Università degli Studi del Piemonte Orientale
Facoltà di Scienze M.F.N.

Corso di Laurea in Informatica

Tesina per il corso d'Algoritmi e Strutture Dati II

Studio sulle tecniche di Compressione Dati

Guido Vicino
Anno Accademico 2003/2004

INDICE

Capitolo 1 Introduzione

Nel trattamento e soprattutto nella trasmissione di dati digitali è molto importante ridurre le dimensioni, questo sia perché permette di memorizzare un maggior numero di dati in uno spazio finito e sia perché permette di trasmetterli più velocemente.

La *Compressione Dati* mira a risolvere questi problemi, alcune volte utilizzando una codifica più efficiente, altre volte eliminando le ridondanze o i dati superflui.

Si vedranno quindi gli algoritmi di compressione più conosciuti, e alcune delle loro implementazioni più diffuse.

Capitolo 2 Tecniche e algoritmi di compressione dati

Il problema principale che ci si trova a dover risolvere in questo campo è l'intrinseca entropia presente nei dati, appunto per questo si sono sviluppati nel tempo algoritmi di compressione "universali" o per specifiche tipologie di dati.

I diversi algoritmi possono essere raggruppati in due gruppi:

- Algoritmi a compressione statistica.
- Algoritmi a compressione tramite sostituzione di testo o con dizionario.

La prima tipologia si basa sullo studio statistico dei simboli e degli elementi presenti nel codice, questi possono essere parole o caratteri. Se ne studia quindi la frequenza assoluta e relativa per associare a pattern frequenti codici più brevi e a frammenti più rari codici più lunghi (vedi Huffman).

La seconda tipologia invece si basa sull'uso di puntatori a porzioni di codice precedenti, basandosi sul fatto che spesso questi puntatori sono più brevi che codificare l'intero pattern. Sono quindi creati degli opportuni dizionari di simboli per realizzare queste codifiche. Simili algoritmi sono chiamati adattativi perché si conformano alla costituzione interna del documento digitale.

Esistono altre due categorie di algoritmi:

- Lossless
- Lossy

Negli algoritmi *Lossless* il dato digitale non subisce perdite è semplicemente codificato in un altra maniera, quindi decifrandolo (decomprimendolo) si ottiene lo stesso dato preso in input.

Gli algoritmi *lossy* non assicurano invece una reversibilità assoluta, questi algoritmi possono essere utilizzati in quei tipi di dati quali immagini o audio, dove sono possibili perdite senza intaccare la visibilità dei contenuti. Il vantaggio di questi algoritmi è che creano un file compresso di dimensioni notevolmente inferiori.

Capitolo 3 Algoritmi Lossless ed Entropia

Nell'andare ad esaminare alcuni algoritmi è bene tenere presente conto dell'Entropia del testo sorgente.

Si definisce entropia di una sorgente dati:

$$x_i \, con \, i = 1, \ldots, n \, con \; probabilità \; P(x_1), .. P(x_n)$$

$$H(X)=\sum_{i=1}^{n} P(x_i)\log_2\left(\frac{1}{(P(x_i))}\right)$$

L'entropia ci dà la misura della quantità d'informazione contenuta nel flusso di bit che deve essere sottoposto a codifica.

Dal teorema di Shannon si ricava che il rapporto di compressione ottenibile ossia la lunghezza media di un codice è limitata inferiormente all'entropia della sorgente:

$$H(X)\leqslant L_{media}$$

Nel caso dei codici Huffman esaminati adesso si può dimostrare che:

$$H(X)\leqslant L_{media}\leqslant H(X)+1$$

Codici di Huffman

Nel 1952 David A. Huffman pubblico un articolo in cui mostrava un metodo per la costruzione di codici con Minima-Rindondanza, questo metodo è stato quindi utilizzato come algoritmo di compressione non distruttivo in diverse implementazioni. Huffman partendo dalla definizione di "ridondanza" di Shannon afferma che:

> A "minimum-redundancy code" will be defined here as an ensemble code which, for a message ensemble consisting of a finite numbers, N, and for a given number of coding digits, D, yields the lowest possible average message length.
> David A. Huffman

Questi codici devono essere caratterizzati dall'essere codici prefissi, cioè quei codici dove nessuna parola è anche un prefisso di qualche altra parola del codice. Questa specifica semplifica molto la codifica (compressione) e la decodifica (decompressione). Alcuni esempi di codici validi sono: 01, 102, 111 e 202, questi possono comporre qualsiasi parola come 20210201111 senza ambiguità di sorta sia per il mittente sia per il ricevente del messaggio.

Ecco riassunto il processo:

1. Si analizza il numero di ricorrenze di ciascun elemento costituente il messaggio originale, quindi ad esempio le occorrenze per i vari caratteri di un file di testo.
2. Si determinano i due elementi meno frequenti e si accomunano in un nuovo gruppo che li rappresenta entrambi, sommandone la frequenza.
3. Per passaggi successivi si crea un albero binario all'interno del quale appaiono con maggiore frequenza ed in combinazioni successive gli elementi più rari all'interno di un file. In questa maniera i simboli più frequenti saranno codificati con parole più corte e viceversa quelli più rari

con parole più lunghe.

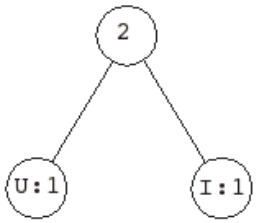
4. Dopo aver creato queste corrispondenze tra elementi e codifica, si produce in output il nuovo testo compresso.

Procediamo con un esempio:

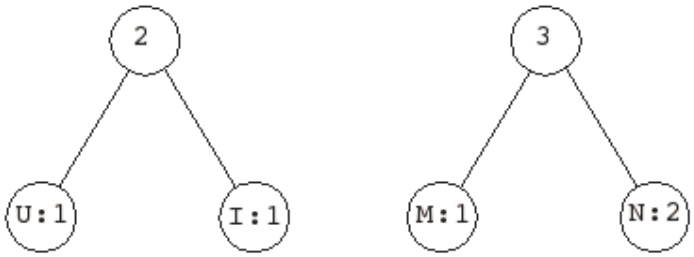
La stringa da codificare è unipppmn si costruisce quindi una tabella di frequenze:

u: 1, i: 1, m: 1, n: 2, p: 3

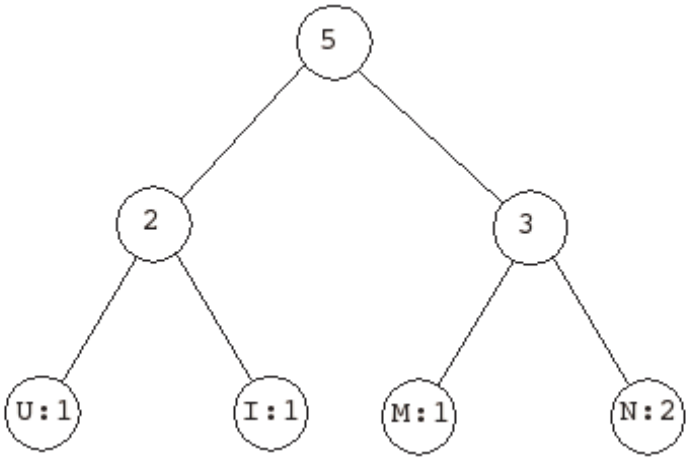
si selezionano gli elementi con frequenza minore e s'inizia a costruire l'albero:



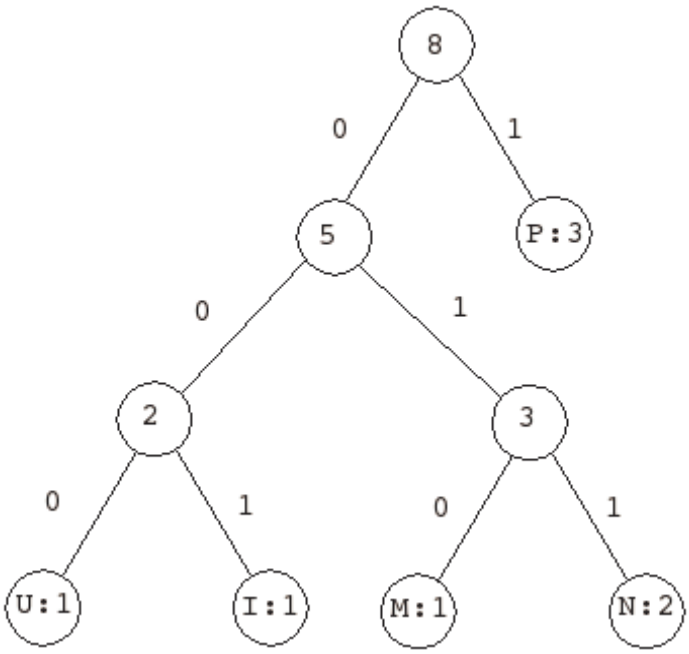
ui: 2, m: 1, n: 2, p: 3



ui: 2, mn: 3, p: 3



a questo punto fondiamo il tutto e otteniamo:



Quest'ultimo grafo ci può aiutare a capire la codifica che sarà la seguente:

u: 000, i: 001, m: 010, n: 011, p: 1

la stringa ricostituita sarà:

000011111010011

avrà una lunghezza di 15 bit, rispetto ai 32 bit della codifica originale (8 bit per carattere), quindi si avrà una compressione del 46% circa.

Passando ad analizzare un ipotetico albero *T* che rappresenta un codice di Huffman è possibile determinarne il costo uguale a:

$$H(T)=\sum_{c\in A} f(c)\,d_t(c)$$

dove *c* è un carattere dell'alfabeto *A* presente nel file, *f (c)* la sua frequenza assoluta e *d (c)* è la lunghezza della parola che codifica quel determinato carattere.

LZ77 – LZ78

Nel 1977 Jacob Ziv e Abraham Lempel pubblicarono un articolo riguardante un algoritmo universale di compressione dati sequenziali, questo prese successivamente il nome di *LZ77*.

Nella prima versione di questa codifica si utilizzava un dizionario definito da una finestra di testo di dimensione fissa, durante la codifica questa scorreva sul testo da comprimere. La possibilità di trovare una corrispondenza nel dizionario per la stringa esaminata è quindi dipendente dai dati contenuti nella window in quell'istante e non

tiene memoria di record precedenti.
L'algoritmo quindi cerca ad ogni passo nella sequenza contenuta nella window una sottostringa già incontrata, se tale sequenza esiste è sostituita con un puntatore alla precedente.
Si potrebbe pensare di aumentare l'efficacia dell'algoritmo aumentando la dimensione della window, questo però oltre a causare per pattern corti codifiche più lunghe causa anche un rallentamento nei tempi di codifica e decodifica.

Per risolvere questi problemi Lempel e Ziv un anno dopo svilupparono la codifica *LZ78*, nell'algoritmo del 1977 il dizionario era limitato al concetto della window che è eliminata in questa nuova versione. Nel LZ78 si costruisce un dizionario inserendo successivamente le stringhe sconosciute, quindi il dizionario diventa illimitato, e il concetto di puntatore è sostituito con quello di un *token* che va' a sostituire man mano le varie sequenze.

L'algoritmo di codifica utilizzato in LZ78 è questo, si consideri il dizionario vuoto all'inizio:

```
prec = c = prendiCodiceInInput();
while(c = prendiCarattereInInput() != -1)
{
     if(verificaPresenzaNelDizionario(prec + c))
     {
          prec = prec +c;
     }
     else
     {
          stampaOutput(codificaDi(prec) + c);
          inserisciNelDizionario(pref + c);
          pref = null;
     }
}
```

La decodifica è analoga alla codifica, all'inizio il dizionario contiene solo la stringa vuota e mentre viene riempito viene anche restituita la codifica originale.

La differenza con il precedente algoritmo è il ridotto numero di confronti tra stringhe e quindi una maggiore velocità, riguardo al coefficiente di compressione non ci sono rilevanti differenze.

LZW(Lempel-Ziv-Welch)

Nel 1984 Terry Welch rivisito l'algoritmo LZ78 cercando di migliorarlo (sotto il punto di vista della velocità), le modifiche riguardano sostanzialmente il dizionario che ora non è più vuoto e l'output in cui sono restituiti solo i codici.

L'algoritmo *LZW* si basa sulla creazione di un dizionario di stringhe ricorrenti in un file basandosi su un dizionario di base che di norma è costituito dai 255 caratteri della codifica ASCII. Leggendo un carattere alla volta si aggiorna il dizionario mantenendo la sequenza codificata più lunga finora incontrata e considerando sempre l'ultimo carattere.

L'algoritmo di compressione si può quindi riassumere in questa maniera:

```
seq = prendiCarattereInInput();
while(c = prendiCarattereInInput() != -1)
{
     if(verificaPresenzaNelDizionario(seq+c))
     {
          seq = seq + c;
     }
     else
     {
          stampaOutput(seq);
          inserisciNelDizionario(seq+c);
          seq = c;
     }
}
stampaOutput(seq);
```

Viceversa l'algoritmo di decompressione si può descrivere in questa maniera:

```
prec = c = prendiCodiceInInput();
stampaOutput(prec);

while(new_c = prendiCodiceInInput() != -1)
{
     if(!verificaPresenzaNelDizionario(new_c))
          string = traduciCodice(prec) + c;
     else
          string = traduciCodice(new_c);

     stampaOutput(string);

     c = restituisciPrimoChar(string);
     inserisciNelDizionario(prec + c);
     prec = new_c;
}
```

Facciamo un esempio ipotizzando però che il nostro dizionario iniziale sia limitato a questo:

```
A/1
B/2
C/3
```

La stringa da codificare sarà

```
A B B A B A B A C
```

il processo di codifica sarà:

```
D = {A/1 B/2 C/3 AB/4}
Output = {1}

D = {A/1 B/2 C/3 AB/4 BB/5}
Output = {1,2}
D = {A/1 B/2 C/3 AB/4 BB/5 BA/6}
Output = {1,2,2}

D = {A/1 B/2 C/3 AB/4 BB/5 BA/6 ABA/7}
Output = {1,2,2,4}

D = {A/1 B/2 C/3 AB/4 BB/5 BA/6 ABA/7 ABAC/8}
Output = {1,2,2,4,7}

D = {A/1 B/2 C/3 AB/4 BB/5 BA/6 ABA/7 ABAC/8}
Output = {1, 2 , 2, 4, 7, 3}
```

Il processo di decodifica sarà:

```
D = {A/1 B/2 C/3}
Output = A

D = {A/1 B/2 C/3 AB/4}
Output = A B

D = {A/1 B/2 C/3 AB/4 BB/5}
Output = A B B

D = {A/1 B/2 C/3 AB/4 BB/5 BA/6}
Output = A B B A B

D = {A/1 B/2 C/3 AB/4 BB/5 BA/6 ABA/7}
Output = A B B A B A B A

D = {A/1 B/2 C/3 AB/4 BB/5 BA/6 ABA/7 ABAC/8}
Output = A B B A B A B A C
```

Per la sua velocità rispetto a LZ78 questo metodo è molto utilizzato, alcuni esempi sono dati da:

- L'utility UNIX `compress/uncompress.`
- Il formato grafico *TIFF*.
- Il formato grafico *GIF*.
- Lo standard *V.42* bis per la trasmissione dati.

RLE – Run Length Encoding

L'algoritmo RLE è una forma molto semplice di codifica per la compressione dati. Esso è basato sul fatto che in ogni flusso sono presenti diversi dati simili (i valori ripetuti sono chiamati *run*), questi dati ridondanti saranno sostituiti con un contatore e il valore ripetuto. Questo tipo d'algoritmo intuitivo funziona bene con tipologie di dati

in cui sono presenti un largo numero d'occorrenze consecutive dello stesso byte pattern. Ovviamente RLE si presterà bene per la compressione di formati grafici come BMP, PCX, TIFF (basandosi sulla ridondanza spaziale delle immagini) e viene anche utilizzato in alcune implementazione di formati editoriali come PDF.

Possiamo trovare diverse implementazioni di quest'algoritmo:

RLE a livello di bit: ogni byte codifica sequenze di bit uguali.

```
Versione non compressa:
000000111111100001111111111111111111100000000000

Versione compressa:
00001100 00001111 00001000 00101001 00010110
```

RLE a livello di byte: ogni run è codificato da due byte.

```
Versione non compressa:
93 23 22 6 6 6 6 6 6 6 6 99 25 2 2 2 2 2 2 2

Versione compressa:
1 93 1 23 1 22 9 6 1 99 1 25 7 2
```

RLE a livello di byte misto: si utilizza il concetto di byte di lunghezza, ossia un byte in cui il bit più indicativo denota il tipo di run (1 = compresso, 0 = non compresso).

```
Versione non compressa:
93 23 22 6 6 6 6 6 6 6 6 99 25 2 2 2 2 2 2 2

Versione compressa:
3 93 23 22 137 6 2 99 25 135 2
```

Ovviamente esistono altre implementazioni, ad esempio si possono codificare i *run* con 4 byte, secondo l'utilizzo che se ne può fare (comprimere pixel, dati ecc).

Capitolo 4 Algoritmi Lossy

Graphics Interchange Format

Negli anni 80 CompuServe sviluppo un formato grafico per lo scambio d'immagini sul WWW, questo formato fu chiamato GIF che sta per "Graphics Interchange Format" ed utilizza di base l'algoritmo non distruttivo LZW.

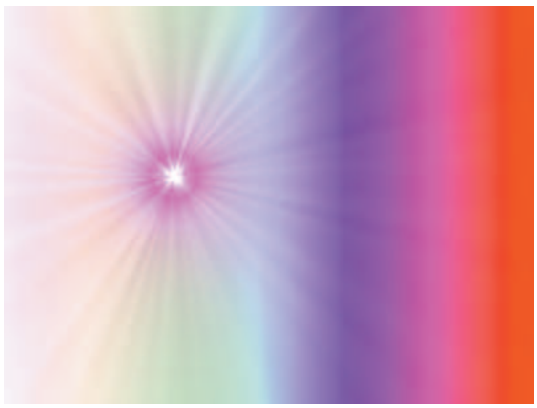
Questo formato però è limitato al fatto che può utilizzare al massimo una tavolozza di 256 colori, quindi se il formato originale è codificato con maggiore risoluzione come RGB o CMYK si subirà una consistente perdita d'informazione.

Il formato GIF supporta un immagine codificata con un minimo di 1 bit per pixel fino ad un massimo di 8 bit, dove 1 corrisponde ad un immagine in bianco e nero (senza sfumature di grigi) e poi si proseguirà con 5, 8,16, 32, 128 e infine 256 colori.

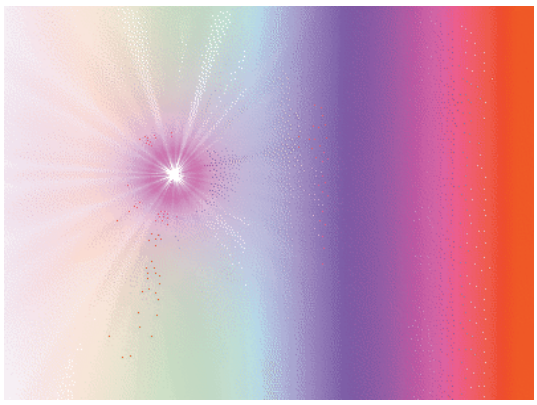
La conversione operata sull'immagine originale trasforma i valori RGB o CMYK originali in una loro approssimazione dipendente dal tipo di tavolozza prescelta per effettuare l'operazione. Nella creazione dell'immagine GIF saranno quindi inglobati i valori cromatici di questa tavolozza in modo da consentirne la generazione dell'immagine. I diversi programmi di grafica permettono a discrezione dell'utente di utilizzare diversi sistemi di conversione e quindi di tavolozze (adattata, ottimizzata, uniforme, personalizzata, web ecc).

Ovviamente nonostante tutto la perdita d'informazioni sarà enorme, come possiamo vedere nelle due immagini seguenti:

La prima in formato tiff quindi RGB:



La seconda in formato gif:



Si può notare la perdita di qualità.

Per i suoi limiti, il formato GIF è consigliato quando si devono creare piccole immagini con poche variazioni di colore.

JPEG

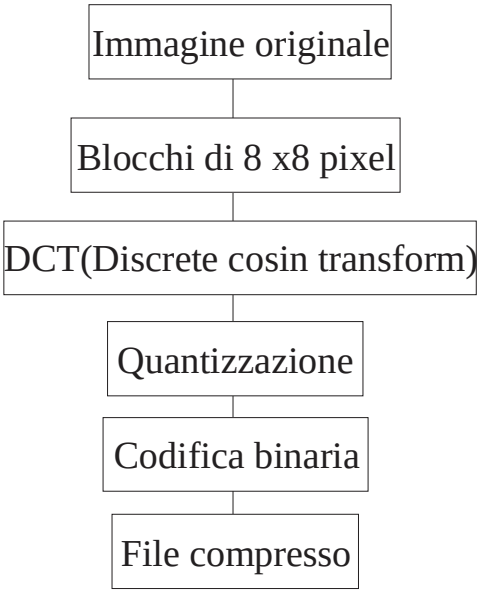
Il formato grafico JPEG è lo standard di compressione sviluppato dal *Joint Photographic Expert Group* e permette di comprimere con un alto tasso di compressione immagini grafiche caratterizzate da tono continuo cioè da fotografie a

colori o a toni di grigio o immagini molto sfumate.

Dal 1986 ad oggi si è formato un complesso insieme di regole che definiscono il formato e sono state standardizzate e approvate come standard ISO nel 1990. Con JPEG si intende lo standard industriale che non va' confuso con il formato di file JPG che rappresenta di volta in volta un'implementazione particolare da parte delle varie software house.

Lo standard JPEG ottiene una buona compressione sfruttando le limitazioni fisiologiche dell'occhio umano, quindi è stato progettato in maniera tale da eliminare i pixel meno rilevanti per il nostro cervello che sono quelli riguardanti le piccole variazioni di colore cui siamo meno sensibili.

Esistono diversi gradi di compressione in questo formato, man mano che si aumenta questo tasso, diminuisce però la qualità perché man mano si perdono informazioni sull'immagine.



Il processo di compressione JPEG

Il processo di compressione di una JPEG è il seguente:

1. Si effettua una trasformazione da modalità *RGB* a modalità *YUV* poiché l'occhio umano è più sensibile alle variazioni di luminosità che alle variazioni di colore.
   Lo spazio colore *YUV* scompone l'informazione d'ogni pixel in due parti che sono la *luminanza* (definente il grado di luminosità nella scala dal bianco al nero) e la *crominanza* (definente il colore base al rapporto tra due assi, dal blu al giallo e dal rosso al verde).
   Questa trasformazione seppure non indispensabile consente di ottenere una maggiore compressione dato che favorisce le operazioni matematiche per l'eliminazione delle informazioni cromatiche senza danneggiare quelle relative alla luminosità.

2. Un'altro passo opzionale è quello relativo alla riduzione, in base alla componente di gruppi di pixel a valori medi. In questo punto la componente cromatica è dimezzata orizzontalmente o verticalmente mantenendo intatta la luminanza. Ci

possono essere diversi coefficienti di riduzione come 1/1. 2/1 4/1/1, 4/2/2.

3. Dopo aver diviso l'immagine in blocchi da 8x8 pixel si applica la DCT.
   La *Discrete Cosine Transform (DCT)* aiuta a separare l'immagine in parti (precisamente in sotto bande spettrali) di differente importanza (rispettando la qualità visuale dell'immagine).

   La DCT è simile alla trasformata di Fourier: essa trasforma un segnale o un immagine da un dominio spaziale ad un dominio di frequenza. Con un immagine in input *A* i coefficienti per l'"immagine" in output *B* sono:

$$B(k_{1,}k_2)=\sum_{i=0}^{N_1-1}\sum_{j=0}^{N_2-1}4\cdot A(i,j)\cdot\cos\left[\frac{\pi\cdot k_1}{2\cdot N_1}\cdot(2\cdot i+1)\right]\cdot\cos\left[\frac{\pi\cdot k_2}{2\cdot N_2}\cdot(2\cdot j+1)\right]$$

   Dove l'immagine in input è larga *N1* e alta *N2*, l'intensità' in pixel della riga *i* e della colonna *j* sarà *A (i, j)*, mentre il coefficiente DCT alla riga *k1* e alla colonna *k2* della matrice creatasi sarà *B (k1, k2)*.

   Tutte le moltiplicazioni DCT sono reali, questo abbassa il numero di moltiplicazioni richieste rispetto a quelle della trasformata discreta di Fourier.

   L'input della DCT deve essere un array di 8 per 8 interi, quest'array contiene per ciascun pixel 255 livelli che saranno trasformati in frequenze che variano da -721 a 721 oppure -1024 a 1023 (a seconda che l'immagine sia a colori o in bianco e nero).

   La trasformazione in frequenze consentirà nei passaggi successivi di tagliare più informazioni senza apparente perdita visiva, prendendo atto che le frequenze più importanti per l'occhio umano (ossia le più basse) saranno memorizzate nell'angolo superiore sinistro del blocco 8x8 e saranno quelle che saranno preservate.

4. Avviene la divisione e l'arrotondamento dei 64 valori ottenuti, ciascuno di essi è diviso per uno specifico coefficiente di quantizzazione codificato in opportune tavole di riferimento, dopodiché avviene un arrotondamento del risultato all'intero più vicino.

5. Ai valori risultanti della divisione e dell'arrotondamento è applicato un algoritmo di compressione non distruttivo. Dato che dopo la quantizzazione, la maggior parte dei coefficienti DCT sono pari eguali a zero, JPEG incorpora una codifica run-lenght per avvantaggiarsi di questo fenomeno. Sono quindi utilizzati codici di Huffman, Shannon-Fano o di codifica aritmetica.

6. Sono inserite nell'intestazione le tabelle contenenti i coefficienti di quantizzazione e i valori di trasformazione della codifica Huffman al fine di poter ricostruire un'immagine fedele dall'originale.

JPEG 2000

L'organo JPEG ha sviluppato un nuovo standard chiamato JPEG2000, questo standard è ancora più rivoluzionario del precedente, costruito appositamente per la

distribuzione su Web, PDA, cellulari, PC e televisioni.

Alcuni esempi delle innovazioni portate da JPEG2000:

1. Supporto per differenti modalità e spazi colore (bianco e nero, scala di grigi, 256 colori, milioni di colori, RGB, CMYK, etc).
2. Supporto per diversi algoritmi di compressione.
3. Standard aperto.
4. Consente sia la compressione con perdita d'informazione (lossy), sia quella senza perdita (lossless).
5. Produce immagini con qualità visive migliori, specialmente a bassi bit-rate, rispetto a quelle ottenibili con JPEG, grazie alle proprietà della trasformata Wavelet.
6. Supporto per immagini più grandi di 64k x 64k pixel.
7. Supporto per la codifica ROI ("Region Of Interest") che enfatizza la codifica di determinate zone dell'immagine salvandole con risoluzione maggiore rispetto al resto dell'immagine.
8. Supporto per la trasmissione dati alla presenza di rumori.
9. Inclusione di diverse risoluzioni per download e per "thumbnail".

Il processo di codifica si può così riassumere:

1. Qui si procede in due maniere secondo il tipo d'immagine:

   Se l'immagine è in bianco e nero viene suddivisa in tante regioni disgiunte chiamate *tiles*, la dimensione di ciascuna tile è medesima tranne per quelle di bordo che possono essere più piccole. Questa suddivisione serve per permettere l'accesso ad una zona determinata dell'immagine riducendo la memoria necessaria per la compressione.

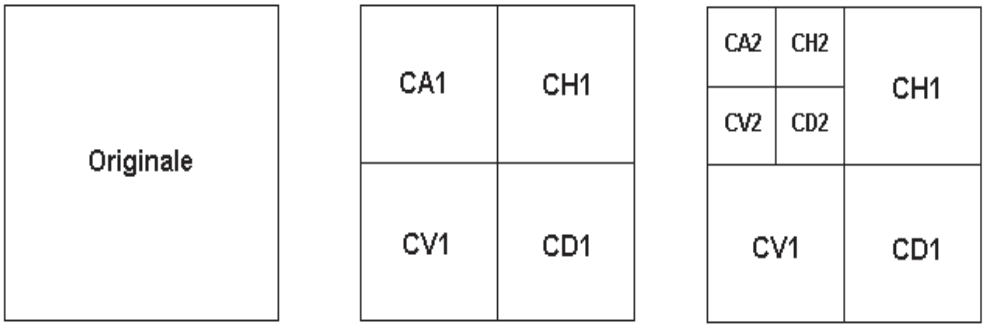
   Se l'immagine è a colori avviene la trasformazione delle componenti RGB ricavando tre schemi diversi per ciascuna di esse, si applicheranno le trasformazioni per ciascun'utile come se lavorassimo su livelli di grigio.

   Sono possibili diverse trasformazioni.

2. Si applica la *Discrete Wavelet Transform (DWT*) che è la principale differenza con il vecchio formato JPEG che applicava la *DCT*.

   Questa trasformata agisce dividendo i contenuti a bassa frequenza da quelli ad alta frequenza. L'immagine originale è suddivisa in quattro immagini alte e larghe ciascuna esattamente la metà dell'originale. Nel quadrante superiore sinistro, grazie all'uso di un filtro *passa-basso* sono salvate le frequenze più basse, mentre negli altri tre quadranti facendo uso di un filtro *passa-alto* si salvano le frequenze più alte.

   In seguito si opera in maniera ricorsiva applicando lo stesso procedimento al quadrante superiore sinistro e così via fino al limite minimo di un'immagine ridotta ad un solo pixel.



In questa maniera le informazioni contenute nell'originale sono state segmentate ricorsivamente per la compressione e la decompressione.

3. Adesso è possibile procedere con la quantizzazione dell'immagine e con la codifica entropica (compressione non distruttiva), operando per cicli si prendono in considerazione i coefficienti maggiori e gli si dimezza arrotondali ad interi, si procede fino a non aver raggiunto una lunghezza predefinita o fino a codificare tutta l'immagine. Questo procedere per stadi e per raffinazione permette la maggiore integrazione con il browser visualizzando man mano che procede il download un'immagine sempre più definita, un po' come nell'interlacciamento per le immagini GIF.

In conclusione cosa presenta JPEG 2000 rispetto a JPEG?

- Nuove funzionalità (ROI, resistenza agli errori,ordine di progressione)
- Possibilità d'algoritmi lossy o lossless.
- Compressione migliore a bassi bitrates.
- Migliore rispetto per immagini composte e grafica palettizzata.
- Enorme guadagno in qualità e non solo in spazio rispetto a JPEG

Al momento si stanno vedendo le varie licenze che saranno disponibili per questo formato che dovrà affrontare formati "free" quali PNG.

MPEG

Quando si parla di multimedialità non si parla solo d'immagini ma anche d'informazioni sonore. Il suono è un'onda e richiede una buona quantità di risorse spaziali per essere memorizzata in digitale, per fare un esempio un minuto di brano musicale della qualità di un cdrom occupa in media 10Mbyte non compresso.

Un gruppo di tecnici informatici ed esperti del suono ha cercato di realizzare uno standard aperto di codifica audio-video per risolvere questi problemi, la sigla MPEG indica appunto *Motion Pictures Expert Group (MPEG)*, in dettaglio parleremo dello standard più famoso realizzato dal gruppo ossia l'*MPEG 1 Layer 3* o più comunemente conosciuto come formato *MP3*.

Attualmente si hanno:

- **MPEG-1**: progettato per la codifica d'immagini in movimento e per l'audio ad esse associato, in forma digitale.

- **MPEG-2**: progettato per la codifica generica d'immagini in movimento e audio ad esse associato.
- **MPEG-3**: confluito in MPEG-2.
- **MPEG-4**:progettato per la codifica audiovisiva.

Il suono

L'onda sonora viene ad interagire con il nostro orecchio formando appunto il suono, quest'onda è di segnale continuo e deve essere campionata al fine di poter essere memorizzata come segnale digitale.
Esistono 3 parametri principali che influenzano lo spazio e la qualità sonora:

1. **Canali audio:** la riproduzione audio può utilizzare più sorgenti, nel caso di una ripartizione unica quindi di un canale unico si ha la modalità *Mono*, nel caso di due ripartizioni o di due canali si ha la modalità *Stereo*, con i moderni sistemi di riproduzione audio come gli impianti surround si possono avere altri canali ancora. Un segnale cresce secondo il numero di canali che possiede, un segnale Stereo occuperà il doppio di un segnale Mono.

2. **La risoluzione:** per ogni campione si può utilizzare una certa quantità di bit, normalmente si utilizzano 1 o 2 byte, avendo quindi una scala di valori pari a 256 o 65000 pari a quella di un CD audio.

3. **Frequenza:** la frequenza di campionamento di un CD audio è pari a 44 khz, il doppio della frequenza di campionamento di un nastro 22 khz e quattro volte la quantità minima richiesta per registrare a voce ossia 11 khz.

   Questo è forse il parametro più importante in quanto bisogna tenere conto del fatto che segnali diversi possono dare segnali campionati identici.

   Il *teorema di Shannon-Nyquist* spiega che per ricostruire un segnale continuo variabile nel tempo, si deve avere una frequenza di campionamento pari o superiore al doppio della massima frequenza presente nel segnale di partenza.

Il prodotto di questi tre parametri definisce la *bitrate* ossia la quantità di bit usati per la rappresentazione d'ogni secondo d'audio, questa quantità si misura in *bps*.

Formato Wave e codifica PCM

Il formato per la memorizzazione audio digitale "storico" è il formato Wave realizzato dalla Microsoft insieme all'IBM, questo formato si basa sulla tecnica PCM, acronimo di Pulse *Code Modulation*. Data una risoluzione di 16 bit si hanno quindi ampiezze di campionamento che variano da -32.768 a 32767. Questa codifica riproduce il segnale in maniera molto accurata ma occupando diverso spazio.

Campionamento e modello percettivo nell'MPEG Layer 3

Normalmente al fine di risparmiare spazio si utilizza rappresentare il segnale nel

dominio delle frequenze e poi campionarlo con pochi bit, questa trasformazione è effettuata scomponendo il segnale in 32 sottobande di dimensione uguale. Questa decisione è la più semplice possibile, anche se a volte criticata dato che è stato studiato che non rispetta fedelmente la percezione sonora dell'uomo.

La suddivisione corretta sarebbe in 26 parti di dimensione crescenti, dato che il nostro orecchio è più sensibile alle variazioni in basse frequenze che in alte:

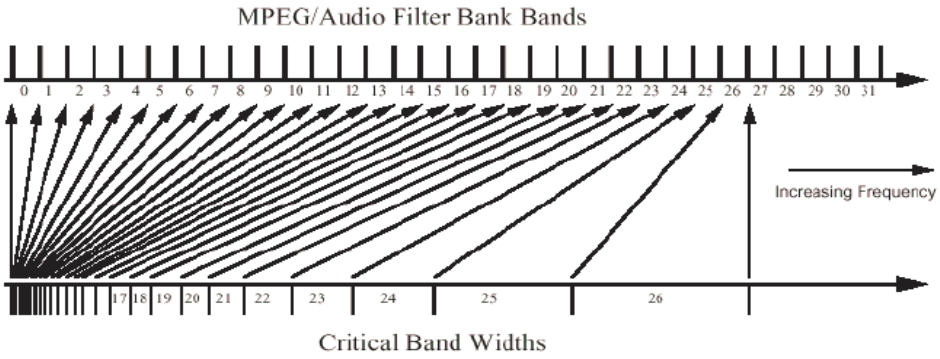


Fig. tratta da: [Pan 95]

La decisione di suddividere in bande uguali ha comunque il vantaggio di possedere un'implementazione più semplice e più veloce.

L'orecchio umano non riesce a percepire tutti i suoni e non tutti alla stessa maniera, sono eliminate tutte le frequenze troppo alte o troppo basse per essere percepite. Innanzi tutto le frequenze udibili dall'uomo vanno dai 16Hz ai 20Khz.

Un altro fenomeno importante è quello del *Mascheramento*, suoni dal tono più forte coprono suoni dall'intensità minore. A causa d'alcuni aspetti fisiologici della membrana presente nell'orecchio si presenta un premascheramento e un postmascheramento dipendenti dalla durata.

Tutti questi fenomeni permettono di scartare gran parte delle frequenze componenti il segnale sonoro, MPEG sfrutta appunto quest'applicazione pratica della *psicoacustica* utilizzando quindi un algoritmo *lossy*.

Lo standard MPEG-1

Innanzi tutto si hanno tre livelli di compressione di complessità crescente:

1. *Layer 1:* questo livello è stato studiato per fornire le migliori prestazioni con bitrate superiori ai 128 kbit/s per canale, e fornisce fattori di compressione di circa 1 a 4.

2. *Layer 2:* questo livello è adatto a bitrate intorno ai 128 kbit/s per canale, con una compressione massima di 1 a 8.

3. *Layer 3:* questo livello da' il nome al famoso MP3 e fornisce prestazioni ottimali con bitrate pari a circa 64 kbit/s per canale, riducendo il file sonoro fino a 12 volte l'originale.

La codifica presenta la possibilità di codificare suoni in diverse modalità:

- Un canale
- Due canali
- Stereo
- Stereo congiunto

Ora passiamo a studiare il processo di codifica del Layer 3, in questa immagine si può osservare la struttura del codificatore:

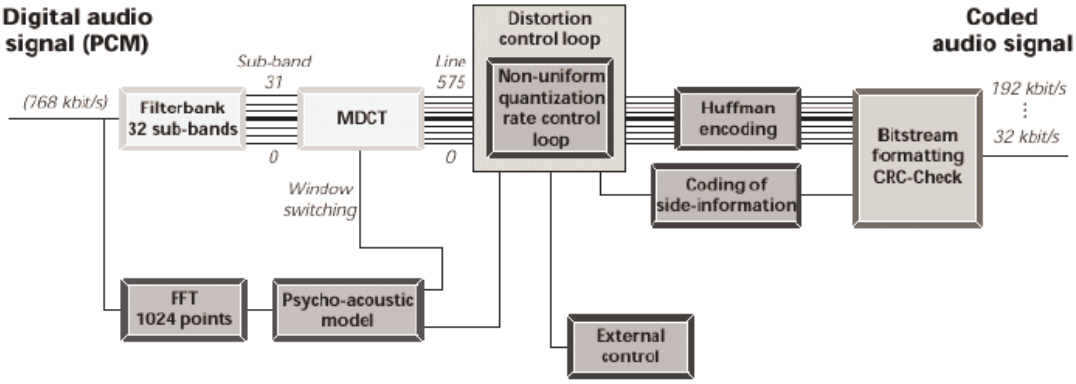


Fig. tratta da [Brandenburg 2000]

Il processo procede in questa maniera:

1. Si analizza il segnale nel campo delle frequenze questo viene scomposto in 32 sotto bande tramite un banco di filtri. Le sottobande del segnale PCM vengono passate ad un ulteriore filtro MDCT (Modified Discrete Cosine Transform) da 6 a 18 punti a seconda che si necessiti maggiore risoluzione frequenziale si desideri velocità nel caso di una previsione d'eco. Tutto questo sistema avviene tramite un sistema di finestre successive, aventi 1152 campioni.

2. Questa fase si occupa di eliminare tutte le frequenze impercettibili per l'uomo, effettuando il processo per il pre e post mascheramento, passando attraversi passaggi molto onerosi in termini computativi.

3. A questo punto si passa a quantizzare il segnale quest'operazione deve essere ben progettata per minimizzare l'inevitabile rumore introdotto.

4. Si procede alla codifica tramite codice di Huffman, se le dimensioni del blocco codificato sono troppo grandi, si ritorna indietro diminuendo il guadagno, questo processo viene definito *Rate Loop*.

5. Viene effettuato un controllo sul rumore, applicando un fattore di scala per ogni sottobanda per calcolare il rumore creatosi, a questo punto si ripete il *Rate Loop* finché non si è ridotto il disturbo fino ad un livello accettabile.

Nel processo di creazione di un dato MPEG viene definito un header di 4 byte contenenti informazioni sulla versione (1 o 2), sul layer (1, 2, 3) sulla modalità di canale.

Inoltre è possibile definire un campo massimo di 28 bytes per inserire informazioni aggiuntive come il titolo del brano, il suo autore, l'album e l'anno di pubblicazione.

Gli standard MPEG2/3 e MPEG4

Lo standard *MPEG2* confluito insieme allo standard *MPEG3* è stato creato espressamente per la trasmissione a distanza per la trasmissione a distanza di flussi video/audio televisivi. Vengono fornite quindi funzionalità come il trasferimento parallelo di più canali audio, e lo zapping tra vari flussi video.

L'innovazione maggiore nel campo audiovisivo è stata portata però dallo standard *MPEG3* che codifica i vari frames con oggetti audiovisuali dotati ciascuno di un proprio comportamento. In questa maniera si inseriscono diversi livelli d'immagine (sfondo, primo piano, secondo piano), per esempio in questa maniera se lo sfondo rimane uguale non verranno memorizzati fotogrammi aggiuntivi risparmiando ulteriore spazio.

Le applicazioni più conosciute in ambito “casalingo” di questi due standard sono nell'ambito dell'industria cinematografica, il primo utilizzato per i filmati memorizzati su supporto DVD e il secondo tramite il codec DivX che ha creato diversi problemi anche in ambito politico a causa del suo utilizzo nello sharing di filmati in rete.

Capitolo 5 Riferimenti

- Articolo originale di Huffman: D.A. Huffman, “A method for the construction of minumum-redundancy codes” http://compression.graphicon.ru/download/articles/huff/huffman_1952_minimum-redundancy-codes.pdf
- Codifica aritmetica: http://en.wikipedia.org/wiki/Arithmetic_coding
- Algoritmi Lossless: http://www.data-compression.com/lossless.shtm
- RLE Compression: http://www.prepressure.com/techno/compressionrle.htm
- H. Cormen, E. Leiserson, L Rivest “Introduzione agli algoritmi”
- Joint Photographic Expert Group Site: http://www.jpeg.org/
- Gif9a Specification: *http://www.w3.org/Graphics/GIF/spec-gif89a.txt*
- K. Brandenburg, H. Popp, “An Introduction to MPEG layer 3”, EBU TECHNICAL REVIEW – Giu. 2000
- D. Pan: “A tutorial on MPEG/Audio compression” IEEE Trans. on Multimedia, vol. 2, no. 2, 1995, pp. 60-74.
- Berkeley Multimedial Research Center Site: http://bmrc.berkeley.edu/frame/research/mpeg
- Pc Magazine n°162 - Agosto 1999
- Linux Journal: "MPEG-1 Movie Players" Maggio 2001
- Dictionary of Algorithms and Data Structures: http://www.nist.gov/dads/