 01 03 11 01 FF C4 00 15 00 01 01 00 00 00 00
000000B0   00 00 00 00 00 00 00 00 00 00 03 02 FF C4 00 1A
000000C0   10 01 00 02 03 01 00 00 00 00 00 00 00 00 00 00
000000D0   00 01 00 12 02 11 31 21 FF C4 00 15 01 01 01 00
000000E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 FF
000000F0   C4 00 16 11 01 01 01 00 00 00 00 00 00 00 00 00
00000100   00 00 00 00 11 00 01 FF DA 00 0C 03 01 00 02 11
00000110   03 11 00 3F 00 AE E7 61 F2 1B D5 22 85 5D 04 3C
00000120   82 C8 48 B1 DC BF FF D9
```

AE	EF	61	F2	1B
10101110	11100111	01100001	11110010	00011011

Т.е. имеем последовательность: 10101110111001110100001111100100011011…

10 – 02 в дереве Хаффмана (DC = 0, id = 0), т.е. берем 2 следующих бита, чтобы узнать коэффициент: $10_2 = 2_{10}$

1110 – 31 в дереве Хаффмана (AC = 0, id = 0), т.е. нужно заполнить 3 нулями, а затем берем 1 бит: $1_2 = 1_{10}$

и т.д., пока не заполнили всю матрицу, или (как в нашем случае) не встретили 100, которому соответствует нулевое значение.

Получили матрицу Y_{11}:

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

Далее:

1B	D5	22
00011011	11010101	00100010

0011011110101010010000010

Получили матрицу Y_{12}:

-2	1	1	1	0	0	0	0
0	0	1	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

Далее:

22	85	5D	04
00100010	10000101	01011101	00000100

0101000010101011101000000100

Получили матрицу Y_{21}:

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

Далее:

3C	82	C8	48
00111100	10000010	11001000	01001000

0011110010000010110010000001001000

Получили матрицу Y_{22}:

-1	2	2	1	0	0	0	0
-1	0	-1	0	0	0	0	0
-1	-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

Аналогично получим матрицы для компонент , Cb и Cr.