Elaborazione di Segnali e Immagini (ESI)

AA 2002-2003

Paola Bonetto
email: bonetto@disi.unige.it
Stanza: #110
Tel: 010 353 6643

Programma

- **Colore** e spazi di colore (CIE, RGB, HSV, gray, …)
- Formati di **immagini** (jpg, gif, png, pcx, …)
  - compressione
  - image processing & filtering
- Formati **audio** (mp3, wav, …)
- Formati **video** (AVI, mpeg, …)
  - compressione

Immagini Raster

- Bitmap, array di valori: per ogni pixel si ha un valore per ogni componente di colore
- Risoluzione-dipendente
- Esempi: GIF, JPEG, PNG, TIFF, PSD

Immagini Vettoriali (vector images)

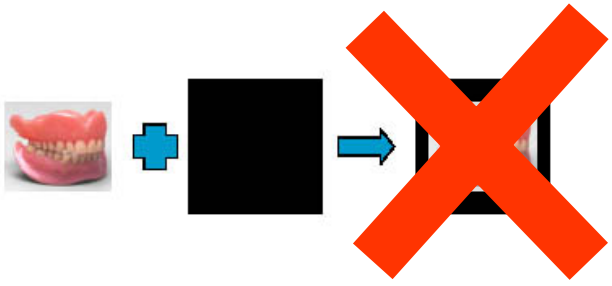
- Descrizione testuale o binaria di una serie di operazioni di disegno
  - MoveTo, LineTo, CurveTo, DrawText
  - SetFont, SetStroke, SetFillColor
- Risoluzione-indipendente
- Esempi: EPS, SVG, WMF, EMF

Immagini Raster: composizione

- Sovrapporre un'immagine all'altra, ottenendo un'immagine "combinata"

Immagini Raster: composizione

- Ma cosa accade per forme irregolari??



Immagini Raster: composizione

- Ma cosa accade per forme irregolari??

Immagini Raster: Composizione e canale/maschera alpha

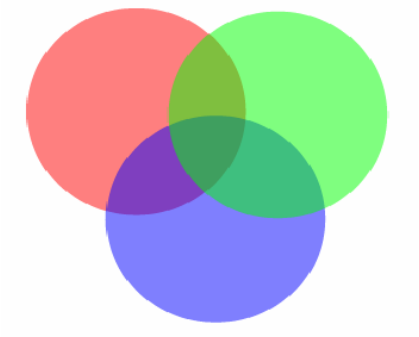
- Un modo per rappresentare trasparenza/ opacità in immagini raster
- Utilizzato **in aggiunta** a un colorspace esistente (es: RGB -> RGBA)
- Determina quali bit "traspaiono"

Immagini Raster: Composizione e canale/maschera alpha

- Esempio in Photoshop

Immagini Raster: alpha blending

- Se alpha è un qualche valore **fra** 0 e 1, i colori vengono "mescolati" come acquarelli



Immagini Raster: Formati Web

- GIF
  - Spazio di colori indicizzato (<= 256 colori)
  - Immagini multiple (frames di animazioni)
- JPG
  - Immagini fotografiche:
    - mln di colori
    - Artefatti su testo e vettori, dovuti a compressione
  - Niente masking / trasparenza

Immagini Raster: Formati Web

- PNG
  - Completo supporto colore (mln di colori)
  - B&W, toni di grigio, RGB
  - Completo supporto trasparenza /alpha

Immagini vettoriali: Formati Web

- SVG – standard W3C basato su XML
  - RGB
  - Trasparenza / alpha
  - Permette referenze a immagini raster
  - Effetti / filtraggi
- SWF (Flash) – formato binario, proprietario
  - RGB
  - Trasparenza / alpha
  - Permette referenze a immagini raster

Formati piattaforma-dipendenti

- BMP (Windows)
  - Raster
  - Supporto completo di colore (mln di colori)
  - B&W, toni di grigio, RGB
- WMF / EMF (Windows Metafile Format)
  - Combinato raster-vettoriale
  - mln di colori
  - RGB

Formati piattaforma-dipendenti

- PICT (Macintosh)
  - Combinato raster-vettoriale
  - Mln di colori
  - B&W e RGB
  - Trasparenza / alpha

Formati di pubblicazione

- TIFF
  - Raster
  - Mln di colori
  - B&W, toni di grigio, RGB, CMYK
  - Trasparenza / alpha
  - Immagini multiple (“pagine”)

Formati di pubblicazione

- PSD
  - Raster
  - Mln di colori
  - B&W, toni di grigio, RGB, CMYK, Lab
  - Trasparenza / alpha
  - Ulteriori canali
  - Immagini multiple (“layer”)
  - Effetti sui layer e di testo

Formati di pubblicazione

- EPS (encapsulated postscript)
  - Toni di grigio, RGB, CMYK, Lab, etc.
  - Modello Adobe
  - Permette referenze ad immagini raster
- PDF
  - Come sopra, inoltre:
  - Trasparenza / alpha
  - Immagini multiple (“pagine”)

JPEG

Joint Photographic Experts Group

(JPEG File Interchange Format (JFIF))

Prima di iniziare…: alcune caratteristiche

- Spazio di colori YCbCr (luminanza/crominanza)
- Lossy …. 1MB -> 25Kb (1:40)!
- Organizzazione in blocchi
- Formato big-endian

- Vedremo:
  - Conversione in JPEG
  - Struttura del file JPEG

Conversione in JPEG

1. Conversione spazio di colori: RGB -> YCbCr
2. Discrete Cosine Transform (DCT)
3. **Quantizzazione dei dati DCT per troncare i valori più piccoli (Unica operazione lossy!!)**
4. Compressione dei dati con Huffman e/o codifica aritmetica
5. Salvataggio JFIF/JPEG in blocchi di informazioni

Conversione spazio di colori: RGB -> YCbCr

$$Y = 0.0299R + 0.587G + 0.114B$$
$$Cb = 0.1687R - 0.3313G + 0.5B$$
$$Cr = 0.5R - 0.4187G + 0.0813B$$

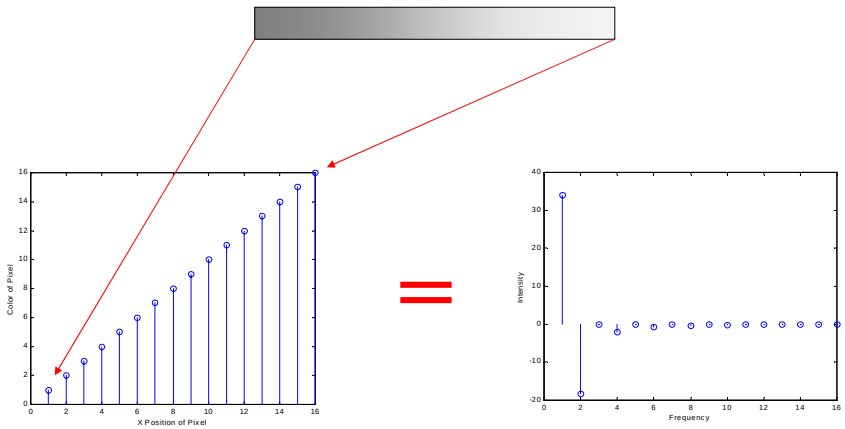
$$R = Cr * (2 - 2 * 0.299) + Y$$
$$G = (Y - 0.114*B - 0.299*R) / 0.587$$
$$B = Cb*(2 - 2* 0.114) + Y$$

Discrete Cosine Transform (DCT)

- Ogni immagine contiene varie frequenze di colore
- Le basse frequenze (bf) corrispondono a colori che cambiano in modo lento e graduale
- le alte (af) a cambiamenti fini e particolareggiati
- L'occhio umano è molto più sensibile alle bf che alle alte
- Alcune af possono essere eliminate dall'immagine senza che l'occhio avverta sensibilmente alcuna differenza
- L'eliminazione di alcune af rappresenta un metodo di compressione (lossy)
- E' dunque necessario trovare un modo per rappresentare l'immagine in termini delle frequenze di cui è composta

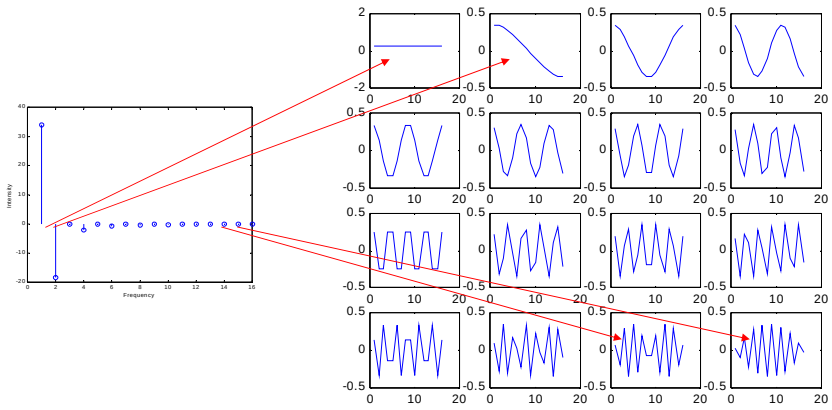
Spazio di colore e spazio di frequenze

(caso 1D, segmento di 16 pixel)



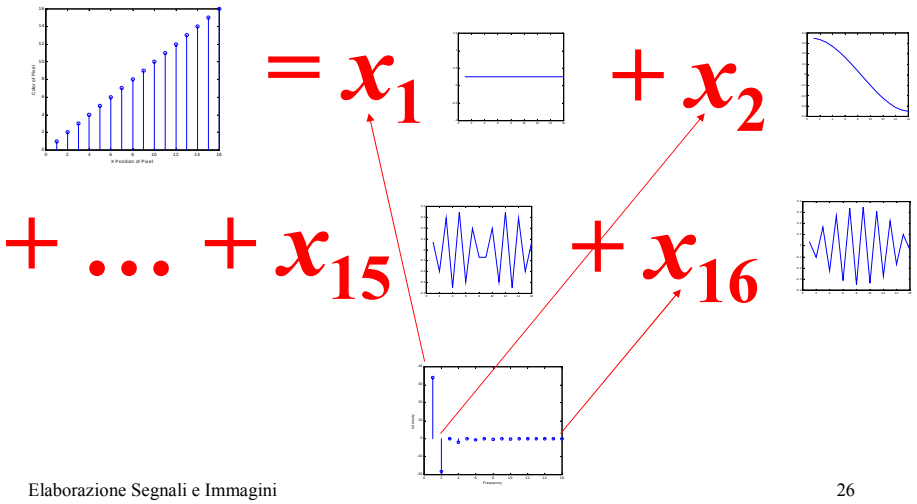
Scomposizione: le **funzioni base**

(caso 1D, segmento di 16 pixel)



Scomposizione: le funzioni base (cont)

(caso 1D, segmento di 16 pixel)



La DCT 1D e la sua inversa

$$S(u) = \sqrt{2/n}\, C(u) \sum_{x=0}^{n-1} s(x) \cos\frac{(2x+1)u\pi}{2n} \qquad u = 0,\ldots,n$$

$$s(x) = \sqrt{2/n} \sum_{u=0}^{n-1} C(u)\, S(u) \cos\frac{(2x+1)u\pi}{2n} \qquad x = 0,\ldots,n$$

Dove $C(u) = 2^{-1/2}$ per $u = 0$

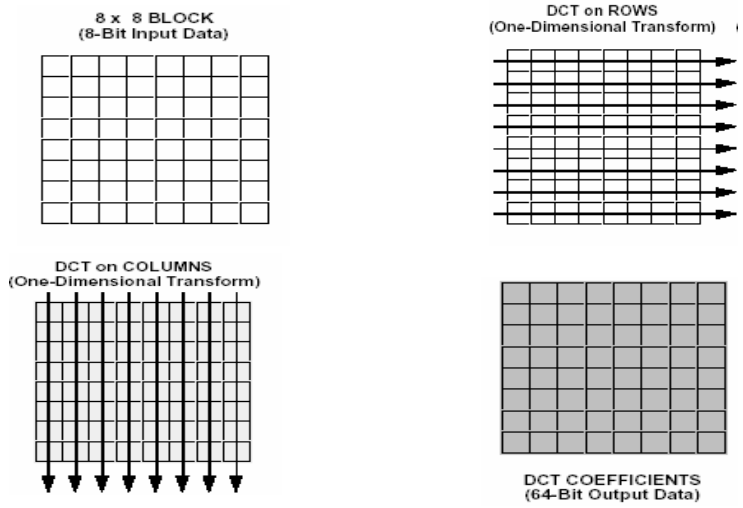
$= 1$ altrimenti

Il caso bidimensionale

- In jpeg si scompone ogni immagine in blocchi di 8x8 pixels.
- La DCT 2D viene applicata ad ogni singolo blocco.

La DCT 2D

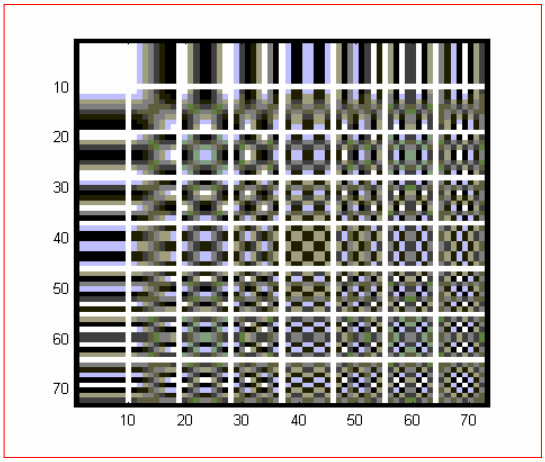
Per ogni blocco si "1D-trasformano" prima le righe, poi le colonne



La DCT 2D (cont)

- Dato che la DCT 2D può essere ottenuta come una combinazione di due DCT 1D si dice che la DCT 2D è *separabile nelle due dimensioni*
- Se lavoro su un blocco di 8x8 pixel, quante saranno le funzioni base? Come sono fatte?

Le 64 immagini base



I coefficienti DCT

7342	199	448	362	342	112	31	22
[illegible]	151	181	264	59	27	14	[illegible]
[illegible]	291	210	07	27	30	27	[illegible]
[illegible]	133	159	119	58	55	36	[illegible]
19	85	217	50	8	3	14	12
58	129	60	40	41	11	2	1
30	121	61	22	30	1	0	1
22	20	[illegible]	33	24	31	44	81

La DCT 2D (cont)

$$S(u,v) = \frac{2}{\sqrt{nm}} C(u)C(v) \sum_{y=0}^{m-1} \sum_{x=0}^{n-1} s(x,y) \cos\frac{(2x+1)u\pi}{2n} \cos\frac{(2y+1)u\pi}{2m}$$

Dove: $u = 0,\ldots,n$

$v = 0,\ldots,m$

$C(u) = 2^{-1/2}$ per $u = 0$

$= 1$ altrimenti

E analogo per C(v)

Riassunto equazioni DCT

$$S(u) = \sqrt{2/n}\, C(u) \sum_{x=0}^{n-1} s(x) \cos\frac{(2x+1)u\pi}{2n} \qquad u = 0,\ldots,n$$

$$s(x) = \sqrt{2/n} \sum_{u=0}^{n-1} C(u)\, S(u) \cos\frac{(2x+1)u\pi}{2n} \qquad x = 0,\ldots,n$$

$$S(u,v) = \frac{2}{\sqrt{nm}} C(u)C(v) \sum_{y=0}^{m-1} \sum_{x=0}^{n-1} s(x,y) \cos\frac{(2x+1)u\pi}{2n} \cos\frac{(2y+1)u\pi}{2m} \qquad u = 0,..,n \quad v = 0,...,m$$

$$C(u) = 2^{-1/2} \quad \text{for } u = 0$$
$$= 1 \quad \text{otherwise}$$

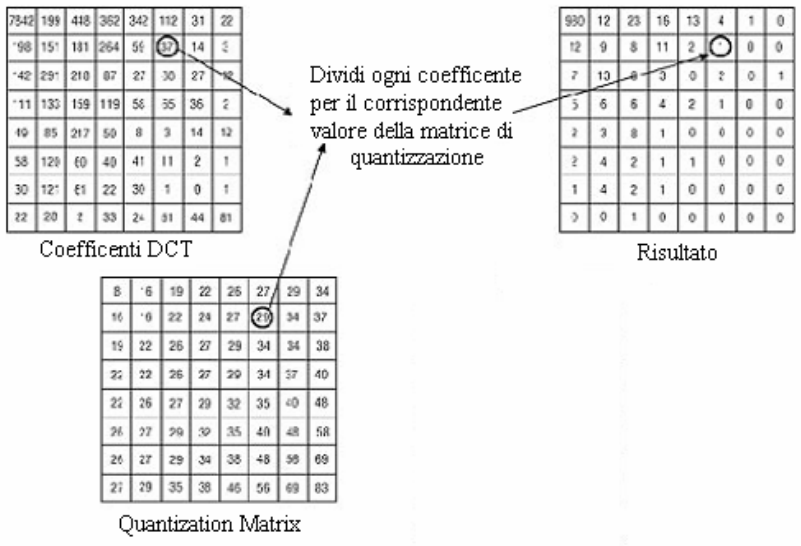
Com'è la trasformata DCT 2D inversa?

Quantizzazione (o…weighting)

- La DCT 2D trasforma ogni blocco 8x8 in un altro blocco 8x8….non è ancora stata ridotta la dimensione dei dati!!
- Si è notato che la vista umana è meno sensibile a frequenze alte, mentre lo è di più a fb
- Questa dipendenza **non** è lineare

Come possiamo manipolare le frequenze??

La *quantization matrix*

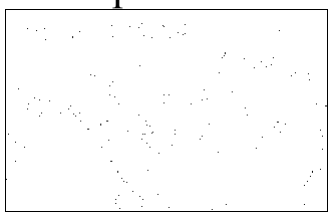


Esempio di quantizzazione

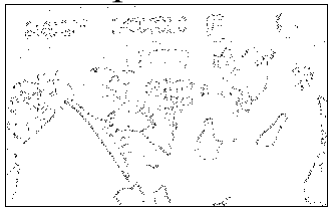
Immagine originale

Esempio di quantizzazione (cont)

Quant.Factor=2, Rapporto di compressione 4:1

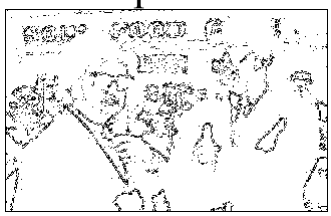


Quant.Factor=5, Rapporto di compressione 7:1



Esempio di quantizzazione (cont)

Quant.Factor=10, Rapporto di compressione 10:1



Quant.Factor=15, Rapporto di compressione 13:1

Un altro esempio

Un altro esempio (cont)

Esempio 3

Immagine originale

Esempio 3 (cont)

Quality Factor = 75

Esempio 3 (cont)

Quality Factor = 20

Esempio 3 (cont)

Quality Factor = 5

Esempio 3 (cont)

Quality Factor = 3

Vedremo:
- Conversione in JPEG
- Struttura del file JPEG

Conversione in JPEG

1. Conversione spazio di colori: RGB -> YCbCr
2. Discrete Cosine Transform (DCT)
3. Quantizzazione dei dati DCT per troncare i valori più piccoli (Unica operazione lossy!!)
4. **Compressione dei dati con Huffman e/o codifica aritmetica**

Compressione

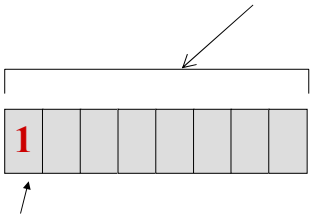
- Dopo DCT e quantizzazione i dati vengono compressi
- Per la compressione si usa l'algoritmo di **Huffman** o quello **aritmetico**
- In ogni caso, l'algoritmo è in genere modificato, per meglio sfruttare la sparsità della matrice:
- si esegue prima un **Run-Length Encoding (RLE)** sui gruppi di valori nulli. Ciò riduce notevolmente la dimensione dei dati.

Run-Length Encoding (RLE)

- Attualmente ha due varianti principali:
- PackBits, per il programma MacPaint
  - MacPaint lavora solo su img monocromatiche
  - Quindi PackBits poteva essere “single-bit oriented”
  - Invece lavora su gruppi di pixel da 8 bit
- Formato PCX della Z-Soft
  - Lavora a livello di singolo bit
  - Tuttavia limitato a 63 bits img -> 2 byte compressi

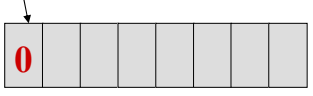
RLE versione PackBits

- PackBits conta pixel(byte) (uguali) adiacenti
- Il numero di pixel viene salvato in un byte, che chiamiamo *contatore* e che è organizzato così:



Indica un pattern di *contatore* pixel tutti uguali; *contatore* è in complemento a due

Il *most significant bit* funziona da FLAG



Seguono *contatore*+1 pixel tutti diversi uno dall’altro

Esempio PackBits

Immagine / Pixel da codificare:

```
10101100  10101100  10101100 10101100
10010110  11110001  01111001 11111111
00000111
```

Contatore: 257_{10}-11111100_2 = 257-253 = 4

Codifica:

```
11111100  10101100  00000100 10010110
11110001  01111001  11111111 00000111
```

Contatore: 00000100_2+1 = 4 + 1 = 5

PackBits & MacPaint

- **In teoria** PackBits può memorizzare fino a un massimo di 1000000_2=257-128=129 pixel consecutivi uguali, o 01111111_2+1=127+1=128 pixel consecutivi diversi, dove ogni pixel è composto da 8 bit (colori o no)
- In realtà però PackBits resetta il contatore ad ogni nuova riga; inoltre
- Macintosh: monitor monocromatico (cioè 1pixel = 1bit) 576 x 720
- Morale, **in pratica** PackBits conta fino ad un massimo di 576 / 8 = 72 gruppi di 8 pixel monocromatici

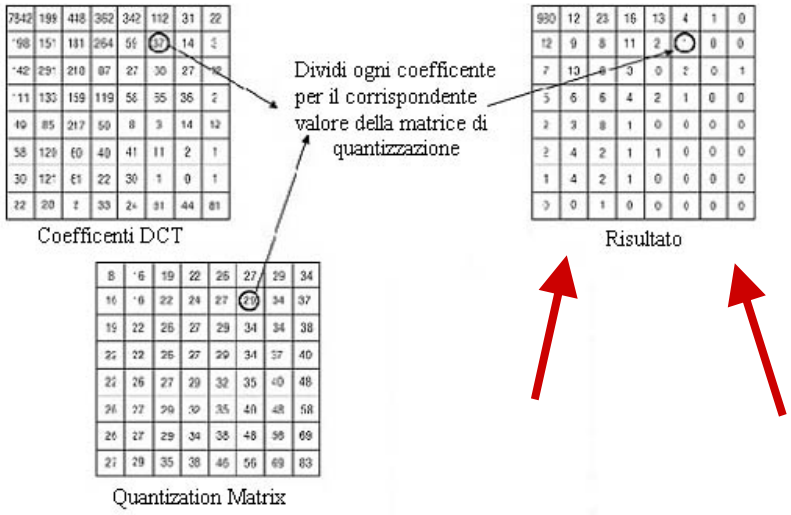
Vedremo:
- Conversione in JPEG
- Struttura del file JPEG

Tornando al JPEG…

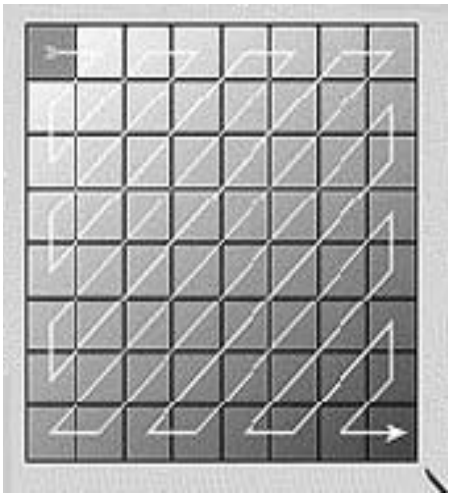
1. Conversione spazio di colori: RGB -> YCbCr
2. Discrete Cosine Transform (DCT)
3. **Quantizzazione** dei dati DCT per troncare i valori più piccoli (Unica operazione lossy!!)
4. **Compressione:** **RLE**
   Huffman / cod. Aritmetica

Dopo la quantizzazione che forma hanno i dati?

La matrice dei dati quantizzati



zig-zag scanning



Codifica Huffman

- Sviluppato nel 1952 da David A. Huffman
- È basato su un dizionario di simboli che
- Tiene conto della frequenza di ogni possibile valore di pixel; più in particolare:
- Il dizionario è organizzato ad albero, in modo tale che pixel più frequenti nell'immagine abbiano i simboli più facili e veloci da accedere

Un esempio di codifica Huffman

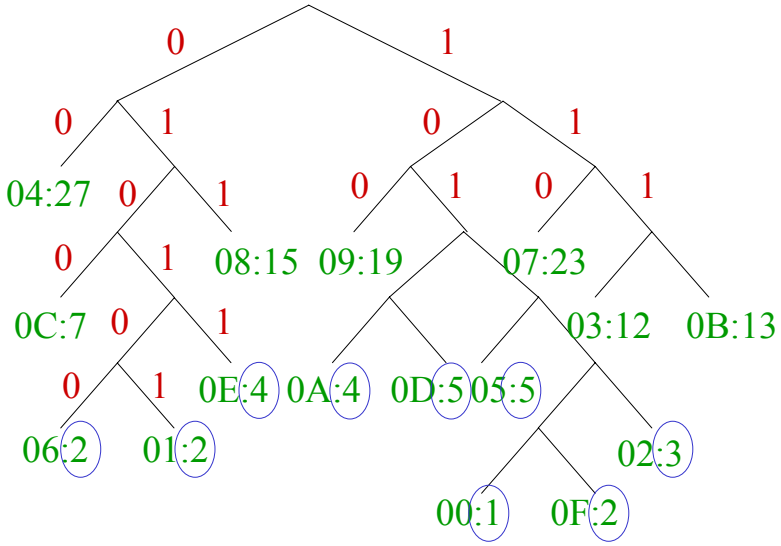
- Si consideri un'immagine 12x12 a 16 colori (quindi occorrono 4 bit, o una sola cifra esadecimale per ogni pixel)
- Per ogni colore, espresso in esadecimale, il corrispondente numero di pixel è (colore : #pixel)

  00:1, 01:2, 02:3, 03:12, 04:27, 05:5, 06:2, 07:23, 08:15, 09:19, 0A:4, 0B:13, 0C:7, 0D:5, 0E:4, 0F:2

Esempio Huffman (cont)

- Organizziamo ora la lista in un albero binario t.c.
- Ogni ramo sinistro è etichettato "0", mentre ogni ramo destro è etichettato "1"
- Ogni coppia della forma (colore:#pixel) è una foglia dell'albero
- Percorrendo le foglie dell'albero partendo dalla più profonda, si ottiene la lista ordinata secondo il numero di occorrenze (o pixel) di ogni colore

Esempio Huffman (cont)



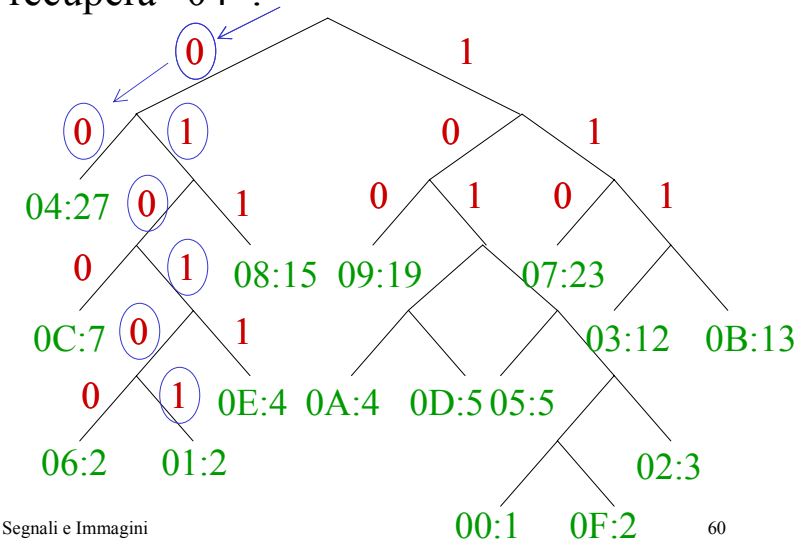
Esempio Huffman (cont)

Come si recupera “04”?

0 0

E “01”?

0 1 0 1 0 1



Gli svantaggi della codifica Huffman

- Si dimostra che la compressione non è ottimale
- Sono necessari due "scan" dell'immagine
- Occorre salvare anche il dizionario (grosso, se l'immagine ha tanti colori)
- Un errore nella trasmissione rende i dati privi di significato
- Huffman non si presta a comprimere immagini in bianco e nero – perché? Quali possibili soluzioni o alternative?

La codifica aritmetica

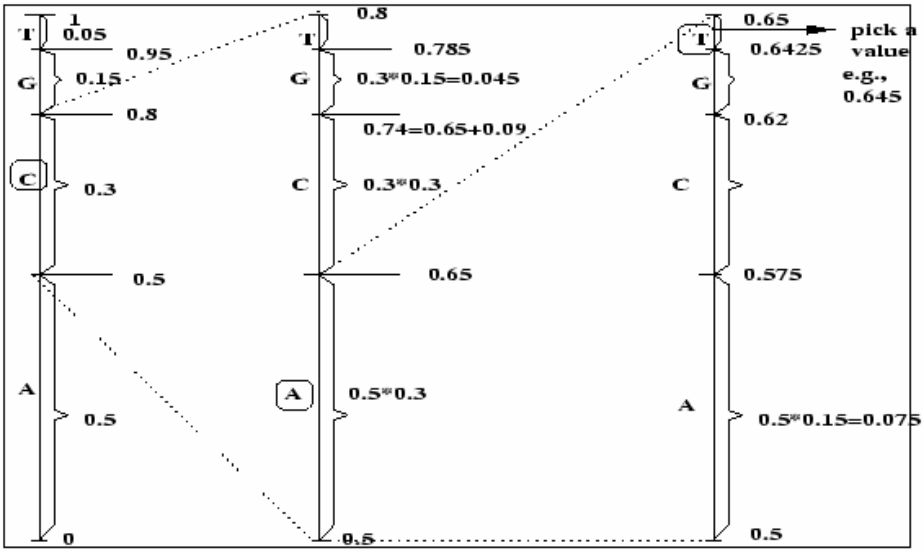
- Invece di produrre un codice per ogni simbolo, si genera un codice per l'intero input
- Il miglioramento è dato dal fatto che ogni simbolo contribuisce alla codifica con un nr frazionario di bit
- Quanto più lungo e complesso è l'input, tanti più bit sono necessari per la codifica
- Il numero risultante dalla codifica può essere univocamente decodificato, recuperando l'esatta sequenza originaria

La codifica aritmetica: il meccanismo alla base

- Ad ogni simbolo si assegna una probabilità (o frequenza di occorrenza) e, sulla base di questa,
- Gli si associa un intervallo su una "retta di probabilità" nel range [0,1)
- L'ordine in cui appaiono gli intervalli sulla retta non è rilevante
- Il codice prodotto è un numero in virgola mobile compreso nell'intervallo [0,1)

Esempio: codifica di CAT

(in una stringa che contiene: 1T, 3G, 6C, 10A)

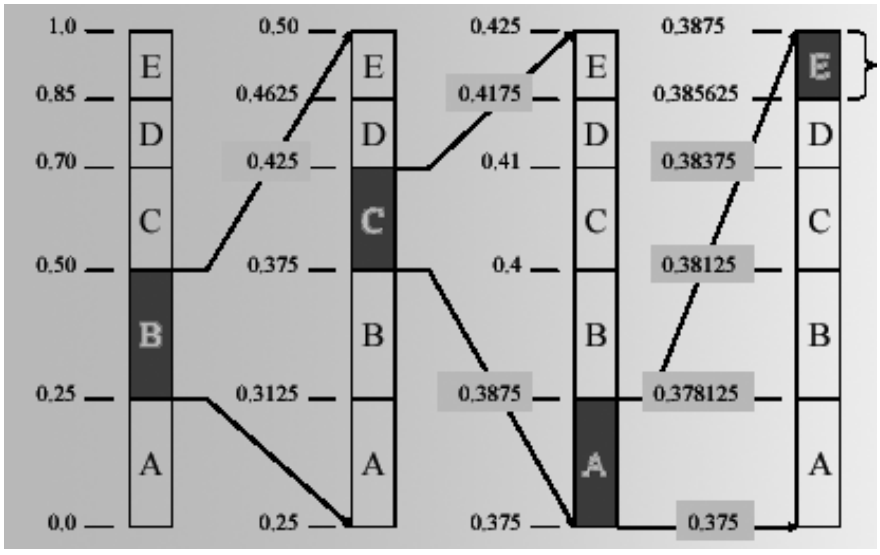


Esempio: codifica di “Bill Gates”

Carattere	Probabilità	Intervallo [_, _)	
		LowRange	HighRange
SPAZIO	1/10	0	0.1
A	1/10	0.1	0.2
B	1/10	0.2	0.3
E	1/10	0.3	0.4
G	1/10	0.4	0.5
I	1/10	0.5	0.6
L	2/10	0.6	0.8
S	1/10	0.8	0.9
T	1/10	0.9	1

Esempio: codifica di BCAE

(in una stringa che contiene 3E, 3D, 4C, 5B, 5A)



L'algoritmo

1. Low = 0
2. High = 1
3. for i = 1 to lunghezza(Input)
4. Range = High – Low
5. High = Low + Range * HighRange(Input[i])
6. Low = Low + Range * LowRange(Input[i])
7. return Low

Esempio: "Bill Gates" passo per passo

Stringa Input	Low	High
inizio	0	1
B	0.2	0.3
I	0.25	0.26
L	0.256	0.258
L	0.2572	0.2576
SPAZIO	0.2572	0.25724
G	0.257216	0.257220
A	0.2572164	0.2572168
T	0.25721676	0.2572168
E	0.257216772	0.257216776
S	**0.2572167752**	0.2572167756

La virgola mobile in binario

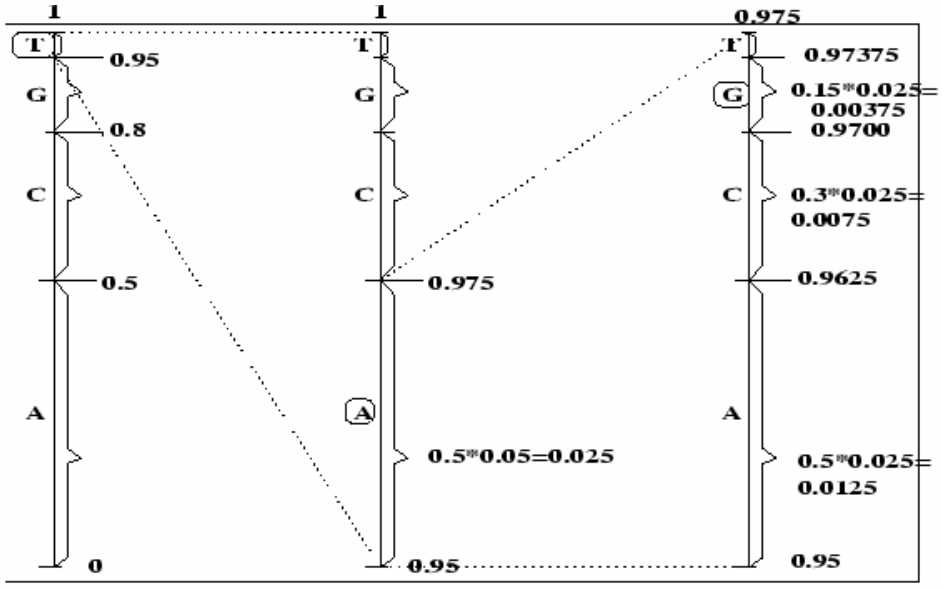
- In generale: … | 2^2 | 2^1 | 2^0 | , | 2^{-1} | 2^{-1} | …
  | | 4 | 2 | 1 | | 0.5 | 0.25 |
- Quindi per esempio:
  - $0.5_{10} = (½)_{10} = (2^{-1})_{10} = 0.1_2$
  - $0.25_{10} = (¼)_{10} = (1/2^2)_{10} = (2^{-2})_{10} = 0.01_2$
  - $0.125_{10} = (1/8)_{10} = (1/2^3)_{10} = (2^{-3})_{10} = 0.001_2$
  - $0.0675_{10} = (1/16)_{10} = (1/2^4)_{10} = (2^{-4})_{10} = 0.0001_2$
  - $0.625_{10} = 0.5_{10} + 0.125_{10} = 0.101_2$
  - $0.875_{10} = (0.5 + 0.25 + 0.125)_{10} = 0.111_2$

"Fine" della codifica

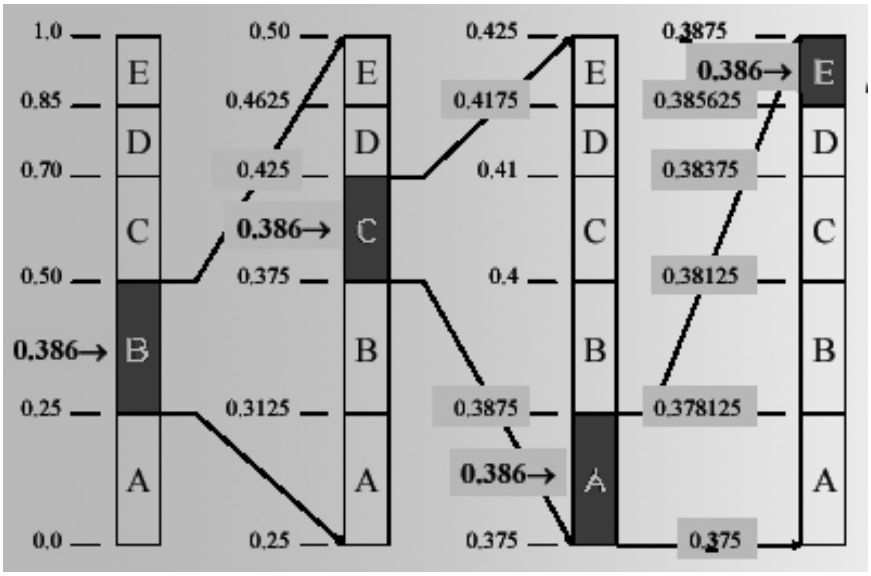
- Ci sono due modi per segnalare la fine della stringa codificata:
  - Specificare in testa il nr di caratteri codificati
  - Assegnare all'alfabeto un nuovo simbolo EOF, con la minore probabilità (è usato solo una volta!) e attaccarlo alla fine del codice

Decodifica: esempio per 0.9715

P(A) = 0.5; P(C) = 0.3; P(G) = 0.15; P(T) = 0.05



Decodifica: esempio per 0.386



L'algoritmo di decodifica

```
1. i = 0
2. do
3.     decodString[i] = ConvToSymbol(codice)
4.     range = HighRange(decodString[i]) –
                LowRange(decodString[i])
5.     codice = codice – LowRange(decodString[i])
6.     codice = codice / Range
7.     i++
8. until decodString[i] == EOF
9. return decodString
```


Gli svantaggi della codifica aritmetica

- Si basa sul formato float:
  - Necessità di processori float
  - Mancanza di compatibilità tra macchine con diversi formati di floating point
  - La precisione float è limitata (tipicamente 64bit, con i quali si possono codificare al più 10-15 simboli)
- Non è incrementale: è necessario avere dall'inizio l'intera stringa da comprimere

Per ogni blocco 8x8…

Tutto questo procedimento (…quale esattamente??) viene applicato ad ogni blocco 8x8 (o, a volte, 16x16) dell'immagine.

Per ogni blocco 8x8…

- Al termine di ogni blocco, JPEG inserisce una sequenza end-of-block.
- Al termine di tutti i blocchi JPEG accoda il marker di fine file (EOF)
- Vediamo più in dettaglio come…

La struttura in blocchi del file JPEG

1. 1 blocco SOI (Start of Image)
2. 1 bl. APP0 (Application type 0) con le info dello header JFIF
3. blocchi APP0 opzionali con le info di estensione JFIF
4. 1 o + bl. DQT (Define Quantization Table) con le tabelle di quantizzazione dell'immagine
5. 1 bl. SOF0 (Start of Frame type 0) con le caratteristiche dei dati dell'immagine "Huffman-codificati"
6. 1 o + bl. DHT (Define Huffman Table) con le informazioni relative alla codifica Huffman
7. 1 bl. DRI (Define Restart Interval) opzionale, che definisce gli intervalli per reinizializzare il decoding
8. 1 o + bl. SOS (Start of Scan) con i dati
9. 1 bl. EOI (End of Image) che marca la fine del file

Il blocco SOI e il blocco APP0 base

Nome campo	**Dimensione**	**Descrizione / valore**
SOI Marker	word	$FFD8 hex
APP0 Marker	word	$FFE0 hex
SegmentLength	word	Lungh.blocco,incl.counter
APP0 ID	Array[0..4] di Char	"JFIF" e $00 hex
Version	word	$0102 hex
Units	Byte	0-Dflt, 1-Pxl/In, 2-Pxl/Cm
X Density	word	pixel ratio orizz.tle (1dflt)
Y Density	word	pixel ratio vertic. (1 dflt)
X Thumb	Byte	Larghezza thumbnail
Y Thumb	Byte	Altezza thumbnail
ThumbNail	3*ThumbSz	Thumbnail (24-bit RGB)

Extension APP0 Block(s)

Nome campo	Dimensione	Descrizione / valore
APP0 Marker	word	$FFE0 hex
SegmentLength	word	Lungh.blocco
APP0 ID	Array[0..4] di Char	“JFXX” e $00 hex
AppType	Byte	Codice (10-12 indica immagine thumbnail (*))
BlockData	block of Byte	Actual extension data

(*)

- 10: thumbnail salvato come immagine compressa JPEG
- 11: thumbnail con palette a 256 colori (inclusa nel blocco)
- 12: thumbnail RGB

I blocchi DQT

Nome campo	Dimensione	Descrizione / valore
DQT Marker	word	$FFDB hex
SegmentLength	word	Lunghezza del segmento
Table ID	Byte	Precisione = 4 bits più signific. Tbl# = 4 bits meno significativi
TableEntries	Byte	64 valori Y o 128 Cr/Cb

Il blocco SOF0

Nome campo	Dimensione	Descrizione / valore
SOF0 Marker	Word	$FFC0 hex
SegmentLength	Word	Lunghezza del segmento
Precisione	Byte	Precisione
RowCount	Word	Nr di righe dell'immagine
PixelCount	Word	Nr di pixel per riga
CompCount	Byte	Nr di componenti (default 3)
Per ognuna delle tre componenti Y, Cb e Cr:		
(Y | Cb | Cr) ID	Byte	$01 | $02 | $03
(Y | Cb | Cr) factor	Byte	4 MS bits = campionam. orizz. 4 LS bits = campionam. Vertic.
(Y | Cb | Cr)Qnt.Table	Byte	$01 | $02 | $03

Le coppie di blocchi DHT

Nome campo	Dimensione	Descrizione / valore
DHT Marker	Word	$FFC4 hex
SegmentLength	Word	Lunghezza del segmento
Le righe seguenti si ripetono per ogni tabella:		
TblClass / TblID	Byte	4 MS bits = TblClass 4 LS bits = TblID
CodeList	Array[0..15] di Byte	Nr di codice di lunghezza I
CodeData	Array di Byte	Codice per lunghezza I

Blocco opzionale DRI

Nome campo	Dimensione	Descrizione / valore
DRI Marker	Word	$FFDD hex
SegmentLength	Word	Lunghezza del segmento
RestartLength	Word	Lunghezza di un intervallo di restart

I blocchi SOS

Nome campo	Dimensione	Descrizione / valore
SOS Marker	Word	$FFDA hex
SegmentLength	Word	Lunghezza del segmento
ComponentNum	Word	#comp.questo scan (1,2,3)
Le 2 righe seguenti si ripetono per ogni componente:		
CompSelector	Byte	Component Selector
DCAC Tbl	Byte	4 Msbits = DCTbl 4 Lsbits = ACTbl
SpectralSel	Byte	Inizio spectr.selection (0)
SpectralEnd	Byte	Fine spectral selection(63)
SpectralBits	Byte	4MSBits = HiSpc 4LSBits=LoSpc: sempre $00

Il blocco EOI

Nome campo	Dimensione	Descrizione / valore
EOI Marker	Word	$FFD9 hex

Domanda finale….

Con riferimento alla seconda immagine dei due amici (lucido 42), salvata sia a colori che in scala di grigio, si commenti la seguente tabella

Dimensione in Kb dell'img originale a colori			313.076
Dimensione in Kb dell'img originale B/W			104.437
Fattore di qualità	JPEG a colori		JPEG B/W	
	Dimensione del file in Kb	Rapporto di compressione	Dimensione del file in Kb	Rapporto di compressione
75	23.039	13.59	21.02	4.97
20	8.457	37.02	7.599	13.74
5	4.009	78.09	3.257	32.07
3	3.268	95.80	2.522	41.41