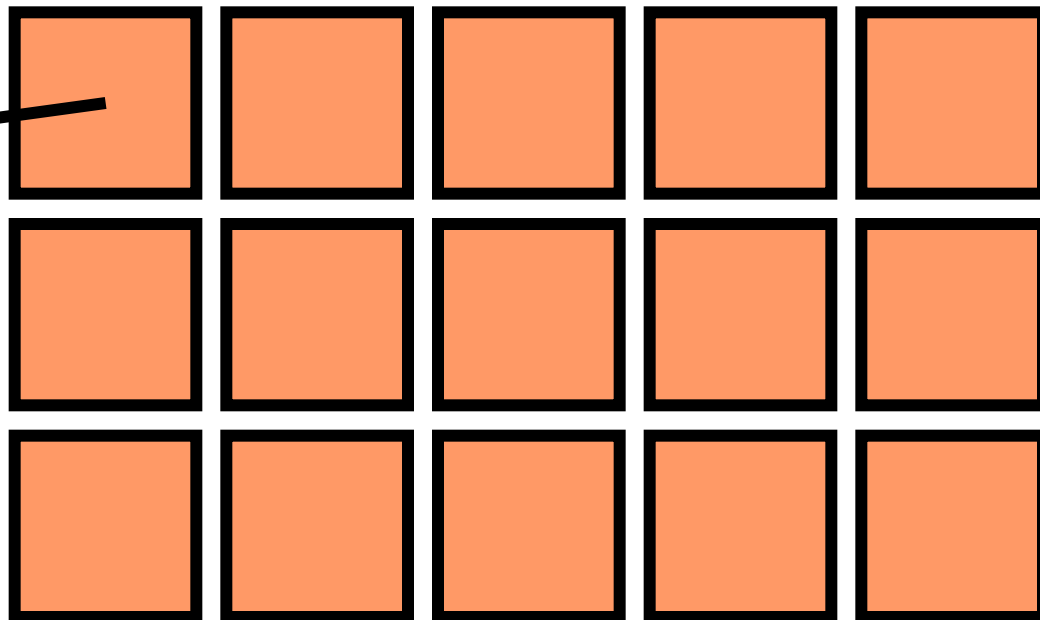


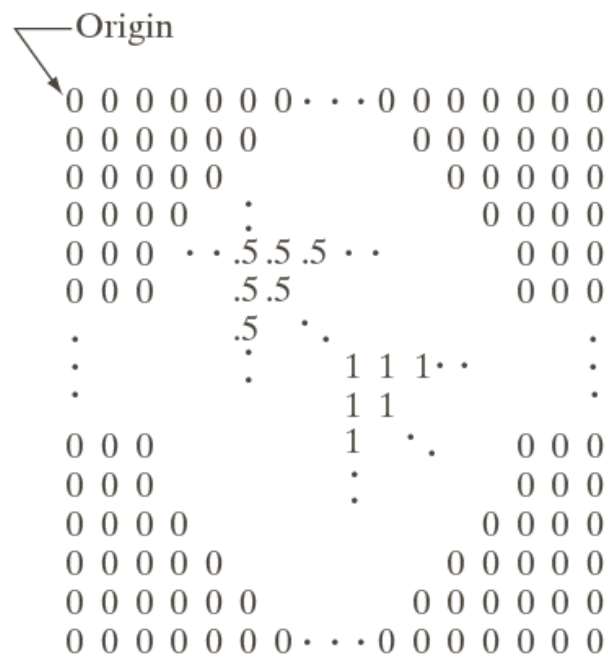
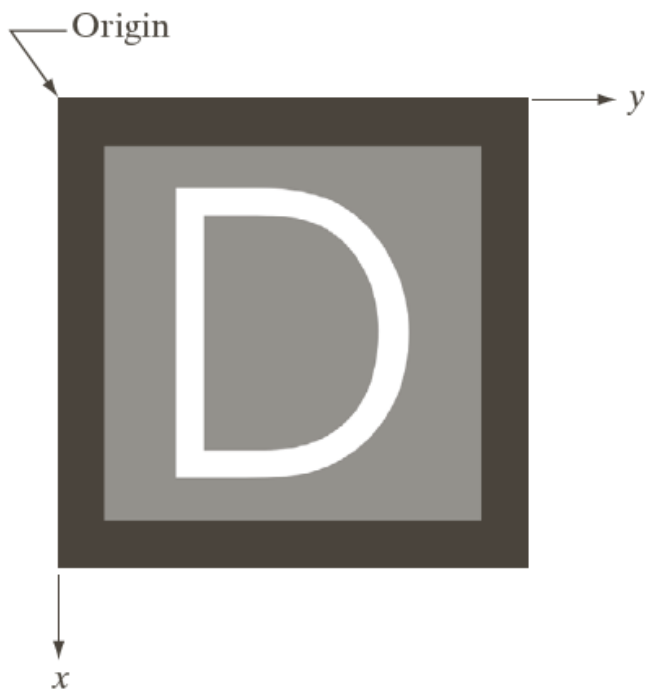
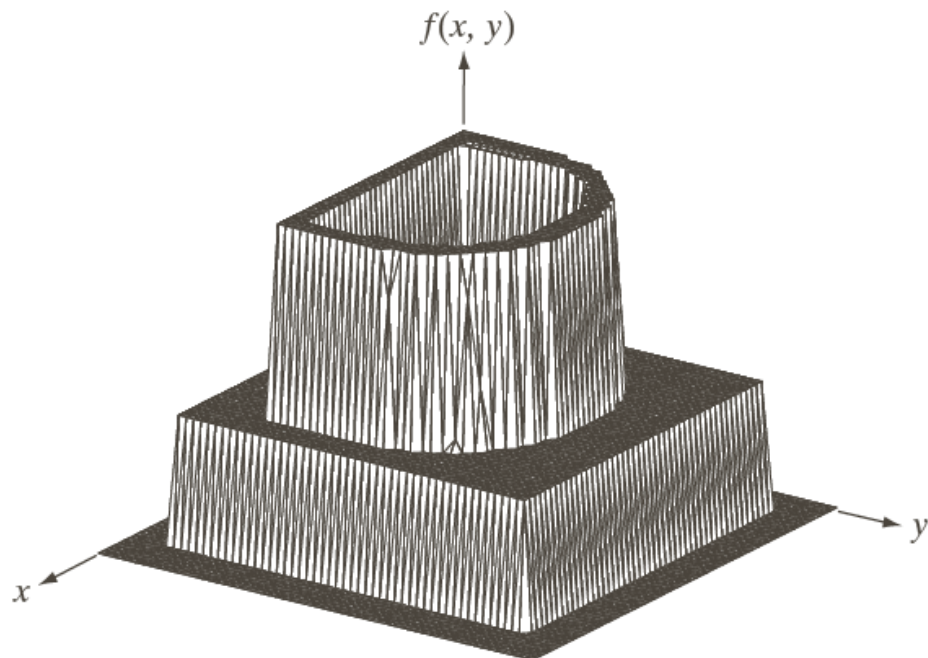


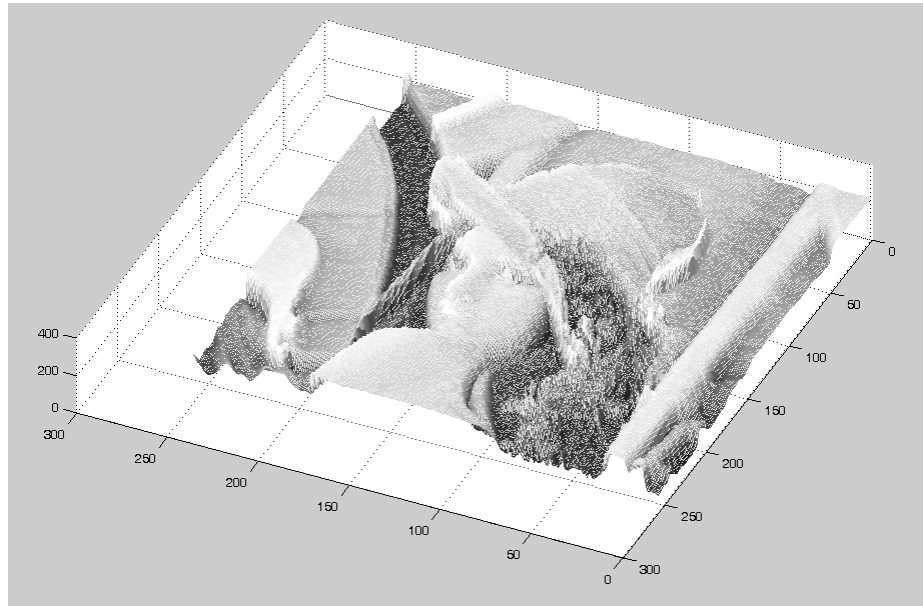
PIXEL

**Il valore quantizzato
misurato da ciascun
sensore diventa un**

**PICTURE ELEMENT =
PIXEL dell'immagine**







Command Window

File	Edit	Debug	Desktop	Window	Help
103	94	70	121	207	198
67	77	117	186	195	191
117	135	171	193	188	190
129	142	155	154	170	163
91	106	120	115	120	117
82	94	111	106	95	99
109	113	121	117	121	127
117	123	126	125	133	140
138	141	140	141	146	149
146	147	145	151	146	145
143	144	142	143	139	142
144	145	143	144	143	145
147	148	146	145	146	147
149	149	145	144	148	147
148	147	142	141	145	143
149	148	143	141	142	140
148	147	140	138	138	136
145	145	139	137	138	136
140	138	137	137	135	132
fx 137	137	137	138	134	132

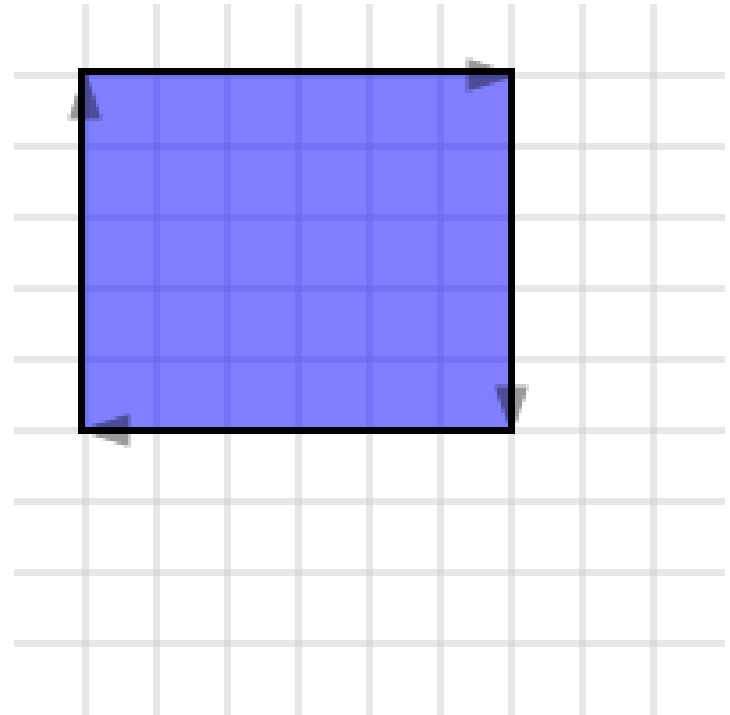
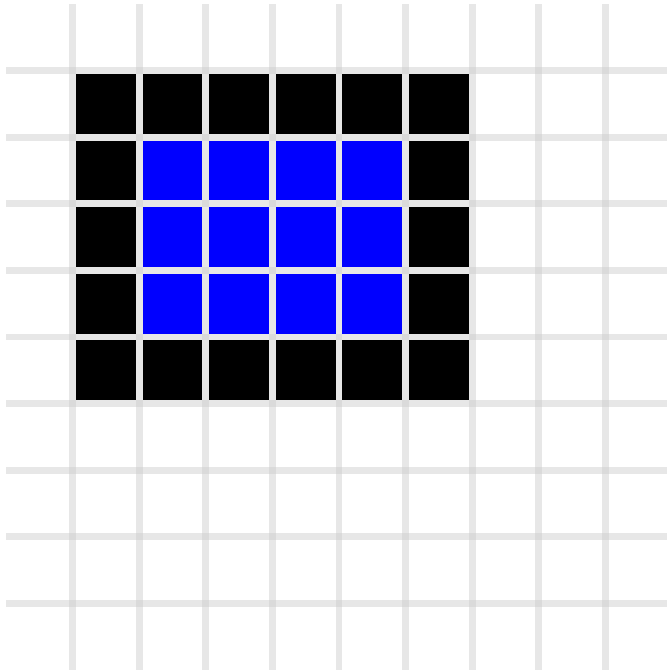
dia



Immagini Vettoriali e Raster



Formati Vettoriali





Grafica Vettoriale

- Nella grafica vettoriale un'immagine è descritta come una serie di forme geometriche.
- Piuttosto che una serie “finita” di pixel, un visualizzatore di immagini vettoriali riceve informazioni su come disegnare l'immagine sul DISPLAY DEVICE in uno specifico sistema di riferimento.
- Le immagini vettoriali possono essere stampate con strumenti appositi (plotter)



Differenze

Raster	Vector
Quali puntini devo colorare ?	Quale linee devo tracciare ?
Disegno a mano libera	Disegno tecnico



Confronto

Raster

Pro

- Fotorealismo
- Standard su Web

Contro:

- Nessuna descrizione semantica.
- Grandi dimensioni

Vector

Pro

- Le trasformazioni sul piano sono semplici (Zooming, Scaling, Rotating)

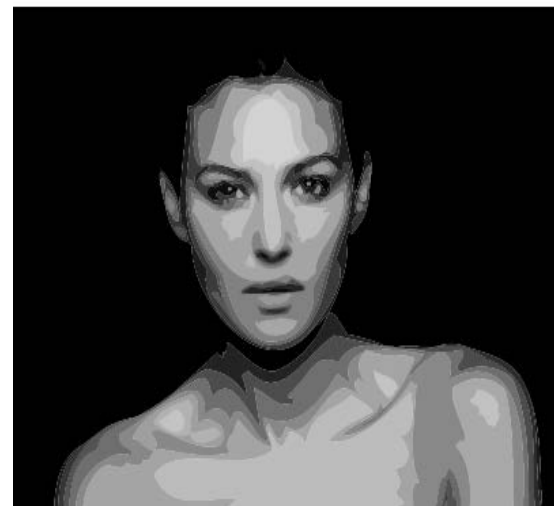
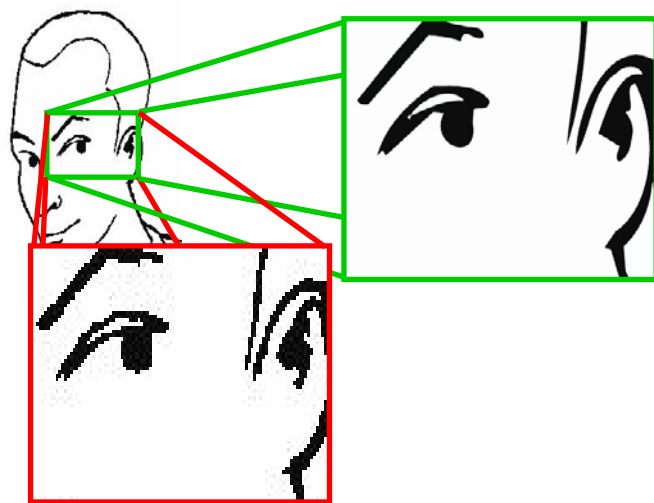
Contro:

- Non fotorealistico
- Formati vettoriali proprietari



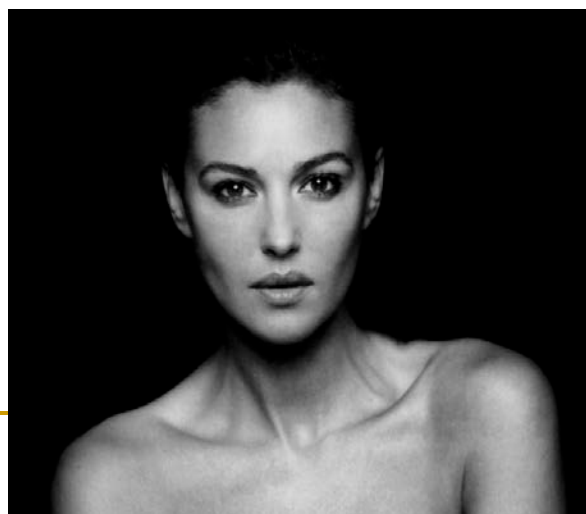
Cui prodest vector ?

- **CAD** (Computer Assisted Drawing) usa grafica vettoriale per misure precise, capacità di zoomare dentro i particolari dei progetti, ecc. (AutoCad,...)
- **Desktop Publishing & Design** (Adobe Illustrator, Macromedia Freehand, Publisher)
- Linguaggio di stampa **Postscript**
- **Animazioni** su web (Macromedia Flash)
- **GIS** (Geographical Information Systems): Arcview, Envi,...



Vettoriale

Raster





Un esempio concreto di scalabilità



Cat

Raster
(140x170)



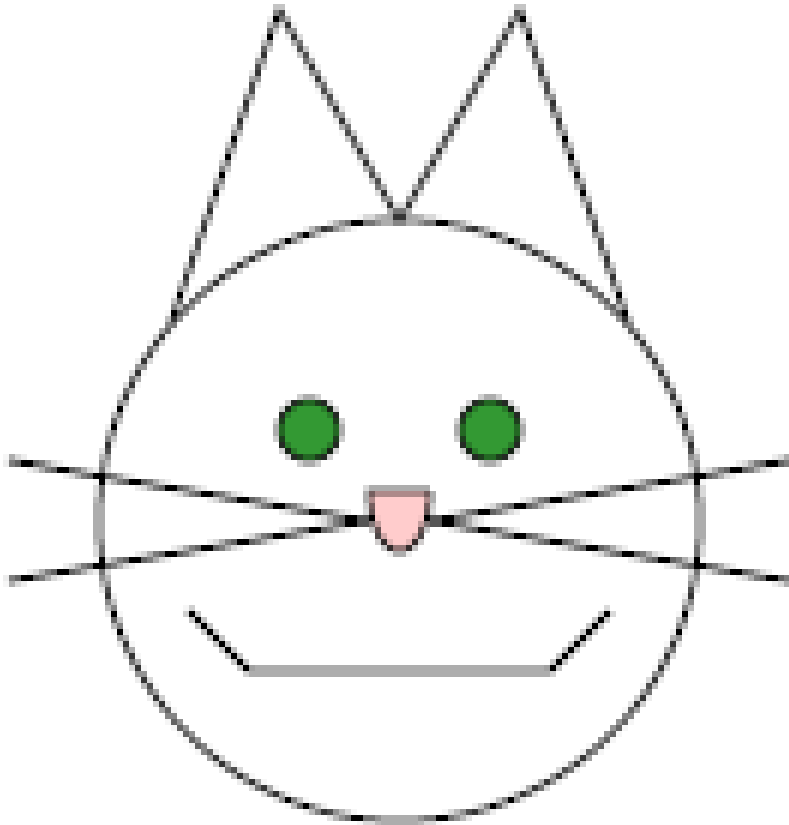
Cat

Vector
(140x170)

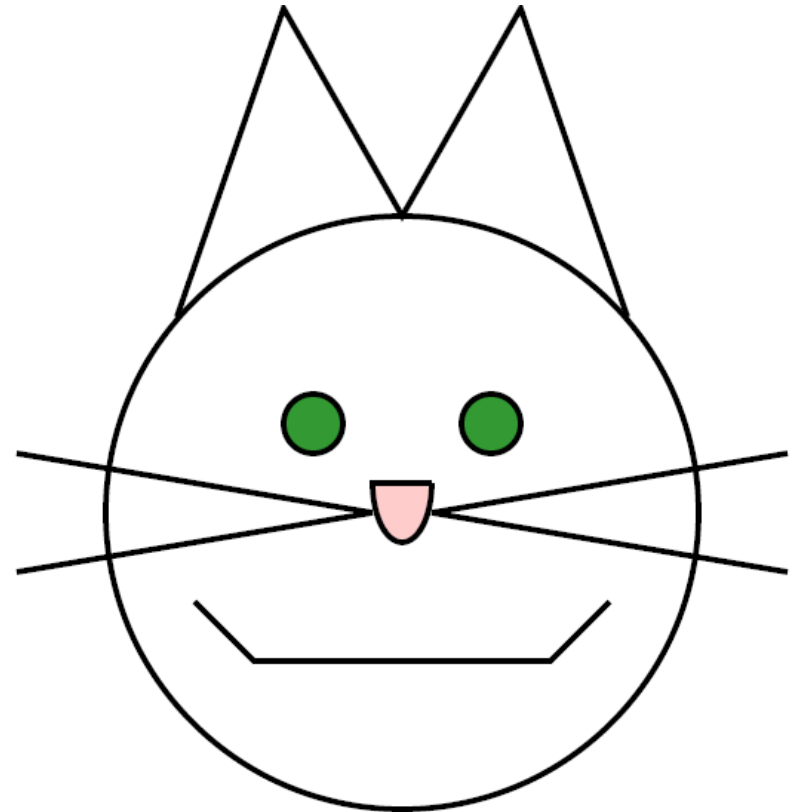


Raster 2x

Vector 2x



Cat



Cat



Vectorialisation



Q=1	19,330	0,976
Q=2	20,178	1,343
Q=4	21,016	1,753
Q=8	21,975	2,251
Q=16	22,670	3,097
Q=32	23,461	4,064
Q=64	24,326	5,693
Q=128	25,180	7,660
Q=256	26,058	10,430
Q=512	26,852	14,169
Q=1024	27,596	19,281
Q=2048	28,174	26,703
Q=4096	28,768	37,641
Q=8192	29,320	53,429

Multimedia

The visual perceived accuracy increases as the Q parameter increase.



Vectorialisation



Q=1	19,330	0,976
Q=2	20,178	1,343
Q=4	21,016	1,753
Q=8	21,975	2,251
Q=16	22,670	3,097
Q=32	23,461	4,064
Q=64	24,326	5,693
Q=128	25,180	7,660
Q=256	26,058	10,430
Q=512	26,852	14,169
Q=1024	27,596	19,281
Q=2048	28,174	26,703
Q=4096	28,768	37,641
Q=8192	29,320	53,429

Multimedia

The visual perceived accuracy increases as the Q parameter increase.



Vectorialisation



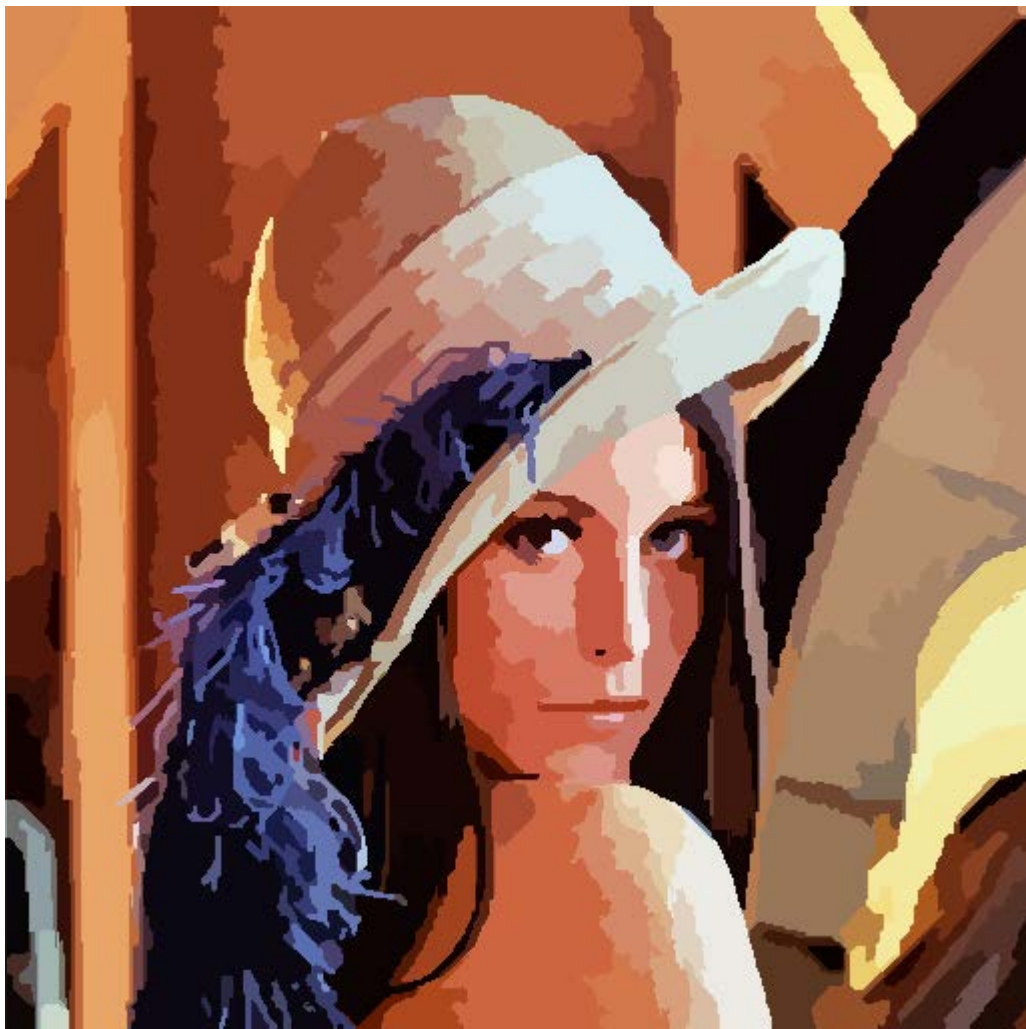
Q=1	19,330	0,976
Q=2	20,178	1,343
Q=4	21,016	1,753
Q=8	21,975	2,251
Q=16	22,670	3,097
Q=32	23,461	4,064
Q=64	24,326	5,693
Q=128	25,180	7,660
Q=256	26,058	10,430
Q=512	26,852	14,169
Q=1024	27,596	19,281
Q=2048	28,174	26,703
Q=4096	28,768	37,641
Q=8192	29,320	53,429

Multimedia

The visual perceived accuracy increases as the Q parameter increase.



Vectorialisation



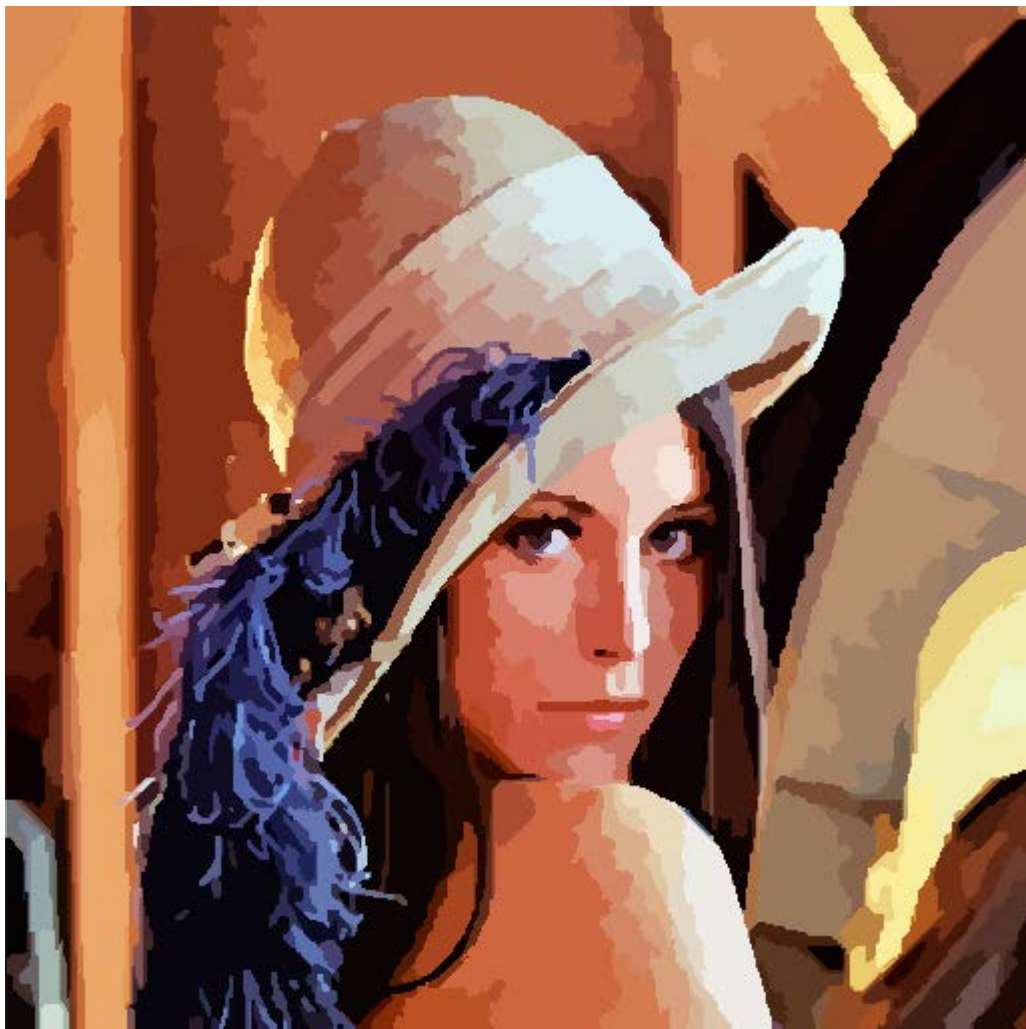
Q=1	19,330	0,976
Q=2	20,178	1,343
Q=4	21,016	1,753
Q=8	21,975	2,251
Q=16	22,670	3,097
Q=32	23,461	4,064
Q=64	24,326	5,693
Q=128	25,180	7,660
Q=256	26,058	10,430
Q=512	26,852	14,169
Q=1024	27,596	19,281
Q=2048	28,174	26,703
Q=4096	28,768	37,641
Q=8192	29,320	53,429

Multimedia

The visual perceived accuracy increases as the Q parameter increase.



Vectorialisation



Q=1	19,330	0,976
Q=2	20,178	1,343
Q=4	21,016	1,753
Q=8	21,975	2,251
Q=16	22,670	3,097
Q=32	23,461	4,064
Q=64	24,326	5,693
Q=128	25,180	7,660
Q=256	26,058	10,430
Q=512	26,852	14,169
Q=1024	27,596	19,281
Q=2048	28,174	26,703
Q=4096	28,768	37,641
Q=8192	29,320	53,429

Multimedia

The visual perceived accuracy increases as the Q parameter increase.



Vectorialisation



Q=1	19,330	0,976
Q=2	20,178	1,343
Q=4	21,016	1,753
Q=8	21,975	2,251
Q=16	22,670	3,097
Q=32	23,461	4,064
Q=64	24,326	5,693
Q=128	25,180	7,660
Q=256	26,058	10,430
Q=512	26,852	14,169
Q=1024	27,596	19,281
Q=2048	28,174	26,703
Q=4096	28,768	37,641
Q=8192	29,320	53,429

Multimedia

The visual perceived accuracy increases as the Q parameter increase.



Vectorialisation



Q=1	19,330	0,976
Q=2	20,178	1,343
Q=4	21,016	1,753
Q=8	21,975	2,251
Q=16	22,670	3,097
Q=32	23,461	4,064
Q=64	24,326	5,693
Q=128	25,180	7,660
Q=256	26,058	10,430
Q=512	26,852	14,169
Q=1024	27,596	19,281
Q=2048	28,174	26,703
Q=4096	28,768	37,641
Q=8192	29,320	53,429

Multimedia

The visual perceived accuracy increases as the Q parameter increase.



Vectorialisation



Q=1	19,330	0,976
Q=2	20,178	1,343
Q=4	21,016	1,753
Q=8	21,975	2,251
Q=16	22,670	3,097
Q=32	23,461	4,064
Q=64	24,326	5,693
Q=128	25,180	7,660
Q=256	26,058	10,430
Q=512	26,852	14,169
Q=1024	27,596	19,281
Q=2048	28,174	26,703
Q=4096	28,768	37,641
Q=8192	29,320	53,429

Multimedia

The visual perceived accuracy increases as the Q parameter increase.



Vectorialisation



Q=1	19,330	0,976
Q=2	20,178	1,343
Q=4	21,016	1,753
Q=8	21,975	2,251
Q=16	22,670	3,097
Q=32	23,461	4,064
Q=64	24,326	5,693
Q=128	25,180	7,660
Q=256	26,058	10,430
Q=512	26,852	14,169
Q=1024	27,596	19,281
Q=2048	28,174	26,703
Q=4096	28,768	37,641
Q=8192	29,320	53,429

Multimedia

The visual perceived accuracy increases as the Q parameter increase.



Vectorialisation



Q=1	19,330	0,976
Q=2	20,178	1,343
Q=4	21,016	1,753
Q=8	21,975	2,251
Q=16	22,670	3,097
Q=32	23,461	4,064
Q=64	24,326	5,693
Q=128	25,180	7,660
Q=256	26,058	10,430
Q=512	26,852	14,169
Q=1024	27,596	19,281
Q=2048	28,174	26,703
Q=4096	28,768	37,641
Q=8192	29,320	53,429

Multimedia

The visual perceived accuracy increases as the Q parameter increase.



Vectorialisation



Q=1	19,330	0,976
Q=2	20,178	1,343
Q=4	21,016	1,753
Q=8	21,975	2,251
Q=16	22,670	3,097
Q=32	23,461	4,064
Q=64	24,326	5,693
Q=128	25,180	7,660
Q=256	26,058	10,430
Q=512	26,852	14,169
Q=1024	27,596	19,281
Q=2048	28,174	26,703
Q=4096	28,768	37,641
Q=8192	29,320	53,429

Multimedia

The visual perceived accuracy increases as the Q parameter increase.



Vectorialisation



Q=1	19,330	0,976
Q=2	20,178	1,343
Q=4	21,016	1,753
Q=8	21,975	2,251
Q=16	22,670	3,097
Q=32	23,461	4,064
Q=64	24,326	5,693
Q=128	25,180	7,660
Q=256	26,058	10,430
Q=512	26,852	14,169
Q=1024	27,596	19,281
Q=2048	28,174	26,703
Q=4096	28,768	37,641
Q=8192	29,320	53,429

Multimedia

The visual perceived accuracy increases as the Q parameter increase.



Vectorialisation



	PSNR	bpp
Q=1	19,330	0,976
Q=2	20,178	1,343
Q=4	21,016	1,753
Q=8	21,975	2,251
Q=16	22,670	3,097
Q=32	23,461	4,064
Q=64	24,326	5,693
Q=128	25,180	7,660
Q=256	26,058	10,430
Q=512	26,852	14,169
Q=1024	27,596	19,281
Q=2048	28,174	26,703
Q=4096	28,768	37,641
Q=8192	29,320	53,429

Multimedia

The visual perceived accuracy increases as the Q parameter increase.



Formati grafici vettoriali

- **PS** (Postscript): Formato sviluppato da Adobe Systems originariamente per la stampa di documenti su stampanti laser, è utilizzato anche per la memorizzazione di immagini vettoriali.
- **EPS** (Encapsulated Postscript): Estensione del formato PostScript che consente di incapsulare immagini bitmap (raster).
- **DCS** (Desktop Color Separation): Un caso speciale di EPS sviluppato originariamente da Quark per tenere separati i dati ad alta risoluzione dall'anteprima a bassa risoluzione.
- **PDF** (Portable Data Format): Sviluppato da Adobe, è il formato più diffuso per condividere, indipendentemente dalla piattaforma, documenti di testi e immagini.
- **PICT**: Formato grafico sviluppato da Apple Computer per la piattaforma Macintosh in grado di memorizzare sia immagini vettoriali che raster.



Standard e formati grafici



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.

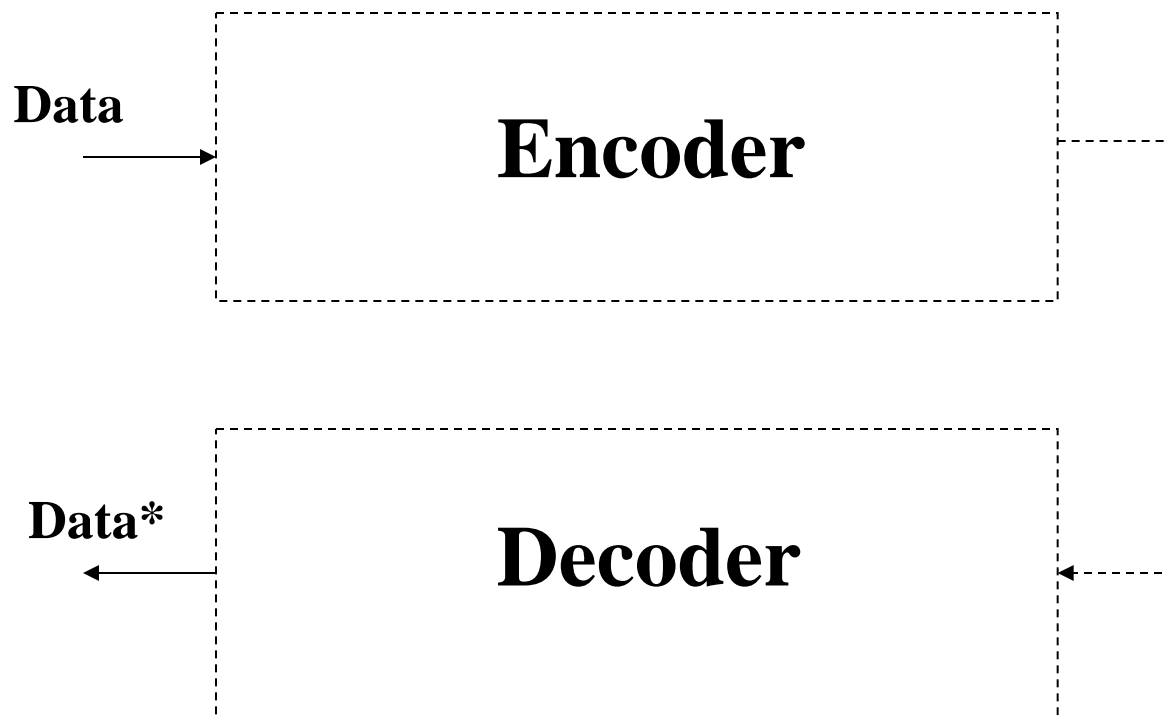


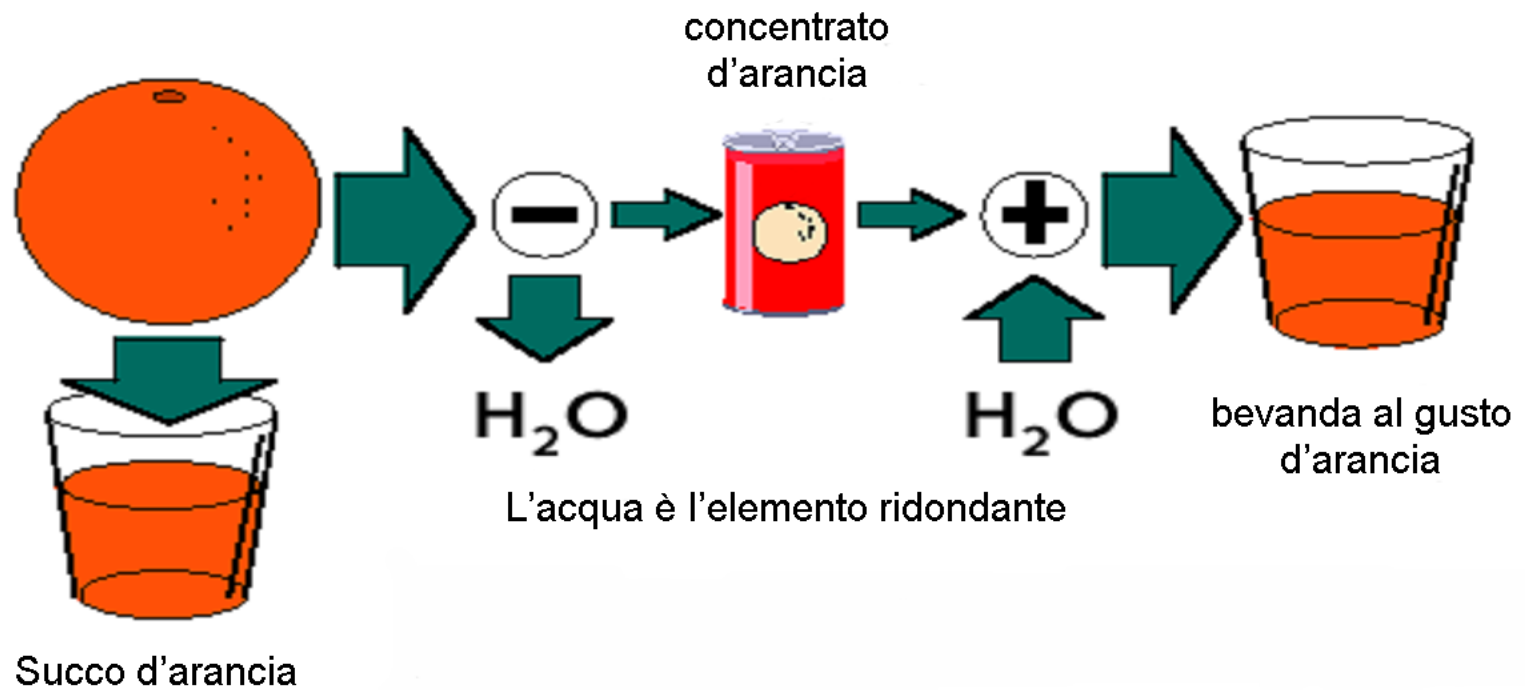
Algoritmo di compressione

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







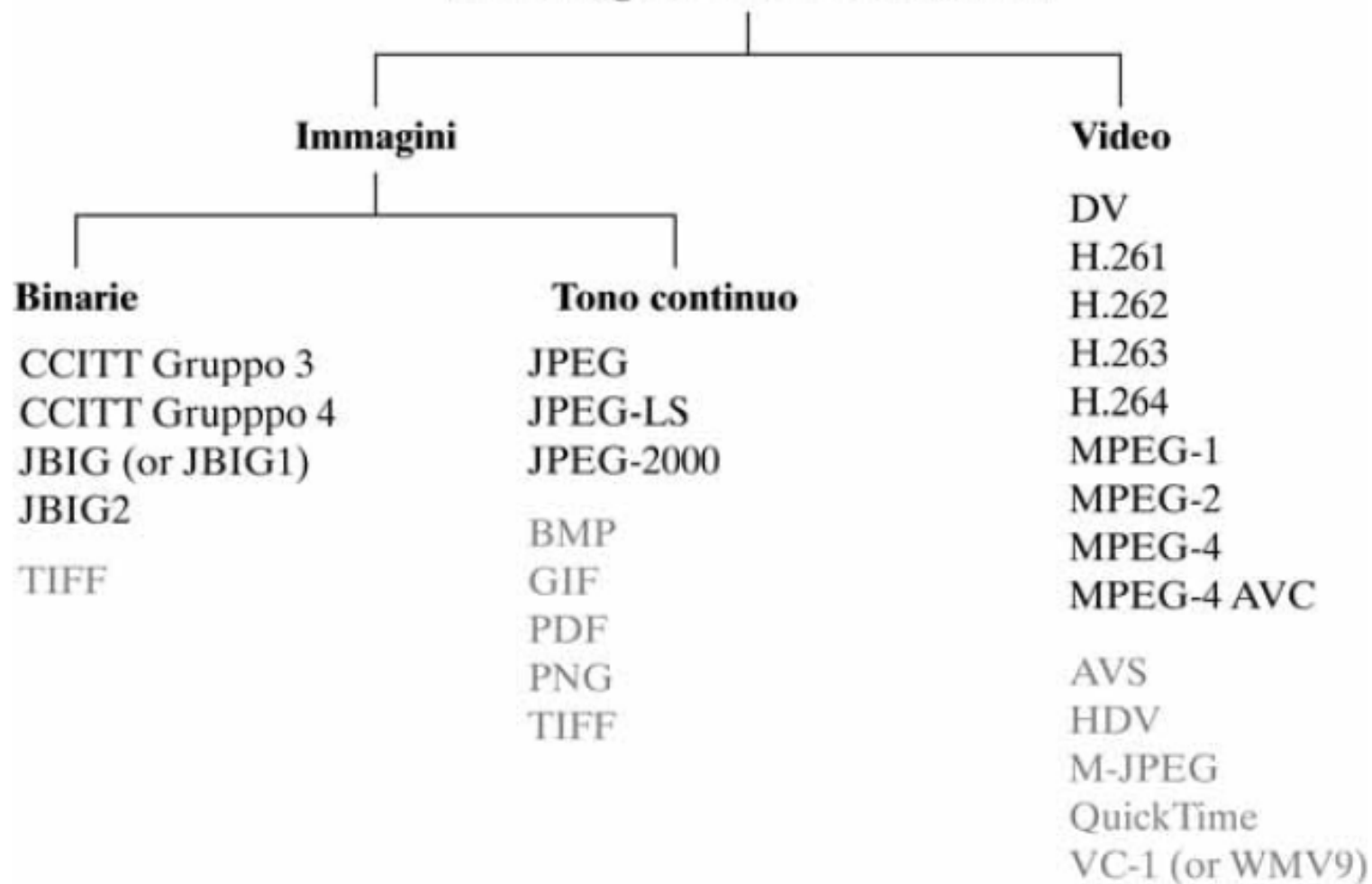
I formati

Un *formato di file di un'immagine* è una modalità standard per organizzare e immagazzinare i dati di un'immagine. Esso definisce come i dati vengono disposti e, se esiste, il tipo di compressione utilizzata.

Gli *standard di compressione* di un'immagine, invece, definiscono le procedure e gli algoritmi per la compressione e la decompressione, cioè per la riduzione della quantità dei dati necessari per rappresentare un'immagine.



Standard di compressione dell'immagine, formati e contenitori





Immagini Binarie (Standard)

CCITT Gruppo 3	ITU-T	Metodo per la trasmissione facsimile (FAX) di documenti attraverso le linee telefoniche. Supporta la codifica di Huffman [8.2.1] e la run-length 1-D e 2-D [8.2.5].
CCITT Gruppo 4	ITU-T	Una versione semplificata dello standard CCITT del gruppo 3 che supporta solo la codifica 2-D run-length.
JBIG o JBIG1	ISO/IEC/ITU-T	Uno standard (<i>Joint Bi-level Image Experts Group</i>) che serve per la compressione delle immagini binarie progressiva e senza perdita. Le immagini a toni continui superiori a 6 bit/pixel possono essere codificate per singoli piani di bit [8.7.2]. Viene utilizzata la codifica aritmetica basata sul contesto [8.2.3] e una versione iniziale dell'immagine a più bassa risoluzione può essere gradualmente migliorata aggiungendo dati compressi.
JBIG2	ISO/IEC/ITU-T	Deriva dallo standard JBIG1 e codifica immagini binarie in applicazioni FAX, per Internet e le immagini nel desktop. Il metodo di compressione utilizzato fa uso del dizionario [8.2.6] per le regioni a halftone o di testo, mentre per le immagini con un altro tipo di contenuto viene usata la codifica di Huffman [8.2.1] o quella aritmetica [8.2.3]. Può essere o meno con perdita.



Immagini a toni continui - Standard

JPEG	ISO/IEC/ITU-T	Standard <i>Joint Photographic Experts Group</i> per le immagini di qualità fotografica. Il suo <i>sistema di codifica</i> detto <i>baseline</i> (più comunemente implementato) utilizza la trasformata discreta del coseno (DCT) quantizzata su blocchi 8×8 [8.2.8], la codifica Huffman [8.2.1] e run-length [8.2.3]. È uno dei metodi più usati per la compressione di immagini su Internet e non solo.
JPEG-LS	ISO/IEC/ITU-T	Uno standard lossless o quasi per le immagini a toni continui che si basa sulla predizione adattativa [8.2.9] sulla modellazione del contesto (context modelling) [8.2.3] e sulla codifica Golomb [8.2.2].
JPEG-2000	ISO/IEC/ITU-T	Deriva dal JPEG per una migliore compressione delle immagini di qualità fotografica. Vengono utilizzate la codifica aritmetica [8.2.3] e la trasformata discreta wavelet (DWT, <i>Discrete Wavelet Transform</i>) quantizzata [8.2.2]. La compressione può essere sia lossless che lossy.



Immagini a toni continui

BMP	Microsoft	<i>Windows Bitmap</i> . Un formato di file utilizzato principalmente per le immagini non compresse.
GIF	CompuServe	<i>Graphic Interchange Format</i> . Un formato di file che utilizza la codifica LZW lossless [8.2.4] per immagini che hanno da 1 a 8 bit. È usato di solito per creare brevi animazioni e brevi filmati a bassa risoluzione.
PDF	Adobe System	<i>Portable Document Format</i> . Un formato per la rappresentazione bidimensionale di documenti indipendente dal dispositivo utilizzato e dalla risoluzione. Può fungere da contenitore per i formati JPEG, JPEG2000, CCITT e altre immagini compresse. Alcune versioni del PDF sono diventate standard ISO.
PNG	<i>World Wide Web Consortium (W3C)</i>	<i>Portable Network Graphics</i> . Un formato di file che comprime le immagini in maniera lossless a colori con trasparenza (fino a 48bit/pixel) codificando la differenza tra il valore di ciascun pixel e il valore predetto basato su pixel precedenti [8.2.9].
TIFF	Aldus	<i>Tagged Image File Format</i> . Un formato di file flessibile che supporta una discreta varietà di standard di compressione, come ad esempio JPEG, JPEG.LS, JPEG-2000, JBIG2, ecc.



Video

AVS	MII	<i>Audio-Video Standard</i> . Simile all'H.264 ma utilizza la codifica esponenziale Golomb [8.2.2]. È stato creato in Cina.
HDV	Company consortium	<i>High definition Video</i> . Estensione del DV per la televisione HD che impiega MPEG-2 per la compressione, compresa la rimozione della ridondanza temporale attraverso la predizione differenziale [8.2.9].
M-JPEG	Varie compagnie	<i>Motion JPEG</i> . Un formato di compressione in cui ciascun frame è compresso indipendentemente utilizzando JPEG.
Quick-Time	Apple Computer	Un contenitore di file multimediali che supporta DV, H.261, H.262, H.264, MPEG-1, MPEG-2 e altri formati di compressione video.
VC-1 WMV9	SMPTE Microsoft	Il formato video più utilizzato su Internet. Viene adottato per l'HD e per i DVD ad alta definizione <i>blu ray</i> . È simile all'H.264/AVC e utilizza la DCT intera su blocchi a dimensione variabile [8.2.8 e 8.2.9] e tabelle con un codice a lunghezza variabile dipendente dal contesto [8.2.1], ma senza predizioni intraframe.

Video

DV	IEC	<i>Digital Video</i> . Uno standard video ideale per la fruizione domestica, per la riproduzione semiprofessionale attraverso videocamere o nuovi strumenti elettronici. I frame vengono compressi indipendentemente utilizzando un approccio per la compressione basato sulla DCT [8.2.8], simile al JPEG.
H.261	ITU-T	Uno standard bidirezionale di videoconferenza per linea ISDN (<i>Integrated Services Digital Network</i>). Supporta immagini con una risoluzione non interlacciata di 352×288 e di 176×144 , chiamate rispettivamente CIF (<i>Common Intermediate Format</i>) e QCIF (<i>Quarter CIF</i>). Viene utilizzato l'approccio per la compressione basato sulla DCT [8.2.8], simile al JPEG, con una predizione differenziale frame a frame [8.2.9], volta a ridurre la ridondanza temporale.
H.262	ITU-T	Si veda MPEG-2 (sotto).
H.263	ITU-T	Una versione migliorata dell'H.261 adatta ai modem ordinari (per esempio 28/8 Kb/s) che supporta alcune risoluzioni aggiuntive: SQCIF (Sub-Quarter CIF 128×96), 4CIF (704×576) e 16CIF (1408×512).
H.264	ITU-T	Un'estensione dell'H.261 e dell'H.263 per le videoconferenze su Internet e per la trasmissione televisiva. Supporta la predizione delle differenze all'interno dei frame [8.2.9], la trasformata intera a blocchi di grandezza variabile (piuttosto che la DCT) e la codifica aritmetica adattativa basata sul contesto [8.2.3].



MPEG-1	ISO/IEC	<i>Motion Pictures Export Group</i> . Uno standard per le applicazioni CD-ROM con video non interlacciati superiori ai 1.5Mb/s. È simile all'H.261 ma la predizione dei frame si fonda o sul frame precedente o sul frame successivo oppure su un'interpolazione di entrambi. È supportato dalla maggior parte dei computer e dai riproduttori DVD.
MPEG-2	ISO/IEC	Un'estensione di MPEG-1 destinata ai DVD con una capacità di trasferimento di 15 Mb/s. Supporta i video interlacciati e l'HDTV. È di gran lunga lo standard video più diffuso e di successo.
MPEG-4	ISO/IEC	Un'estensione di MPEG-2 che supporta blocchi di dimensione variabile e la predizione differenziale all'interno dei frame [8.2.9]
MPEG-4 AVC	ISO/IEC	MPEG-2 Part 10 <i>Advanced Video Coding</i> (AVC). Identico all'H.264.



.BMP (Bitmap)

- Utilizzato (principalmente) per immagini **non compresse**
- Utilizzato per rappresentare **immagini raster**
- Utilizzato (principalmente) per immagini a **tono continuo**
- Sviluppato da Microsoft nel 1990
- Esistono 3 versioni di questo formato: 3, 4 e 5



BMP versione 3

- E' la più diffusa, permette R/W su disco veloce, ma occupa più spazio (immagini non compresse).

	Profondità (bit/pixel)	
	1, 4, 8	16, 24, 32
Gamma colori	2, 16, 256	65.536, 16MIn, 4MId
Rappresentazione	Colore indicizzato (tavolozza)	RGB

- Nessun **canale alfa** per la trasparenza.



BMP versioni 4 e 5

- Possibilità di utilizzare il **canale alfa**
- Possibilità di utilizzare spazi di colore personalizzati
- Possibilità di incorporare immagini JPEG e PNG
- Utilizzate maggiormente come formato interno di altre applicazioni e raramente come file indipendenti



BMP - Struttura

Header del file

Blocco d'informazioni

Header della bitmap

opzionale: modello di colore

opz.: tavolozza

opz.: spazio inutilizzato

Mappa dei pixel

opz.: spazio inutilizzato

Informazioni sul file (come la dimensione in byte). E' assente se la bitmap è integrata all'interno di una applicazione o libreria.



BMP - Struttura

Header del file

Blocco d'informazioni

Header della bitmap

opzionale: modello di colore

opz.: tavolozza

opz.: spazio inutilizzato

Mappa dei pixel

opz.: spazio inutilizzato

Informazioni su dimensioni (in pixel) dell'immagine e gamma di colore utilizzata.

Le informazioni sono relative sia al dispositivo di input che a quello di output (da utilizzare per un'eventuale attività di stampa).



BMP - Struttura

Header del file

Blocco d'informazioni

Header della bitmap

opzionale: modello di colore

opz.: tavolozza

opz.: spazio inutilizzato

Mappa dei pixel

opz.: spazio inutilizzato

Per le versioni 4 e 5

Per profondità ≤ 8
bit/pixel.
Nella tavolozza ogni
colore è
rappresentato da
una struttura di 4
byte detta
RGBQuad:
RGB+byte
proprietario (a 0).



BMP - Struttura

Header del file

Blocco d'informazioni

Header della bitmap

opzionale: modello di colore

opz.: tavolozza

opz.: spazio inutilizzato

Mappa dei pixel

opz.: spazio inutilizzato

Struttura di dati principale della bitmap: ad ogni pixel si fa corrispondere un colore sotto forma di indice nella tavolozza, oppure nelle sue componenti cromatiche. Nel caso delle versioni 4 e 5 potrebbero essere incorporate delle immagini al suo interno.



BMP e compressione? (1)






- Le BMP non compresse permettono R/W su disco veloce, perché il processore non deve effettuare ulteriori calcoli nella (de)compressione. Il compromesso sta nel maggiore spazio di memoria richiesto.



BMP e compressione? (2)

- E' possibile comprimere le BMP utilizzando un algoritmo RLE, ma le versioni BMP compresse utilizzano comunque più spazio rispetto ad altri formati appositamente sviluppati per immagini compresse

Dimensioni in byte della stessa immagine (128 colori) in diversi formati raster

	BMP non compresso		24030
	BMP compresso		8764
	GIF		5365
	PNG		4029



.GIF (Graphics Interchange Format)

- Utilizzato per **immagini compresse**
- Utilizzato per rappresentare **immagini raster**
- Utilizzato (principalmente) per immagini a **tono continuo**
- Sviluppato da CompuServe nel 1987 (versione *87a*) e migliorato (v *89a*) con l'aggiunta della **trasparenza** e il supporto per le **immagini multiple** (animazioni)



GIF – Metodi di Compressione

- GIF utilizza principalmente l'algoritmo di compressione LZW al posto della codifica RLE
- E' possibile interlacciare le linee dell'immagine per fornire un'anteprima a dimensione ridotta (utile per il web)
- Dispute legali sugli standard di compressione hanno portato alla nascita del formato PNG
- Dal 2004 è scaduto il brevetto sull'algoritmo LZW: 'GIF Liberation Day'



GIF – Fasi di Compressione

1. Encoder

- ❑ Crea il flusso dati GIF a partire da dati raster
- ❑ Include informazioni necessarie per riprodurre gli elementi grafici
- ❑ Ottimizza la memorizzazione dell'informazione (rimozione della ridondanza)

2. Decoder

- ❑ Processa il flusso dati GIF
- Conformità: se la versione dell'Encoder e del Decoder coincidono



GIF – Colore

- Basato su una tavolozza VGA a 256 colori
 - Ogni entry della tavolozza occupa 1 byte
 - Le entry sono un sottoinsieme dei 16Mln di colori possibili fra le terne RGB complete (256x256x256)
 - Si può ottimizzare la scelta dei 256 colori per migliorare la qualità dell'immagine compressa
- Ad ogni pixel dell'immagine corrisponde un indice (1 byte) nella tavolozza



GIF – Dithering

- L'immagine compressa potrebbe presentare delle zone di colore uniforme a bande
- Si può applicare il dithering per inserire una 'sfumatura' nell'immagine:
 - I colori della tavolozza vengono accostati e l'occhio percepisce nuovi colori in realtà non presenti
- Il dithering abbassa la comprimibilità dell'immagine (riduce la ridondanza)



GIF – Esempi

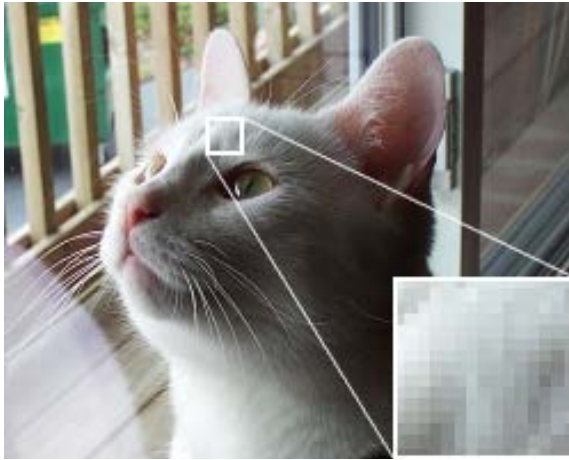


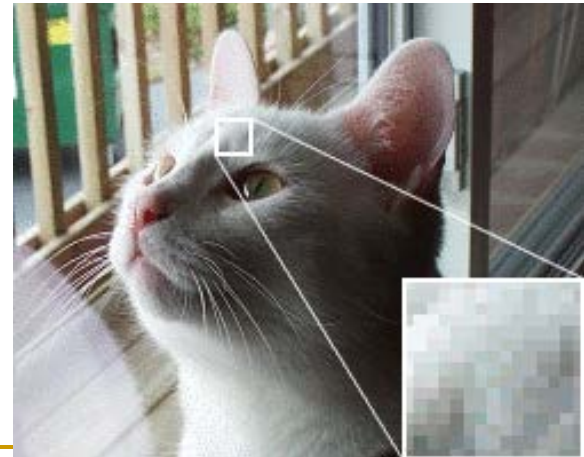
Immagine originale



Immagine indicizzata



Indicizzata + Dithering



Palette ottimizzata + Dithering



GIF – Trasparenza

- Una entry della tavolozza può essere definita come ‘trasparente’:
 - Assume il colore dello sfondo
 - E’ differente dal canale alfa: non permette la trasparenza parziale
 - Simile alle tecniche di ‘green screen’ (chroma key) usate in Computer Grafica



GIF animate

- Dalla v 89a è possibile inserire semplici animazioni nelle immagini grazie alla *Graphics Control Extension (GCE)*
- Una GIF animata è formata da vari frames che si alternano secondo un delay preimpostato
 - Ogni frame ha la sua GCE che gestisce la durata
- Di default il ciclo di animazione non è ripetuto
 - Netscape nel 1990 ha implementato la ripetizione



.PNG

(Portable Network Graphics)

- Utilizzato per **immagini compresse**
- Utilizzato per rappresentare **immagini raster**
- Utilizzato (principalmente) per immagini a **tono continuo**
- Sviluppato dal World Wide Web Consortium (W3C) nel 1996 per sostituire GIF
 - Non supporta immagini animate



PNG – Colore

- Supporta immagini **truecolor** (RGB) da 24 a 48 bit per pixel (bpp): maggiore degli 8 bpp di GIF
- Supporta il **canale alfa** per la trasparenza: 32 bpp RGB-A
- Supporta immagini **grayscale** fino a 16 bpp
- Non supporta spazi di colore diversi dall'RGB, come CMYK



PNG – Bits per pixel e per canale

- Come per BMP, i pixel dell'immagine possono contenere un puntatore ad una palette oppure le coordinate grayscale o RGB, con o senza canale alfa

Bits per channel (bpc) = bpp / #Channels						
Color option	Channels	Bits per pixel (bpp)				
Indexed	1	1	2	4	8	
Grayscale	1	1	2	4	8	16
Grayscale + alpha	2				16	32
Truecolor	3				24	48
Truecolor + alpha	4				32	64



PNG – Struttura

1. Header:

- 8 byte, contiene informazioni sul file

2. Chunks:

- Durante la decompressione deve essere possibile leggere e interpretare i chunks per risalire all'immagine non compressa
- Distinguiamo fra chunks ***ausiliari*** e ***critici***
 - I chunks ausiliari contengono informazioni secondarie come valori gamma dell'immagine, colore di background, metadati testuali, ...



PNG – Chunks critici

4 tipi di chunks critici principali:

1. **Header**: informazioni sull'immagine (dimensioni, canali, colori, ...)
2. **Palette** (se si usa indicizzazione)
3. **Dati**: l'immagine potrebbe essere suddivisa in più chunks critici
4. **Marker di fine immagine**



PNG – Compressione

La compressione è realizzata in due fasi:

1. Pre-compressione: *filtraggio predittivo*

- ❑ Il filtro predice il valore di ciascun pixel basandosi sui valori dei pixels vicini, e memorizza la differenza fra il valore predetto e quello effettivo
- ❑ Se l'immagine non è molto fotorealistica (c'è molta ridondanza), tramite il filtraggio predittivo può diventare più comprimibile

2. Compressione: *algoritmo Deflate*



PNG – Compressione: Deflate

- Algoritmo di compressione **lossless**
- E' una variante dell'algoritmo LZW (GIF!)
- Opera su dati con dimensione massima di 64KB
- Può utilizzare la **codifica di Huffman** per ridurre la dimensione dei dati (se risulta conveniente farlo)



PNG – Interlacciamento e Animazione

- E' possibile interlacciare le linee dell'immagine per fornire un'anteprima a dimensione ridotta (utile per il web). Rispetto a GIF le immagini a bassa risoluzione sono più chiare.
- Non supporta immagini animate
 - **MNG**: variante di PNG che le supporta



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

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

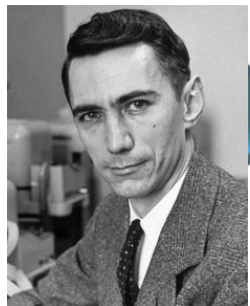


Entropia

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

$$E = - \sum_{i \in S} f_i \log_2 (f_i)$$

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.



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:

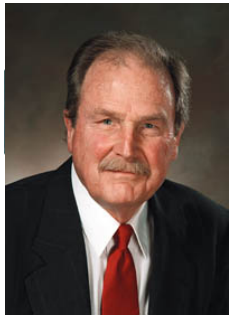
$$N * E$$

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 perché tende al limite imposto dal teorema di Shannon.



Codifica di Huffman (1)

Illustro l'algoritmo con un esempio.

Dati: AABABCAACAAADDDD.

A : frequenza pari a $8/16 = 1/2$

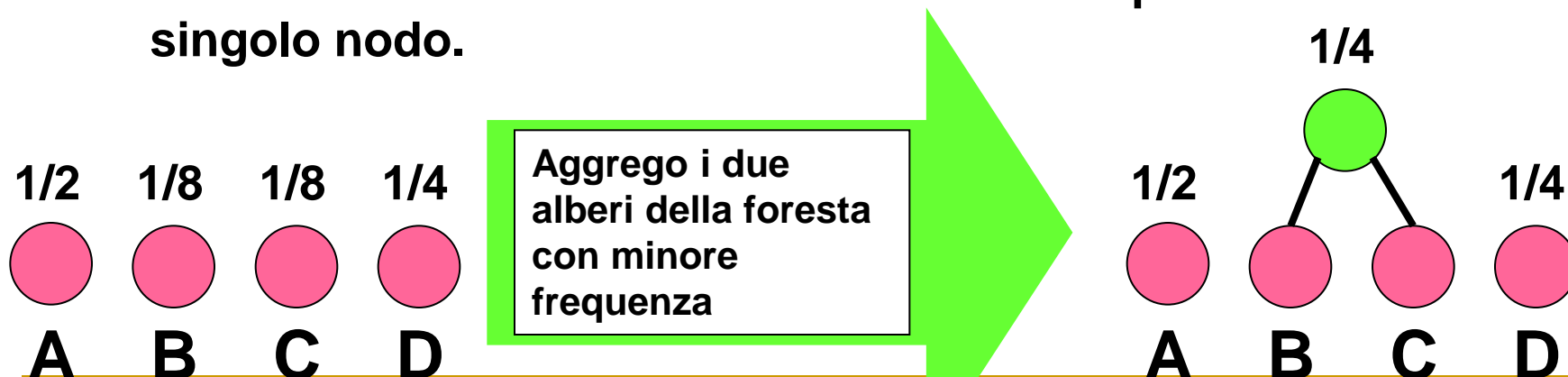
B : frequenza pari a $2/16 = 1/8$

C : frequenza pari a $2/16 = 1/8$

D : frequenza pari a $4/16 = 1/4$

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

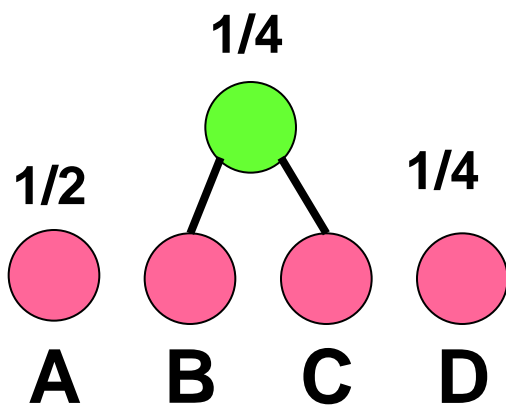
INIZIO: una foresta con 4 alberi ciascuno composto di un singolo nodo.



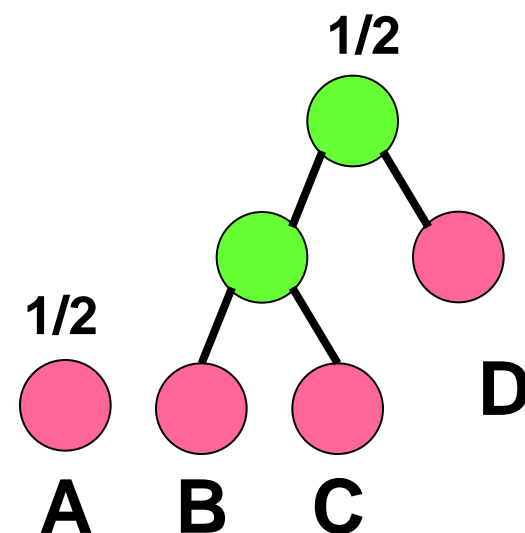


Codifica di Huffman(2)

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



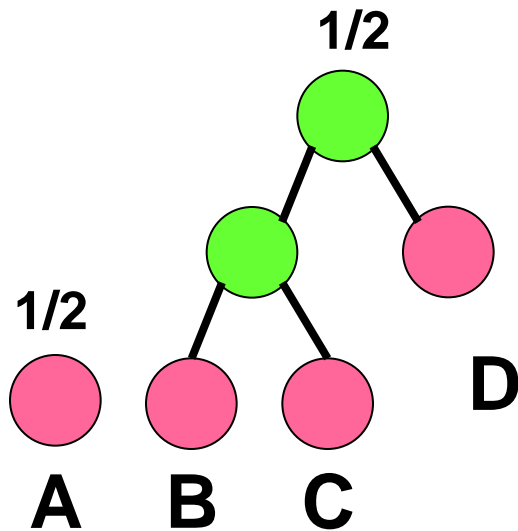
Aggrego i due alberi della foresta con minore frequenza



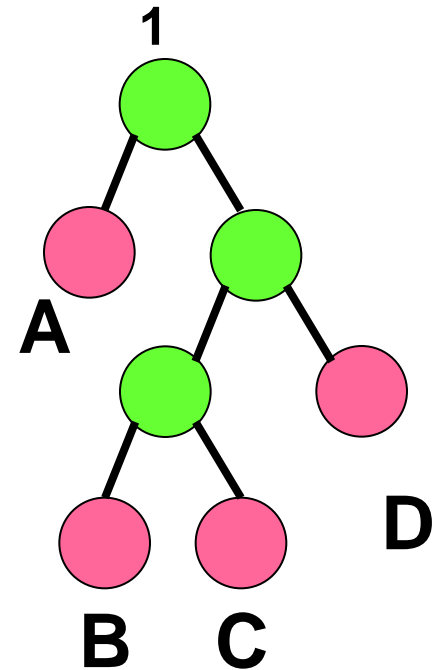


Codifica di Huffman(3)

In questo caso ecco il risultato finale:

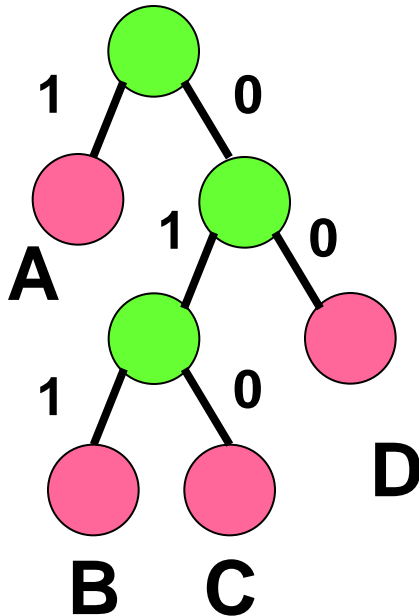


Aggrego i due alberi della foresta con minore frequenza





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)

- Codice per la sequenza

AABABCAACAAADDDD.

A : 1

B : 011

C : 010

D : 00

- 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.
- Per le immagini a toni di grigio i “caratteri” sono i livelli di grigio (256) e tale tabella è assai poco pratica.
- Huffman viene usato per comprimere alcune informazioni *nella fase finale* della codifica JPEG (dopo che è stata fatta una riduzione con altre tecniche)



Codifica di Golomb

- Codifica di input **interi non negativi** con distribuzioni di probabilità che decadono esponenzialmente.
- Sono più facili da calcolare rispetto ad Huffman
- sono utilizzati negli standard di compressione JPEG-LS e AVS



Codifica di Golomb

Dato un numero intero n non negativo e un suo *divisore* intero positivo $m > 0$, il codice di Golomb di n rispetto a m , denotato come $\mathbf{G}_m(n)$, è la combinazione di un *codice unario** del quoziente $\lfloor n/m \rfloor$ e della *rappresentazione binaria del resto $n \bmod m$* .

$\mathbf{G}_m(n)$ si costruisce nel modo seguente:

1. Si forma il codice unario* del quoziente $\lfloor n/m \rfloor$
2. Sia $k = \lceil \log_2 m \rceil$, $c = 2^k - m$, $r = n \bmod m$ e si calcola il resto r^t in modo che:

$$r^t = \begin{cases} r \text{ troncato a } t = k - 1 \text{ bit} & 0 \leq r < c \\ r + c \text{ troncato a } t = r & \text{altrimenti} \end{cases}$$

3. Si concatenano i risultati di cui sopra.

*:Il **codice unario** di un numero intero q si definisce come q numeri 1 seguiti da uno 0



Codifica di Golomb - Esempio

Calcoliamo $G_2(9)$, $n=9$, $m=2$:

1. Determiniamo il *codice unario del quoziente* $[9/2] = 4$ che corrisponde al codice unario 11110.
2. Sia $k = \lceil \log_2 2 \rceil = 1$, $c = 2^1 - 2 = 0$, $r = 9 \bmod 2 = 1 = 0001$

Siamo nel secondo caso, $r > c$

$$r + c = 0001 + 0000 = 0001$$

tronchiamo $t = 1$ bit

$$r^1 = 1$$

3. Concateniamo il codice unario a r^1

111101



Codifica di Golomb

- Alcuni codici di Golomb per i numeri interi da 0 a 9

n	$G_1(n)$	$G_2(n)$	$G_4(n)$
0	0	00	000
1	10	01	001
2	110	100	010
3	1110	101	011
4	11110	1100	1000
5	111110	1101	1001
6	1111110	11100	1010
7	11111110	11101	1011
8	111111110	111100	11000
9	1111111110	111101	11001



Codifica Aritmetica

- La codifica aritmetica fu proposta da Elias e presentata da Abramson nel suo paper dal titolo “*Information Theory*” (1963)
- Sono codifiche **protette da copyright** ed è per questo motivo che se lo standard prevede sia la codifica aritmetica che quella di Huffman, spesso viene supportata solo la seconda.
- la codifica aritmetica è usata in JBIG1, JBIG2, JPEG-2000, H.264, MPEG-4 AVC



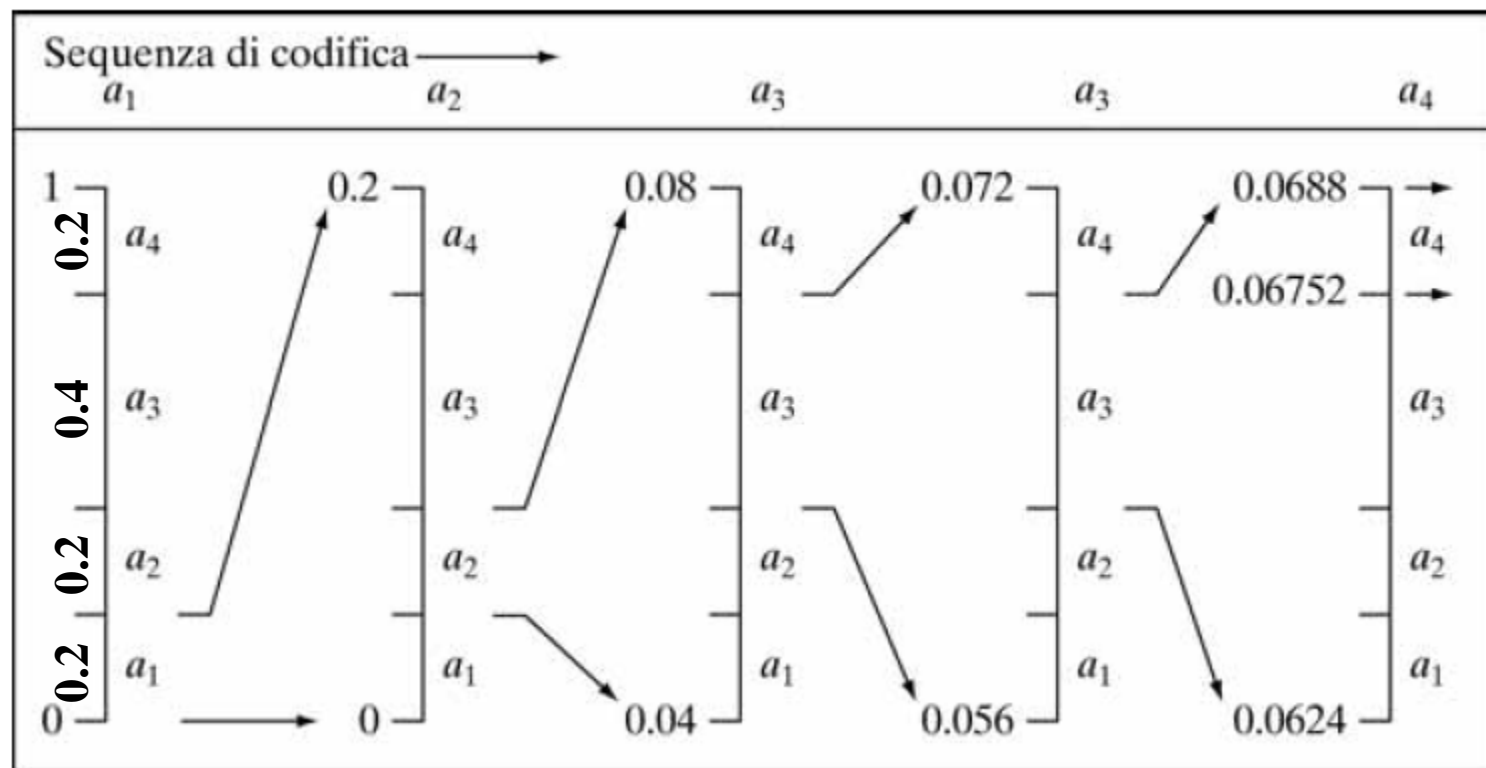
Codifica Aritmetica

- Codici di **lunghezza variabile**
- Nella codifica aritmetica all'alfabeto è associata una distribuzione di probabilità.
- L'intervallo iniziale è **$[0,1)$**
- L'algoritmo ricalcola ad ogni passo l'intervallo dei propri simboli dell'alfabeto, sulla base della distribuzione iniziale, riducendo di volta in volta la “finestra” sull'intervallo $[0,1)$



Codifica Aritmetica

- Ad ogni passo le proporzioni delle probabilità iniziali in $[0,1)$ sono mantenute nel sottointervallo selezionato





Codifica Aritmetica

Algoritmo di Codifica

- **INPUT:** Stringa da codificare & alfabeto con distribuzione di probabilità
 - 1. Inizializza l'intervallo a $[0,1)$
 - 2. Per ogni simbolo della stringa di input:
 - 2.1 Suddividi l'intervallo corrente in n sottointervalli ($n = \text{length dell'alfabeto}$). Ogni sottointervallo avrà un'ampiezza proporzionale alla probabilità associata a ciascun elemento nell'alfabeto iniziale
 - 2.2 Seleziona come intervallo corrente quello corrispondente al simbolo in analisi (input)
 - 3. Il lower bound dell'ultimo intervallo (convertito in binario) è il risultato della nostra codifica (output)



Codifica Aritmetica

Algoritmo di Codifica

- Il numero di bit necessari a codificare un intervallo $[a,b)$ di dimensione s è $\lceil -\log_2 s \rceil$
- La lunghezza dell'intervallo finale è uguale al prodotto delle probabilità dei singoli simboli di input
- Avremo quindi: $\lceil -\log_2 s \rceil = \left\lceil -\sum_{i=1}^N p(a(i)) \log p(a(i)) \right\rceil$
- Il numero di bit generati dalla codifica aritmetica è esattamente pari all'**entropia**
- **La codifica aritmetica è dunque (sub)ottimale**



Codifica Aritmetica

Algoritmo di Codifica – Esempio (1)

- Supponiamo di avere il seguente alfabeto:

A	B	EOF
0.4	0.5	0.1

- Input: BBB# (# = End Of File – carattere di stop)

Intervallo	A	B	EOF	Input
[0,1)	[0,0.4)	[0.4,0.9)	[0.9,1)	B
[0.4,0.9)	[0.4,0.6)	[0.6,0.85)	[0.85,0.9)	B
[0.6,0.85)	[0.6,0.7)	[0.7,0.825)	[0.825,0.85)	B
[0.7,0.825)	[0.7,0.75)	[0.75,0.812)	[0.8125,0.825)	#



Codifica Aritmetica

Algoritmo di Codifica – Esempio (2)

- L'intervallo finale è quindi $[0.8125, 0.825)$
- Prendiamo il lower bound e cioè 0.8125
- Lo convertiamo in binario* ed otteniamo 0.1101 ($1/2 + 1/4 + 1/16 = 0.8125$)
- Inoltre, essendo
 $s = 0.825 - 0.8125 = 0.0125 \Rightarrow \lceil -\log_2 0.0125 \rceil = 7 \text{ bits}$
- Il risultato della codifica (output) sarà

1101000

*: conversione decimale (non intero!) \Leftrightarrow binario. Vedi Appendice A.



Codifica Aritmetica

Algoritmo di Decodifica

- **INPUT:** Stringa da decodificare & alfabeto con distribuzione di probabilità
 - 1. Inizializza l'intervallo a $[0,1)$
 - 2. Per ogni simbolo della stringa di input:
 - 2.1 Suddividi l'intervallo corrente in n sottointervalli (n =length dell'alfabeto). Ogni sottointervallo avrà un'ampiezza proporzionale alla probabilità associata a ciascun elemento nell'alfabeto iniziale
 - 2.2 Seleziona il simbolo corrispondente al sottointervallo nel quale ricade il valore dell'input in analisi e seleziona tale intervallo come corrente



Codifica Aritmetica

Algoritmo di Decodifica – Esempio

- Supponiamo di avere il seguente alfabeto:

A	B	EOF
0.4	0.5	0.1

- Input: $.1101000 = 0.8125$

Intervallo	A	B	EOF	Input	Output
$[0,1)$	$[0,0.4)$	$[0.4,0.9)$	$[0.9,1)$	0.8	B
$[0.4,0.9)$	$[0.4,0.6)$	$[0.6,0.85)$	$[0.85,0.9)$	0.81	B
$[0.6,0.85)$	$[0.6,0.7)$	$[0.7,0.825)$	$[0.825,0.85)$	0.812	B
$[0.7,0.825)$	$[0.7,0.75)$	$[0.75,0.812)$	$[0.8125,0.825)$	0.8125	#

- Output: **BBB#**

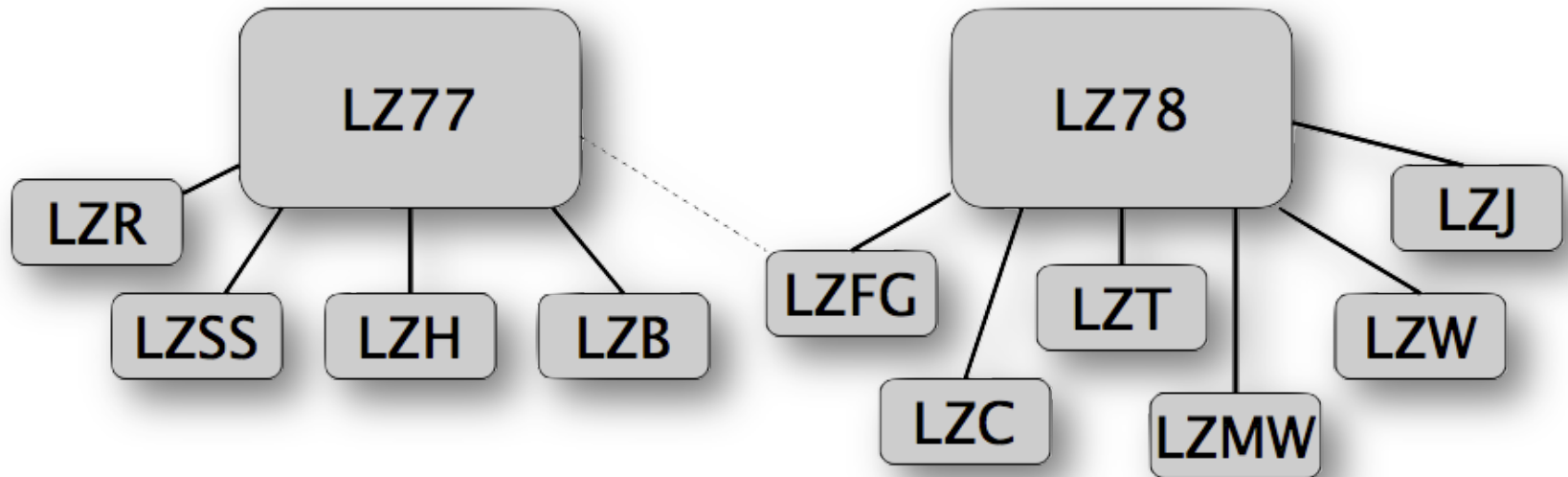


LZ77 – LZ78 – LZW

- LZW (Lempel, Ziv, Welch) è un algoritmo per la **compressione “lossless”**.
- Fu pubblicato per la prima volta nel ‘77 (LZ77 – “*sliding window*”),
 - poi modificato nel ‘78 (LZ78 – aggiunta del *dizionario*)
 - e infine modificato da Welch nel 1984 e quindi chiamato **LZW**



LZ77 – LZ78 – LZW





LZ77 – Sliding Window

- L'algoritmo opera su un'unica struttura dati (tipicamente un vettore di input)
- Su tale vettore scorre una window formata da:
 - Una parte già codificata (“**search buffer**”)
 - Una parte da codificare (“**lookahead buffer**”)
- ***L'algoritmo cerca il più lungo prefisso del lookahead presente nella finestra a partire dal search buffer***
- La “finestra” scorre ad ogni iterazione



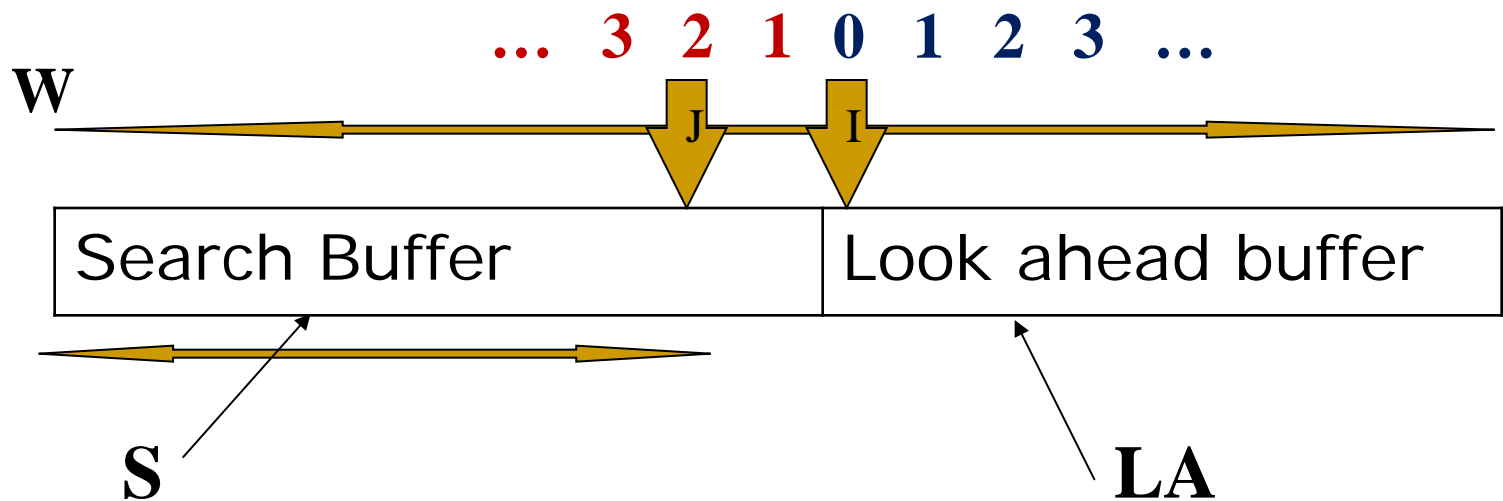
LZ77 – Sliding Window

- Se la ricerca va a buon fine l'algoritmo restituirà la tripla (**D**, **L**, **<s>**)
 - **D** = distanza o offset tra la parola nel lookahead e quella nel search buffer
 - **L** = lunghezza del prefisso trovato
 - **<s>** = simbolo che compare nel lookahead buffer dopo il prefisso
- Altrimenti l'algoritmo restituirà la tripla (**0,0,<s>**)
 - **<s>** = primo simbolo nel lookahead buffer
- **W** (dimensione della “window”) = Search + LookAhead



LZ77 – Sliding Window

Search pointer:



Sequenza codificata in precedenza

Porzione della “finestra” da codificare



LZ77 – Esempio Codifica

- $|S|=7, |LA|=6$
- $[\text{c a b r a c}] \text{ a d a b r a r r a r r a d} \rightarrow (0,0,"c")$
- $[\text{c}] [\text{a b r a c a}] \text{ d a b r a r r a r r a d} \rightarrow (0,0,"a")$
- $[\text{c a}] [\text{b r a c a d}] \text{ a b r a r r a r r a d} \rightarrow (0,0,"b")$
- $[\text{c a b}] [\text{r a c a d a}] \text{ b r a r r a r r a d} \rightarrow (0,0,"r")$
- $[\text{c a b r}] [\text{a c a d a b}] \text{ r a r r a r r a d} \rightarrow (3,1,"c")$
- $[\text{c a b r a c}] [\text{a a b r a}] \text{ r r a r r a d} \rightarrow (2,1,"d")$
- $\text{c} [\text{a b r a c a d}] [\text{a b r a r}] \text{ a r r a d} \rightarrow (7,4,"r")$
- $\text{c a b r a c} [\text{a d a b r a r}] [\text{r a r r a}] \rightarrow (3,5,"d")$

L'algoritmo cerca il più lungo prefisso del lookahead presente nella finestra a partire dal search buffer



LZ77 – Esempio Decodifica

- (0,0,"c") → c
- (0,0,"a") → c a
- (0,0,"b") → c a b
- (0,0,"r") → c a b r
- (3,1,"c") → c a b r a c
- (2,1,"d") → c a b r a c a d
- (7,4,"r") → c a b r a c a d a b r a r
- (3,5,"d") → c a b r a c a d a b r a r r a r r a d

Passo di decodifica di (D,L,<s>)

- Se $D \geq L$:

1. Vai indietro di D caratteri
2. Scrivi L caratteri
3. Aggiungi <s>

- Altrimenti:

1. Vai indietro di D caratteri
2. Scrivi D caratteri ciclicamente fino a L caratteri
3. Aggiungi <s>



Codifica LZW

- La *codifica Lempel-Ziv-Welch (LZW)* assegna delle **codeword a lunghezza fissa** a sequenze di simboli a lunghezza variabile.
- Una caratteristica chiave della codifica LZW è che essa **non necessita di una conoscenza a priori della probabilità di occorrenza dei simboli** che devono essere codificati.
- è stata integrata in numerosi formati di file per l'imaging, quali ad esempio GIF, TIFF, e PDF.
- Il formato PNG è infine stato creato anche per ovviare gli eventuali problemi di copyright della codifica LZW.



LZW

- LZW ha un **dizionario “esplicito”** che viene costruito dinamicamente in fase di codifica (e decodifica)
- Fasi dell’algoritmo:
 - 1. Inizializza il dizionario con tutte le stringhe di lunghezza 1
 - 2. Trova la più lunga stringa ***W*** nel dizionario che “matcha” con l’input corrente
 - 3. Restituisci *dictionary(W)* al posto di *W*
 - 3. Salva *W* + carattere successivo della stringa di input nel dizionario
 - 4. Torna al passo 2



Codifica LZW

La codifica LZW è molto semplice concettualmente (Welch [1984]). All'inizio del processo di codifica, si costruisce un codebook o un *dizionario* contenente i simboli della sorgente da codificare. Per le immagini monocromatiche a 8 bit, vengono assegnate alle intensità 0, 1, 2, ..., 255 le prime 256 parole del dizionario. Quando l'encoder esamina sequenza dopo sequenza i pixel dell'immagine, le sequenze dell'intensità che non sono nel dizionario vengono posizionate (per esempio la successiva inutilizzata) in determinate locazioni. Se i primi due pixel dell'immagine sono bianchi, per esempio, la sequenza "255-255" deve essere assegnata alla locazione 256, posizione che segue quelle riservate ai livelli di intensità da 0 a 255. La prossima volta che ci si imbatte in due pixel consecutivi bianchi, viene utilizzato l'indirizzo della locazione contenente la sequenza 255-255 per rappresentarli. Se viene utilizzato un dizionario a 9 bit e 512 parole nel processo di codifica, i bit originali ($8 + 8$), utilizzati per rappresentare i due pixel, vengono sostituiti con una singola codeword a 9 bit. Chiaramente, la dimensione del dizionario è un importante parametro del sistema. Se infatti esso è troppo piccolo, il rilevamento delle sequenze dei livelli di intensità combacianti sarà meno probabile; se è troppo grande, la dimensione delle codeword comprometterebbe le performance della compressione.



Codifica LZW

Codifica LZW

Si consideri la seguente immagine a 8 bit di dimensione 4×4 di un edge verticale.

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Locazione del dizionario	Voce
0	0
1	1
⋮	⋮
255	255
256	—
⋮	⋮
511	—



Codifica LZW

Sequenza attualmente riconosciuta	Pixel da processare	<i>dictionary</i> (W) Output codificato	Locazione nel dizionario (codeword)	Eventuale nuova Voce del dizionario
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126	126			
126-126	39	258	261	126-126-39
39	39			
39-39	126			
39-39-126	126	260	262	39-39-126-126
126	39			
126-39	39	259	263	126-39-39
39	126			
39-126	126	257	264	39-126-126
126		126		



RLE: Codifica Run-Length

- 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



RLE: Codifica Run-Length

- Come funziona:
 - Consideriamo l'immagine (matrice) come un vettore
 - Ipotizziamo di leggere un'immagine binaria (0 o 1)
 - Generalizzabile ad immagini con più di 1 bpp: la codifica si applica a livello di bit
 - Si salvano in un vettore di output il numero di occorrenze di 0 e 1 nell'ordine in cui vengono letti, ipotizzando di cominciare da uno 0
 - I vari "count" saranno la nostra immagine codificata



RLE: Codifica Run-Length

Esempio

Si voglia comprimere la sequenza:

00000111001011101110101111111 → *29 bit*

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 → *con 3 bit [1,...,8] → 36 bit*

Tali valori vanno adesso scritti in binario. Non sempre tale codifica porta un risparmio rispetto a quella di input. Ciò accade 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 eguali) piuttosto lunghe ricordare la sequenza delle loro lunghezze potrebbe portare un risparmio.



RLE: Codifica Run-Length

- Altra implementazione
 - Ipotizziamo di leggere un'immagine in scala di grigi, quindi non binaria
 - Una possibile implementazione di RLE è il salvataggio di una coppia per ogni carattere letto contenente il carattere stesso e il numero di occorrenze di esso
 - **ATTENZIONE: NON SEMPRE RLE HA UN BUON COMPRESSION RATIO**
 - Compression Ratio =
$$\frac{\text{DimensioneNonCompressa}}{\text{DimensioneCompressa}}$$



RLE: Codifica Run-Length

Esempio

- Supponiamo di voler codificare la seguente stringa

A = 1 2 3 1 1 1 33 33 4 6 5 6 6 6 → *14 numeri*

La codifica di A sarà:

A' = (1,1)(2,1)(3,1)(1,3)(33,2)(4,1)(6,1)(5,1)(6,3) → *18 numeri!*

- Come si evince da questo “toy example” sprechiamo più spazio con la codifica piuttosto che con la stringa “uncompressed”, quindi risulta inutile la codifica RLE in casi simili.



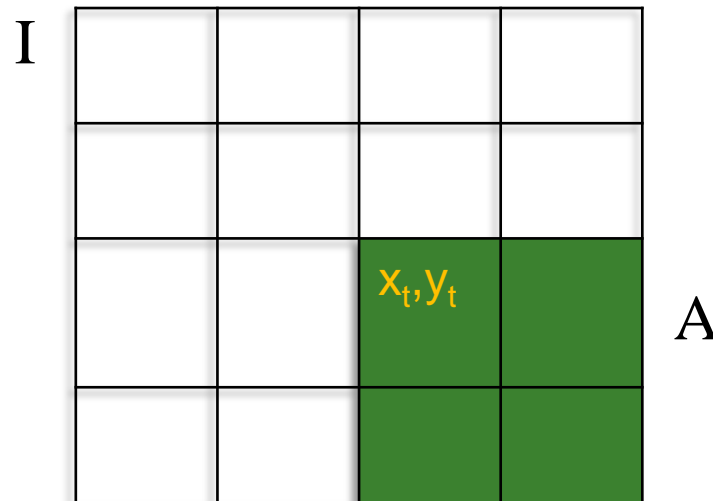
Codifica basata su i simboli

- In una codifica *basata su simboli*, o *su token*, un'immagine viene rappresentata come una serie di sottoimmagini ricorrenti, chiamati *simboli*.
- Ciascun simbolo è memorizzato come un *dizionario di simbolo* e l'immagine è codificata come un insieme di terzine $\{ (x_1, y_1, t_1), (x_2, y_2, t_2), \dots \}$, in cui ciascuna coppia (x_i, y_i) determina la posizione di un simbolo nell'immagine e il *token* t_i è l'indirizzo del simbolo della sottoimmagine nel dizionario.
- Immagazzinando solo una volta i simboli ripetuti si possono comprimere significativamente le immagini, dove i simboli sono spesso delle vere e proprie bitmap e sono ripetuti più volte.
- La codifica basata sui simboli è usata nella compressione JBIG2



Codifica basata su simboli (riassumendo...)

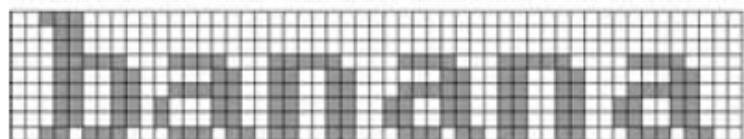
- $(x_t, y_t, t) \rightarrow$ a t corrisponderà una matrice A (un simbolo) all'interno della matrice I (l'immagine) nelle coordinate (x_t, y_t)





Codifica basata su i simboli

Esempio



(a)

Token	Simbolo
0	
1	
2	

(b)

Terzina
(0, 2, 0)
(3, 10, 1)
(3, 18, 2)
(3, 26, 1)
(3, 34, 2)
(3, 42, 1)

(c)

Figura 8.17 (a) Un documento a due livelli; (b) dizionario di simboli; (c) le terzine utilizzate per localizzare i simboli nel documento.



Codifica basata su i Bit plane

- Le tecniche run-length e basate su simboli dei paragrafi precedenti possono essere applicate alle immagini che hanno più di due intensità attraverso l'elaborazione singola del loro piano di bit.
- La tecnica, chiamata *codifica per piani di bit* (bit-plane) si basa sul concetto di scomposizione di un'immagine a livelli multipli (a toni di grigio o a colori) in una serie di immagini binarie e di compressione di ciascuna immagine binaria attraverso uno dei metodi di compressione binaria conosciuti.
- La codifica mediante piani di bit è usata negli standard di compressione JBIG1 e JPEG-2000



Codifica basata sui Bit plane

- Le intensità di una immagine a toni di grigio a m bit possono essere rappresentate con il seguente **polinomio a base 2**:

$$a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \dots + a_12^1 + a_02^0$$

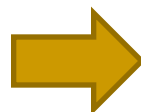
Per scomporre l'immagine in una serie di immagini binarie basta separare gli m coefficienti del polinomio in **m piani a 1 bit**



Codifica basata sui Bit plane

- Supponiamo di avere un'immagine e di considerare ad esempio alcuni dei suoi pixel, come indica la figura a sinistra:

215	180	111
77	161	211
0	194	144



11010111	10110100	1101111
01001101	10100001	11010011
00000000	11000010	10010000

- Ad ogni valore di grigio corrisponde una rappresentazione in numeri binari (8 bit) come indicato nella figura a destra



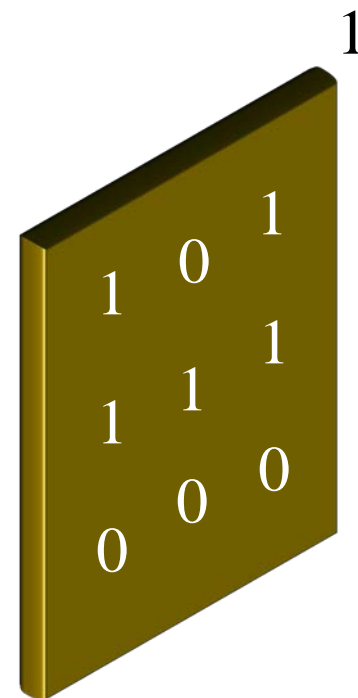
Codifica basata sui Bit plane

215	180	111
77	161	211
0	194	144

- L'immagine a 8-bit è scomposta in 8 piani ad un bit
- Il piano 1 contiene il bit meno significativo di tutti i pixel nell'immagine



11010111	10110100	01101111
01001101	10100001	11010011
00000000	11000010	10010000





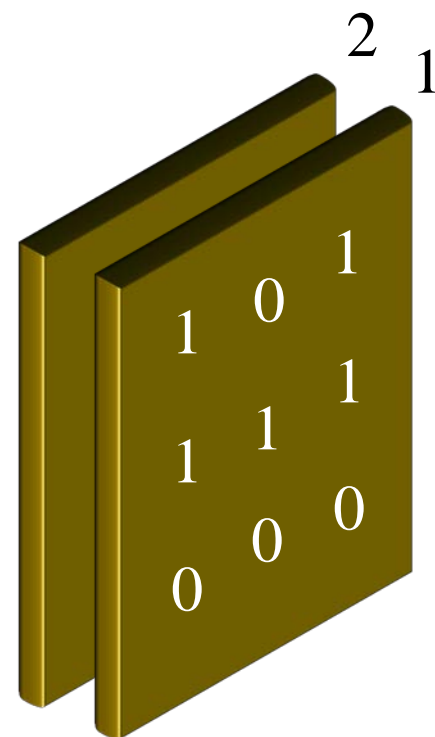
Codifica basata sui Bit plane

215	180	111
77	161	211
0	194	144

- L'immagine a 8-bit è scomposta in 8 piani ad un bit
- Il piano 1 contiene il bit meno significativo di tutti i pixel nell'immagine



11010111	10110100	01101111
01001101	10100001	11010011
00000000	11000010	10010000






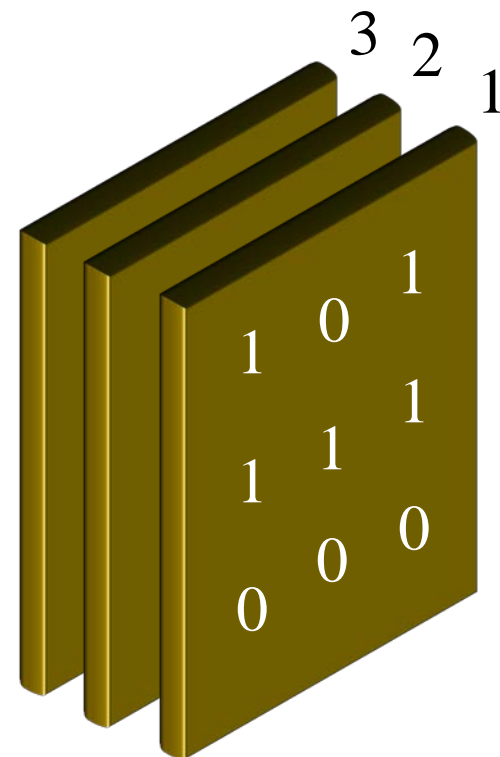
Codifica basata sui Bit plane

215	180	111
77	161	211
0	194	144

- L'immagine a 8-bit è scomposta in 8 piani ad un bit
- Il piano 1 contiene il bit meno significativo di tutti i pixel nell'immagine



11010111	10110100	01101111
01001101	10100001	11010011
00000000	11000010	10010000






Codifica basata sui Bit plane

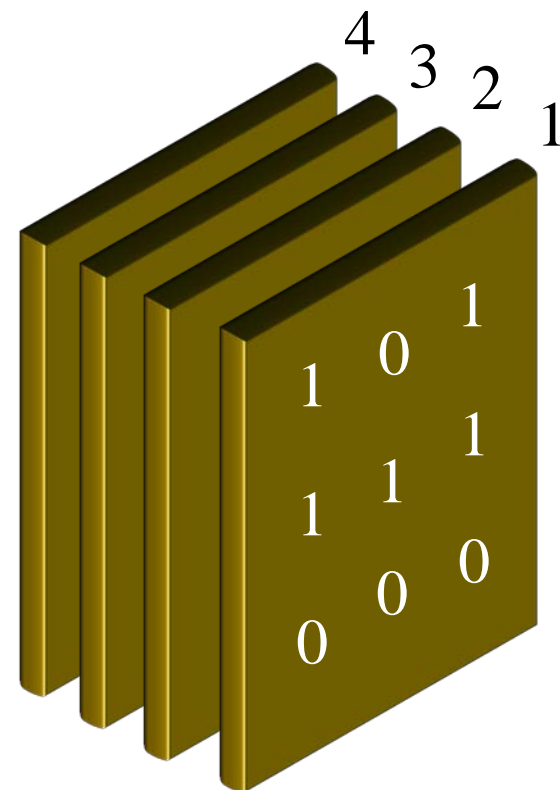
215	180	111
77	161	211
0	194	144

- L'immagine a 8-bit è scomposta in 8 piani ad un bit
- Il piano 1 contiene il bit meno significativo di tutti i pixel nell'immagine



11010111	10110100	01101111
01001101	10100001	11010011
00000000	11000010	10010000

Red vertical boxes highlight the least significant bit (the rightmost bit) of each pixel in the 3x3 grid.






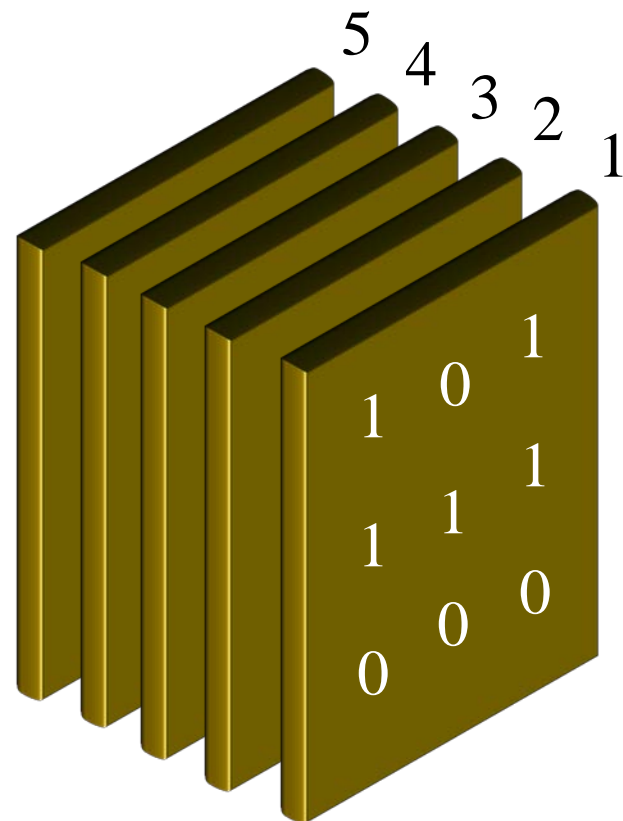
Codifica basata sui Bit plane

215	180	111
77	161	211
0	194	144

- L'immagine a 8-bit è scomposta in 8 piani ad un bit
- Il piano 1 contiene il bit meno significativo di tutti i pixel nell'immagine



11010111	10110100	01101111
01001101	10100001	11010011
00000000	11000010	10010000






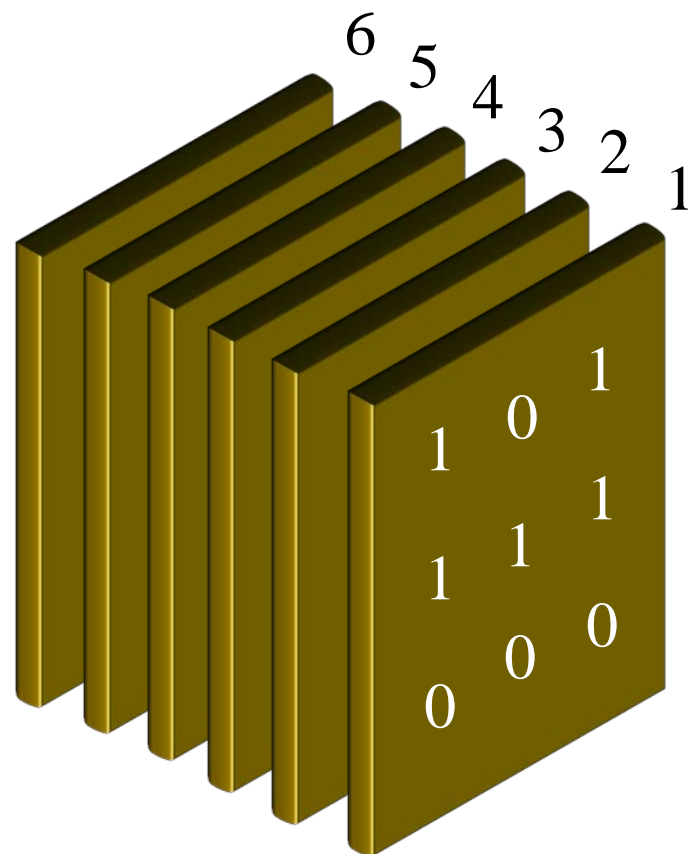
Codifica basata sui Bit plane

215	180	111
77	161	211
0	194	144

- L'immagine a 8-bit è scomposta in 8 piani ad un bit
- Il piano 1 contiene il bit meno significativo di tutti i pixel nell'immagine



11010111	10110100	01101111
01001101	10100001	11010011
00000000	11000010	10010000





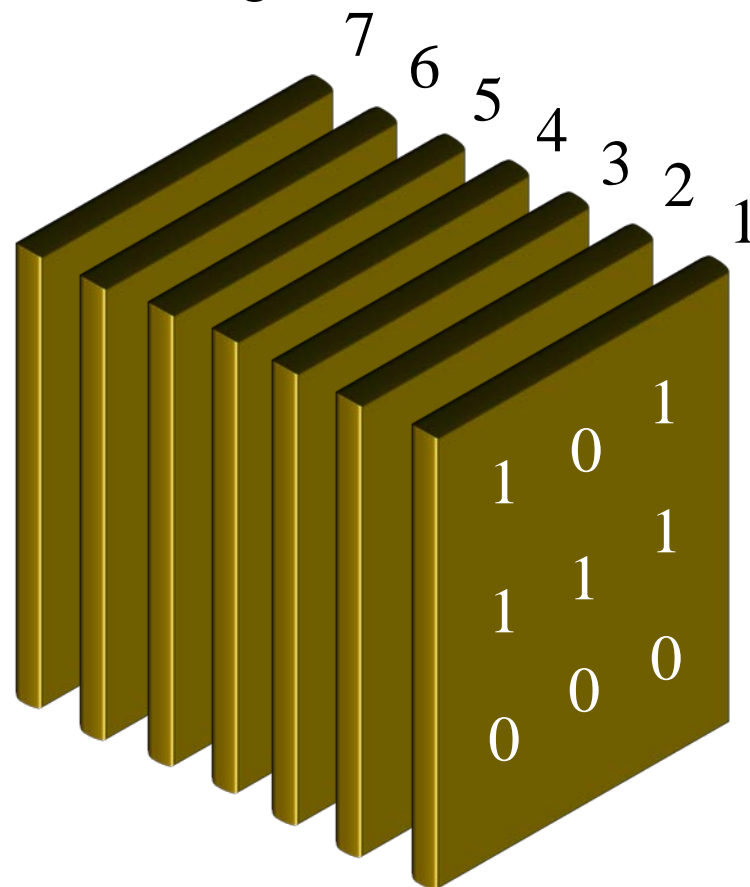
Codifica basata sui Bit plane

215	180	111
77	161	211
0	194	144

- L'immagine a 8-bit è scomposta in 8 piani ad un bit
- Il piano 1 contiene il bit meno significativo di tutti i pixel nell'immagine



11010111	10110100	01 101111
01001101	10100001	11010011
00000000	11000010	10010000






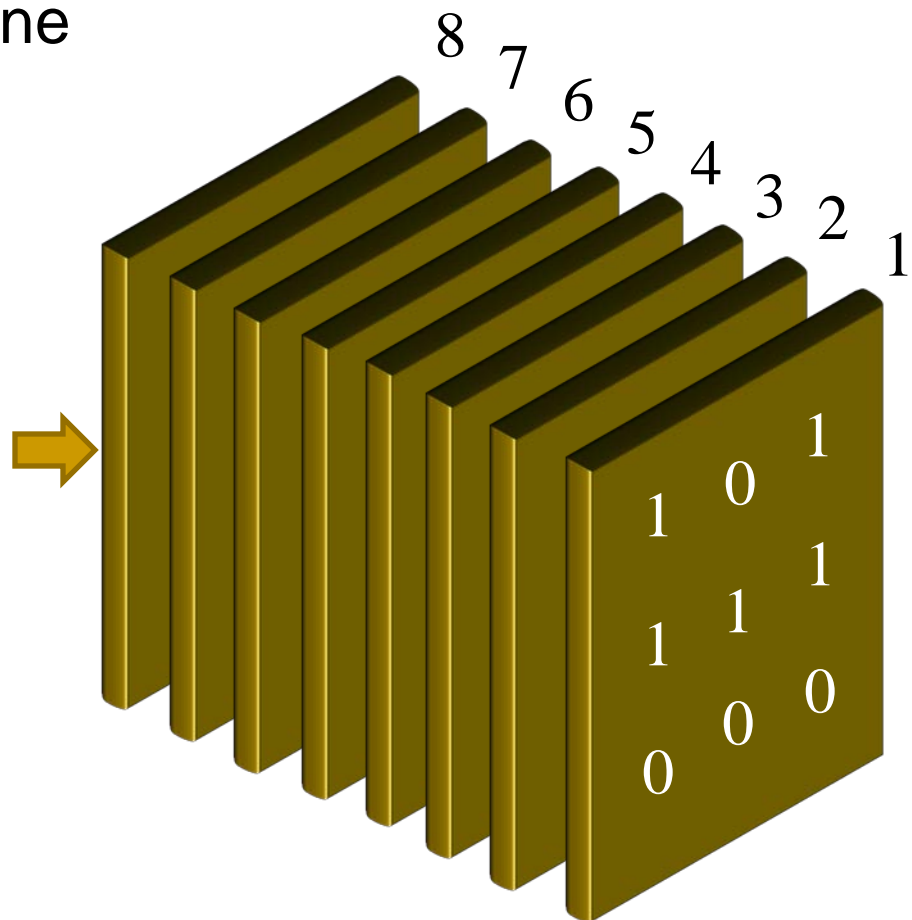
Codifica basata sui Bit plane

215	180	111
77	161	211
0	194	144

- L'immagine a 8-bit è scomposta in 8 piani ad un bit
- Il piano 1 contiene il bit meno significativo di tutti i pixel nell'immagine



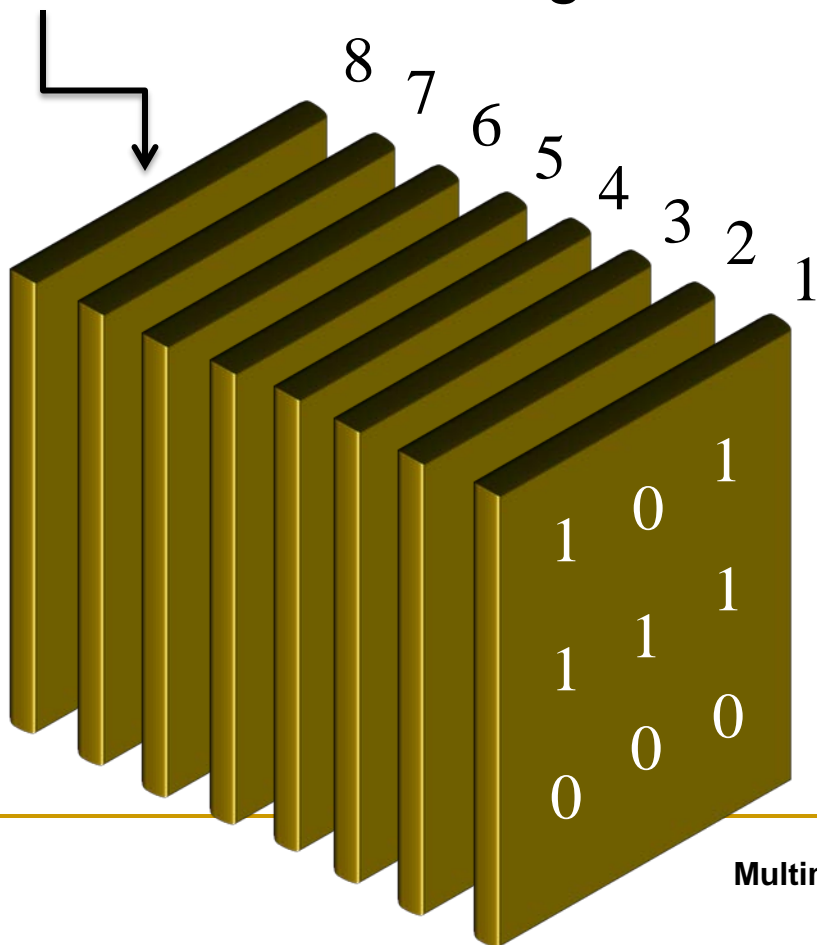
11010111	10110100	01101111
01001101	10100001	11010011
00000000	11000010	10010000





Codifica basata sui Bit plane

- I piani di ordine più alto contengono le aree più uniformi e con meno dettagli dell'immagine

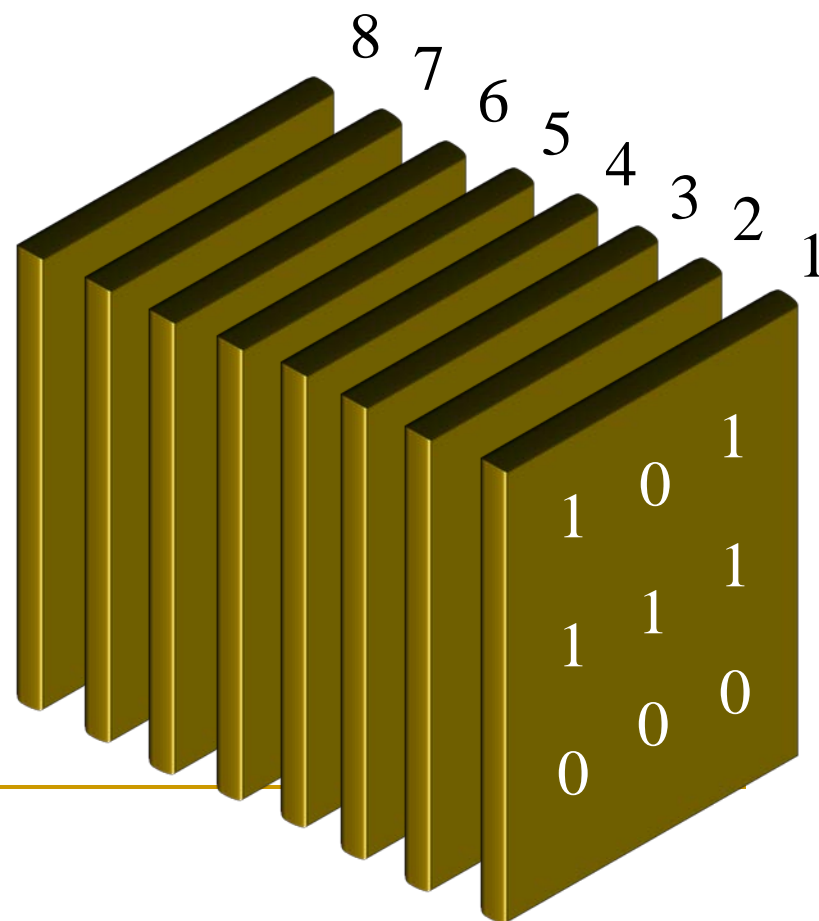


- I piani di ordine più basso contengono i dettagli dell'immagine



Codifica basata sui Bit plane

- La scomposizione in piani di bit ha lo **svantaggio** che se si ha la presenza di piccole variazioni di intensità, questa si ripercuote su tutti i piani di bit
- Se un pixel ha ad esempio intensità 127 (01111111) e il suo adiacente ha intensità 128 (10000000) allora la transizione tra 0 e 1 si ripercuote su tutti i piani di bit. Per risolvere questo problema si usa la **variante con XOR della scomposizione bitplanes**





Codifica basata sui Bit plane Variante con XOR e Gray Code

- Il codice di Gray a m -bit $g_{m-1} \dots g_2 g_1 g_0$ che corrisponde al numero in binario $a_{m-1} \dots a_2 a_1 a_0$ può essere calcolato da

$$g_i = a_i \oplus a_{i+1} \quad 0 \leq i \leq m - 2$$
$$g_{m-1} = a_{m-1}$$

- dove \oplus denota l'operatore di OR esclusivo.
- Questo codice gode delle proprietà per cui ogni codeword successiva differisce dalla precedente per solo un bit.



Codifica basata sui Bit plane

Variante con XOR – Esempio

Vantaggio: piccole variazioni di intensità hanno bassa probabilità di influenzare tutti gli m piani di bit

Considerando i pixel di intensità 127 e 128 avremo:

<i>Bit</i>		<i>Gray code</i>
01111111	→	01000000
10000000		11000000

solo il piano di ordine più alto è influenzato

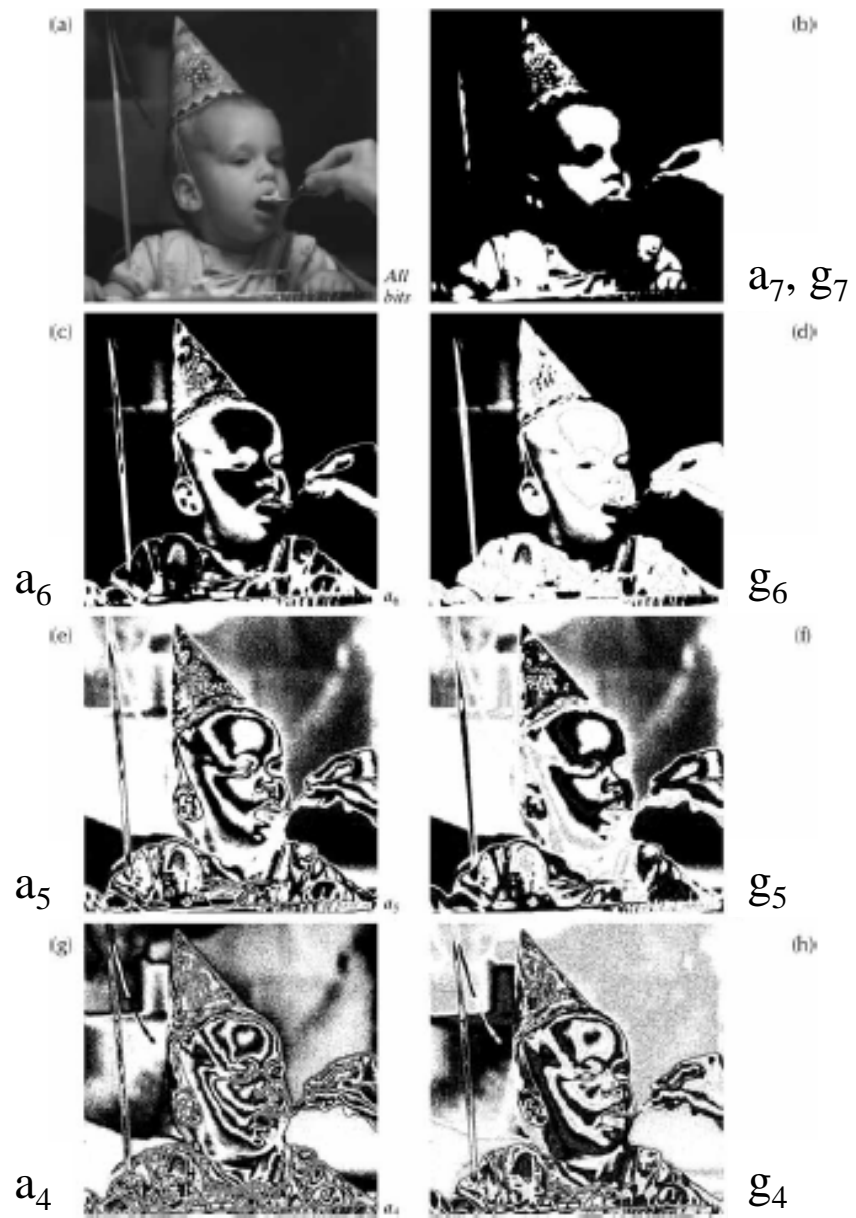


Figura 8.19 (a) Immagine a toni di grigio a 256 bit. (b)-(h) I quattro più significativi piani di bit gray coded dell'immagine in (a).

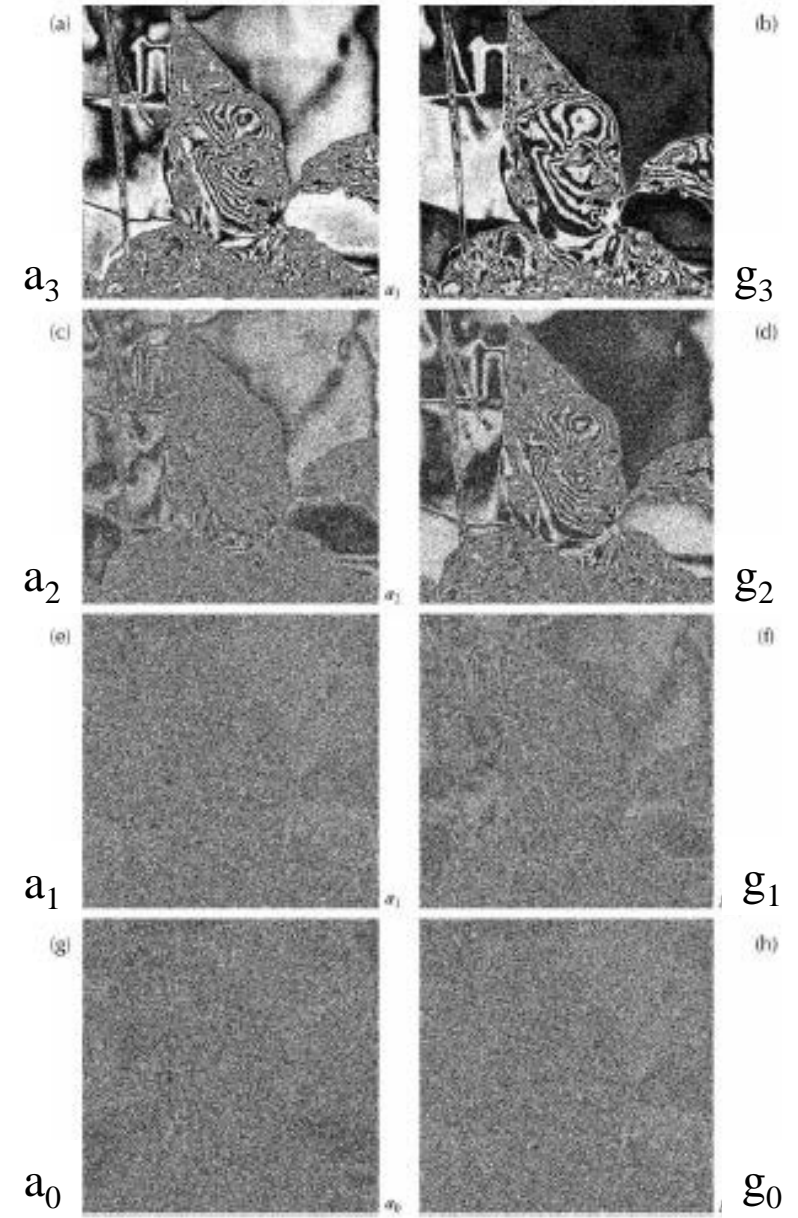


Figura 8.20 (a)-(h) I quattro piani di bit binari (colonna di sinistra) e gray coded (colonna di destra) meno significativi dell'immagine nella Figura 8.19a.