Embedded Systems Course

JPEG Algorithm Development

Eng. Dante Casalena

Outline

- Introduction to JPEG
- Digital Camera Example Development

JPEG

- JPEG coding standard (1988): DCT (discrete cosine transform) based transform coding to compress bit-map images.

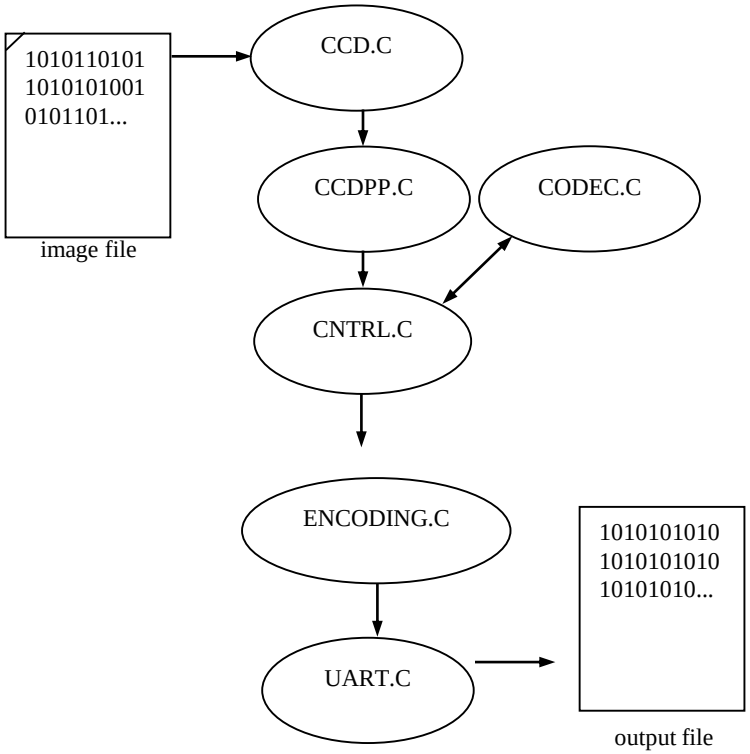
- JPEG compression technique
  - “Lossy” compression: Perfect data recovery not possible

Three Steps

- Transformation
  - 8× 8 DCT
- Quantization
  - uniform quantization
  - DC/AC coefficients
- Coding
  - Zigzag scan
  - Run length/Huffman coding

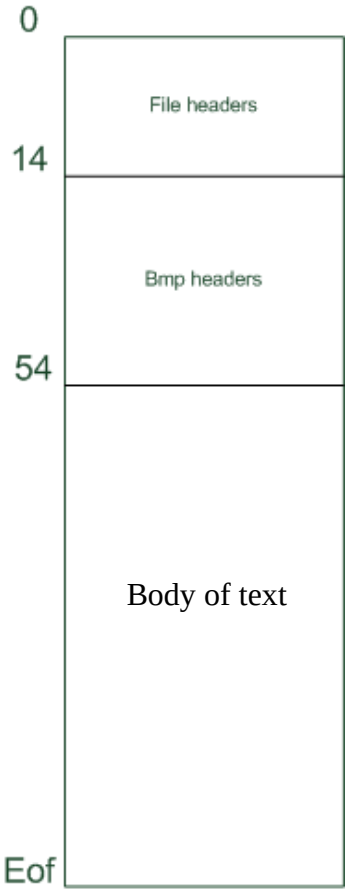
Digital Cam

- Digital Cam model



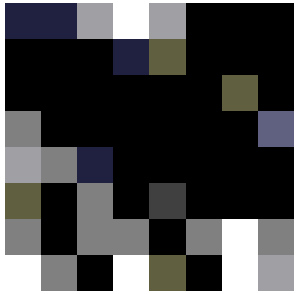
File BMP

- BMP format
  - File headers (offset 0 - 14)
  - Bmp headers (offset 14 - 54)
  - Body of text (offset 54 - eof)



Module CCD: *CcdBmpInfo()*

- File BMP 8x8 pixels, grey scale



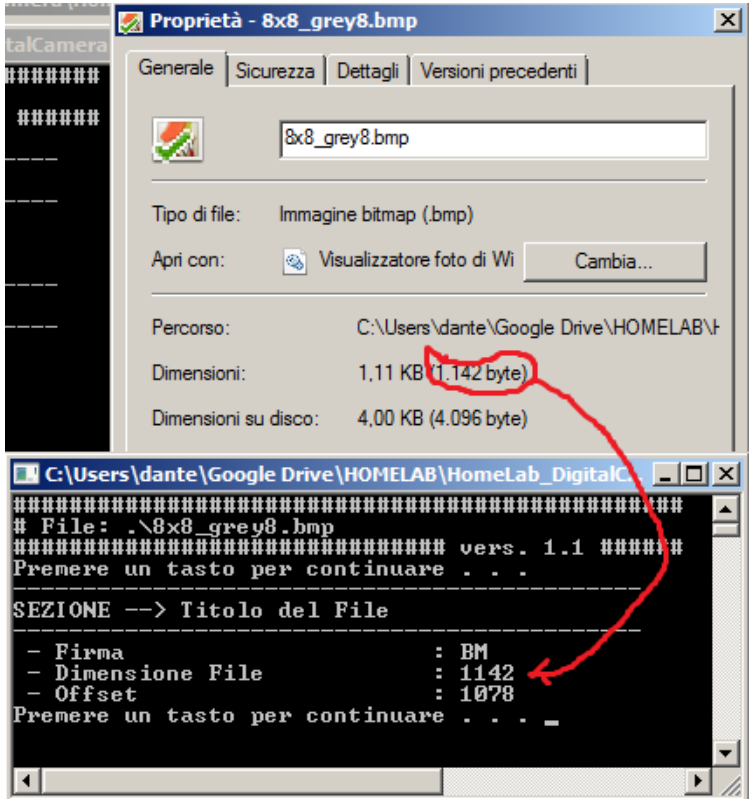
	0001	0203	0405	0607	0809	0A0B	0C0D	0E0F	1011	1213	1415	1617	1819	1A1B	1C1D	1E1F	2021	2223
000	424D	7604	0000	0000	0000	3604	0000	2800	0000	0800	0000	0800	0000	0100	0800	0000	0000	4000
024	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	8000	0080	0000	0080	8000	8000
048	0000	8000	8000	8080	0000	C0C0	C000	C0DC	C000	F0CA	A600	0020	4000	0020	6000	0020	8000	0020
06C	A000	0020	C000	0020	E000	0040	0000	0040	2000	0040	4000	0040	6000	0040	8000	0040	A000	0040
090	C000	0040	E000	0060	0000	0060	2000	0060	4000	0060	6000	0060	8000	0060	A000	0060	C000	0060
0B4	E000	0080	0000	0080	2000	0080	4000	0080	6000	0080	8000	0080	A000	0080	C000	0080	E000	00A0
0D8	0000	00A0	2000	00A0	4000	00A0	6000	00A0	8000	00A0	A000	00A0	C000	00A0	E000	00C0	0000	00C0
0FC	2000	00C0	4000	00C0	6000	00C0	8000	00C0	A000	00C0	C000	00C0	E000	00E0	0000	00E0	2000	00E0
120	4000	00E0	6000	00E0	8000	00E0	A000	00E0	C000	00E0	E000	4000	0000	4000	2000	4000	4000	4000
144	6000	4000	8000	4000	A000	4000	C000	4000	E000	4020	0000	4020	2000	4020	4000	4020	6000	4020
168	8000	4020	A000	4020	C000	4020	E000	4040	0000	4040	2000	4040	4000	4040	6000	4040	8000	4040
18C	A000	4040	C000	4040	E000	4060	0000	4060	2000	4060	4000	4060	6000	4060	8000	4060	A000	4060
1B0	C000	4060	E000	4080	0000	4080	2000	4080	4000	4080	6000	4080	8000	4080	A000	4080	C000	4080
1D4	E000	40A0	0000	40A0	2000	40A0	4000	40A0	6000	40A0	8000	40A0	A000	40A0	C000	40A0	E000	40C0
1F8	0000	40C0	2000	40C0	4000	40C0	6000	40C0	8000	40C0	A000	40C0	C000	40C0	E000	40E0	0000	40E0
21C	2000	40E0	4000	40E0	6000	40E0	8000	40E0	A000	40E0	C000	40E0	E000	8000	0000	8000	2000	8000
240	4000	8000	6000	8000	8000	8000	A000	8000	C000	8000	E000	8020	0000	8020	2000	8020	4000	8020
264	6000	8020	8000	8020	A000	8020	C000	8020	E000	8040	0000	8040	2000	8040	4000	8040	6000	8040
288	8000	8040	A000	8040	C000	8040	E000	8060	0000	8060	2000	8060	4000	8060	6000	8060	8000	8060
2AC	A000	8060	C000	8060	E000	8080	0000	8080	2000	8080	4000	8080	6000	8080	8000	8080	A000	8080
2D0	C000	8080	E000	80A0	0000	80A0	2000	80A0	4000	80A0	6000	80A0	8000	80A0	A000	80A0	C000	80A0
2F4	E000	80C0	0000	80C0	2000	80C0	4000	80C0	6000	80C0	8000	80C0	A000	80C0	C000	80C0	E000	80E0
318	0000	80E0	2000	80E0	4000	80E0	6000	80E0	8000	80E0	A000	80E0	C000	80E0	E000	C000	0000	C000
33C	2000	C000	4000	C000	6000	C000	8000	C000	A000	C000	C000	C000	E000	C020	0000	C020	2000	C020
360	4000	C020	6000	C020	8000	C020	A000	C020	C000	C020	E000	C040	0000	C040	2000	C040	4000	C040
384	6000	C040	8000	C040	A000	C040	C000	C040	E000	C060	0000	C060	2000	C060	4000	C060	6000	C060
3A8	8000	C060	A000	C060	C000	C060	E000	C080	0000	C080	2000	C080	4000	C080	6000	C080	8000	C080
3CC	A000	C080	C000	C080	E000	C0A0	0000	C0A0	2000	C0A0	4000	C0A0	6000	C0A0	8000	C0A0	A000	C0A0
3F0	C000	C0A0	E000	C0C0	0000	C0C0	2000	C0C0	4000	C0C0	6000	C0C0	8000	C0C0	A000	F0FB	FF00	A4A0
414	A000	8080	8000	0000	FF00	00FF	0000	00FF	FF00	FF00	0000	FF00	FF00	FFFF	0000	FFFF	FF00	FFA4
438	00FF	5B00	FFF7	A400	A4A4	00A4	FFA4	5B00	A400	5200	0000	F7A4	4900	0000	0000	A400	0000	0000
45C	009B	0000	0000	0000	5B00	0000	0049	5B00	0000	4949	F7FF	F700	0000					

Module CCD: *CcdBmpInfo()*

- Read File Headers

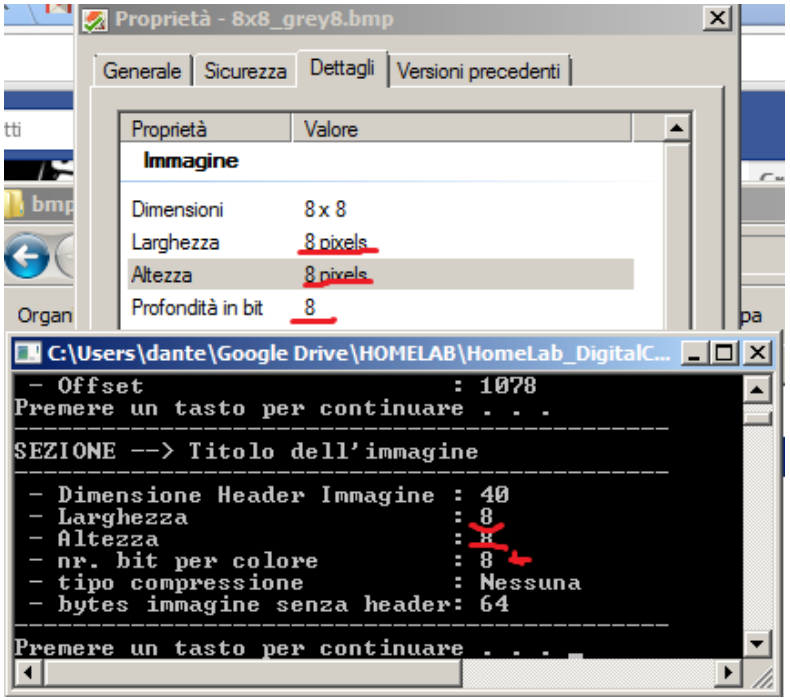
```
void CcdBmpInfo(void) {

    int n, j;
    int int_var;
    short short_var;
    float float_var;
    char char_2[2];
  static FILE *fpo;
   fpo=imageFileHandle;
    printf("-------------------------------------------\n");
    printf("SEZIONE --> Titolo del File\n");
    printf("-------------------------------------------\n");
    fseek(fpo,0,0);
    n = fread(char_2,2, 1, fpo);
    printf(" - Firma                    : %c%c\n",char_2[0],char_2[1]);
    fseek(fpo,2,0);
    fread(&int_var, sizeof(int), 1, fpo);
    printf(" - Dimensione File          : %d\n",int_var);
    fseek(fpo,10,0);
    fread(&int_var, sizeof(int),1 , fpo);
    printf(" - Offset                   : %d\n",int_var);
    iOffSet=int_var;
```



Module CCD: *CcdBmpInfo()*

- Read File Headers



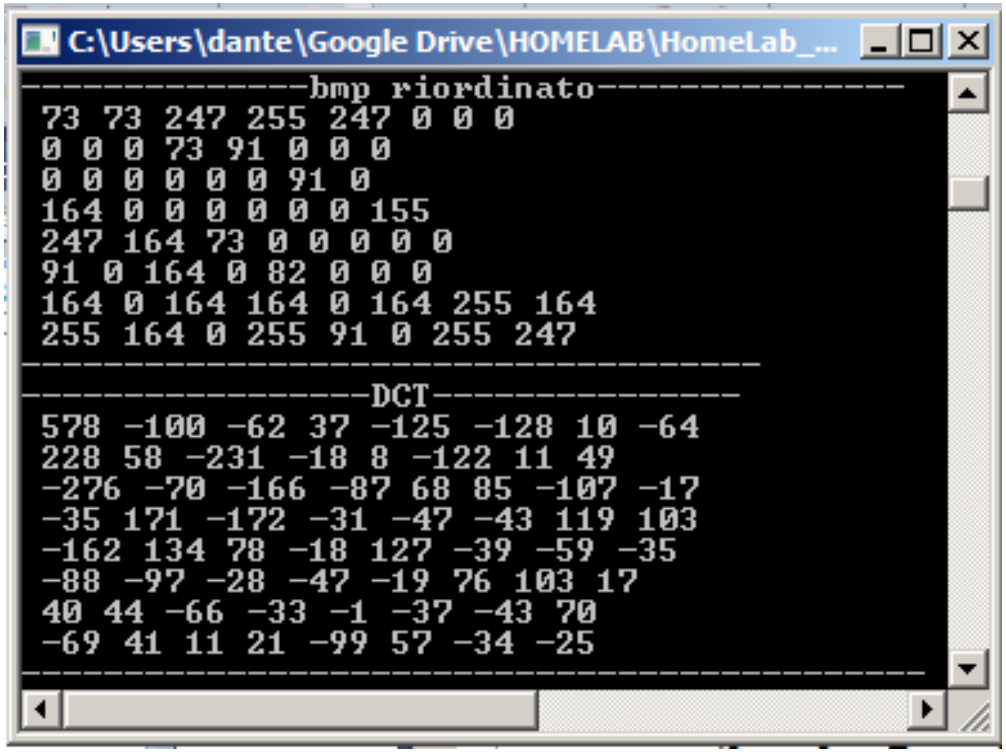

```
printf("-----------------------------------------------\n");
printf("SEZIONE --> Titolo dell'immagine\n");
printf("-----------------------------------------------\n");
fseek(fpo,14,0);
n = fread(&int_var,sizeof(int), 1, fpo);
printf(" - Dimensione Header Immagine : %d\n",int_var);
headers_size=int_var+14;
n = fread(&int_var,sizeof(int), 1, fpo);
printf(" - Larghezza                  : %d\n",int_var);
i_w=int_var;
n = fread(&int_var,sizeof(int), 1, fpo);
printf(" - Altezza                    : %d\n",int_var);
i_h=int_var;
fseek(fpo,28,0);
n = fread(&short_var,sizeof(short), 1, fpo);
printf(" - nr. bit per colore         : %hd\n",short_var);
nBit=short_var;
n = fread(&int_var,sizeof(int), 1, fpo);
if (int_var==0)
   printf(" - tipo compressione          : Nessuna\n");
else if (int_var==1)
   printf(" - tipo compressione          : RLE 8 Bit\n");
else if (int_var==2)
   printf(" - tipo compressione          : RLE 4 Bit\n");
else if (int_var==3)
   printf(" - tipo compressione          : Bitfields\n");
else
   printf(" - tipo compressione          : %d\n",int_var);
fread(&int_var, 1, sizeof(int), fpo);
printf(" - bytes immagine senza header: %d\n",int_var);
printf("-----------------------------------------------\n");
```

DCT step

- Transforms original 8 x 8 block into a cosine-frequency domain
  - Upper-left corner values represent more of the essence of the image
  - Lower-right corner values represent finer details
    - Can reduce precision of these values and retain reasonable image quality
- FDCT (Forward DCT) formula
  - C(h) = if (h == 0) then 1/sqrt(2) else 1.0
    - Auxiliary function used in main function F(u,v)
  - $F(u,v) = \frac{1}{4} \times C(u) \times C(v) \Sigma_{x=0..7} \Sigma_{y=0..7} D_{xy} \times \cos(\pi(2u + 1)u/16) \times \cos(\pi(2y + 1)v/16)$
    - Gives encoded pixel at row u, column v
    - D_{xy} is original pixel value at row x, column y
- IDCT (Inverse DCT)
  - Reverses process to obtain original block (not needed for this design)

DCT step

- Call the function **CntrlCompressImage()**



Quantization step

- Achieve high compression ratio by reducing image quality
  - Reduce bit precision of encoded data
    - Fewer bits needed for encoding
    - One way is to divide all values by a factor of 2
      - Simple right shifts can do this
  - Dequantization would reverse process for decompression

1150	39	-43	-10	26	-83	11	41
-81	-3	115	-73	-6	-2	22	-5
14	-11	1	-42	26	-3	17	-38
2	-61	-13	-12	36	-23	-18	5
44	13	37	-4	10	-21	7	-8
36	-11	-9	-4	20	-28	-21	14
-19	-7	21	-6	3	3	12	-21
-5	-13	-11	-17	-4	-1	7	-4

After being decoded using DCT

Divide each cell's value by 8

144	5	-5	-1	3	-10	1	5
-10	0	14	-9	-1	0	3	-1
2	-1	0	-5	3	0	2	-5
0	-8	-2	-2	5	-3	-2	1
6	2	5	-1	1	-3	1	-1
5	-1	-1	-1	3	-4	-3	2
-2	-1	3	-1	0	0	2	-3
-1	-2	-1	-2	-1	0	1	-1

After quantization

Quantization step

- How to choose the quantization matrix?
  - Luminance quantization matrix
  - Chrominance quantization matrix
  - Compression level

Reference:
http://www.impulseadventure.com/photo/jpeg-quantization.html

Quantization step

- Gimp luminance quantization matrix

```
const unsigned short QuantShiftTable[8][8] = {
 {3, 2,  2,  3,  5,  8,  10, 12},
 {2, 2,  3,  4,  5,  12, 12, 11} ,
 {3, 3,  3,  5,  8,  11, 14, 11},
 {3, 3,  4,  6,  10, 17, 16, 12},
 {4, 4,  7,  11, 14, 22, 21, 15},
 {5, 7, 11,  13, 16, 21, 23, 18},
 {10,13,16,  17, 21, 24, 24, 20},
 {14,18, 19, 20, 22, 20, 21, 20}
};
```

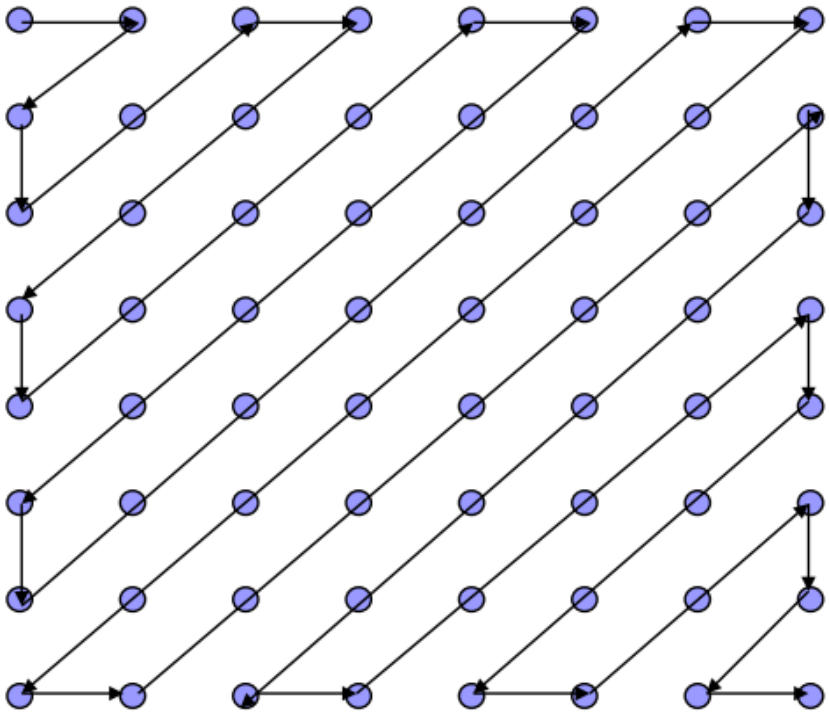
```
FCD
-444 -100 -62 37 -125 -128 10 -64
228 58 -231 -18 8 -122 11 49
-276 -70 -166 -87 68 85 -107 -17
-35 171 -172 -31 -47 -43 119 103
-162 134 78 -18 127 -39 -59 -35
-88 -97 -28 -47 -19 76 103 17
40 44 -66 -33 -1 -37 -43 70
-69 41 11 21 -99 57 -34 -25

quantization
-148 -50 -31 12 -25 -16 1 -5
114 29 -77 -4 1 -10 0 4
-92 -23 -55 -17 8 7 -7 -1
-11 57 -43 -5 -4 -2 7 8
-40 33 11 -1 9 -1 -2 -2
-17 -13 -2 -3 -1 3 4 0
4 3 -4 -1 0 -1 -1 3
-4 2 0 1 -4 2 -1 -1
```

Coding

- There are two steps:
  - ZigZag scan and reorder
  - Huffman coding
    - *run length encoding* technique
    - Luminace and Chrominance Huffman tables

ZigZag step



Zigzag Scan

$$\begin{bmatrix} 20 & 5 & -3 & 1 & 3 & -2 & 1 & 0 \\ -3 & -2 & 1 & 2 & 1 & 0 & 0 & 0 \\ -1 & -1 & 1 & 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Zigzag Scan

(20,5,-3,-1,-2,-3,1,1,-1,-1,
0,0,1,2,3,-2,1,1,0,0,0,0,0,
0,1,1,0,1,EOB)

End Of the Block:
All following coefficients are zero

ZigZag step

```
quantization
 -148 -50 -31 12 -25 -16 1 -5
 114 29 -77 -4 1 -10 0 4
 -92 -23 -55 -17 8 7 -7 -1
 -11 57 -43 -5 -4 -2 7 8
 -40 33 11 -1 9 -1 -2 -2
 -17 -13 -2 -3 -1 3 4 0
 4 3 -4 -1 0 -1 -1 3
 -4 2 0 1 -4 2 -1 -1
Order zig zag
 -148 -50 114 -92 29 -31 12 -77 -23 -11 -40 57 -55 -4 -25 -16 1 -17 -43 33 -1
7 4 -13 11 -5 8 -10 1 -5 0 7 -4 -1 -2 3 -4 2 -4 -3 9 -2 -7 4 -1 7 -1 -1 -1 0
1 0 3 -2 8 -2 4 -1 -4 2 -1 0 3 -1 -1
```

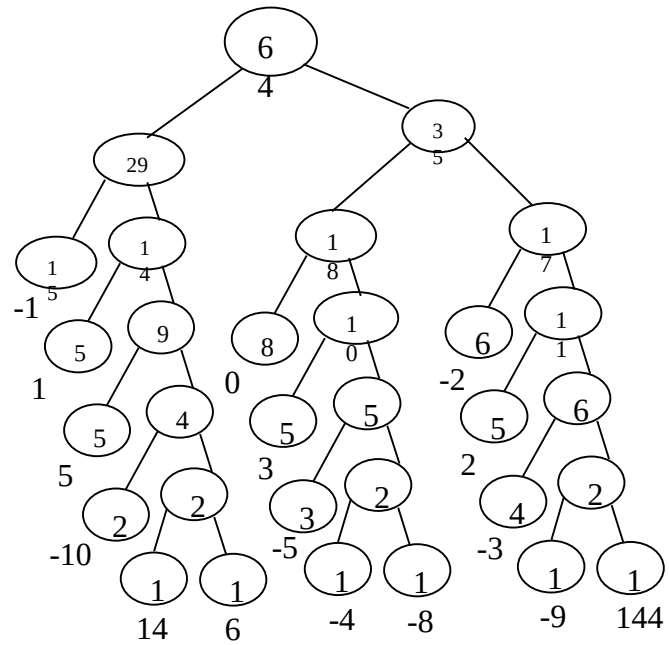
Huffman encoding example

- Pixel frequencies on left
  - Pixel value –1 occurs 15 times
  - Pixel value 14 occurs 1 time
- Build Huffman tree from bottom up
  - Create one leaf node for each pixel value and assign frequency as node's value
  - Create an internal node by joining any two nodes whose sum is a minimal value
    - This sum is internal nodes value
  - Repeat until complete binary tree
- Traverse tree from root to leaf to obtain binary code for leaf's pixel value
  - Append 0 for left traversal, 1 for right traversal
- Huffman encoding is reversible
  - No code is a prefix of another code

Pixel frequencies

-1	15x
0	8x
-2	6x
1	5x
2	5x
3	5x
5	5x
-3	4x
-5	3x
-10	2x
144	1x
-9	1x
-8	1x
-4	1x
6	1x
14	1x

Huffman tree



Huffman codes

-1	00
0	100
-2	110
1	010
2	1110
3	1010
5	0110
-3	11110
-5	10110
-10	01110
144	111111
-9	111110
-8	101111
-4	101110
6	011111
14	011110

Huffman Table

- … but actually

Table K.3 – Table for luminance DC coefficient differences

Category	Code length	Code word
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Table K.4 – Table for chrominance DC coefficient differences

Category	Code length	Code word
0	2	00
1	2	01
2	2	10
3	3	110
4	4	1110
5	5	11110
6	6	111110
7	7	1111110
8	8	11111110
9	9	111111110
10	10	1111111110
11	11	11111111110

Huffman Table

- And…

Table K.5 – Table for luminance AC coefficients (sheet 1 of 4)

Run/Size	Code length	Code word
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
2/1	5	11100
2/2	8	11111001
2/3	10	1111110111
2/4	12	111111110100
2/5	16	1111111110001001
2/6	16	1111111110001010
2/7	16	1111111110001011
2/8	16	1111111110001100
2/9	16	1111111110001101
2/A	16	1111111110001110
3/1	6	111010
3/2	9	111110111
3/3	12	111111110101
3/4	16	1111111110001111
3/5	16	1111111110010000
3/6	16	1111111110010001
3/7	16	1111111110010010
3/8	16	1111111110010011
3/9	16	1111111110010100
3/A	16	1111111110010101

Table K.6 – Table for chrominance AC coefficients (sheet 1 of 4)

Run/Size	Code length	Code word
0/0 (EOB)	2	00
0/1	2	01
0/2	3	100
0/3	4	1010
0/4	5	11000
0/5	5	11001
0/6	6	111000
0/7	7	1111000
0/8	9	111110100
0/9	10	1111110110
0/A	12	111111110100
1/1	4	1011
1/2	6	111001
1/3	8	11110110
1/4	9	111110101
1/5	11	11111110110
1/6	12	111111110101
1/7	16	1111111110001000
1/8	16	1111111110001001
1/9	16	1111111110001010
1/A	16	1111111110001011
2/1	5	11010
2/2	8	11110111
2/3	10	1111110111
2/4	12	111111110110
2/5	15	111111111000010
2/6	16	1111111110001100
2/7	16	1111111110001101
2/8	16	1111111110001110
2/9	16	1111111110001111
2/A	16	1111111110010000
3/1	5	11011
3/2	8	11111000
3/3	10	1111111000
3/4	12	111111110111
3/5	16	1111111110010001
3/6	16	1111111110010010
3/7	16	1111111110010011
3/8	16	1111111110010100
3/9	16	1111111110010101
3/A	16	1111111110010110
4/1	6	111010

Huffman Table

- To encode what we need is:
  - DC
    - Difference between DC $block_i$ and DC $block_{i-1}$
  - AC
    - Run Length technique

- Example

Huffman Table Example

- Consider two subsequent 8x8 blocks:

  29,-2,1,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0…..

  22,-2,1,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0…..

DC elements (22-29=-7)-> C1(-7)=000

[3,-7]

Number of bits	Range
0	0
1	-1,1
2	-3...-2; 2...,3
3	-7...,-4; 4...7
4	-15...-8; 8...15
5	-31...-16; 16...31
6	-63...-32; 32...63
7	-127...-64; 64...127
8	-255...-128; 128...255
9	-511...-256; 256...511
10	-1023...512; 512...1023
11	-2047...-1024; 1024...2047

Run Length

- 1° step: count of zeros before the number

  29,-2,1,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0…..

(0,-2);(0,1); (0,1); (0,1); (3,1);…….

([0,2],-2);([0,1],1)….. ([3,1],1);…….

Number of bits	Range
0	0
1	-1,1
2	-3...-2; 2...,3
3	-7...,-4; 4...7
4	-15...-8; 8...15
5	-31...-16; 16...31
6	-63...-32; 32...63
7	-127...-64; 64...127
8	-255...-128; 128...255
9	-511...-256; 256...511
10	-1023...512; 512...1023
11	-2047...-1024; 1024...2047

Apply Huffman tables

- …now

[3,000];([0,2],01);([0,1],1)….. ([3,1],1);…….

Table K.3 – Table for luminance DC coefficient differences

Category	Code length	Code word
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

[011,000] DC

Apply Huffman tables

- …now

([0,2],01);([0,1],1)….. ([3,1],1);…….

Run/Size	Code length	Code word
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
2/A	16	1111111110001110
3/1	6	111010
3/2	9	111110111

[01,01]

[111010,1]

Apply Huffman tables

- Result

0110000101…1110101……

Come back to application

```
runlenght
 -148 -50 114 -92 29 -31 12 -77 -23 -11 -40 57 -55 -4 -25 -16 1 -17 -43 33 -1
7 4 -13 11 -5 8 -10 1 -5 0 7 -4 -1 -2 3 -4 2 -4 -3 9 -2 -7 4 -1 7 -1 -1 -1 0
1 0 3 -2 8 -2 4 -1 -4 2 -1 0 3 -1 -1
DC_runlenght_buffer -148 8
AC_runlenght_buffer -50 0 6
AC_runlenght_buffer 114 0 7
AC_runlenght_buffer -92 0 7
AC_runlenght_buffer 29 0 5
AC_runlenght_buffer -31 0 5
AC_runlenght_buffer 12 0 4
AC_runlenght_buffer -77 0 7
AC_runlenght_buffer -23 0 5
AC_runlenght_buffer -11 0 4
AC_runlenght_buffer -40 0 6
AC_runlenght_buffer 57 0 6
AC_runlenght_buffer -55 0 6
AC_runlenght_buffer -4 0 3
AC_runlenght_buffer -25 0 5
AC_runlenght_buffer -16 0 5
AC_runlenght_buffer 1 0 1
AC_runlenght_buffer -17 0 5
AC_runlenght_buffer -43 0 6
AC_runlenght_buffer 33 0 6
AC_runlenght_buffer -17 0 5
AC_runlenght_buffer 4 0 3
AC_runlenght_buffer -13 0 4
AC_runlenght_buffer 11 0 4
AC_runlenght_buffer -5 0 3
AC_runlenght_buffer 8 0 4
AC_runlenght_buffer -10 0 4
AC_runlenght_buffer 1 0 1
AC_runlenght_buffer -5 0 3
AC_runlenght_buffer 7 1 3
AC_runlenght_buffer -4 0 3
AC_runlenght_buffer -1 0 1
AC_runlenght_buffer -2 0 2
AC_runlenght_buffer 3 0 2
AC_runlenght_buffer -4 0 3
AC_runlenght_buffer 2 0 2
AC_runlenght_buffer -4 0 3
AC_runlenght_buffer -3 0 2
AC_runlenght_buffer 9 0 4
AC_runlenght_buffer -2 0 2
AC_runlenght_buffer -7 0 3
AC_runlenght_buffer 4 0 3
AC_runlenght_buffer -1 0 1
AC_runlenght_buffer 7 0 3
AC_runlenght_buffer -1 0 1
AC_runlenght_buffer -1 0 1
AC_runlenght_buffer -1 0 1
AC_runlenght_buffer 1 1 1
AC_runlenght_buffer 3 1 2
AC_runlenght_buffer -2 0 2
AC_runlenght_buffer 8 0 4
AC_runlenght_buffer -2 0 2
```

```
-148 8 111110
DC binary 01101011
0 6 1111000
AC binary -50  001101
0 7 11111000
AC binary 114  1110010
0 7 11111000
AC binary -92  0100011
0 5 11010
AC binary 29  11101
0 5 11010
AC binary -31  00000
0 4 1011
AC binary 12  1100
0 7 11111000
AC binary -77  0110010
0 5 11010
AC binary -23  01000
0 4 1011
AC binary -11  0100
0 6 1111000
AC binary -40  010111
0 6 1111000
AC binary 57  111001
0 6 1111000
AC binary -55  001000
0 3 100
AC binary -4  011
0 5 11010
AC binary -25  00110
0 5 11010
AC binary -16  01111
0 1 00
AC binary 1  1
0 5 11010
AC binary -17  01110
0 6 1111000
AC binary -43  010100
0 6 1111000
AC binary 33  100001
0 5 11010
AC binary -17  01110
0 3 100
AC binary 4  100
0 4 1011
AC binary -13  0010
0 4 1011
AC binary 11  1011
0 3 100
AC binary -5  010
0 4 1011
AC binary 8  1000
0 4 1011
AC binary -10  0101
0 1 00
AC binary 1  1
0 3 100
AC binary -5  010
1 3 1111001
```

End coding

```
completo byte stuffing
1111100110101111110000011011111100011100101111100001000111101011101110100000
1011110011111000011001011010010001011010011111000010111111100011100111111000001
0001000111101000110110100111110011101001110111110000101001111000100001110100111
0100100101100101011101110001010111000101101010011000101111001111100011000010 1
0111100011011010001101001011110010101100000100100000100111000000000011001110111
1010110111000010110010000010001101100001101111000000101011111111
 count_byte 448

 completo byte stuffing HEX
F9AFC1BF1CBE11EBBA05E7C32D22D3C2FE39F0447A369E74EF0A78874E92CAEE2B8B53179F185
78DA34B95824138033BD6E16411B0DE057F
 count_hex HEX 112
```

Jpeg File

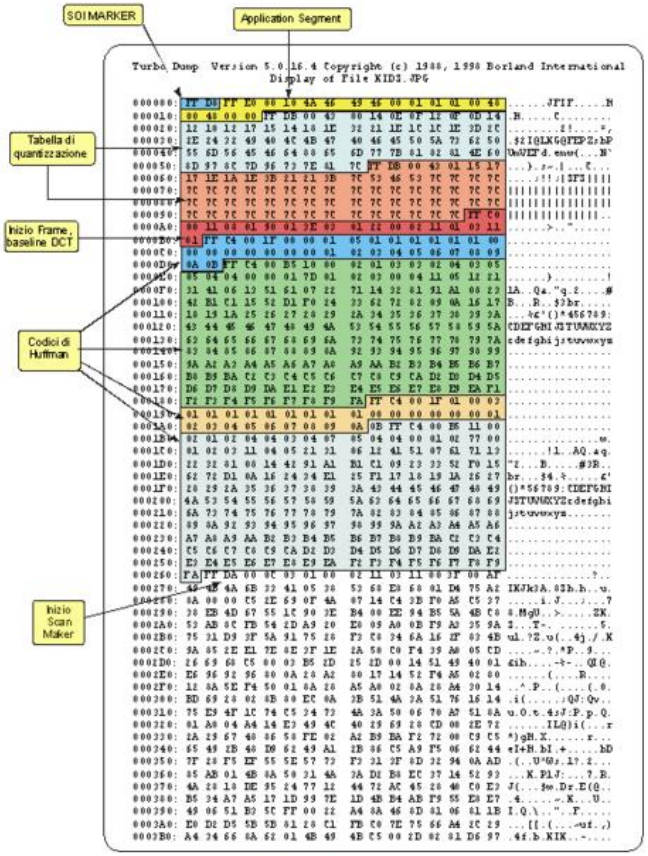


Fig. 15 - File JPEG.

Thanks for the attention