Заметки Крикса об алгоритме декодирования JPEG.
Авторское Право 1999 Cristi Cuturicu.

ОТКАЗ (ПРЕДУПРЕЖДЕНИЕ)

Вы получаете этот файл свободно, так что Вы не можете иметь любые юридические запросы от меня. Если Вы не соглашаетесь, не читайте дальше. Никакая гарантия не обеспечивается этим документом, могут быть дефекты или ошибки, сбои (хотя я попытался избегать их), так что использовать информацию, содержащуюся в этом файле - Ваш собственный риск. Это неофициальная документация, для дальнейшей информации, пожалуйста, ссылайтесь на JPEG стандарт ISO. Все имена, упомянутые в этом файле - торговые марки или зарегистрированные фирменные знаки их соответствующих владельцев. Не для коммерческого, персонального использования.

JPEG СЖАТИЕ и ФОРМАТ ФАЙЛА JPG

Давно, я искал через сеть хороший документ, который объяснял бы мне JPEG сжатие, и особенно файловый формат JPG. И недавно я обнаружил ISO-ITU JPEG стандарт в файле, названном itu-1150.ps (JPEG стандарт = ISO стандарта 10918-1 или стандартная рекомендация CCITT T.81: "Информационные Технологии - Цифровое сжатие и кодирование фотографических статичных изображений - Требования и руководящие принципы"). Хотя этот стандарт совсем завершен, в руководстве есть много лишнего на 186 страницах, и я копался в них, а затем написал свой собственный просмотрщик JPG.

Сначала примечание: Главным образом из-за того, что большинство файлов JPG сжаты способом Baseline Sequential DCT, этот документ рассматривает только этот формат сжатия и особенно его JFIF реализацию. Он НЕ раскрывает JPG Прогрессивное или Иерархическое сжатие.
(Если нужна дополнительная информация, читайте стандарт itu-1150.
Его можно найти на www.wotsit.org или где-нибудь в www.jpeg.org/jpeg)

Я подумал, что было бы легче для читателя понять сжатие JPG, если я объясню шаги шифратора. (Шаги дешифратора будут инверсией шагов шифратора, но в обратном порядке, конечно)

ШАГИ ШИФРАТОРА JPEG

1) Плавное преобразование цветового пространства: [R G B] -> [Y Cb Cr]

(R,G,B - 8-битовые величины без знака)

```
      | Y  |     |  0.299      0.587      0.114  |    | R |     | 0 |
      | Cb |  =  | -0.1687    -0.3313     0.5    | *  | G |  +  |128|
      | Cr |     |  0.5       -0.4187    -0.0813 |    | B |     |128|
```

Новая величина Y = 0.299*R + 0.587*G + 0.114*B названа яркостью. Это – величина, использованная монохромными мониторами, чтобы представить цвет RGB. Физиологически, передает интенсивность цвета RGB воспринятого глазом.
Вы видите, что формула для Y, подобно средневзвешенному значению с разным весом для каждого спектрального компонента: глаз наиболее чувствителен на Зеленый цвет, затем следует Красный компонент и в последнюю очередь - Синий.

```
Величины Cb = - 0.1687*R - 0.3313*G + 0.5    *B + 128
       и Cr =   0.5    *R - 0.4187*G - 0.0813*B + 128
```

названы цветовыми величинами и представляют 2 координаты в системе, которая измеряет оттенок и насыщение цвета ([Приближенно], эти величины указывают количество синего и красного в этом цвете).
Эти 2 координаты кратко названы цветоразностью.

Преобразование [Y,Cb,Cr] в [R,G,B] (обратно предыдущему преобразованию)

RGB-цвет может быть вычислен непосредственно из YCbCr (8-битовые величины без знака) следующим образом:

```
R = Y                    + 1.402  *(Cr-128)
G = Y - 0.34414*(Cb-128) - 0.71414*(Cr-128)
B = Y + 1.772  *(Cb-128)
```

Примечание, связывающее Y,Cb,Cr в человеческой визуальной системе

Глаз, особенно сетчатка, имеет как визуальные анализаторы два типа ячеек: ячейки для ночного видения, воспринимающие только оттенки серого (от ярко-белого до темно-черного) и ячейки дневного видения, которые воспринимают цветовой оттенок. Первые ячейки, дающие цвет RGB, обнаруживают уровень яркости, подобный величине Y. Другие ячейки, ответственные за восприятие цветового оттенка, - определяют величину, связанную с цветоразностью.

2) Дискретизация

JPEG Стандарт принимает во внимание то, что глаз более чувствителен к яркости цвета, чем к оттенку этого цвета. (Черно-белые ячейки вида имеют больше влияния, чем ячейки дневного видения)

Так, для большинства JPG, яркость взята для каждого пикселя, тогда как цветоразность – как средняя величина для блока 2x2 пикселей. Имейте в виду, что это не обязательно, но при этом можно достичь хороших результатов сжатия, с незначительным убытком в визуальном восприятии нового обработанного изображения.

Примечание: JPEG стандарт определяет, что для каждого компонента образа (подобно, например Y) должно быть определено 2 коэффициента дискретизации: один для горизонтальной дискретизации и один для вертикальной дискретизации. Эти коэффициенты дискретизации определяются в файле JPG как относительно максимального коэффициента дискретизации (дополнительно об этом позже).

3) Сдвиг Уровня

Все 8-битовые величины без знака (Y,Cb,Cr) в изображении - "смещенные по уровню": они преобразовываются в 8-битовое знаковое представление вычитанием 128 из их величины.

4) 8x8 Дискретное Косинусоидальное Преобразование (DCT)

Изображение делится на блоки 8x8 пикселей, затем для каждого блока 8x8 применяется DCT-трансформация. Заметьте, что если размер X исходного образа не делится на 8, шифратор должен сделать его делимым, дополняя остальные правые столбцы (пока X не станет кратным 8). Аналогично, если размер Y не делимо на 8, шифратор должен дополнить строки.
Блоки 8x8 обрабатываются слева направо и сверху вниз.

Примечание: Поскольку каждый пиксель в блоке 8x8 имеет 3 компонента (Y,Cb,Cr), DCT приложен отдельно в трех блоках 8x8:
Первый блок 8x8 является блоком, который содержит яркость пикселей в исходном блоке 8x8;
Второй блок 8x8 является блоком, который содержит величины Cb;
И, аналогично, третий блок 8x8 содержит величины Cr.

Цель DCT-трансформации в том, что вместо обработки исходных изображений, Вы работаете с пространством частот изменения яркости и оттенка. Эти частоты очень связаны с уровнем детализации изображения. Высокие частоты соответствуют высокому уровню детализации.

DCT-трансформация очень похожа на 2-мерное преобразование Фурье, которое получает из временного интервала (исходный блок 8x8) частотный интервал (новые коэффициенты 8x8=64, которые представляют амплитуды проанализированного частотного пространства)

Математическое определение прямого DCT (FDCT) и обратного DCT (IDCT):

FDCT:

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

u,v = 0,1...7

c(u,v)=1/2, когда u=v=0;
c(u,v)= 1 – в остальных случаях.

IDCT:

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

x,y = 0,1...7

Применение этих формул непосредственно в вычислительном отношении дорого, особенно, когда имеются разработанные более быстрые алгоритмы для прямого или обратного DCT. Один, названный AA&N, имеет только 5 операций умножения и 29 операций сложения. Больше информации и реализацию этого можно найти в свободном программном обеспечении для JPEG кодировщиков от Независимой JPEG Группы (IJG), их C-источники могут быть найдены на www.ijg.org.

5) Зигзагообразная перестановка 64 DCT коэффициентов

Так, после того, как мы выполнили DCT-преобразование над блоком величин 8x8, у нас есть новый блок 8x8. Затем, этот блок 8x8 просматривается по зигзагу подобно этому:

(Числа в блоке 8x8 указывают порядок, в котором мы просматриваем 2-мерную матрицу 8x8)

```
 0, 1, 5, 6,14,15,27,28,
 2, 4, 7,13,16,26,29,42,
 3, 8,12,17,25,30,41,43,
 9,11,18,24,31,40,44,53,
10,19,23,32,39,45,52,54,
20,22,33,38,46,51,55,60,
21,34,37,47,50,56,59,61,
35,36,48,49,57,58,62,63
```

Как Вы видите, сначала - верхний левый угол (0,0), затем величина в (0,1), затем (1,0), затем (2,0), (1,1), (0,2), (0,3), (1,2), (2,1), (3,0) и т.п.

После того, как мы прошли по зигзагу матрицу 8x8, мы имеем теперь вектор с 64 коэффициентами (0..63) Смысл этого зигзагообразного вектора – в том, что мы просматриваем коэффициенты 8x8 DCT в порядке повышения пространственных частот. Так, мы получаем вектор отсортированный критериями пространственной частоты: первая величина на векторе (индекс 0) соответствует самой низкой частоте в изображении – она обозначается термином DC. С увеличением индекса на векторе, мы получаем величины соответствующие высшим частотам (величина с индексом 63 соответствует амплитуде самой высокой частоте в блоке 8x8). Остальная часть коэффициентов DCT обозначается AC.

6) Квантование

На этом этапе, у нас есть отсортированный вектор с 64 величинами, соответствующими амплитудам 64 пространственных частот в блоке 8x8.

Эти 64 величины квантованы: Каждая величина делится на число, определенное для вектора с 64 величинами - таблицу квантования, затем округляется до ближайшего целого.

```
для (i = 0; i<=63; i++)
   вектор[i] = (округлить) (вектор[i] / таблица_квантования[i] + 0.5)
```

Вот пример таблицы квантования для яркости(Y) данной в приложении JPEG стандарта. (Дается в форме блока 8x8; полученного из 64 векторных величин, зигзагообразным преобразованием)

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

Эта таблица опирается на "психовизуальный порог", это "используется с хорошими результатами для изображений с 8-битовой яркостью и оттенками". Большинство существующих шифраторов просто копируют этот пример, но величины не оптимизируются (шифратор может использовать ЛЮБУЮ ДРУГУЮ таблицу квантования) таблица определяется в JPEG файле с DQT (Определение Таблицы Квантования) маркером. Обычно присутствует одна таблица для Y, и другие для оттенка (Cb и Cr).

Процесс квантования играет ключевую роль в JPEG сжатии. Это - процесс, который удаляет высокие частоты, представленные в исходном изображении - впоследствии высокую детализацию. Мы делаем это из-за того, что глаз более чувствителен к низким частотам, чем к высоким, так что мы можем удалить, с очень небольшим визуальным убытком, высокие частоты. Это сделано посредством деления величин в высоких индексах на векторе (амплитуды высоких частот) на большие величины, чем величины, на которыми разделены амплитуды более низких частот. Больше величины в таблице квантования - больше потери (впоследствии визуальные потери) введенные этим процессом, и меньше – лучше визуальное качество.

Другой важный факт - в большинстве изображений цвет изменяется медленно от одного пикселя к другому, так что большинство образов будут иметь небольшое количество высокой детализации -> небольшая сумма (небольшие амплитуды) высоких пространственных частот - но у них есть много информации об изображении, содержащейся на низких пространственных частотах.

Впоследствии на новом квантованном векторе, на высоких пространственных частотах, мы будем иметь много последовательных нулей.

7) RunLength кодирование нулей (RLC)

Теперь у нас есть квантованный вектор с длинной последовательностью нулей. Мы можем использовать это, кодируя последовательные нули. ВАЖНО: Вы увидите позже почему, но здесь мы пропускаем кодировку первого коэффициента вектора (коэффициент DC), который закодирован по-другому. (Я представлю его кодирование позже в этом документе) Рассмотрим исходный 64 вектор как 63 вектор (это - 64 вектор без первого коэффициента)

Допустим, мы имеем 57,45,0,0,0,0,23,0,-30,-16,0,0,1,0,0,0,0,0,0, только 0,...,0

Здесь - как RLC JPEG сжатие сделано для этого примера:

(0,57); (0,45); (4,23); (1,-30); (0,-16); (2,1); EOB

Как Вы видите, мы кодируем для каждой величины, отличающейся от 0 количество последовательных ПРЕДШЕСТВУЮЩИХ нулей перед величиной, затем мы добавляем величину. Другое примечание: EOB - короткая форма для Конца Блока, это - специальная кодированная величина (маркер). Если мы достигли в позиции на векторе, от которого мы имеем до конца только нули вектора, мы выделим эту позицию с EOB и завершим сжатие RLC квантованного вектора.

[Заметьте, что если квантованный вектор не оканчивается нулями (имеет последний элемент не 0), мы не будем иметь маркер EOB.]

(0,57); (0,45); (4,23); (1,-30); (0,-16); (2,1); (0,0)

Другая ОСНОВНАЯ вещь: Допустим, где-нибудь на квантованном векторе мы имеем:
57, восемнадцать нулей, 3, 0,0 ,0,0 2, тридцать-три нуля, 895, EOB

Кодирование Хаффмана JPG делает ограничение (Вы увидите позже почему), по которому число предшествующих нулей должно кодироваться как 4-битовая величина - не может превысить 15.

Так, предшествующий пример должен быть закодирован как:
(0,57); (15,0) (2,3); (4,2); (15,0) (15,0) (1,895), (0,0)

(15,0) - специальная кодированная величина, которая указывает, что там следует за 16 последовательными нулями. Примечание: 16 нулей не 15 нулей.

8) Конечный шаг - кодирование Хаффмана

Сначала ВАЖНОЕ примечание: Вместо хранения фактической величины, JPEG стандарт определяет, что мы храним минимальный размер в битах, в котором мы можем держать эту величину (это названо категория этой величины) и затем битно кодированное представление этой величины подобно этому:

Величины	*Категория*	*Биты для величины*
0	0	-
-1,1	1	0,1
-3,-2,2,3	2	00,01,10,11
-7,..,-4,4,..,7	3	000,001,010,011,100,101,110,111
-15,..,-8,8,..,15	4	0000,..,0111,1000,..,1111
-31,..,-16,16,..,31	5	00000,..,01111,10000,..,11111
-63,..,-32,32,..,63	6	.
-127,..,-64,64,..,127	7	.
-255,..,-128,128,..,255	8	.
-511,..,-256,256,..,511	9	.
-1023,..,-512,512,..,1023	10	.
-2047,..,-1024,1024,..,2047	11	.
-4095,..,-2048,2048,..,4095	12	.
-8191,..,-4096,4096,..,8191	13	.
-16383,..,-8192,8192,..,16383	14	.
-32767,..,-16384,16384,..,32767	15	.

Впоследствии для предшествующего примера:
(0,57); (0,45); (4,23); (1,-30); (0,-8); (2,1); (0,0)

давайте закодируем ТОЛЬКО правую величину этих пар, кроме пар, которые являются специальными маркерами подобно (0,0) или (если мы должны иметь) (15,0)

57 - в категории 6 и это – битно кодированный 111001, так что мы закодируем это как (6,111001)
45, аналогично, будет закодирован как (6,101101)
23 -> (5,10111)
-30 -> (5,00001)
-8 -> (4,0111)
1 -> (1,1)

И теперь, мы напишем снова строку пар:

(0,6), 111001; (0,6), 101101; (4,5), 10111; (1,5), 00001; (0,4), 0111; (2,1), 1; (0,0)

Пары 2 величин, заключенные в скобки, могут быть представлены в байте, так как фактически каждая из 2 величин может быть представлена в 4-битном кусочке (счетчик предшествующих нулей - всегда меньше, чем 15 и также как и категория [числа закодированные в файле JPG - в области -32767..32767]). В этом байте, старший кусочек представляет число предшествующих нулей, а младший кусочек - категорию новой величины, отличной от 0.

КОНЕЧНЫЙ шаг кодировки состоит в кодировании Хаффмана этого байта, и затем записи в файле JPG, как поток из битов, кода Хаффмана этого байта, сопровождающийся битовым представлением этого числа.

Например, для байта 6 (эквивалент (0,6)) у нас есть код Хаффмана = 111000;
для байта 69 = (4,5) (например) у нас есть 1111111110011001
21 = (1,5) — 11111110110
4 = (0,4) — 1011
33 = (2,1) — 11011
0 = EOB= (0,0) — 1010

Конечный поток битов записанных в файле JPG на диск для предшествующего примера 63 коэффициентов (запомните, что мы пропустили первый коэффициент) -

```
111000 111001  111000 101101  1111111110011001 10111   11111110110 00001
        1011 0111   11011 1   1010
```

Кодирование коэффициента DC

DC является коэффициентом на квантованном векторе соответствующем самой низкой частоте в образе (это - 0 частота), и (перед квантованием) - математически = (сумма 8x8 составляющих)/8. (Это - подобно средней величине для этого блока образцов изображения). Сказано, что он содержит много энергии, присутствующей в исходном изображении блока 8x8. (Обычно, это большие величины). Авторы JPEG стандарта обращали внимание, что есть очень близкое соотношение между коэффициентом DC последовательных блоков, так что они решили кодировать в файле JPG различие между DC последовательными блоками 8x8 (Примечание: последовательные блоки 8x8 одинаковых компонентов изображения, подобно последовательным блокам 8x8 для Y, или последовательные блоки для Cb, или для Cr)

Diff = DC_i – DC_{i-1}; Так DC текущего блока (DC_i) равен: DC_i = DC_{i-1} + Diff

Декодируя JPG, Вы начнете с 0 - Вы рассмотрите первым коэффициентом DC = 0; DC_0 = 0

А затем Вы добавите к текущей величине величину, декодированную из JPG (Diff величину)

Так, в файле JPG, первый коэффициент = коэффициент DC является действительно отличительным, и это – Хаффман-кодирование отличается от кодирования коэффициентов AC.

Здесь это - как это сделано: (Запомните, что мы теперь кодируем Diff величину)

Diff предшествует, как Вы видели, представлению категории и, это – бит-кодированное представление. В файле JPG будет Хаффманом закодирована только величина категории, подобно этому:

Diff = (Категория, битно кодированное представление)
Затем Diff будет закодирован как (код_Хаффмана (категория), битно кодированное представление)

Например, если Diff равняется -511, тогда Diff соответствует
(9, 000000000)
Допустим, что 9 имеет код Хаффмана = 1111110 (В файле JPG, есть 2 таблицы Хаффмана для каждого компонента изображения: для DC и для AC)

В файле JPG, биты соответствующие коэффициенту DC будут:
1111110 000000000
И, относившийся к этому примеру DC и в предшествующем примере AC, для этого вектора с 64 коэффициентами, КОНЕЧНЫЙ ПОТОК БИТОВ, записанных в файле JPG, будет:

```
   1111110 000000000 111000 111001  111000 101101  1111111110011001 10111
       11111110110 00001 1011 0111   11011 1   1010
```

(В файле JPG, сначала закодировано DC, затем AC)

ДЕШИФРАТОР ХАФФМАНА (краткий итог)
для 64 коэффициентов (Единиц Данных) компонента изображения (Например Y)

Так, когда Вы декодируете поток битов с изображения в файле JPG, Вы сделаете:

Инициализируйте DC = 0.

1) Сначала декодируйте коэффициент DC:

а) Выберите правильный код Хаффмана (проверьте, существует ли он в DC таблице Хаффмана)
б) Посмотрите, какой категории этот код Хаффмана соответствует
в) Пусть N равно категории битов,
и определите, какая величина представлена (категория, выбранные биты N) = Diff
г) DC + = Diff
д) Запишите DC в 64 вектор: "вектор[0]=DC"

2) Декодируйте 63 коэффициента AC:

- ДЛЯ каждого коэффициента AC ДО ТЕХ ПОР, ПОКА (EOB_достигнут ИЛИ AC_счетчик=64)

а) Выберите правильный код Хаффмана (проверьте существование в таблице Хаффмана)
б) Декодируйте этот код Хаффмана: Код Хаффмана соответствует (число_предшест_0,категория)
   [Помните: EOB_достигнут = ВЕРНО, если (число_предшест_0, категория) = (0,0)]
в) Пусть N равно категории битов, и определите, какая величина представлена
   (категория, выбранные биты N) = AC_коэффициент
г) Запишите в 64 векторе, число нулей = число_предшест_0
д) Увеличьте AC_счетчик на число_предшест_0 (число предшествующих нулей)
е) Запишите AC_коэффициент в векторе: "вектор[AC_счетчик]=AC_коэффициент"

Следующие Шаги

Так, теперь у нас есть вектор из 64 элементов.
Мы сделаем возобновление шагов представленных в этом док.:

1) Выполните обратное квантование для 64 вектора:
   "для (i=0; i<=63; i++) вектор[i]*=квант[i]"
2) Перестройте 64 элемента из зигзагообразного вектора в блок 8x8
3) Примените обратное DCT-преобразование в блок 8x8
   Повторите предыдущие операции [дешифратор Хаффмана, шаги 1), 2) и 3)]
   для каждого блока 8x8 каждого компонента образа (Y,Cb,Cr).
4) Обработайте, если требуется
5) Сдвиньте по уровню величины (добавьте 128 ко всем 8-битовых знаковым величинам
   в блоках 8x8, полученным IDCT-преобразованием (обратным DCT))
6) Преобразуйте YCbCr в RGB
7) - И... JPG изображение

JPEG Маркеры и/или как организована информация изображения в файле JPG (Байтовый уровень)

ПРИМЕЧАНИЕ: JPEG/JFIF файловый формат использует формат Motorola для целых (двухбайтовых) значений, НЕ формат Intel, то есть: сначала старший байт, затем - младший (например: двухбайтовое целое значение FFA0 будет записано в JPEG файле в порядке: FF в младшем смещении, A0 в старшем смещении)

Стандарт JPG определяет, что JPEG файл сформирован в основном из частей, названных "сегментами". Сегмент является потоком байтов с длиной <= 65535. Сегментное начало определяется маркером. Маркер = 2 байта, начиная с 0xFF (С-шестнадцатеричная запись для 255), и заканчивая байтом, отличным от 0 и 0xFF. Например: 'FFDA', 'FFC4', 'FFC0'. Каждый маркер имеет значение: второй байт (отличный от 0 и 0xFF) определяет функцию этого маркера. Например, есть маркер, который определяет, что Вы должны запускать декодирование процесса, это обозначается (стандартная терминология JPG): SOS = Начало Сканирования = 'FFDA'

Другой маркер, названный DQT = Определение Таблицы Квантования = 0xFFDB, определяет, что в файле JPG, после маркера (и после 3 байтов, дополнительно об этом позже) следуют 64 байта = коэффициенты таблицы квантования.

Если, в течение обработки файла JPG, Вы встречаете 0xFF, затем снова байт, отличный от 0 (Я сообщил Вам, что второй байт маркера - не 0) и этот байт не имеет значение маркера (Вы не можете найти маркер, соответствующий этому байту), тогда байт 0xFF, который Вы встретили, должен игнорироваться и пропускаться. (В некоторых JPG-файлах, последовательности последовательных 0xFF - для некоторого заполнения и должны пропускаться)

Вы видите, что всякий раз, когда Вы встречаете 0xFF, Вы проверяете следующий байт и смотрите, имеет ли значение маркера или должно быть пропущено. Что случается, если нам действительно нужно кодировать байт 0xFF в файле JPG как *обычный* байт (не маркер или заполнение)? (Допустим, что нам нужно писать код Хаффмана, который начинается с 11111111 (8 битов 1) на перестановке байтов) Стандарт сообщает, что мы просто делаем следующий байт 0, и пишем последовательность 'FF00' в файл JPG. Так, когда Ваш дешифратор JPG встречает 2-байтовую последовательность 'FF00', он должен рассмотреть ее как обычный байт 0xFF.

Другая вещь: Вы понимаете, что эти маркеры являются выровненными по байту в файле JPG. Что случается, если в течение вашего кодирования Хаффмана и, включая биты в файловых байтах JPG, Вы не завершили включать биты в байт, но Вам нужно писать маркер, который является выровненным по байту? Для перестановки байтов маркеров, Вы УСТАНАВЛИВАЕТЕ ОСТАЛЬНЫЕ БИТЫ ДО НАЧАЛА СЛЕДУЮЩЕГО БАЙТА НА 1, тогда Вы пишите маркер в следующем байте.

Короткое объяснение некоторых важных маркеров, содержащихся в файле JPG.

SOI = Начало Изображения = 'FFD8'
  Этот маркер должен присутствовать в любом файле JPG *однажды* в начале файла.
  (Любой файл JPG начинается с последовательности FFD8.)
EOI = Конец Изображения = 'FFD9'
  Подобно EOI: любой файл JPG заканчивается на FFD9.

RSTi = FFDi (где i - значения 0..7) [RST0 = FFD0, RST7=FFD7]
     = Маркеры Перезапуска
Эти маркеры перезапуска используются для синхронизации. В регулярных интервалах, они появляются в потоке JPG байтов, в течение декодирования процесса (после SOS)
(Они появляются в порядке: RST0 - интервал - RST1 - интервал - RST2 -...
                      ...- RST6 - интервал - RST7 - интервал - RST0 -...)
(Замечание: Большинство JPG-файлов не имеет маркеров перезапуска)

Проблема с этими маркерами – в том, что они прерывают нормальный битовый порядок в Хаффман-кодированном битовом потоке JPG. Запомните, что для байтового выравнивания маркеров остальные биты устанавливаются на 1, так что Ваш дешифратор должен пропустить в регулярные интервалы бесполезные биты заполнения (установленные в 1) и маркеры RST.

Маркеры...

В конце этого документа, я включил очень хорошее письменное техническое объяснение JPEG/JFIF файлового формата, записанное Oliver Fromme, автором QPEG-просмотрщика. Там Вы найдете довольно хорошее и полное определение для маркеров.

Но, во всяком случае, здесь - список маркеров, которые Вы должны проверять:

SOF0	=	Начало Кадра 0	= FFC0
SOS	=	Начало Сканирования	= FFDA
APP0	=	маркер для идентификации файла JPG, который использует JFIF спецификацию	= FFE0
COM	=	Комментарий	= FFFE
DNL	=	Определение Количество Строк	= FFDC
DRI	=	Определение Интервала Перезапуска	= FFDD
DQT	=	Определение Таблицы Квантования	= FFDB
DHT	=	Определение Таблицы Хаффмана	= FFC4

Таблица Хаффмана, сохраненная в файле JPG

Здесь - как JPEG содержит дерево Хаффмана: вместо дерева, он определяет таблицу в файле JPG после DHT (Определение Таблицы Хаффмана) маркера. ПРИМЕЧАНИЕ: длина кодов Хаффмана ограничивается 16 битами.

В основном есть 2 типа таблиц Хаффмана в файле JPG: одна для DC и одна для AC (действительно есть 4 таблицы Хаффмана: 2 для DC, AC для яркости и 2 для DC, AC цветоразности)

Они сохраняются в файле JPG в том же формате, который состоит из:
1) 16 байтов:

байт i содержит число кодов Хаффмана длины i (длина в битах)
   i изменяется от 1 до 16

2) Таблица с длиной (в байтах) = $\Sigma_{i=1}^{16}$ число_кодов_длины_i

которая содержит в позиции [k][j] (k в 1..16, j в 0..(число_кодов_с_длиной_k-1)) БАЙТОВУЮ величину, связанную с j-ым кодом Хаффмана длины k. (Для фиксированной длины k, величины сохранены, отсортированными по величине кода Хаффмана)

В этой таблице Вы можете найти фактический код Хаффмана, связанный с конкретным байтом.

Например: (Примечание: количество кодов для данной длины - здесь для этого конкретного примера, чтобы считать это, они могут иметь любые другие величины)
ДОПУСТИМ что,

```
       Для длины 1 мы имеем число_кодов[1]=0, мы пропускаем эту длину
       Для длины 2 у нас есть 2 кода 00
                                     01
       Для длины 3 у нас есть 3 кода 100
                                     101
                                     110
       Для длины 4 у нас есть 1 код  1110
       Для длины 5 у нас есть 1 код  11110
       Для длины 6 у нас есть 1 код  111110
       Для длины 7 у нас есть 0 кодов – пропускаем
 (если у нас был 1 код для длины 7,
                     мы должны иметь 1111110)
       Для длины 8 у нас есть 1 код  11111100 (Вы видите, что код все еще перемещается
                                               на оставленное, хотя мы пропустили
                                               кодовую величину для 7)
       .....
       Для длины 16,... (та же вещь)
```

Я сообщил Вам, что в таблице Хаффмана в файле JPG загружены БАЙТОВЫЕ величины для данного кода.

Для этого конкретного примера кодов Хаффмана:
Допустим, что в таблице Хаффмана в файле JPG на диске мы имеем (после тех 16 байтов, которые содержат число кодов Хаффмана с данной длиной):

```
    45 57 29 17 23 25 34 28
```

Эти величины соответствуют конкретным длинам, которые я Вам дал перед этим, в кодах Хаффмана это будет:

```
    нет величины для кода длины 1
    для кодов длины 2 : у нас есть 45 57
    для кодов длины 3 : 3 величины (например : 29,17,23)
    для кодов длины 4 : только 1 величина (например: 25)
    для кодов длины 5 : 1 величина (например: 34)
    ...
    для кода длины 7, снова никакая величина, переходим к коду с длиной 8
    для кода длины 8 : 1 величина 28
```

ВАЖНОЕ примечание:

```
  Для кодов длины 2:
       величина 45 соответствует коду 00
                57                коду 01
  Для кодов длины 3:
       величина 29 соответствует коду 100
                17      ---||---       101
                23      ---||---       110
  И Т.П...
```

(Я сообщил Вам, что для данной длины байтовые величины хранятся в порядке увеличения величины кода Хаффмана.)

Четыре таблицы Хаффмана соответствуют DC и AC таблицам яркости, и DC и AC таблицам для цветоразности, даются в приложении JPEG стандарта как предложение для шифратора. Стандарт сообщает, что эти таблицы протестированы с хорошими результатами сжатия для многих изображений и рекомендует их, но шифратор может использовать любую другую таблицу Хаффмана. Много шифраторов JPG используют эти таблицы. Некоторые предлагаются на Ваш выбор: оптимизация энтропии - если возможно, используйте таблицы Хаффмана, оптимизированные для этого конкретного изображения.

JFIF (Jpeg Формат Взаимозаменяемого Файла) файл

JPEG стандарт (что в файле itu-1150.ps) как-нибудь очень обобщенный, JFIF реализация является конкретным случаем этого стандарта (и он является, конечно, совместимым со стандартом). JPEG стандарт определяет некоторые маркеры, зарезервированные для приложений (под приложениями я подразумеваю конкретные случаи осуществления стандарта) Эти маркеры названы APPn, где n изменяется от 0 до 0xF; APPn = FFEn. JFIF спецификация использует маркер APP0 (FFE0), чтобы идентифицировать файл JPG, который использует эту спецификацию. Вы увидите в JPEG стандарте, что он имеет отношение к "компонентам образа". Эти компоненты образа могут быть (Y,Cb,Cr) или (YIQ) или все, что угодно. JFIF реализации использует только (Y,Cb,Cr) для цветного JPG, или только Y для монохромного JPG.

Показатели (факторы) дискретизации

Примечание: следующее объяснение раскрывает кодировку цветных (3 компонента) JPG-файлов; для черно-белых JPG-файлов есть один компонент (Y), который - обычно никак необработанный совсем, и не требует любого обратного преобразования подобно инверсии (Y,Cb,Cr) -> (R,G,B). Впоследствии, черно-белые JPG-файлы - самые простейшие для декодирования: для каждого блока 8x8 в изображении Вы делаете декодирование Хаффмана кодированного вектора RLC, который затем Вы преобразовываете из зигзага, деквантуете 64 вектор и, наконец, Вы применяете к нему обратное DCT и прибавляете 128 (сдвиг уровня) к новым величинам 8x8.

Я сообщил Вам, что компоненты изображения обработаны. Обычно Y взято для каждый пикселя, и Cb, Cr взяты для блока пикселей 2x2. Но есть некоторые JPG-файлы, в которых Cb, Cr взяты для каждого пикселя, или некоторые JPG-файлы где Cb, Cr взяты для каждых 2 пикселей (горизонтальная дискретизация на 2 пикселях, и вертикальная дискретизация на каждом пикселе) Показатели дискретизации для компонента образа в файле JPG определяются в отношении (относительном) в самом высшем показателе дискретизации.

Вот показатели дискретизации для наиболее обычного примера: Y взято на каждый пиксель, и Cb, Cr взяты для блока пикселей 2x2 (JFIF спецификация дает формулу для показателей дискретизации, которая я думаю, что работает только когда максимальный показатель дискретизации для каждого измерения X или Y <=2) JPEG стандарт не определяет показатели дискретизации.

```
Вы видите, что Y будет иметь самую высшую частоту дискретизации:
               Горизонтальный показатель дискретизации = 2 = HY
               Вертикальный показатель дискретизации   = 2 = VY
     Для Cb,   Горизонтальный показатель дискретизации = 1 = HCb
               Вертикальный показатель дискретизации   = 1 = VCb
     Для Cr,   Горизонтальный показатель дискретизации = 1 = HCr
               Вертикальный показатель дискретизации   = 1 = VCr
```

Действительно эта форма определения показателей дискретизации вполне полезна. Вектор 64 коэффициентов для компонента изображения, кодированных Хаффманом, назван
     DU = Единица Данных (стандартная терминология JPEG)

```
В файле JPG, порядок кодировки Единиц Данных:
     1) закодируйте Единицу Данных для первого компонента изображения:
           для (счетчик_y=1; счетчик_y<=VY; счетчик_y++)
              для (счетчик_x=1; счетчик_x<=HY; счетчик_x++)
                 { кодировать Единицу Данных для Y }
     2) закодируйте Единицу Данных для второго компонента изображения:
           для (счетчик_y=1; счетчик_y<=VCb ; счетчик_y++)
              для (счетчик_x=1; счетчик_x<=HCb; счетчик_x++)
                 { кодировать Единицу Данных для Cb }
     3) наконец, для третьего компонента, аналогично:
           для (счетчик_y=1; счетчик_y<=VCr; счетчик_y++)
              для (счетчик_x=1; счетчик_x<=HCr; счетчик_x++)
                 { кодировать Единицу Данных для Cr }
```

Для примера я дал Вам (HY=2, VY=2; HCb=VCb =1, HCr, VCr=1)
здесь это – цифра:

```
        YDU YDU   CbDU   CrDU
        YDU YDU
```

(YDU - Единица Данных (DU) для Y, и аналогично CbDU DU для Cb, CrDU = DU для Cr) Эта обычная комбинация показателей дискретизации обозначается 2:1:1 как для вертикальных, так и для горизонтальных показателей дискретизации.
И, конечно, в файле JPG порядок кодирования будет:

```
      YDU,YDU,YDU,YDU,CbDU,CrDU
```

Вы знаете, что DU (64 коэффициента) определяют блок величин 8x8, таким образом, здесь мы определили кодирующий порядок для блока пикселей изображения 16x16 (Пиксель изображения (Y,Cb,Cr)): Четыре блока 8x8 величин Y (4 YDUs), один блок 8x8 величин Cb (1 CbDU) и один блок 8x8 величин Cr (1 CrDU)

(Hmax = Максимальный горизонтальный показатель дискретизации, Vmax = максимальный вертикальный показатель дискретизации) Впоследствии для этого примера показателей дискретизации (Hmax = 2, Vmax=2), шифратор должен обрабатывать ОТДЕЛЬНО каждый 16x16 = блок пикселей образа (Hmax*8 x Vmax*8) в упомянутом порядке.

Этот блок пикселей изображения размерами (Hmax*8,Vmax*8) называется, в стандартной терминологии JPG, MCU = Минимальная Кодированная Единица.
Для предшествующего примера: MCU = YDU,YDU,YDU,YDU,CbDU,CrDU

Другой пример показателей дискретизации:

```
      HY =1, VY =1
      HCb=1, VCb=1
      HCr=1, VCr=1
```

Цифра/порядок: YDU CbDU CrDU
Вы видите, что здесь определены пиксель образа блока 8x8 (MCU) с 3 8x8 блоками:
      один для Y, один для Cb и один для Cr (Совсем нет обработки)
Здесь (Hmax=1, Vmax=1) MCU имеет измерение (8,8), и MCU = YDU,CbDU,CrDU

In the JPG file, the sampling factors for every image component are defined after the marker SOF0 = Start Of Frame 0 = FFC0

Для черно-белых JPG-файлов Вы не должны побеспокоиться о порядке кодировки единиц данных в MCU. Для этих JPG, MCU = 1 Единица Данных (MCU = YDU)

В файле JPG, показатели дискретизации для каждого компонента образа определены после маркера SOF0 = Начало Кадра 0 = FFC0

Краткая схема декодирования файла JPG

Дешифратор читает из файла JPG показатели дискретизации, он обнаруживает размеры MCU (Hmax*8,Vmax*8) => сколько MCU - в целом изображении, затем декодирует каждую MCU, присутствующую в исходном изображении (цикл для всех этих блоков, или пока не обнаружится маркер EOI [это должно обнаружиться, когда кончится цикл, в противном случае Вы получите неполное изображение]) - он декодирует MCU, декодируя каждую Единицу Данных в MCU в порядке упомянутом перед этим, и наконец, пишет расшифрованный цветной блок пикселей (Hmax*8 x Vmax*8) в (R,G,B) буфер изображения.

Видео MPEG-1 и JPEG

Интересная часть спецификации MPEG-1 (и вероятно MPEG-2) - то, что она полагается сильно на JPEG спецификацию. Он использует много понятий, представленных здесь. Причина в том, что каждые 15 кадров, или, когда это нужно, есть независимый кадр названный I-кадр (Интра фрейм), который JPEG закодирован. (Между прочим, блок 16x16 пикселей изображения в примере, который я Вам дал, называется, в стандартной терминологии MPEG, макроблок) За исключением алгоритмов для компенсации движения, видео MPEG-1 положились во многом на спецификацию JPG (DCT-преобразование, квантование, и т.п.)

Надеюсь, Вы готовы теперь начать программирование вашего просмотрщика JPG или шифратора.

Об авторе этого документа

Автором этого документа является Cristi Cuturicu, студент Политехнического Университета в Бухаресте (UPB), Отделе Информатики. Вы можете связаться с ним по электронной почте:
cccrx@kermit.cs.pub.ro; cryx@ulise.cs.pub.ro
И если Вы – компания по программированию, требующая программиста, тогда свяжитесь.

Техническое объяснение JPEG/JFIF файлового формата, записанного Oliver Fromme, автором QPEG-просмотрщика

Юридическое ПРИМЕЧАНИЕ: юридические правила, упомянутые в Опровержении вначале этого файла, прилагаются также к следующей информации так никакой Oliver Fromme, я не могу быть признан ответственным за ошибки или дефекты в следующей информации.

Автор следующей информации:

Oliver Fromme
Leibnizstr. 18-61
38678 Clausthal
GERMANY

JPEG/JFIF формат файла:

- заголовок (2 байта): $ff, $d8 (SOI) (эти два идентифицируют JPEG/JFIF файл)
- для JFIF файлов, сегмент APP0 немедленно следует за маркером SOI, смотрите ниже
- любое число "сегментов" (подобно IFF (Image File Format) кускам), смотрите ниже
- хвостовик (2 байта): $ff, $d9 (EOI)

Формат сегмента:

- заголовок (4 байта):

$ff	идентифицирует сегмент
n	тип сегмента (один байт)
sh, sl	размер сегмента, включая эти два байта, но не включая $ff и байт типа. Примечание, не порядок Intel: сначала старший байт, затем младший байт!

- содержимое сегмента, максимум 65533 байта.

Примечания:

- Есть сегменты без параметров (обозначенные '*'), которые не имеет спецификацию размера (и никакого содержания), просто $ff и байт типа.
- Любое число $ff байтов между сегментами легальное и должно быть пропущено.

Типы сегментов:

*TEM	= $01	обычно вызывает ошибки декодирования, может быть проигнорировано
SOF0	= $c0	Начало Кадра (Baseline JPEG), относительно деталей смотрите ниже
SOF1	= $c1	dito
SOF2	= $c2	обычно неподдерживаемый
...		
SOF9	= $c9	для арифметического кодирования, обычно не поддерживается
SOF10	= $ca	обычно неподдерживаемый
...		
DHT	= $c4	Определение Таблицы Хаффмана, относительно деталей смотрите ниже
JPG	= $c8	неопределенный/зарезервированный (вызывает ошибки декодирования)
DAC	= $cc	Определение Арифметической Таблицы, обычно не поддерживается
*RST0	= $d0	RSTn используются для синхронизации, может быть проигнорировано
...		
*RST7	= $d7	
SOI	= $d8	Начало Изображения
EOI	= $d9	Конец Изображения
SOS	= $da	Начало Сканирования, относительно деталей смотрите ниже
DQT	= $db	Определение Таблицы Квантования, относительно деталей смотрите ниже
DNL	= $dc	обычно неподдерживаемый, игнорировать
DRI	= $dd	Определение Интервала Перезапуска, относительно деталей смотрите ниже
APP0	= $e0	JFIF маркеры сегментов приложения, относительно деталей смотрите ниже
APP15	= $ef	игнорировать
COM	= $fe	Комментарий, относительно деталей смотрите ниже

Все другие типы сегментов резервируются и должны быть проигнорированы (пропущены).

SOF0: Начало Кадра 0:

- $ff, $c0 (SOF0)
- длина (старший байт, младший байт), 8+components*3
- точность данных (1 байт) в битах/единицу, обычно 8 (12 и 16 не поддерживается большинством программного обеспечения)
- высота изображения (2 байта, High-Low), должно быть >0, если DNL не поддерживается
- ширина изображения (2 байта, High-Low), должно быть >0, если DNL не поддерживается
- количество компонентов (1 байт), обычно 1 = черно-белый, 3 = цвет YCbCr или YIQ, 4 = цвет CMYK)
- для каждого компонента: 3 байта
  ü идентификатор компонента (1 = Y, 2 = Cb, 3 = Cr, 4 = I, 5 = Q)
  ü показатели дискретизации (бит 0-3 вертикальной, 4-7 горизонтальной)
  ü номер таблицы квантования

Замечания:

ü JFIF использует также 1 компонент (Y, черно-белое изображение) или 3 компонента (YCbCr, иногда называемое YUV, цветное).

APP0: JFIF маркер сегмента:

- $ff, $e0 (APP0)
- длина (старший байт, младший байт), должна быть >= 16
- 'JFIF'#0 ($4a, $46, $49, $46, $00), идентифицирует JFIF
- номер общей версии, должен быть 1 (в противном случае ошибка)
- номер подверсии, должен быть 0..2 (в противном случае попытка декодирования по-другому)
- единица для x/y разрешения:
  0 = нет никаких единиц, x/y-отношение определяет соотношение аспекта
  1 = x/y-разрешение в точках/дюйм
  2 = x/y-разрешение в точках/сантиметр
- x-разрешение (старший байт, младший байт), должно быть <> 0
- y-разрешение (старший байт, младший байт), должно быть <> 0
- ширина предварительного просмотра (1 байт)
- высота предварительного просмотра (1 байт)
- n байт для предварительного просмотра (RGB 24 бит) n = ширина*высота*3

Замечания:

- Если нет 'JFIF'#0, или длина <16, тогда это - вероятно не JFIF сегмент и должен быть проигнорирован.
- Нормально единицы=0, x-разрешение=1, y-разрешение=1, означает, что отношение числа точек - 1:1 (равномерно масштабированное).
- JFIF файлы с предварительным просмотром встречаются очень редко, предварительный просмотр может обычно быть проигнорирован. Если нет предварительного просмотра, тогда ширина=0 и высота=0.
- Если длина не соответствует размеру предварительного просмотра, можно выдать предупреждение, затем продолжить декодирование.

DRI: Определение Интервала Перезапуска:

- $ff, $dd (DRI)
- длина (старший байт, младший байт), должна быть = 4
- интервал перезапуска (старший байт, младший байт) в единицах блоков MCU, означает, что каждые n блоков MCU может быть найден маркер RSTn.
- Первый маркер будет RST0, затем RST1 и т.п., после RST7 повторится с RST0.

DQT: Определение Таблицы Квантования:

- $ff, $db (DQT)
- длина (старший байт, младший байт)
- информация о таблице квантования (1 байт):
  биты 0..3: номер таблицы (0..3, в противном случае ошибка)
  биты 4..7: точность таблицы, 0 = 8 бит, в противном случае 16 бит
- n байт таблицы n = 64*(точность+1)

Замечания:

- Единственный сегмент DQT может содержать несколько таблиц, каждая со своим собственным байтом информации.
- Для точности=1 (16 бит), порядок является High-Low (Старшим-Младшим) для каждого из 64 целых значений.

DAC: Определение Арифметической Таблицы:

Текущее программное обеспечение не поддерживает арифметическое кодирование по законным причинам. JPEG файлы, использующие арифметическое кодирование не могут быть обработаны.

DHT: Определение Таблицы Хаффмана:

- $ff, $c4 (DHT)
- длина (старший байт, младший байт)
- информация о таблице Хаффмана (1 байт):
  биты 0..3: число таблиц (0..3, в противном случае ошибка)
  бит 4 : тип таблицы, 0 = таблица DC, 1 = таблица AC
  биты 5..7: не используются, должны быть 0
- 16 байт: количество символов с кодами длины 1..16, сумма этих байтов является общим числом кодов, которые должны быть <= 256
- n байт: таблица, содержащая символы в порядке повышения длины кода (n = общее число кодов)

Замечания:

- Единственный сегмент DHT может содержать несколько таблиц, каждая со своим собственным байтом информации.

COM: Комментарий:

- $ff, $fe (COM)
- длина (старший байт, младший байт) комментария = L+2
- Комментарий = байтовый поток длиной L

SOS: Начало Сканирования:

- $ff, $da (SOS)
- длина (старший байт, младший байт), должна быть 6+2*(количество компонентов сканирования)
- количество компонентов сканирования (1 байт), должно быть >=1 и <=4 (в противном случае ошибка), обычно 1 или 3
- для каждого компонента: 2 байта
  ü идентификатор компонента (1 = Y, 2 = Cb, 3 = Cr, 4 = I, 5 = Q), смотри SOF0
  ü использование таблицы Хаффмана:
  биты 0..3: таблица AC (0..3)
  биты 4..7: таблица DC (0..3)
- 3 байта, которые должны игнорироваться

Замечания:

- Данные об изображении (данные сканирования) следуют сразу за сегментом SOS.