

# Compressione



# Compressione

- Con il termine compressione dati si indica la tecnica di elaborazione dati che, attuata a mezzo di opportuni algoritmi, permette la riduzione della quantità di bit necessari alla rappresentazione in forma digitale di una informazione.
- La compressione riduce le dimensioni di un file e quindi lo spazio necessario alla sua memorizzazione.
- La compressione riduce l'occupazione di banda necessaria in una generica trasmissione dati digitale.



- Dal momento che differenti quantità di dati possono essere usate per rappresentare la stessa quantità di informazione le rappresentazioni che contengono informazioni irrilevanti o ripetute contengono i cosiddetti *dati ridondanti*.



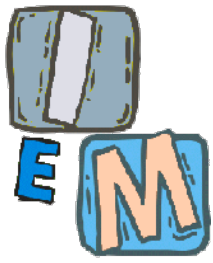
# Ridondanza della codifica

- Un codice è un sistema di simboli (lettere, numeri, bit, ecc) utilizzati per rappresentare una certa quantità di informazioni.
- Ad ogni “pezzo” d’informazione e a ogni singolo evento è assegnata una sequenza di *simboli codificati*, chiamati *codeword*.
- Il numero di simboli che costituisce ciascun codice è la sua *lunghezza*.



# Ridondanza spaziale e temporale

- Dal momento che la maggior parte degli array di intensità 2-D sono relazionati spazialmente (per es. ciascun pixel è simile ai pixel del suo intorno, o dipende da esso), l'informazione è replicata inutilmente nei pixel correlati.
- In una sequenza video, i pixel correlati temporalmente (per es. simili o dipendenti dai pixel dei frame vicini) rappresentano anch'essi un'informazione duplicata.



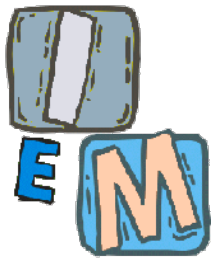
# Informazione percettivamente irrilevante

- Spesso i dati contengono informazioni ignorate dal sistema sensoriale umano.
- E' ridondante nel senso che non viene utilizzata.

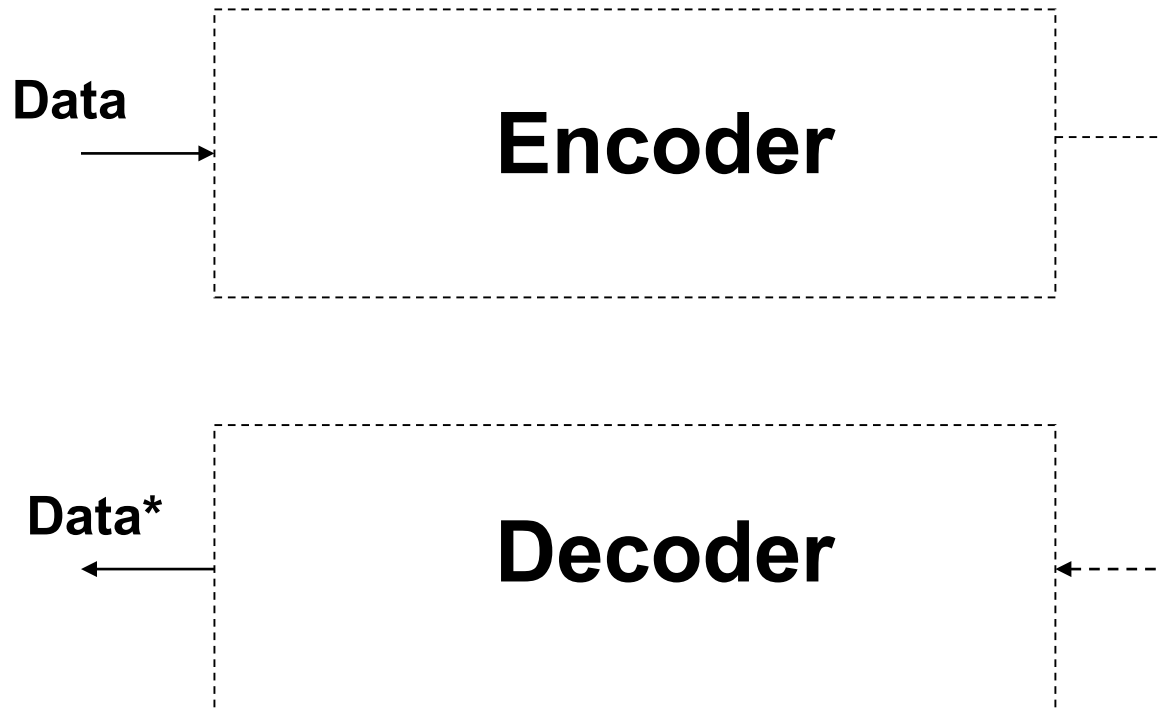


# Algoritmo di compressione

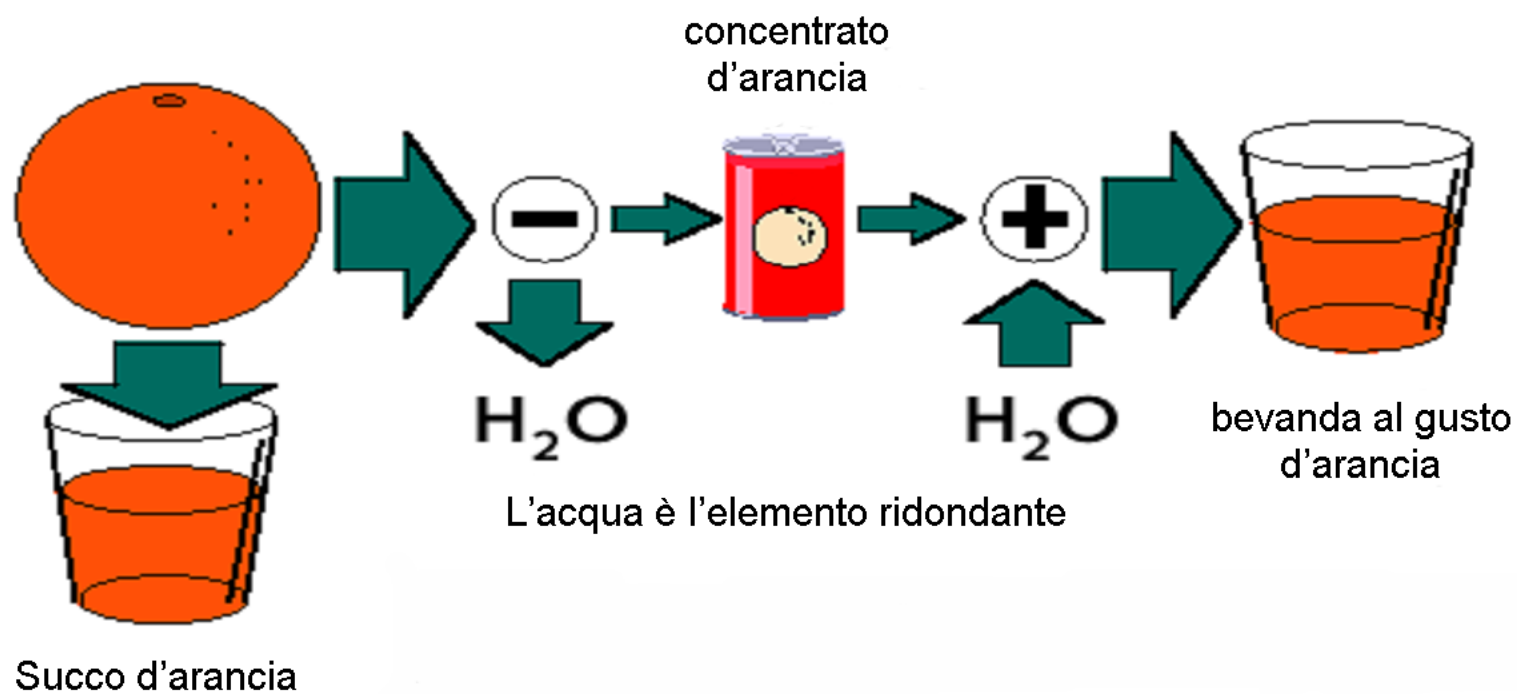
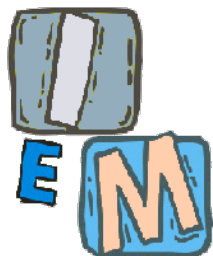
**Un algoritmo di compressione** è una tecnica che elimina la ridondanza di informazione dai dati e consente un risparmio di memoria



# Il processo di compressione e decompressione









# Classificazione dei metodi di compressione

Basata sul tipo di dati (Generico, Audio, Immagini, Video);

Basata sul tipo di compressione:

- REVERSIBILE o **lossless**, cioè senza perdita di informazione;
- IRREVERSIBILE o **lossy**, con eventuale perdita di informazione.



# Compressione LOSSLESS

Si parla di compressione **LOSSLESS** quando i dati possono essere trasformati in modo da essere memorizzati con risparmio di memoria e successivamente ricostruiti perfettamente, senza errore e senza perdita di alcun bit di informazione.

Tale tipo di compressione è ovviamente necessario per ridurre lo spazio occupato da documenti, programmi eseguibili, eccetera.



# Criterio per una buona compressione di tipo lossless

- Cercare di raggiungere il limite teorico per la compressione senza perdita che viene fornito dal primo teorema di Shannon (che vedremo tra poco).



# Frequenza

Sia data una sequenza  $S$  di  $N$  caratteri tratti da un alfabeto di  $M$  possibili caratteri:  $a_1, \dots, a_M$ .

Sia  $f_i$  la frequenza del carattere  $a_i$  cioè

$$f_i = (\# \text{ occorrenze } a_i) / N$$



# Entropia

Definiamo entropia  $E$  della sequenza di dati  $S$  la quantità media di informazione associata alla singola generazione di un simbolo nella sequenza  $S$ :

$$E = - \sum f_i \log_2 (f_i) \quad i \in S$$

Più è grande l'incertezza della sequenza maggiore è l'entropia. Il massimo valore di entropia (e quindi di incertezza) lo si ha quando i simboli della sequenza sono equiprobabili.



# ATTENZIONE

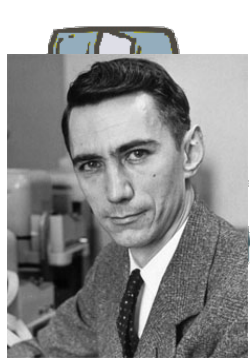
- Se la vostra calcolatrice non ha il log in base 2 occorre cambiare la base usando questa formula:

## FORMULA DEL CAMBIAMENTO DI BASE PER I LOGARITMI

Ecco la formula per il cambiamento di base nei logaritmi: dati  $a, b, c$  positivi e con  $a, c \neq 1$

$$\log_a(b) = \frac{\log_c(b)}{\log_c(a)}$$

Grazie a questa formula possiamo scrivere il logaritmo in una data base  $a$  con una nuova base  $c$  **scelta a piacere**, purché sia maggiore di zero e diversa da uno. 😊



# Teorema di Shannon (1948)

«per una sorgente discreta e a memoria zero, il bitrate minimo è pari all'entropia della sorgente»

*I dati possono essere rappresentati senza perdere informazione (lossless) usando almeno un numero di bit pari a:*

$$N * E$$

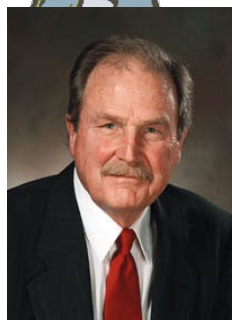
Dove N è il numero di caratteri mentre E è l'entropia.





# Attenzione!

- Il teorema di Shannon fissa il numero minimo di bit, ma non ci dice come trovarli.
- Occorre usare un algoritmo che permetta di codificare i nostri caratteri usando esattamente il numero di bit ricavati con il teorema di Shannon.
- Un algoritmo che fa ciò è stato proposto da Huffman.



# Codifica di Huffman (1953)

**David Huffman** ha proposto un semplice algoritmo greedy che permette di ottenere un “dizionario” (cioè una tabella carattere-codifica\_binaria) per una compressione quasi ottimale dei dati cioè pari al limite di Shannon con un eccesso di al più qualche bit.



# Proprietà

- Si tratta di codifica a lunghezza variabile che associa a simboli meno frequenti i codici più lunghi e a simboli più frequenti i codici più corti.
- Si tratta di una codifica in cui nessun codice è prefisso di altri codici.
- È una codifica ottimale perchè tende al limite imposto dal teorema di Shannon.



# Codifica di Huffman

- È un algoritmo iterativo. Ad ogni iterazione si scelgono i due nodi con frequenza più bassa.
- Questi due nodi vengono collegati per formare un sottoalbero la cui frequenza del nodo padre è la somma delle frequenze dei due nodi.
- Se ci fossero più nodi con peso minimo se ne scelgono solo due.
- Se c'è un solo nodo con frequenza più bassa si seleziona tale nodo e poi si prende da seconda frequenza più bassa e si seleziona un nodo con quella frequenza.



# Codifica di Huffman

- Si deve creare un albero binario bilanciato.
- Al termine delle iterazioni la radice avrà peso 1.
- Si etichetteranno i rami a sinistra con codice 1 e quelli a destra con codice 0.
- Il codice che si forma procedendo dalla radice alla foglia è il codice abbinato al carattere presente nella foglia stessa.



# Codifica di Huffman (1)

Illustro l'algoritmo con un esempio.

Dati: AABABCAACAAADDDD.

A : frequenza pari a  $8/16 = \frac{1}{2}$

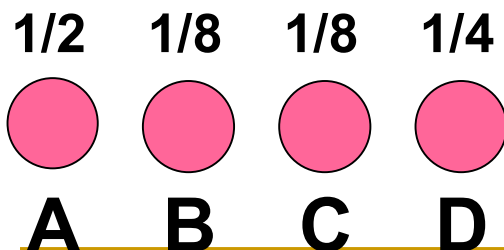
B : frequenza pari a  $2/16 = \frac{1}{8}$

C : frequenza pari a  $2/16 = \frac{1}{8}$

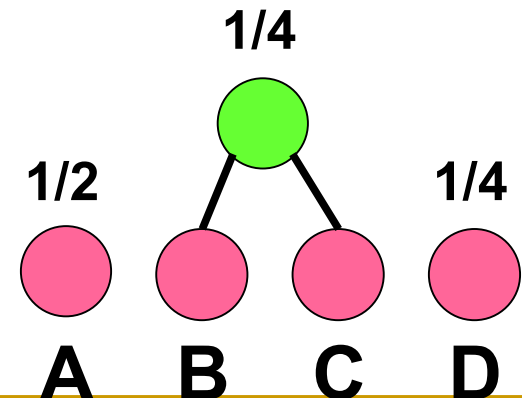
D : frequenza pari a  $4/16 = \frac{1}{4}$

L'algoritmo procede costruendo un albero binario le cui foglie sono i caratteri da codificare come segue:

INIZIO: una foresta con 4 alberi ciascuno composto di un singolo nodo.



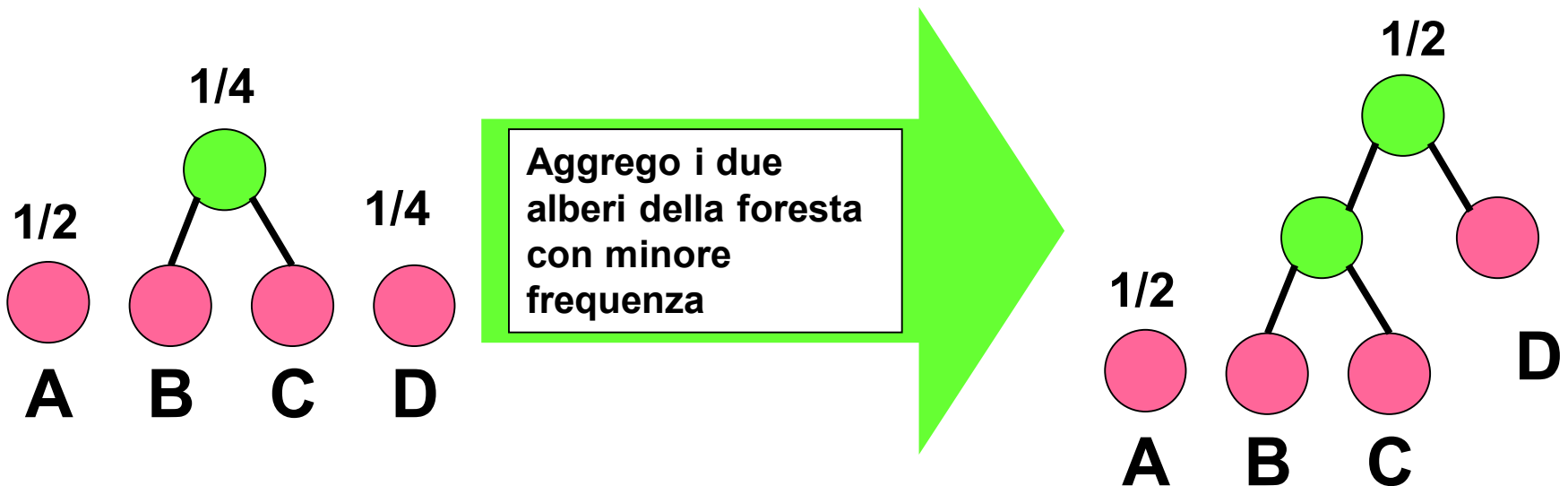
Aggrego i due alberi della foresta con minore frequenza





## Codifica di Huffman(2)

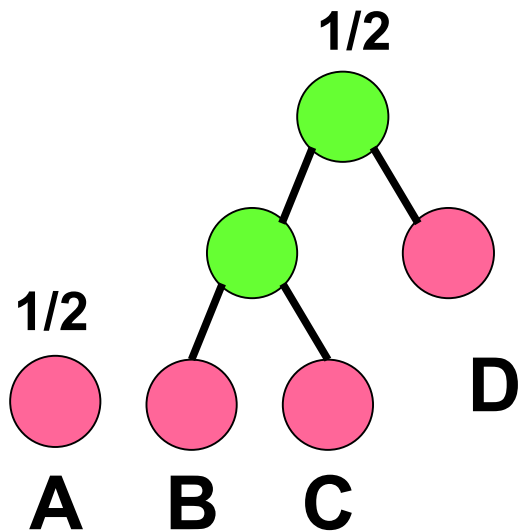
Procedo in tal modo fino ad avere un solo albero che aggrega tutte le foglie



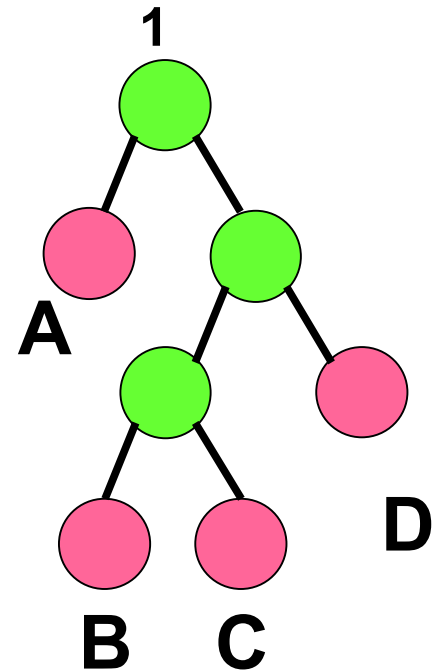


# Codifica di Huffman(3)

In questo caso ecco il risultato finale:



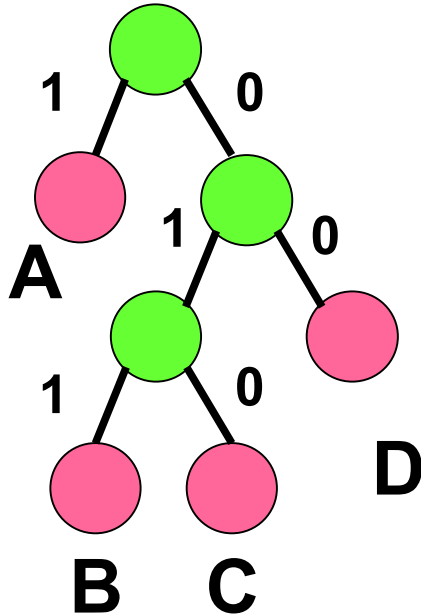
Aggrego i due  
alberi della foresta  
con minore  
frequenza







# Codifica di Huffman(4)



Etichetto con 1 i rami sinistri e 0 i destri. Il cammino dalla radice al simbolo fornisce la parola del dizionario che “codifica” in maniera ottimale il simbolo.

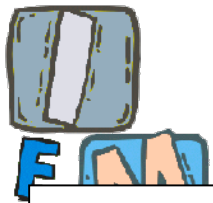
Si osservi che ho parole di codice di varia lunghezza, ma nessuna è prefissa delle altre. Inoltre i simboli più frequenti richiedono meno bit, i meno frequenti più bit.

**A : 1**

**B : 011**

**C : 010**

**D : 00**



# Codifica di Huffman(5)

**A : 1**  
**B : 011**  
**C : 010**  
**D : 00**

- **Codice per la sequenza  
AABABCAACAAADDDD.**
  - A : frequenza pari a  $8/16 = 1/2$
  - B : frequenza pari a  $2/16 = 1/8$
  - C : frequenza pari a  $2/16 = 1/8$
  - D : frequenza pari a  $4/16 = 1/4$

**Codifica:**

**1-1-011-1-011-010-1-1-010-1-1-1-00-00-00-00**      pari a **28 bit**

**(si osservi che i trattini sono del tutto superflui perché nessun codice per i caratteri è prefisso degli altri, cioè posso decodificare senza fare errori se ho solo:**

**1101110110101101011100000000**

**Quale è il limite previsto da Shannon?**

$$16 * (-1/2 * \log_2(1/2) - 1/8 * \log_2(1/8) - 1/8 * \log_2(1/8) - 1/4 * \log_2(1/4)) =$$

$$16 * (1/2 + 3/8 + 3/8 + 2/4) = 8 + 6 + 6 + 8 = \mathbf{28 \text{ bit}} - \mathbf{CODIFICA OTTIMALE!}$$



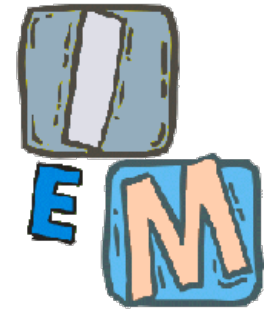
# Attenzione in pratica!

- **Costo aggiuntivo:** si deve memorizzare la tabella caratteri-codici. Se i caratteri sono tanti questo può essere costoso.
- Huffman viene usato per comprimere alcune informazioni *nella fase finale* della codifica JPEG (dopo che è stata fatta una riduzione con altre tecniche)
- Huffman è usato nei seguenti standard di compressione CCITT, JBIG2, JPEG, MPEG-1,2,4.



# Esercizio

- Applicare la codifica di Huffman alla stringa “gazzella”. Quanti bit sono effettivamente usati?
- Quanti bit si dovrebbero usare secondo il teorema di Shannon per la codifica della stringa “gazzella”?



# Altri algoritmi di compressione LOSSLESS

- Run-Length-Encoding (RLE)
- Codifica differenziale



# Run-Length-Encoding

- le immagini che hanno delle ripetizioni di intensità lungo le righe (o colonne) possono spesso essere compresse rappresentando tali sequenze (run) sottoforma di *coppie di run-length*, in cui ciascuna coppia individua l'inizio di una nuova intensità e il numero di pixel consecutivi ne condividono il valore in questione.
- la codifica run-length è usata in CCITT, JBIG2, JPEG, M-JPEG-1,2,4, BMP



# Run-Length-Encoding

Si voglia comprimere la sequenza formata da 29 bit:

00000111001011101110101111111

Si potrebbe ricordare in alternativa:

5 volte 0, 3 volte 1, 2 volte 0 etc.

O meglio basterebbe accordarsi sul fatto che si inizia con il simbolo 0 e ricordarsi solo la lunghezza dei segmenti (run) di simboli eguali che compongono la sequenza:

5,3,2,1,1,3,1,3,1,1,1,7

Tali valori vanno adesso scritti in binario. Sono sufficienti 3 bit per numero. La sequenza finale (senza spazi) sarà la seguente:

101 011 010 001 001 011 001 011 001 001 001 111

Occorrono quindi 36 bit.



# Run-Length-Encoding

- Non sempre tale codifica porta un risparmio rispetto a quella di input. Infatti nell'esempio della slide precedente, alla fine della codifica abbiamo usato 36 bit contro i 29 della sequenza iniziale.
- La compressione RLE funziona solo se la lunghezza della run è molto grande e prevede un numero di bit superiore a quelli necessari per scrivere il numero che rappresenta la run.
- Se ci sono molte “run” (sequenze di simboli uguali) piuttosto lunghe ricordare la sequenza delle loro lunghezze potrebbe portare un risparmio.

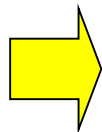


## Run Length + bit planes

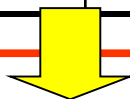
Debbo memorizzare la seguente immagini a 8 toni di grigio (3 bit depth):

In binario

2	5	7	0
5	5	5	5
4	4	3	4
5	4	2	0



010	101	111	000
101	101	101	101
100	100	011	100
101	100	010	000



Bit planes

0	1	1	0
1	1	1	1
1	1	0	1
1	1	0	0

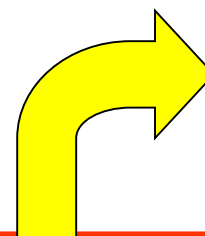
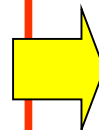
Primo bit

1	0	1	0
0	0	0	0
0	0	1	0
0	0	1	0

Secondo bit

0	1	1	0
1	1	1	1
0	0	1	0
1	0	0	0

Terzo bit



To run-length encoding

0110111111011100  
1010000000100010  
0110111100101000



# Codifica “differenziale”

- Si tratta di una tecnica assai usata.
- Se la sequenza dei valori varia lentamente, invece di registrare i valori è sufficiente ricordarsi del valore iniziale e delle differenze successive.

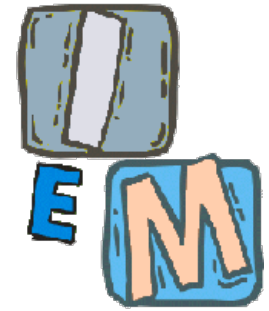
## Esempio:

134, 137, 135, 128, 130, 134, 112, ...

ricorderò il valore iniziale 134 e poi la sequenza delle differenze successive:

-3, +2, 7, -2, -4, 22, ...

- Si dimostra sperimentalmente che per le immagini la sequenza delle differenze ha una entropia minore di quella dei valori originali e quindi richiede meno bit per essere memorizzata.



# Compressione Lossy



# Compressione LOSSY

Si parla di compressione **LOSSY** quando i dati possono essere trasformati in modo da essere memorizzati con risparmio di memoria ma con perdita di informazione.

Tale tipo di compressione produce un maggiore risparmio di memoria!

# La compressione lossy: ridondanza nei dati

*Nl mzz dl cmmn d nstr vt*  
*M rtrv pr n slv scr*

Gli uomini hanno grande flessibilità nell'interpretare correttamente segnali rumorosi o incompleti ...

Alcuni linguaggi NON prevedono affatto che si scrivano le vocali!

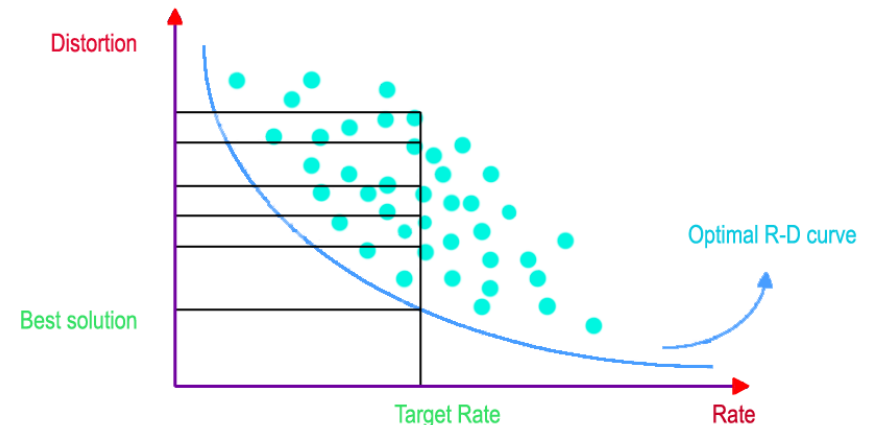
Se si ha una “competenza” spesso si possono completare da soli i dettagli che non sono stati trasmessi in maniera completa (esempio di ridondanza semantica).

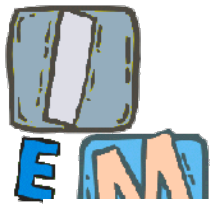
Alcuni telefonini fanno “scegliere” quando si compone un SMS quale è la sequenza di caratteri più “probabile” nella nostra lingua (esempio di ridondanza statistica).



# Criterio per una buona compressione di tipo lossy

- Fissata la massima distorsione accettabile l'algoritmo di compressione deve trovare la rappresentazione con il più basso numero di bit.
- Viceversa, fissato il massimo numero di bit accettabile occorre trovare il miglior algoritmo di compressione che a parità di numero di bit mi dia la minima distorsione





## Per le immagini:



Una di queste immagini richiede su disco 26 Kb e una 8 Kb.

Quale è l'una o l'altra? Il nostro occhio non coglie alcuni dettagli e il nostro cervello “integra” l'informazione mancante o corrotta.



# Idea della compressione lossy!

- **Se “percettivamente” non è importante: buttalo via!**
  - ❑ **MP3** : applicano questa idea al caso del suono e della musica;
  - ❑ **JPEG** : applicano questa idea alle immagini fisse (still images)
  - ❑ **MPEG, AVI, DVX etc**: applicano questa idea alle sequenze di immagini (filmati)

**Ovviamente una volta buttata via l'informazione non può essere ricostruita: si tratta di compressione IRREVERSIBILE.**





# In questo corso vedremo soltanto:

- Un algoritmo di requantization;
- Il JPEG.



# Un algoritmo lossy: requantization

- Si tratta molto semplicemente di una riduzione del numero di livelli disponibili in modo da risparmiare bit per pixel.
- La si realizza “dimenticando” n bit meno significativi per canale.
- Per esempio:
  - RED: da 8 bit si conservano solo i 4 più significativi;
  - GREEN: da 8 bit si conservano solo i 6 più significativi;
  - BLUE: da 8 bit a si conservano solo i 2 più significativi.

**Si risparmia così il 50% dei bit inizialmente necessario!**

**In più: se ci sono meno simboli ... la compressione LZW o Huffman è più efficiente!**

**Ma che perdita di qualità nelle immagini!**



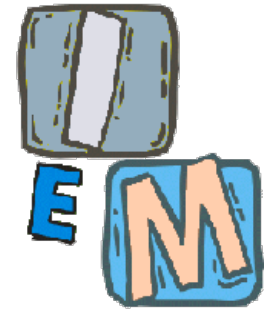
# Requantization

originale



requantization

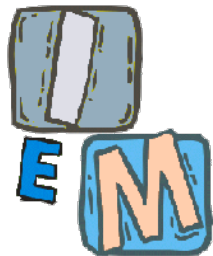




---

Lo standard JPEG

---



# Storia

- ▶ JPEG è l'acronimo di “Joint Photographic Experts Group”.  
([www.jpeg.org](http://www.jpeg.org))
- ▶ lo standard JPEG è stato sviluppato per la compressione di immagini.
- ▶ Nel 1988, JPEG ha scelto uno schema di codifica adattativo basato sulla tecnica DCT (Discrete Cosine Transform)
- ▶ Nel 1991 è stata presentata ufficialmente una proposta di standard che è stata approvata dai membri del consorzio ed è diventata uno standard ISO nel 1992 (ISO –International Standard Organization).

# Passi fondamentali della codifica JPEG

## ► Pre-processing:

- i. Color Transform ( $RGB \rightarrow YC_bC_r$ );
- ii. Sottocampionamento della cromaticanza
- iii. Suddivisione della immagine in sottoimmagini.

## ► Trasformazione:

- i. Discrete Cosine Transform;
- ii. Quantization;

## ► Codifica:

- i. DC Coefficient Encoding;
- ii. Zig-zag ordering of AC Coefficients;
- iii. Entropy Coding (Huffman).

## Preprocessing (i): da RGB a Y C<sub>b</sub> C<sub>r</sub>

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

La Y è la luminanza, gli altri due canali codificano i colori.

Si tratta di una trasformazione reversibile.

Il modello Y C<sub>b</sub> e C<sub>r</sub> è un modello di colori che permette di avvantaggiarsi della “debolezza” del sistema visivo umano.

Infatti...



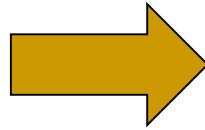
## Preprocessing (ii): sottocampionamento della crominanza

- L'occhio umano è più sensibile alla luminanza che alla crominanza.
- JPEG prende **TUTTE** le informazioni sulla luminanza ma sceglie **solo un campione** delle informazioni degli altri canali.
- **Si sceglie** 1 valore ogni 4 per  $C_b$  e  $C_r$  (*tralasciando metà dei valori originari da comprimere*)
- **Questo passo è ovviamente con perdita di informazione ed è irreversibile!**





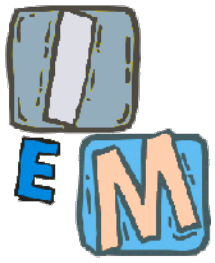
**Da RGB a YCbCr**



**Prendo tutti i pixel  
del canale Y**

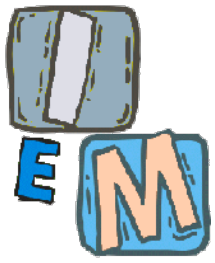


**Prendo solo  
un pixel ogni  
quattro per i  
canali  $C_b$  e  $C_r$**



## Preprocessing (iii): partizione della immagine

- Per approfittare al meglio della ridondanza (che localmente è maggiore che sulla intera immagine) e per semplificare i calcoli, JPEG procede dividendo l'immagine in “quadrotti” **8 x 8** di 64 pixel non sovrapposti.
- “Quadrotti” diversi subiranno una elaborazione differente: è qui l'origine del noto problema “quadrettatura” spesso visibile in ingrandimenti o stampe di immagini che sono state compresse con JPEG.

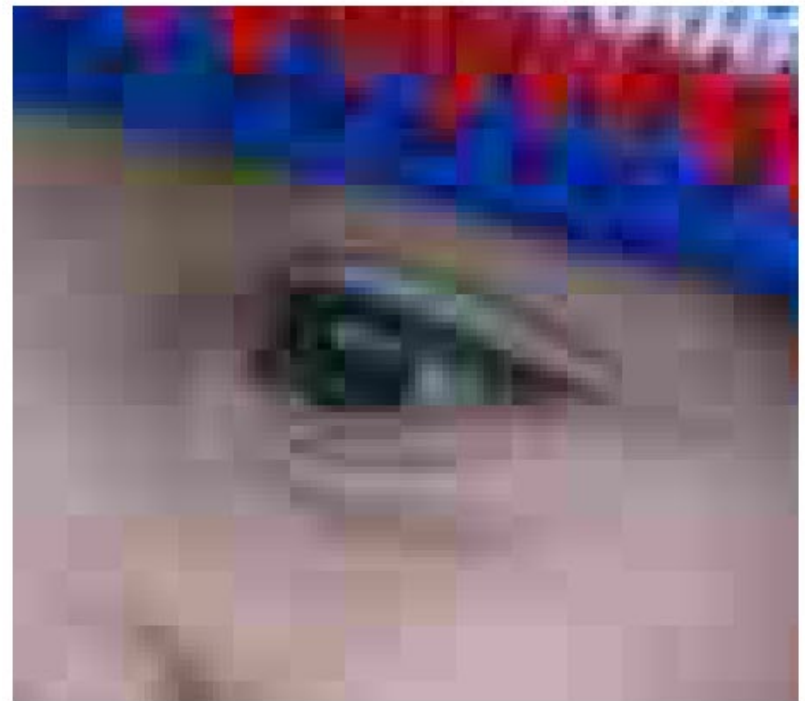


La «quadrettatura» è molto evidente se la compressione è elevata.





La «quadrettatura» è molto evidente se la compressione è elevata.





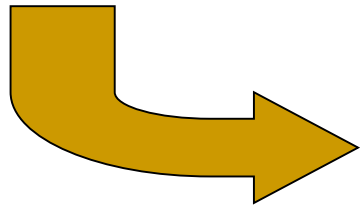
# Prima della DCT

- Prima della applicazione della DCT ai 64 pixel di ciascun blocco viene sottratta una quantità pari a  $2^{n-1}$ , dove  $2^n$  rappresenta il numero massimo di livelli di grigio dell'immagine.
- Se il blocco considerato presenta  $256 = 2^8$  possibili livelli di grigio, a ciascun pixel di tale blocco verrà sottratto un offset pari a  $128 = 2^7$ .
- Con questo processo, noto come **shift dei livelli di grigio**, il grigio medio (128) diventa 0.

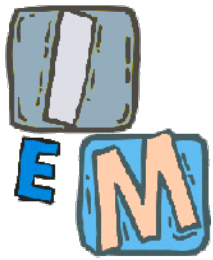


# Esempio

52	55	61	66	70	61	64	73
63	59	66	90	109	85	69	72
62	59	68	113	144	104	66	73
63	58	71	122	154	106	70	69
67	61	68	104	126	88	68	70
79	65	60	70	77	68	58	75
85	71	64	59	55	61	65	83
87	79	69	68	65	76	78	94



-76	-73	-67	-62	-58	-67	-64	-55
-65	-69	-62	-38	-19	-43	-59	-56
-66	-69	-60	-15	16	-24	-62	-55
-65	-70	-57	-6	26	-22	-58	-59
-61	-67	-60	-24	-2	-40	-60	-58
-49	-63	-68	-58	-51	-60	-70	-53
-43	-57	-64	-69	-73	-67	-63	-45
-41	-49	-59	-60	-63	-52	-50	-34



# Trasformazione (i): la DCT

- Il JPEG trasforma i blocchi 8 x 8 di pixel secondo un algoritmo detto della Trasformata Discreta del Coseno (**Discrete Cosine Transform**, DCT)
- Si tratta di un algoritmo della famiglia delle trasformate di Fourier.
- E' stato dimostrato che, statisticamente, **tale trasformazione “decorrela” al massimo i dati permettendo maggiori rapporti di compressione nella fase successiva di codifica.**
- Il nome non deve confondere: si tratta di una trasformazione del “vettore” di 64 pixel dalla base impulsiva (canonica) ad una più adatta alle immagini (vedi slide successiva)



# Trasformazione (i): la DCT

- Come osservato una immagine di 8 x 8 pixel si può pensare come un vettore nello spazio a 64 dimensioni.
- Ogni immagine è quindi la somma pesata di 64 immagini impulsive (tutte nere tranne in un pixel di valore 1) ove i “pesi” rappresentano l’effettivo livello di luminosità di ogni pixel.
- Tali immagini impulsive costituiscono una base, detta “base impulsiva” per le immagini.
- La “base impulsiva” non è l’unica base. La trasformata del coseno esprime l’immagine in un’altra base.





La formula della DCT per un blocco di dimensioni  $N \times N$  ( $N=8$  nel jpeg)

$$F(u, v) = \frac{2}{N} \left[ \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} C(u)C(v)f(x, y) \cos \frac{(2x+1)u\pi}{2 * N} \cos \frac{(2y+1)v\pi}{2 * N} \right]$$
$$f(x, y) = \frac{2}{N} \left[ \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u, v) \cos \frac{(2x+1)u\pi}{2 * N} \cos \frac{(2y+1)v\pi}{2 * N} \right]$$

where :

$$C(u) = \frac{1}{\sqrt{2}} \text{ for } u = 0; C(u) = 1 \text{ otherwise}$$

$$C(v) = \frac{1}{\sqrt{2}} \text{ for } v = 0; C(v) = 1 \text{ otherwise}$$

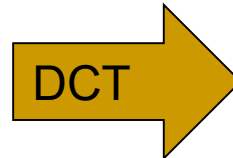
**Una implementazione diretta delle formule sopra richiede  $O(N^2)$  Esistono algoritmi “fast” per calcolare i coefficienti in  $O(N \log(N))$  derivati dalla Fast Fourier Transform.**



# Esempio

- Dopo l'applicazione della DCT, i nostri coefficienti diventano:

-76	-73	-67	-62	-58	-67	-64	-55
-65	-69	-62	-38	-19	-43	-59	-56
-66	-69	-60	-15	16	-24	-62	-55
-65	-70	-57	-6	26	-22	-58	-59
-61	-67	-60	-24	-2	-40	-60	-58
-49	-63	-68	-58	-51	-60	-70	-53
-43	-57	-64	-69	-73	-67	-63	-45
-41	-49	-59	-60	-63	-52	-50	-34



-415	-29	-62	25	55	-20	-1	3
7	-21	-62	9	11	-7	-6	6
-46	8	77	-25	-30	10	7	-5
-50	13	35	-15	-9	6	0	3
11	-8	-13	-2	-1	1	-4	1
-10	1	3	-3	-1	0	2	-1
-4	-1	2	-1	2	-3	1	-2
-1	-1	-1	-2	-1	-1	0	-1



# Esempio

- E sono i coefficienti che devono essere moltiplicati alle basi della DCT (NON a quella canonica!) per ottenere il blocco precedente.

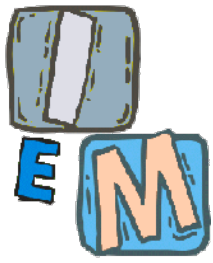
-415	-29	-62	25	55	-20	-1	3
7	-21	-62	9	11	-7	-6	6
-46	8	77	-25	-30	10	7	-5
-50	13	35	-15	-9	6	0	3
11	-8	-13	-2	-1	1	-4	1
-10	1	3	-3	-1	0	2	-1
-4	-1	2	-1	2	-3	1	-2
-1	-1	-1	-2	-1	-1	0	-1



# Per la prima riga si ottiene:

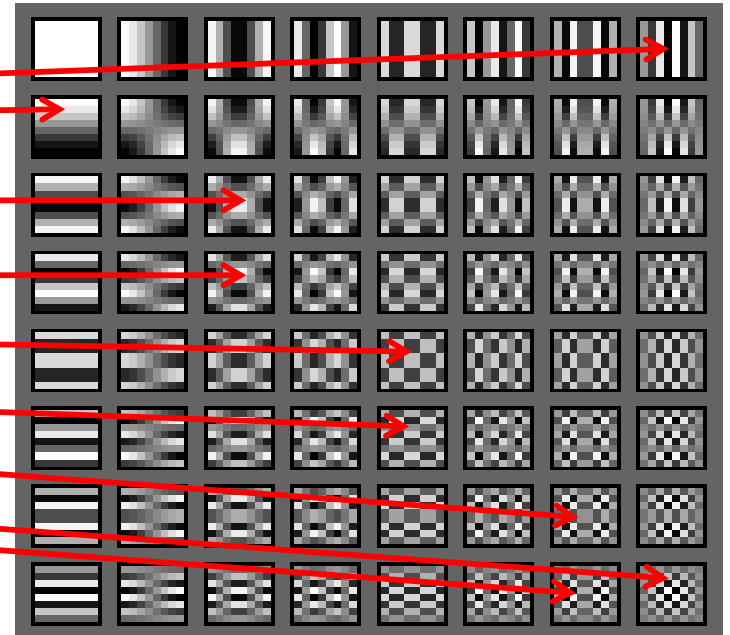
$$\begin{aligned} & -415 * \begin{array}{|c|} \hline \text{L-shaped corner} \\ \hline \end{array} - 29 * \begin{array}{|c|} \hline \text{8 vertical bars of varying shades} \\ \hline \end{array} - 62 * \begin{array}{|c|} \hline \text{4 vertical bars of varying shades} \\ \hline \end{array} + 25 * \begin{array}{|c|} \hline \text{4 vertical bars of varying shades} \\ \hline \end{array} \\ & + 55 * \begin{array}{|c|} \hline \text{4 vertical bars of varying shades} \\ \hline \end{array} - 20 * \begin{array}{|c|} \hline \text{8 vertical bars of varying shades} \\ \hline \end{array} - 1 * \begin{array}{|c|} \hline \text{4 vertical bars of varying shades} \\ \hline \end{array} + 3 * \begin{array}{|c|} \hline \text{8 vertical bars of varying shades} \\ \hline \end{array} + \dots \end{aligned}$$

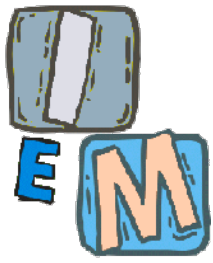
- Ognuna delle basi (cioè delle piccole immagini riportate sopra) è grande 8x8 pixel



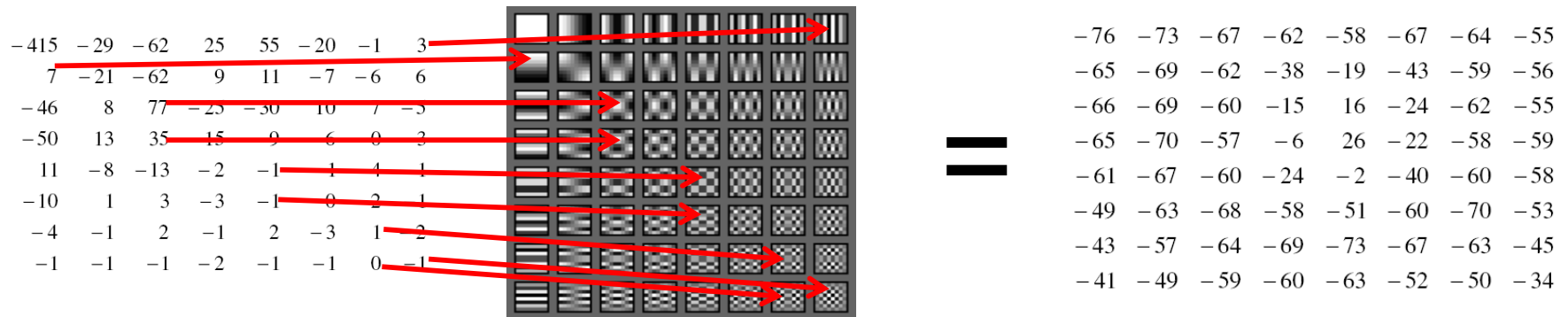
Lo stesso calcolo va ripetuto per tutti gli elementi della matrice.

-415	-29	-62	25	55	-20	-1	3
7	-21	-62	9	11	-7	-6	6
-46	8	77	-25	-30	10	7	-5
-50	13	35	15	9	6	0	3
11	-8	-13	-2	-1	1	4	1
-10	1	3	-3	-1	0	2	-1
-4	-1	2	-1	2	-3	1	-2
-1	-1	-1	-2	-1	-1	0	-1





# Quindi, se volessimo tornare indietro:




Somma di prodotti tra 64 coefficienti e 64 basi della DCT di 8x8 elementi ciascuno!



# Risultato della DCT

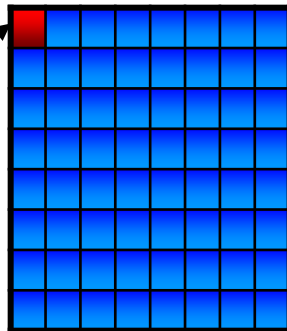
**DC Coefficient**



-415	-29	-62	25	55	-20	-1	3
7	-21	-62	9	11	-7	-6	6
-46	8	77	-25	-30	10	7	-5
-50	13	35	-15	-9	6	0	3
11	-8	-13	-2	-1	1	-4	1
-10	1	3	-3	-1	0	2	-1
-4	-1	2	-1	2	-3	1	-2
-1	-1	-1	-2	-1	-1	0	-1

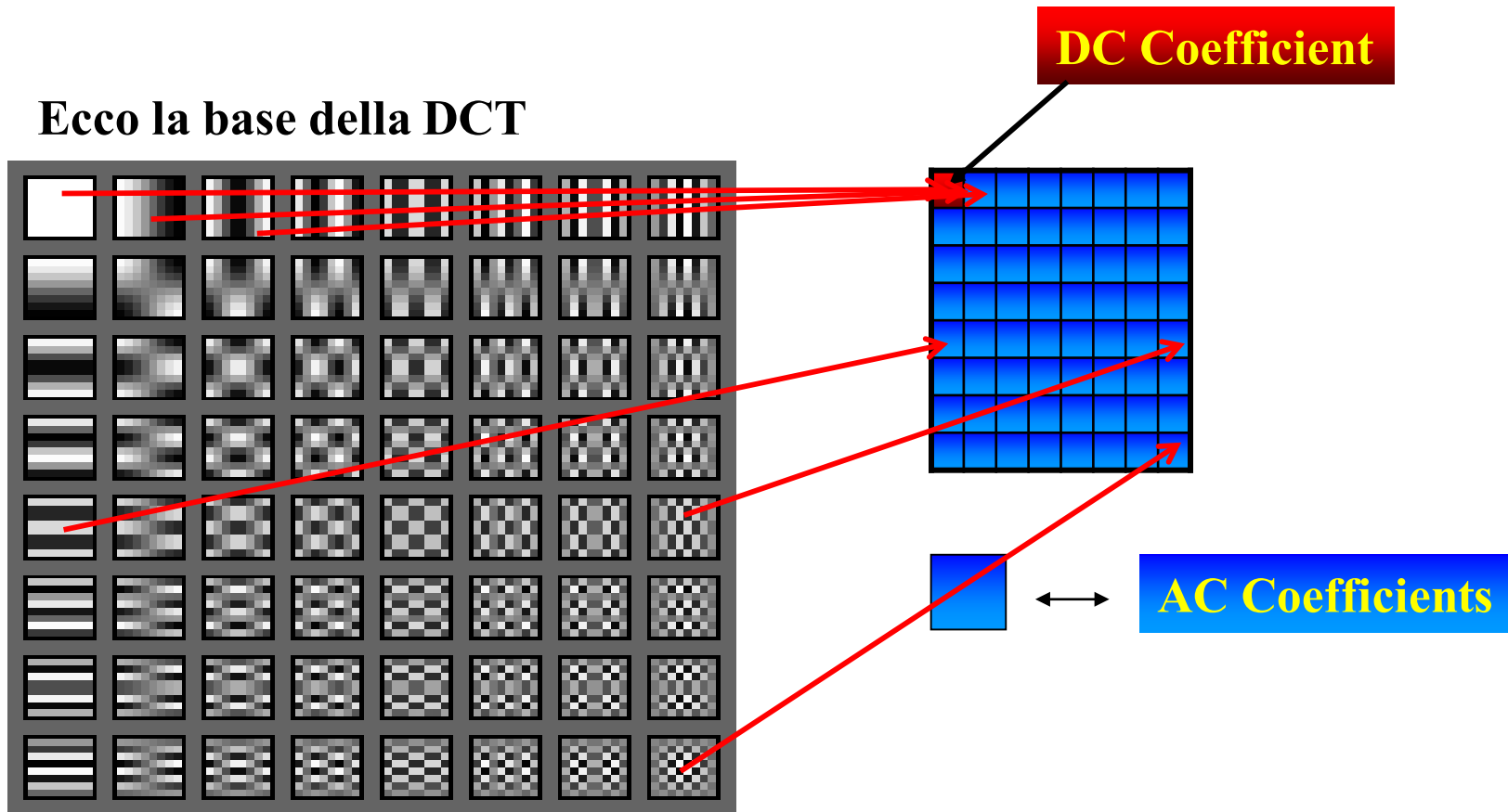
- Il coefficiente DC è solo il primo in alto a sinistra.
- Tutti gli altri 63 coefficienti sono detti coefficienti AC.

**DC Coefficient**



**AC Coefficients**

Ecco la base della DCT



Ogni immagine 8 x 8 si ottiene moltiplicando ciascuna delle immagini a sinistra per un coefficiente e sommando tutte le immagini.

I coefficienti di tale somma sono i coefficienti della DCT.

Il coefficiente in alto a sinistra è un valore proporzionale al valor medio della luminanza dell'immagine. E' detto anche coefficiente DC





## Trasformazione (ii): Quantizzazione

Un vantaggio in termini di simboli da usare (e quindi in termini di lunghezza dei codici Huffman) si ottiene se si riduce il numero di “livelli” su cui i coefficienti della DCT possono variare.

Tale operazione permette di rappresentare i diversi coefficienti incrementando il fattore di compressione, più precisamente avviene un processo di riduzione del numero di bit necessari per memorizzare un valore intero riducendone la precisione.



## Trasformazione (ii): Quantizzazione

- Si usa il seguente “formalismo” per la quantizzazione. Dato un fattore di quantizzazione  $Q$  e un numero  $F$  il valore  $F_{\text{quantizzato}}$  si ottiene come:

$$F_{\text{quantizzato}} = \text{round}(F/Q)$$

- Il valore ricostruito si ottiene moltiplicando  $F_{\text{quantizzato}}$  per  $Q$ .
- Ovviamente la quantizzazione è un processo irreversibile (perdita di informazione)



## Trasformazione (ii): Quantizzazione

Si è scoperto sperimentalmente che **non è conveniente usare un unico fattore di quantizzazione per tutti i 64 coefficienti della DCT della luminanza, o per quantizzare i valori provenienti dalla DCT delle crominanze.**

Si preferisce adottare per il coefficiente  $F(i,j)$  un fattore di quantizzazione  $Q(i,j)$  scelto a priori (fornito dallo standard) o scelto dall'utente (*in questo caso la tabella di quantizzazione deve essere trasmessa assieme ai dati compressi per consentire una corretta ricostruzione*). I fattori  $Q(i,j)$  costituiscono la cosiddetta “tabella di quantizzazione”.

Nella successiva slide le due tabelle di fattori di quantizzazione standard

**Tabella di quantizzazione luminanza**

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

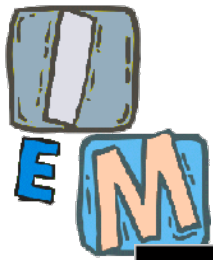
**Tabella di quantizzazione crominanza**

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

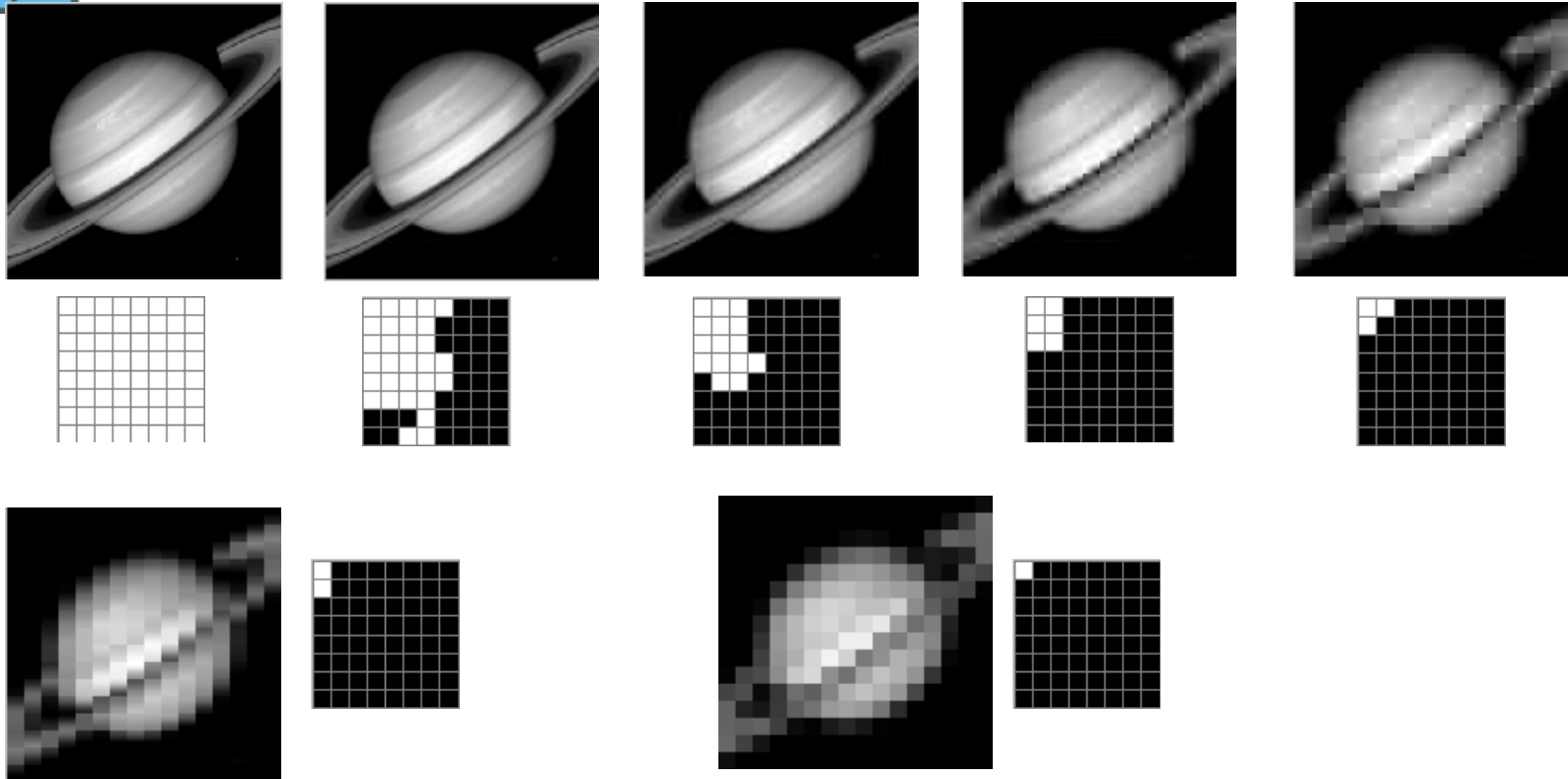
**Si osservi che un fattore di compressione maggiore comporta una maggiore perdita di informazione.**

**L'utente del JPEG può scegliere il “grado” di quantizzazione da adottare fornendo un “**quality factor**” **QF** che va da 1 a 100. La tabella di quantizzazione adottata sarà una copia delle tabelle sopra i cui elementi sono divisi per QF.**

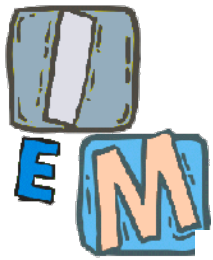
**Maggiore il QF, minore i fattori di quantizzazione e minore la perdita di informazioni.**



# Effetto della quantizzazione



**I quadretti neri rappresentano i coefficienti DCT che la quantizzazione ha portato a zero per ogni blocco 8x8 della immagine di Saturno.**



# Nel nostro esempio

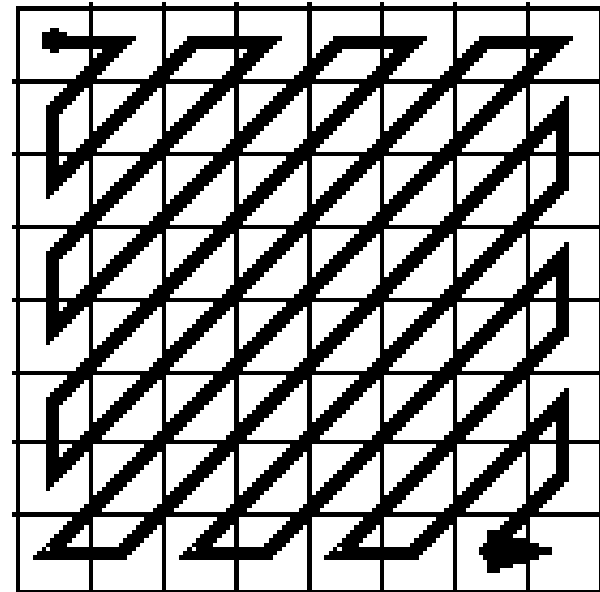
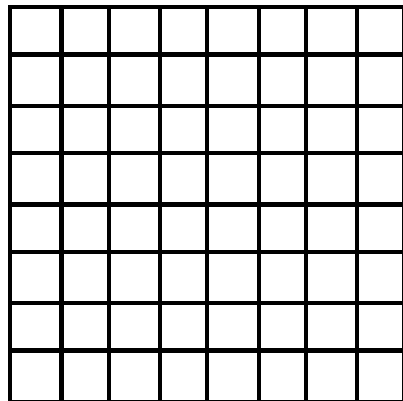
-415	-29	-62	25	55	-20	-1	3		16	11	10	16	24	40	51	61
7	-21	-62	9	11	-7	-6	6		12	12	14	19	26	58	60	55
-46	8	77	-25	-30	10	7	-5		14	13	16	24	40	57	69	56
-50	13	35	-15	-9	6	0	3		14	17	22	29	51	87	80	62
11	-8	-13	-2	-1	1	-4	1		18	22	37	56	68	109	103	77
-10	1	3	-3	-1	0	2	-1		24	35	55	64	81	104	113	92
-4	-1	2	-1	2	-3	1	-2		49	64	78	87	103	121	120	101
-1	-1	-1	-2	-1	-1	0	-1		72	92	95	98	112	100	103	99

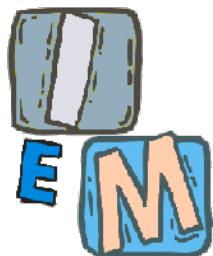
	-26	-3	-6	2	2	0	0	0
	1	-2	-4	0	0	0	0	0
	-3	1	5	-1	-1	0	0	0
	-4	1	2	-1	0	0	0	0
	1	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0



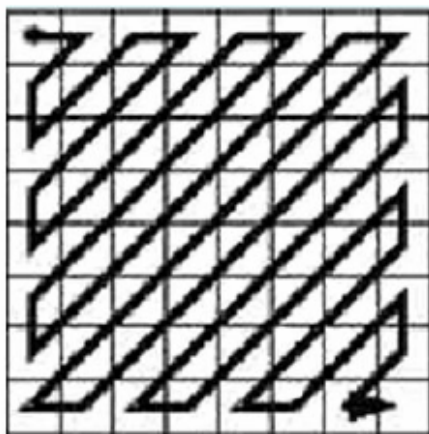
# Codifica

- Tutti i coefficienti vengono riordinati in un vettore  $64 \times 1$  seguendo l'ordinamento "a serpentina" (per creare lunghe run di zeri) e codificati in un altro stream.





# Codifica zig-zag



ordinamento a zigzag

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

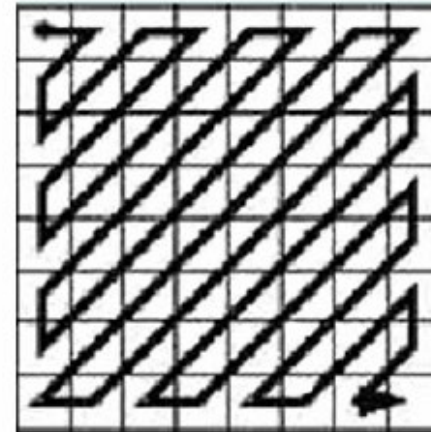
Gli indici che determinano  
l'ordinamento a zig-zag  
dei coefficienti quantizzati





# Nel nostro esempio

-26	-3	-6	2	2	0	0	0
1	-2	-4	0	0	0	0	0
-3	1	5	-1	-1	0	0	0
-4	1	2	-1	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



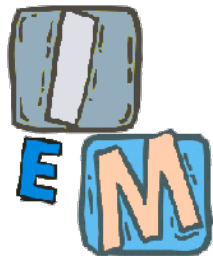
ordinamento a zigzag

-26 -3 1 -3 -2 -6 2 -4 1 -4 1 1 5 0 2 0 0 -1 2 0 0 0 0 0 -1 -1 EOB



# Differenti codifiche

- A questo punto si hanno due differenti codifiche.
- I coefficienti DC, cioè quelli che stanno nella posizione (1,1) del blocco 8x8, sono codificati usando una codifica differenziale;
- I coefficienti AC, cioè tutti gli altri del blocco, sono codificati usando una codifica run-length.
- Le tabelle usate di seguito forniscono i codici di Huffman ottenuti sulla base di calcoli statistici preventivi e sono fornite dallo standard.



# Codifica coefficienti DC

- I coefficienti del blocco 8x8 messi in sequenza sono:

-26 -3 1 -3 -2 -6 2 -4 1 -4 1 1 5 0 2 0 0 -1 2 0 0 0 0 0 -1 -1 EOB

- Il coefficiente DC è il primo e vale -26.
- Assumendo che il coefficiente DC del blocco successivo sia -17, otterremo l'**evento**  
 $\Delta = -26 - (-17) = -9$



# Codifica coefficienti DC

- Nella tabella delle categorie, -9 sta nella posizione  $n = SSSS = 4$ .

SSSS	$\Delta$
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

SSSS = categoria

$\Delta$  = differenza tra due coefficienti DC (evento)

n.b.: le tabelle complete sono disponibili su teams



# Codifica coefficienti DC

- Il valore  $n=SSSS=4$  ha come codice base 101. Ed  $n$  è anche il numero di bit mancanti che occorre aggiungere.
- La corrispondenza tra il valore e il codice è fissata dalla tabella dei codici di Huffman che varia in base al fatto che stiamo trattando la luminanza o la cromaticità.

SSSS	Codice base
0	010
1	011
2	100
3	00
4	101
5	110
6	1110
7	11110
8	111110
9	1111110
10	11111110
11	111111110

n.b.: le tabelle complete sono disponibili su teams



# Codifica coefficienti DC

- A questo punto occorre completare il codice.
- Per fare ciò si usa la seguente regola:
- Se  $\Delta > 0$  allora i bit da aggiungere sono gli  $n$  bit meno significativi del valore  $\Delta$  in binario.
- Se  $\Delta < 0$  allora i bit da aggiungere sono gli  $n$  bit meno significativi del valore in binario di  $\Delta$  (con complemento a due) ai quali occorre sottrarre il valore 1.
- $\Delta = 0$  allora anche SSSS è uguale a zero, pertanto non viene aggiunto alcun bit.



# Codifica coefficienti DC

- Nell'esempio considerato i quattro bit meno significativi del valore in binario di  $\Delta$  (-9) sono 0111; essendo  $\Delta < 0$  si sottrae il valore 1 e si ottengono i quattro bit 0110 che completano il codice base trovato in precedenza (101).
- Il codice completo del coefficiente DC è 1010110.



# Codifica coefficienti AC

-3 1 -3 -2 -6 2 -4 1 -4 1 1 5 0 2 0 0 -1 2 0 0 0 0 0 -1 -1 EOB

- Dalla sequenza si è eliminato il primo coefficiente e si passa alla codifica di tutti gli altri.
- Poiché i coefficienti quantizzati AC sono spessissimo nulli, si usa una trasformazione in skip-value. Cioè, data una sequenza di valori, si memorizza il numero degli zeri seguito dal primo valore non zero che si incontra.
- Esempio: si debba codificare  
0,0,0,0,11,0,0,0,3,0,0,0,0,0,0,0,12,17...  
memorizzo:  
(4,11),(3,3),(8,12),(0,17),...



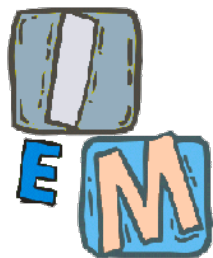


## Nel nostro caso

-3 1 -3 -2 -6 2 -4 1 -4 1 1 5 0 2 0 0 -1 2 0 0 0 0 0 -1 -1 EOB

### ■ Nel nostro caso sarebbe

$(0, -3), (0, 1), (0, -3), (0, -2), (0, -6), (0, 2), (0, -4),$   
 $(0, 1), (0, -4), (0, 1), (0, 1), (0, 5), (1, 2), (2, -1),$   
 $(0, 2), (5, -1), (0, -1) \dots$



# Codifica coefficienti AC

- La prima coppia del tipo  $(0,v)$  da codificare è  $(0,-3)$ .
- Il coefficiente  $-3$  ha come SSSS il valore 2.

SSSS	Coefficiente AC
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
A	-1023 ... -512, 512 ... 1023

n.b.: le tabelle complete  
sono disponibili su  
teams

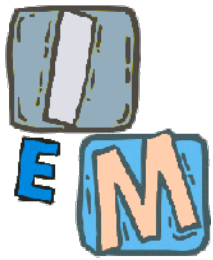


# Codifica coefficienti AC

- La classe dell'evento è espressa mediante una coppia del tipo (run, categoria).
- Nel nostro caso abbiamo (0,2)

(run, categoria)	Codice base	Lunghezza codice completo
(0, 0)	1010 (= EOB)	4
(0, 1)	00	3
(0, 2)	01	4
(0, 3)	100	6
....	....	....
(F, 0)	111111110111	12
....	....	....
(F, A)	1111111111111110	26

n.b.: le tabelle complete sono disponibili su teams



# Codifica coefficienti AC

- Alla coppia (0, 2) corrisponde il codice base 01, tale codice sarà completato dall'aggiunta di un numero di bit fino a raggiungere il numero totale di bit che è riportato nell'ultima colonna della tabella.
- I bit che completano il codice base sono scelti con lo stesso criterio enunciato per la codifica dei coefficienti DC, in base al valore  $v$  del coefficiente AC ( $v > 0$ ,  $v < 0$ ,  $v = 0$ ).
- Se  $v > 0$  allora i bit da aggiungere sono i bit meno significativi del valore  $v$  in binario.
- Se  $v < 0$  allora i bit da aggiungere sono i bit meno significativi del valore in binario di  $v$  (con complemento a due) ai quali occorre sottrarre il valore 1.
- $v = 0$  allora anche SSSS è uguale a zero, pertanto non viene aggiunto alcun bit.



# Codifica coefficienti AC

- Nell'esempio considerato i due bit meno significativi del valore in binario del coefficiente AC  $v$  (-3) sono 01, essendo  $v < 0$  si sottrae il valore 1 e si ottengono i due bit 00 che completano il codice base trovato in precedenza (01).
- Il codice completo è quindi 0100.



# Sequenza finale del blocco 8x8

- La sequenza finale sarà

-26 -3 1 -3 -2 -6 2 -4 1 -4 1 1 5 0 2 0 0 -1 2 0 0 0 0 0 -1 -1 EOB

1010110 0100 001 0100 0101 100001 0110 100011 001 100011 001 001 100101 11100110 110110 0110 11110100 000 1010

- Dove gli spazi sono inseriti solo per migliorare la leggibilità.
- Senza spazi la sequenza diventa (72 bit):

10101100100001010001011000010110100011001100011001001100  
101111001101101100110111101000001010

Usando 72 bit contro i 512 bit del blocco di 8 x 8 pixel da 8 bit ciascuno.



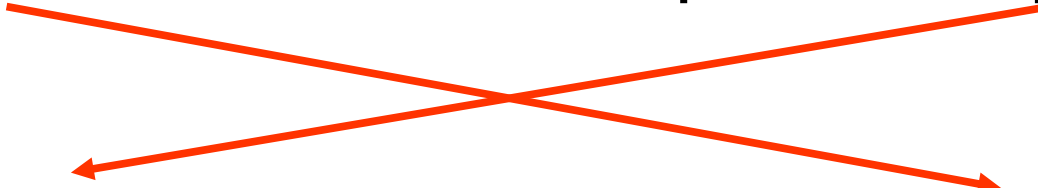
## Nella ricostruzione...

- Si deve tornare indietro “ricostruendo” i dati originali (o le loro approssimazioni per i passi irreversibili).
- Esistono diverse “strategie” per la ricostruzione in modo da abilitare la ricostruzione progressiva, gerarchica o lossless.
- Non tratteremo queste strategie in questo corso.



# Confronto tra blocchi

- Attenzione, il Jpeg è lossy! Quindi il blocco ricostruito è diverso da quello in input.



52	55	61	66	70	61	64	73
63	59	66	90	109	85	69	72
62	59	68	113	144	104	66	73
63	58	71	122	154	106	70	69
67	61	68	104	126	88	68	70
79	65	60	70	77	68	58	75
85	71	64	59	55	61	65	83
87	79	69	68	65	76	78	94

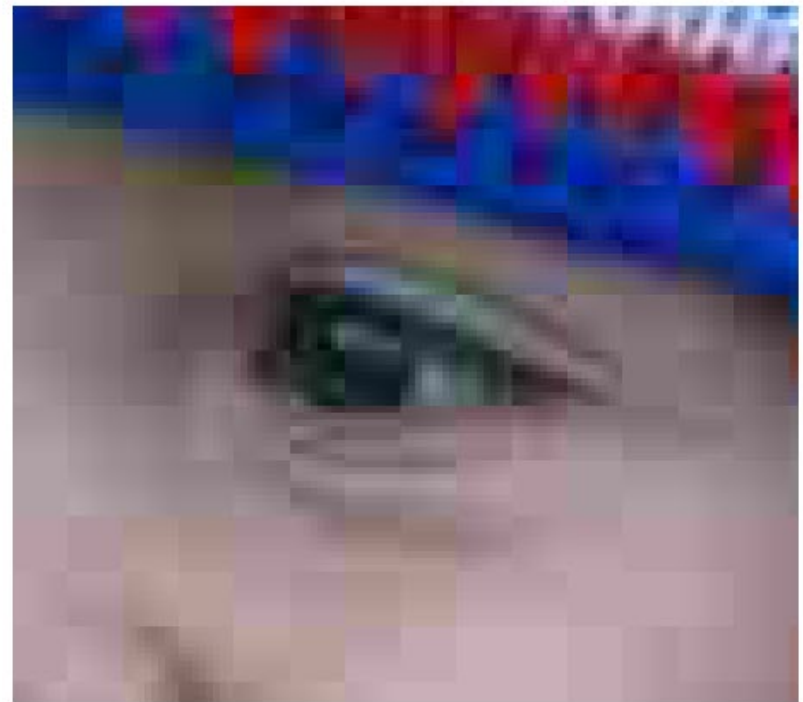
58	64	67	64	59	62	70	78
56	55	67	89	98	88	74	69
60	50	70	119	141	116	80	64
69	51	71	128	149	115	77	68
74	53	64	105	115	84	65	72
76	57	56	74	75	57	57	74
83	69	59	60	61	61	67	78
93	81	67	62	69	80	84	84





# Esempi

- Fattori di qualità pari a 1, 16

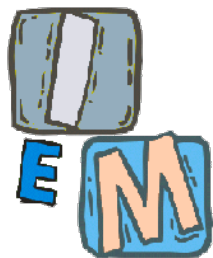




# Esempi

- Fattori di qualità pari a 50, 100





# Esempio

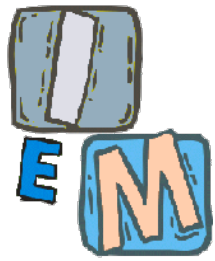
- Fattore di compressione 1, 16, 50, 100

Quiuis horrentia pilis agn  
los: aut labentis equo descri  
parum est: uel nimium si  
mo ueterum dicendi more:

Quiuis horrentia pilis agn  
los: aut labentis equo descri  
parum est: uel nimium si  
mo ueterum dicendi more:

Quiuis horrentia pilis agn  
los: aut labentis equo descri  
parum est: uel nimium si  
mo ueterum dicendi more:

Quiuis horrentia pilis agn  
los: aut labentis equo descri  
parum est: uel nimium si  
mo ueterum dicendi more:



# Esempio



**Originale**



**CR. 75:1 QF=10**



**CR. 110:1 QF=5**





# Esempio



**Original Uncompressed (3.2MB)**



**JPEG High level (179 KB)**



**JPEG Low level (15 KB)**



# Esempio



Immagini “grafiche” con pochi colori e con testi non sono compresse con buona qualità dal JPEG!

Inoltre, per il WEB, JPEG non gestisce la trasparenza (GIF lo fa)



# Nuovi standard

## **JPEG2000:**

sostituisce la DCT con le wavelets.

Alloca più bit nelle zone con più informazione e permette il controllo esplicito di tale allocazione.

Raggiunge rapporti di compressione più elevati.

Non è stato un successo commerciale (inerzia tecnologica)



# 2-D Wavelet decomposition

