10-01-2023

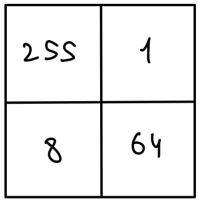
BIT PLANE

E' un modo per rappresentare un segnale (*nel nostro caso un immagine*) **come insieme di piani dove ogni piano ha solo un bit**.
In caso di immagine, quindi, si rappresenta l'immagine nella versione come un insieme di immagini binarie, quindi come n immagini a 1 bit.

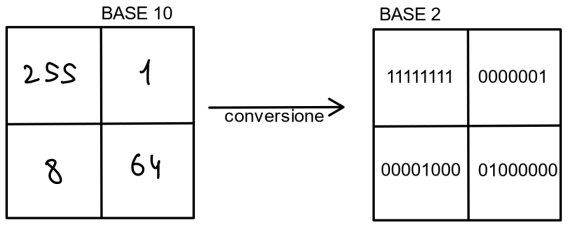
> Se l'immagine è a scala di grigi (8 bit) allora serviranno 8 immagini binarie dove in ogni immagine c'è 1 bit in una posizione specifica

Esempio

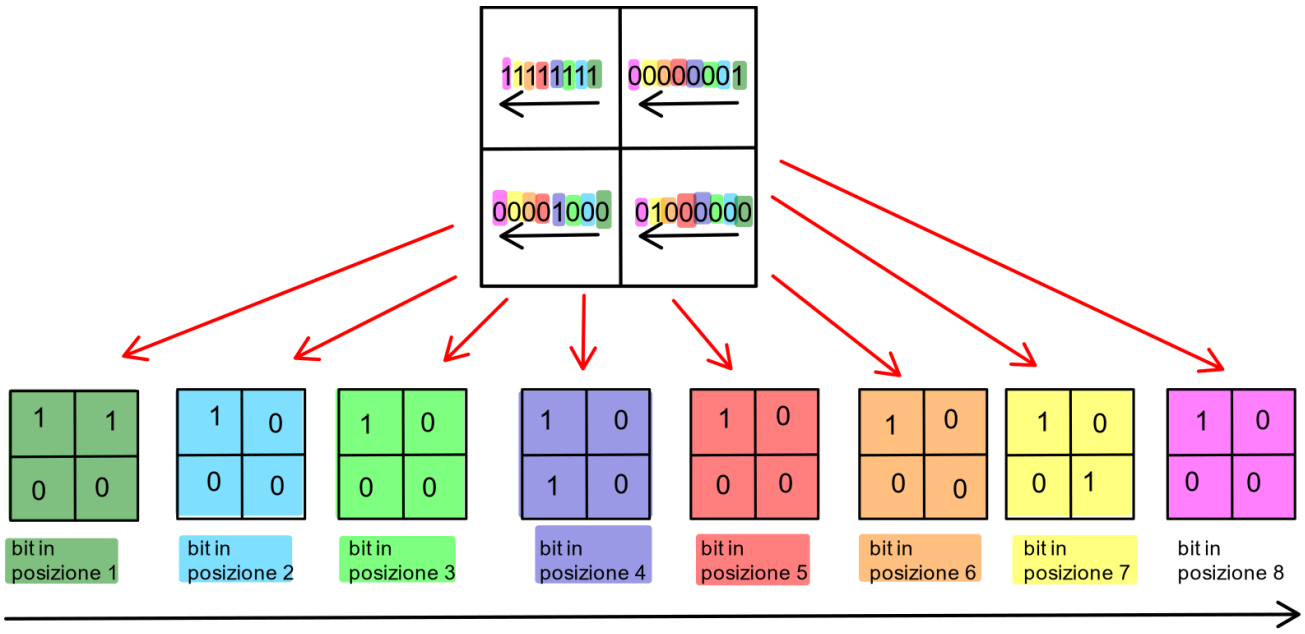
Data un immagine a 8 bit (con valori in base 10):



Questi numeri possono essere convertiti da base 10 a base 2, quindi si ottiene una codifica in **BINARIO PURO** ("*puro*" vuol dire "la base 2 del numero"). L'immagine in base 2 viene:



Dopo la conversione, l''immagine continua ad avere lo stesso significato.
Una volta rappresentata in base 2, allora è più semplice andare a rappresentarla in rappresentazione in **bit plane**, ovvero scomporla in 8 immagini. Si ottengono 8 immagini binarie:

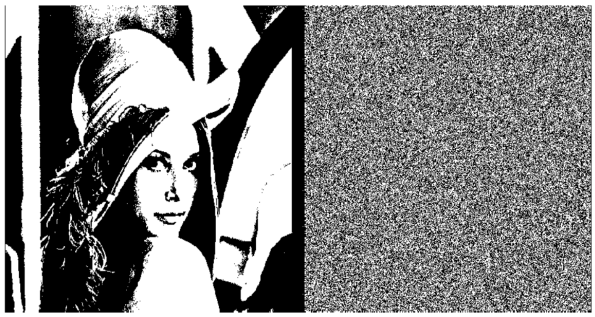


Cioè vado a vedere nella corrispondente cella il bit che corrisponde alla posizione della matrice che sto considerando. E si ottiene cosi l'insieme di n immagini binarie

Definizione generale

Il bit plane di un'immagine digitale a N bit, è un'insieme di N immagini binarie (piani), in cui l'immagine i-esima contiene i valori dell' i-esimo bit della codifica scelta (nel nostro caso la codifica è binaria).

Esempio con Lena:



Most Significant bit (**MSB**) Least Significant bit (**LSB**)

> *Perchè a destra non capisco nulla e a sinistra si?*
> In generale, nell'immagine binaria un pixel è bianco e uno è nero anche se valori di grigio sono molto simili.
> Una piccola variazione emerge immediatamente(nell'immagine di destra).
> Vengono **presentate tutte le microvariazioni di grigio** (LSB)
> Nell'MSB per far variare da bianco a nero serve una variazione maggiore.

> Nella codifica binaria, i bit più significativi sono quelli a sinistra e quelli meno significativi sono a destra.

Fatte queste considerazioni si nota subito che

- il **RUMORE** si presenta nei **bit meno significativi** perchè variano primi.

- i **bit più significativi** permettono di riconoscere le **forme e altri dettagli**.

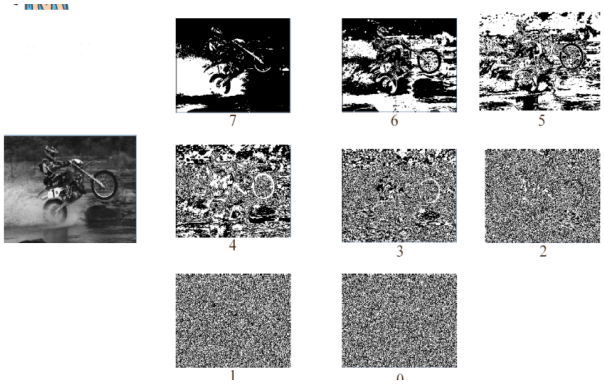
Possibile algoritmo di compressione

Un possibile algoritmo di compressione è:

- Se scopro che dei dettagli non sono cosi importanti (meno significativi) considero che quei bit **NON sono utili** (quindi butto via dei dettagli) e **considero** solo alcuni **bit più a sinistra**, *(per esempio dal 5 in poi)*.
- Quindi butto alcuni bit per comprimere meglio e **risparmiare memoria**.

Esempio:

- Dall'immagiene sottostante si nota che **SOLO dal bit 4 in poi** si iniziano a distinguere geometrie e forme dell'immagine di partenza



> *Perchè si usa bit plan?*
> **Per agire in range particolari**: Ad esempio, se si vogliono eliminare tutti i grigi compresi tra 32 e 64, è necessario porre a 0 il quinto bit, e quindi tutto il piano 5.

Ricostruzione solo con alcuni bit plane

Data la seguente immagine e il suo bit plane:

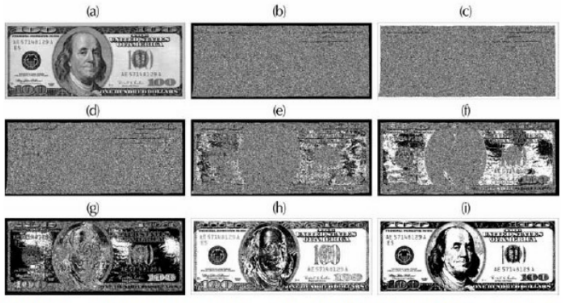


Figura 3.14 (a) Immagine a 8 bit in scala di grigio di 500×1192 pixel. Da (b) a (i) i piani di bit da 1 a 8; il piano 1 corrispondente al bit meno significativo. Ogni piano è un'immagine binaria.

La si vuole ricostruire utilizzando solo con alcuni piani. (quindi buttandone via alcuni meno significativi) Il risultato è il seguente:

Figura 3.15 Immagini ricostruite usando (a) i piani di bit 8 e 7; (b) i piani di bit 8, 7 e 6; (c) i piani di bit 8, 7, 6 e 5. Si confronti (c) con la Figura 3.14a.

- L'**immagine risultante** a sembra **abbastanza significativa** (*anche se si usano meno bit*).
- Si ha un risparmio del 50% di spazio senza perdere cosi tanto come dettagli quindi non è una scelta cosi grave.

Problema bit plane

- Se la codifica usata è quella in binario puro, allora risulta evidente uno svantaggio: una piccola variazione può ripercuotersi su tutti i piani. *Capita quando ci sono passaggi da certi valori ad altri*:
  - **CASO PEGGIORE**: quando passo da 127 a 128 nella binarizzazione. Cioè da 01111111 passo a 1000000 ed è un problema.
    - **Esempio pratico** : 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.

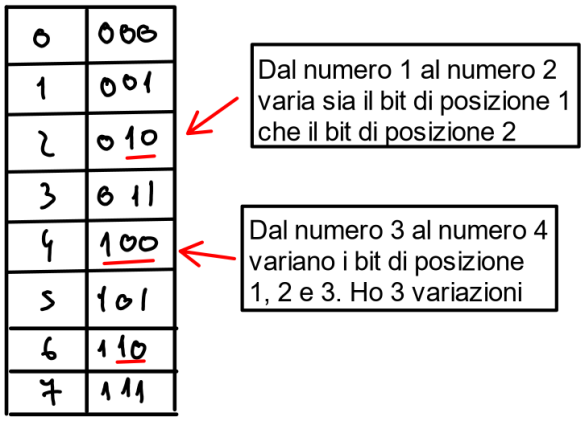
L'obiettivo è quello di avere **piccole variazioni di bit** se l'**intensità di grigio varia poco**

> Ciò che è diverso, ***ridondante*** e quindi si **comprime peggio**

Soluzione: cerco di costruire un codice avente la seguente premessa:

- quando avanzo di 1, fra parola di bit attuale e precedente, deve variare solo un bit, ovvero la codifica di un valore e il successivo deve variare di 1

Spiegazione con disegno: converto i numeri da decimali a binario e vedo come variano i bit.



Gray Code

Si usa lo **XOR** e la sua tabella di verità è:

IN	IN	XOR
0	0	0
1	0	1
1	0	1
1	1	0

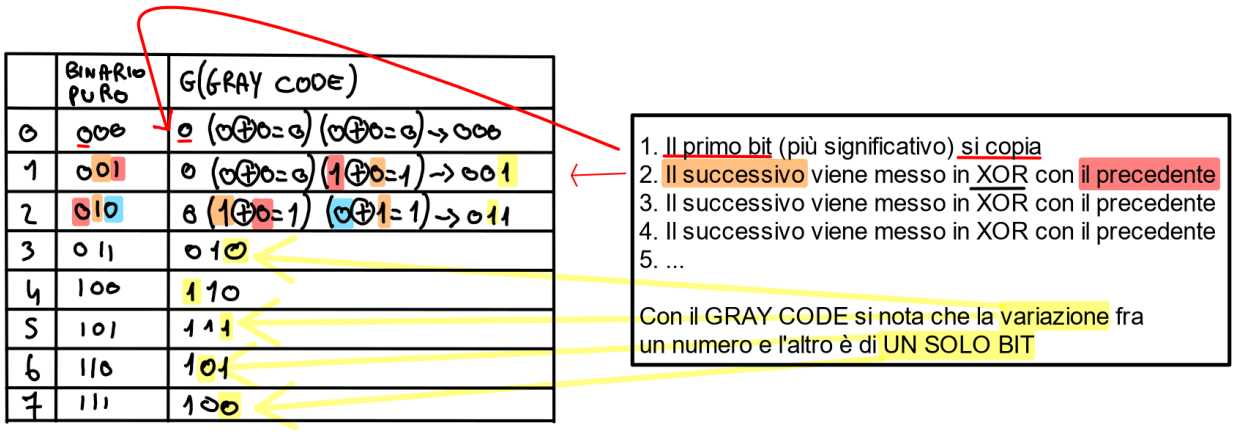
Il livello di grigio g_i in Gray Code da assegnare è dato dalla seguente formula:

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

$$g_{m-1} = a_{m-1}$$

dove:

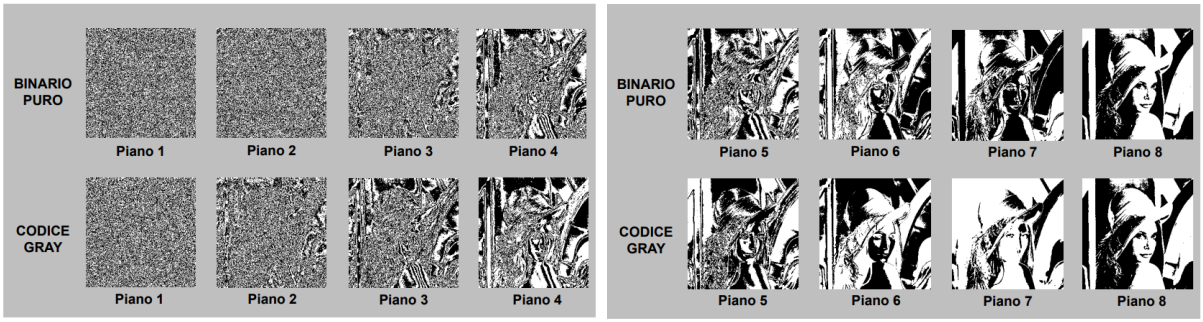
- $\oplus$ denota l'operatore XOR (detto anche *OR esclusivo*)
- g_i denota il nuovo valore in **GRAY CODE** da assegnare dopo aver effettuato lo XOR.
- a_i indica il bit di posizione i nella codifica in binario puro a
  L'idea del Gray Code è la seguente:
- **il più significativo si copia** cosi e com'è;
- **il successivo bit** è il **risultato** fra il **bit attuale** e il **precedente**
  Si ottiene:



Vantaggi Gray Code

- **A livello visivo**: si ha **maggiore coerenza** rispetto al binario puro (quindi capisco un po' di più l'immagine di partenza)
- **A livello hardware**: (*in caso di trasmissione in rete di immagini*)
  - con il **Binario Puro**, il passaggio da 127 a 128 (*per esempio*) comporta la **commutazione di 8 flip flop** (*o altro componente hardware*) visto che le **variazioni di piccoli livelli di grigio comportano la variazione di molti bit**.

    - con il **Gray Code**, il passaggio da 127 a 128 (*per esempio*) comporta la **commutazione di un solo flip flop** (o *altro componente hardware*) visto che **la variazione di piccoli livelli di grigio comportano la variazione di un solo bit**.
- Queste caratteristiche indicano una **minore entropia**. Ciò significa che diventa più **semplice comprimere** a partire da immagini così codificate. Quindi posso **conservare i dati usando meno bit** per rappresentarli, quindi risparmio spazio



ATTENZIONE!

- Dato che il significato associato ai bit è diverso tra le due codifiche, alcune proprietà di una non valgono per l'altra!
- Se si azzerano dei piani di bit in Gray code, si eliminano range di valori **diversi** (e meno significativi) rispetto a quelli in binario puro.
- Nonostante i dettagli e il rumore tenderanno a concentrarsi nei piani più bassi anche con il codice Gray, eliminare direttamente tali piani potrebbe introdurre artefatti indesiderati.

Dopo aver codificato in Gray Code **NON** vale più la proprietà (la **posizione** identifica l'"*importanza del bit*") che valeva nel binario puro.
Ovvero che **"minor importanza" NON VUOL DIRE** più che posso buttare i bit meno significativi perchè loro stessi potrebbero diventare significativi.

- Di conseguenza si può affermare che **i dettagli trovo nei bit meno significativi SOLO IN BINARIO PURO**.
- In caso di altra codifica (*tipo gray code*) non si sa più dove si trovano i bit piu significativi.

> *Esempio domanda a trabocchetto esame:*
>
> - Domanda: Ho 1 codifica in gray code, cosa succede se si eliminano i bit meno significativi?
> - Risposta: potrebbe comportare la distruzione completa dell'immagine visto che tale proprietà vale solo per il binario puro.

COMPRESSIONE

Per compressione si intende un insieme di **tecniche** che ha come **scopo la riduzione** il **numero di bit** per rappresentare una certa informazione.

- (*ridurre* = si è già scelta una rappresentazione)

Per trasmettere, archiviare, elaborare certe informazioni, le rappresentiamo con sequenze di bit.

> *Posso rappresentare la stessa informazione con meno bit di quelli che ho effettivamente deciso di usare?*
> Se la risposta è sì allora posso comprimere.

La compressione può essere:

1. **LOSSY** - con perdite: posso rappresentare l'informazione nonostante a livello **PERCETTIVO** sia **uguale all'originale** (anche se si hanno meno bit) (*butto **qualcosa** di percettivamente **NON RILEVANTE***)
2. **LOSSLESS** - senza perdite: posso rappresentare l'informazione esattamente così e com'è con meno bit senza buttare alcuna informazione che la riguarda.

Vantaggi della compressione

- Si ha un'**ottimizzazione dei supporti di archiviazione**.
- Il **disco ha più valore** se lo posso occupare con la stessa informazione ma sfruttando meno la capacità. (*sfrutto meno memoria per memorizzare la stessa informazione*)

Ridondanze della codifica

Un'informazione si comprime quando si vede che c'è *qualcosa* di ripetuto, ovvero qualcosa in più. Questo "qualcosa in più" viene denominato **RIDONDANZA**. (*qualcosa di ripetuto che non è necessario ripeterlo*)

> Esempio ridondanza **spaziale**: 2 pixel vicini i pixel hanno valori simili in una foto fotorealistica.
>
> - Se guardo un pixel so che *MOLTO* probabilmente il successivo è ridondante.
> - Si preferisce **conservare la differenza fra un pixel e un altro** piuttosto che il valore stesso.

Codeword e lunghezza di essa

- Un **CODICE** è un **sistema di simboli** (*lettere, numeri, bit, ecc*) utilizzati per rappresentare una certa **quantità di informazioni**.

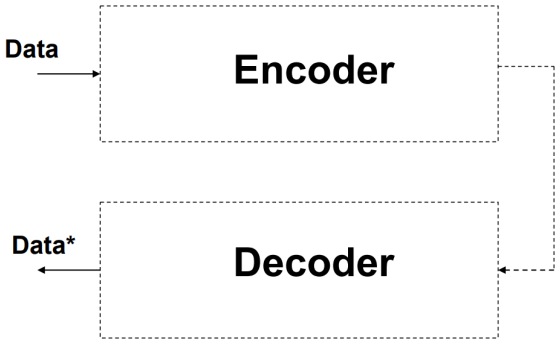
Si parla di:

- **CODEWORD** = codice che associo a un ***simbolo*** (ad ogni ***evento*** che vedo nell'informazione che voglio rappresentare) che voglio memorizzare:
  - **simbolo**= **la più piccola quantità informativa** che desidero memorizzare *(come un valore in scala di grigi)*;
    - Esempio: codeword per il colore bianco = `11111111`.
- **LUNGHEZZA** = fissate delle codeword, è il numero di elementi che costituisce la codeword.

Nei contenuti multimediali c'è spesso una quantità di informazione che posso decidere di buttare perchè non è rilevante a livello **PERCETTIVO (chiaramente non si preserva il contenuto originale).**

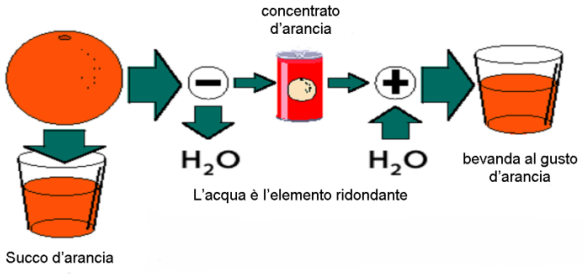
ALGORITMO DI COMPRESSIONE

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



> L'asterisco serve per far capire che $Data$ e $Data^*$ non sono la stessa cosa perchè si parla di **compressione in generale**.
>
> - Se si parlasse di compressione senza perdita (*LOSSLESS*) le due parole $Data$ in INPUT e $Data^*$ in OUTPUT potrebbero essere uguali, appunto.

Altro esempio per capire la compressione:



Classificazione dei metodi di compressione

Il metodo è basato sul tipo di compressione:

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

Lossless

> - Il formato `.png` è un formato che ha un algoritmo di compressione lossless.
> - In **caso di un testo** o di una **serie di parole alfanumeriche** non è possibile buttare dei caratteri quindi l'algoritmo di compressione dovrebbe essere **lossy forzatamente**

Shannon è riuscito a stabilire **il numero minimo di bit** necessari per poter rappresentare una qualunque sequenza di simboli.
Esiste un *TEOREMA* che dice che:

- Data una sequenza di simboli (*possono anche essere pixel*), se riesco a memorizzarla, **si devono usare almeno N bit**.
- Ciò implica che **NON** si può trovare una codifica con meno bit di quelli indicati dal teorema.
  - Se trovo una codifica che usa meno bit degli N bit indicati dal teorema, vuol dire che ho scartato delle informazioni.

L'obiettivo è cercare di raggiungere il **limite teorico** sotto il quale non si può scendere per la compressione senza perdita che viene fornito dal primo teorema di Shannon (*che vedremo tra poco*).

Frequenza

Data una sequenza S di N caratteri (*colori o qualunque sequenza di informazioni*) tratti da un alfabeto di M possibili caratteri: $a_1, \ldots, a_M$, allora la frequenza f_i del carattere a_i è data da:

$$f_i = \frac{n\ occorrenze\ a_i}{N}$$

Esempio:

Dato un alfabeto fatto da {A, B, C, D} e costruisco la stringa $AAABCCCB$

- Allora $N = 8$
- $a_0 = A$
- $a_1 = B$
- $a_2 = C$

- $a_3 = D$
- Mentre le frequenze dei caratteri nella stringa sono:
  - $f_0 = f_A = f_{a_0} = \frac{3}{8}$
  - $f_1 = f_B = f_{a_1} = \frac{2}{8}$
  - $f_2 = f_C = f_{a_2} = \frac{3}{8}$
  - $f_3 = f_D = f_{a_3} = \frac{0}{8}$

ENTROPIA

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

$$E = -\sum f_i \, log_2(f_i)$$

> **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** (cioè si presentano tutti con la stessa frequenza) e questo rappresenta il **CASO PEGGIORE**

L'idea è quella di associare la frequenza di un evento ad un numero di bit inferiore o maggiore. Più in pratica:

- fenomeno più **RARO** = uso **più bit** per rappresentarlo;
- fenomeno più **COMUNE** = uso **meno bit** per rappreentarlo.

Esempio pratico

1. Data una moneta non truccata essa viene lanciata:

- può uscire testa con 50% di probabilità $\rightarrow 0.5$;
- può uscire croce con 50% di probabilità $\rightarrow 0.5$
  Se calcolo l'entropia, mediamente serve 1 bit per memorizzare l'esito del lancio di una moneta, cioè serve per 1 bit per ogni lancio. In questo caso l'entropia risulta massima e rappresenta il caso peggiore.

2. Data una moneta truccata che agisce nel seguente modo: su 1000 lanci esce 800 volte testa e 200 croce, allora:

- può uscire testa con probabilità $80\% \rightarrow 0.8$;
- può uscire croce con probabilità $20\% \rightarrow 0.2$.
- L'entropia varrà <1, cioè serve meno di un bit per memorizzare il lancio di una moneta truccata per un singolo esito.
  - Mediamente servono 0.7 bit per esito. Ma non posso usare 0.7 bit per un esito, perchè non è un numero intero, ma tipo potrei usarlo per 10 esiti e userei 7 bit che

sono comunque meno dei precedenti 10 (*caso facce equiprobabili*).

> $N \times E$ è **il limite teorico** per memorizzare una sequenza di lunghezza N

TEOREMA DI SHANNON(1948)

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 $N \times E$ dove:

- N è il numero di caratteri;
- E è l'entropia.

Significato delle parole:

- **BITRATE** = numero di bit per simbolo.
- **SORGENTE** = ciò che genera la sequenza (ciò che genera l'immagine, ovvero la sequenza di pixel). Si preferisce parlare di entropia della sorgente
- **DISCRETA** = il numero di simboli possibili è **FINITO**
- **MEMORIA ZERO** = **la probabilità** di osservare un simbolo **NON dipende** da **quello che si è visto prima**
  - In italiano dopo la "gn" è molto probabile che ci sia una vocale e NON una consonante e quindi questa probabilità dipende da quello che si osserva prima.
  - Se osservo un pixel posso stimare quello che c'è dopo quindi non si tratta di memoria zero.

> Occorre usare un algoritmo che permetta di codificare i nostri caratteri usando esattamente il numero di bit ricavati con il teorema di Shannon.
> **Ci si deve avvicinare molto al limite teorico** e raramente si riesce a beccare.
> L'algoritmo che fa questo è Huffman.

Esempio 1:

Dato un alfabeto {A, B, C, D} e la stringa $AAAABBCC$:

- Si scrivono le relative **frequenze** dei caratteri:
  - $f_A = \frac{4}{8}$
  - $f_B = \frac{2}{8}$
  - $f_C = \frac{2}{8}$
- Si calcola l'**entropia** $E = -\sum f_i \, log_2(f_i)$

- $E = -(\frac{4}{8}\ log_2(\frac{4}{8}) + \frac{2}{8}\ log_2(\frac{2}{8}) + \frac{2}{8}\ log_2(\frac{2}{8})) = -(\frac{2}{4}(-1) - \frac{1}{4}(-2) + \frac{1}{4}(-2)) = -(-\frac{6}{4}) =$
- Quindi mediamente, per rappresentare un simbolo, mi servono 1.5 bit. Per **rappresentare tutta la sequenza** mi serve $N \times E = 8 \times 1.5 = 12\ bit$ cioè **AL PIU'** DI **12bit**. minimo si devono usare 12 bit. si codifica esattamente la stessa stringa usando meno bit.

Esempio 2: codifica con lunghezza variabile

Data la codifica:

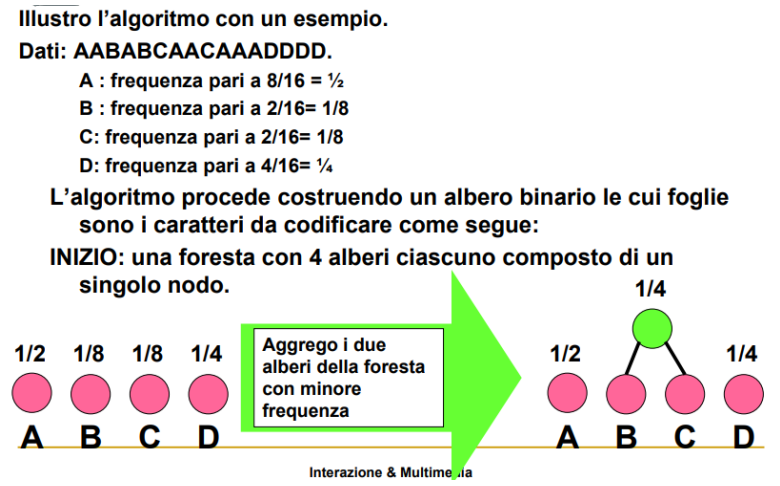
- $A = 1$
- $B = 00$
- $C = 01$
  La stringa $AAAABBCC$ viene codificata nel modo seguente:
- 1 1 1 1 00 00 01 01 e questa è la migliore codifica LOSSLESS.
  L'idea è quella di assegnare **codeword più corte** a **chi si presenta più spesso** perchè quello **occupa più memoria**.

> Con l'algoritmo Greedy di Huffman, appunto, si assegnano dei codici di lunghezza più corta ai simboli più frequenti.

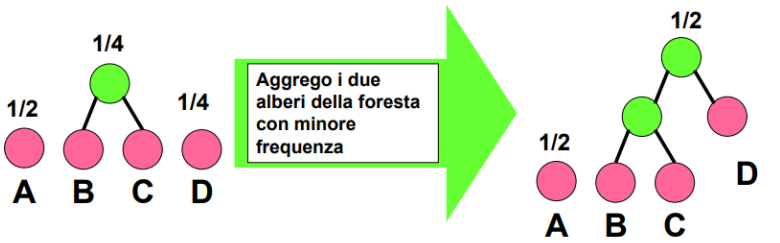
Proprietà Huffman

- 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**. (per evitare problemi di ambiguità in caso di decodifica.
  - Per esempio, nel caso precedente, per capire l'ambiguità si può assegnare $C = 11$ piuttosto che $C = 01$
- È una **codifica ottimale** perchè tende al limite imposto dal teorema di Shannon.

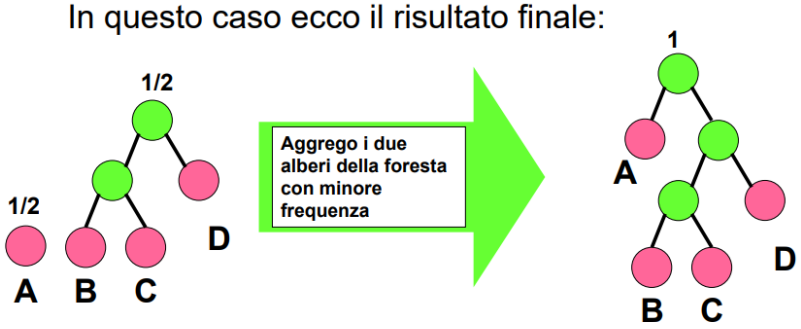
Esempio Huffman



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

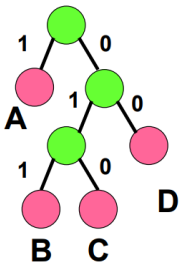


In questo caso ecco il risultato finale:



In altre parole:

- le **foglie** dell'albero sono i **caratteri disponibili** nell'alfabeto (A, B, C, D)
- iterativamente si **accoppiano** a 2 a 2 i nodi dell'albero
  - si prendono i 2 nodi con frequenza più bassa e si uniscono, in questo caso B e C.
- il nuovo nodo creato (verde in figura) ha come frequenza la somma delle frequenze, cioè $\frac{4}{16}$.
- *fra i rimanenti nodi (insieme al nuovo nodo creato) chi ha frequenza più bassa?* in questo caso $\frac{4}{16}$ e li fondo in $\frac{8}{16}$
- Si itera tale processo fino all'esaurimento delle coppie formabili.
- Tutti i nodi destri li etichetto con 0, tutti i sinistri li etichetto con 1 (*scelta arbitraria*)
- Trovo la codifica nel seguente modo:
  - partendo dalla radice, il percorso che seguo per arrivare al simbolo T, trovo la codifica (lungo gli archi che percorro) del simbolo T in maniera ottimale.



Riassumendo si ha che:

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

- **Codice per la sequenza**
  AABABCAACAAADDDD.
  - **A : frequenza pari a 8/16 = ½**
  - **B : frequenza pari a 2/16= 1/8**
  - **C: frequenza pari a 2/16= 1/8**
  - **D: frequenza pari a 4/16= ¼**

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 = 28 bit – CODIFICA OTTIMALE!

Il limite per Shannon è $N \times E$, cioè $N \times (-\Sigma f_i log_2(f_i)) = 28$

- Huffman serve per **trovare un algoritmo** che sia **MOLTO vicino** al **limite teorico** esposto da *Shannon*

Attenzione in pratica!

Se si usa Huffman per comprimere, **si deve memorizzare anche la tabella** che si crea.

- **Costo aggiuntivo**: si deve memorizzare la tabella $caratteri - codici$. Se i caratteri sono tanti la memorizzazione può essere costosa. Se, invece, la tabella è piccola allora il costo della memorizzazione è trascurabile.
- 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 $CCITT$, $JBIG2$, $JPEG$, $MPEG-1,2,4$.

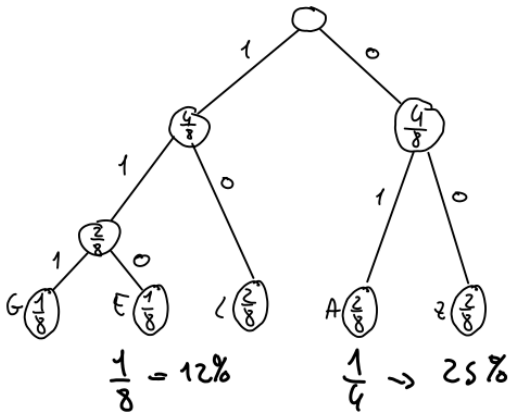
Esercizio (esame)

- 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”?

PUNTO 1:

ALFABETO: $\{G, A, z, E, L\} \rightarrow N = 8$

FREQUENZE: $f_G = \frac{1}{8}$ $f_A = \frac{2}{8}$ $f_z = \frac{2}{8}$ $f_E = \frac{1}{8}$ $f_L = \frac{2}{8}$



	COD
G	111
A	01
z	00
E	110
L	10

Codifica di "gazzella":

"111 01 00 00 110 10 10 110"

Si usano 18 bit che corrispondono esattamente al limite teorico di Shannon.
Per tale motivo questo è il miglior algoritmo di codifica della stringa.

ALFABETO: $\{G, A, z, E, L\} \rightarrow N = 8$

FREQUENZE: $f_G = \frac{1}{8}$ $f_A = \frac{2}{8}$ $f_z = \frac{2}{8}$ $f_E = \frac{1}{8}$ $f_L = \frac{2}{8}$

PUNTO 2

$$E = -\sum f_i \log_2(f_i)$$

$$= -\left(\frac{1}{8}\log_2\frac{1}{8} + \frac{2}{8}\log_2\left(\frac{2}{8}\right) + \frac{2}{8}\log_2\left(\frac{2}{8}\right) + \frac{1}{8}\log_2\left(\frac{1}{8}\right) + \frac{2}{8}\log_2\left(\frac{2}{8}\right)\right) =$$

$$= -\left(\frac{1}{8}(-3) + \frac{2}{8}(-2) + \frac{2}{8}(-2) + \frac{1}{8}(-3) + \frac{2}{8}(-2)\right) =$$

$$= -\left(-\frac{3}{8} - \frac{4}{8} - \frac{4}{8} - \frac{3}{8} - \frac{4}{8}\right) =$$

$$= -\left(-\frac{18}{8}\right) = \frac{18^{9}}{8_{4}} = \frac{9}{4} = 2{,}25 \ (E)$$

Per ogni singola informazione servono in media 2.25 bit.

LIMITE TEORICO $N \cdot E = 8 \cdot 2{,}25 = 16$

Secondo il teorema di Shannon,
Per rappresentare la stringa servono minimo 16 bit per avere una compressione LOSSLESS

RUN-LENGTH-ENCODING (RLE)

- In questo algoritmo un **certo simbolo** si **può ripetere molte volte di fila** (*disegni con ampie aree con lo stesso colore, pixel uguali uno dopo l'altro*);
- **Si memorizzano delle coppie** dove ogni coppia è fatta da (**simbolo**, **quante volte esso viene ripetuto**)

Per esempio:

- invece di memorizzare la stringa $AAAAAABBB$ memorizzo $(A, 6), (B, 3)$ che potrebbe occupare meno spazio (*potrebbe* perchè non è detto che sia effettivamente cosi).

Note RLE

- Nella maggior parte dei casi **ci fa occupare più bit dei previsti**. E' raro che si migliori la situazione.
- E' **utile** quando si **hanno pochi valori con sequenza molto lunghe** di caratteri uguali:
- Si potrebbe fare pre-processing affinché compaiono delle sequenze lunghe proprio come le vorrei io.

Esempio:

Si voglia comprimere la sequenza:

0000011100101111011101011111111

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:

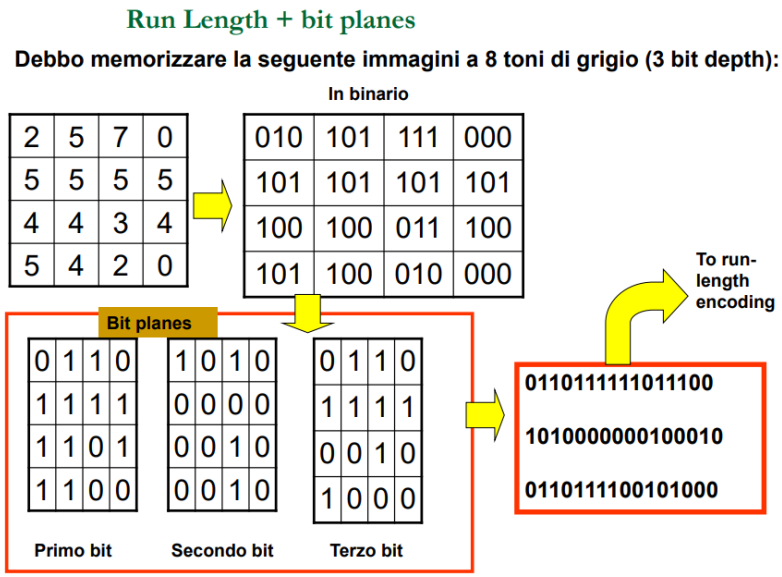
5,3,2,1,1,3,1,3,1,1,1,7

Tali valori vanno adesso scritti in binario. 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.

- Posso fare questo lavoro solo quando ho solo **2 possibilità**, quindi nella codifica posso trovare **0 oppure 1**:
  - Se leggo una codifica RLE e trovo un "*successivo*" numero (se ho una sequenza di 11111 e poi trovo uno 0, per esempio 1111**0**00) vuol dire che è cambiato simbolo ed è superfluo indicare altro.
  - **Serve** solo **sapere il primo simbolo** e poi è implicito che, quando si trova un simbolo diverso dalla sequenza che si sta analizzando, vuol dire che è cambiato simbolo (visto che si sta analizzando la codifica di solo 2 simboli).

Diventa sensato codificare con RLE nel seguente caso:



> In questo caso noto che cambiano molto frequentemente i valori;

- Usando i bit plane e mettendo le righe in modo sequenziale (*come un array*) trovo **run** più lunghe. (come Lena binarizzata);
- Ne segue che se ho run molto lunghe allora posso applicare RLE

CODIFICA DIFFERENZIALE

L'idea è la seguente:

- Se 2 pixel vicini, per come è fatto il dato, sono nella maggior parte delle volte simili, mi conservo le differenze fra i valori adiacenti.

Esempio:
Dati i seguenti valori: 134, 137, 135, 128, 130, 134, 112, ... ricorderò il valore iniziale 134 e poi la sequenza delle differenze successive: $-3, +7, 1, 7, -2, -4, 22, \ldots$.

- *Perchè conservo le differenze e non la sequenza originale?*
- Si dimostra che **l'entropia nella sequenza delle differenze è più bassa dell'entropia della sequenza originale**.
- Il limite di Shannon si abbassa.
- Si ha il vantaggio quando si possono usare meno bit e quando le **differenze sono piccole**