Teoria dell'informazione e crittografia

Riassunto videolezioni

Contents

Lezione 2

Supponiamo di avere una sorgente S che emette un simbolo ogni colpo di clock. Questo simbolo subisce una codifica di sorgente (per compattare la rappresentazione dell'informazione) e successivamente, prima di immettere nel canale, bisogna effettuare una codifica di canale. Il canale è rumoroso per definizione, quindi bisogna aggiungere al simbolo delle cifre di controllo. Il messaggio quindi sarà composto da m bit di messaggio e k bit di controllo, il totale farà n.

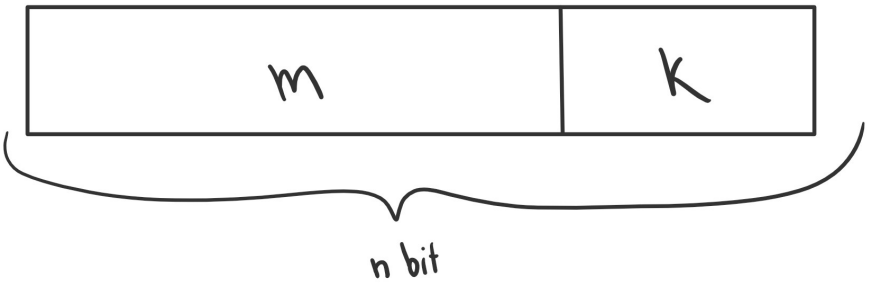


Figure 1: Struttura messaggio da inserire nel canale

Le cifre di controllo servono ad aumentare la ridondanza del messaggio, in questo modo per il ricevente sarà più facile capire se ci sono stati errori.
La **ridondanza** serve per capire meglio il messaggio (esempio, se togli una lettera dall'alfabeto italiano le parole si capiscono comunque, però con quella lettera è più veloce capire).

Controllo di parità singolo - Codifica di canale

Si aggiunge una cifra alla fine del messaggio. Ci sono due tipi di parità:

- Parità pari: il bit equivale a 1 se il numero di 1 nel messaggio è dispari (in questo modo si arriva ad avere un numero pari di 1). il bit equivale a 0 se il numero di 1 nel messaggio è pari (il numero è già pari, quindi non aggiungo altri 1)
- Parità dispari: stesso ragionamento ma al contrario.

Di base useremo la parità pari, hanno stesse proprietà (in certi casi si usa quella dispari, es. mando pacchetto di zeri su linea con 0 volt).

Per il calcolo del bit di parità si usa una funzione di parità:

$$y = \text{Parity}(x_1, x_2, ..., x_n) = x_1 \oplus x_2 \oplus ... \oplus x_n \tag{1}$$

In questo modo quando il ricevente legge il messaggio, controlla che il numero di 1 sia pari. Se non lo è, allora c'è stato un errore singolo.

La stessa funzione può essere vista come:

$$y = \text{Parity}(x_1, x_2, ..., x_n) = x_1 + x_2 + ... + x_n \bmod 2 \tag{2}$$

Questo è utile quando si vuole trattare 0 e 1 con i loro significati numerici. Isomorfismo fra queste due forme:

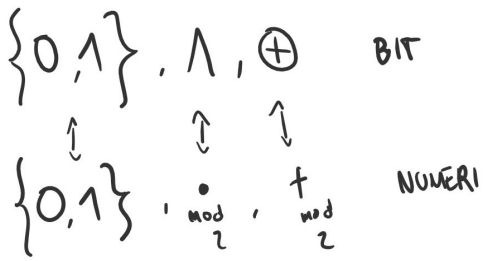


Figure 2: Isomorfismo bit/numeri

Per fare il calcolo della funzione Parity(x) di solito si basa l'hardware (visto che è un operazione molto diffusa, ogni hardware ha supporto per questo tipo di calcoli). In caso l'hardware supporti solo ad esempio lo xor fra due soli bit, posso scomporre in questo modo ed effettuarne uno ad uno:

$$x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 = (x_1 \oplus x_2) \oplus (x_3 \oplus x_4) \oplus (x_5 \oplus 0) \tag{3}$$

n.b.
$x \oplus 0 = x$
$x \oplus 1 = \neg x$

Si potrebbe anche utilizzare un automa a stati finiti per modellare il problema:

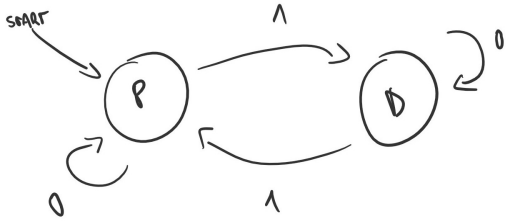


Figure 3: Automa a stati finiti per il calcolo pari/dispari

Ridondanza

Con ridondanza intendiamo, data la Figura 1, questo rapporto:

$$R = \frac{\text{numero di cifre totali}}{\text{numero di cifre di controllo}} = \frac{n}{m} \geq 1 \tag{4}$$

Con $k = 0$ la ridondanza vale 1, e praticamente stiamo inserendo il messaggio senza bit di parità.
In altri termini si dice che:

$$R = 1 + \text{eccesso di ridondanza} \tag{5}$$

$$R = 1 + \frac{k}{m}$$

Quando si definisce un metodo per il controllo degli errori, si cerca di avere un eccesso di ridondanza minore possibile (meno bit si usano nel canale, meno errori avverranno; ma in generale meglio ottimizzare).

Quanto vale la ridondanza nel caso del controlli di parità?

$$R_{\text{parity}} = \frac{n}{n-1} = \frac{(n-1)+1}{n-1} = 1 + \frac{1}{n-1} \tag{6}$$

Quindi l'eccesso di ridondanza è $\frac{1}{n-1}$.

Ovviamente, osservando come l'eccesso di ridondanza sia uguale a $\frac{k}{m}$, a parità di m, il numero di bit del messaggio, il modo di ridurre al minimo la ridondanza è quello di minimizzare il numeratore, quindi la k. In gergo si dice che ogni cifra di controllo dovrà *coprire* il maggior numero di bit di messaggio possibile, per avere una bassa ridondanza.

Canale - modelli di rumore

Il canale più semplice e facile da gestire è il **rumore bianco**, definito da queste proprietà:

- La probabilità che avvenga un errore in ciascuna posizione è uguale a p per ogni bit. Di conseguenza la probabilità che la trasmissione avvenga in modo corretto è $1 - p$.
- Il fatto che avvenga un errore in posizione i non influisce sulle altre probabilità, quindi si parla di eventi statisticamente indipendenti (condizione che rende poco realistico il rumore bianco, es. sbalzo di corrente).

Nell'intero pacchetto, qual è la probabilita di avere k errori, dato $0 \leq k \leq n$? Ragioniamo prima sul valore di p

- $p = 0$? No, impossibile che in un canale ci sia completa assenza di errore (qualunque supporto io utilizzi, col tempo si deteriora).
- $p = 1$? No, se no basta una porta not al ricevitore per avere tutti i bit corretti (come prima).

- $p > \frac{1}{2}$? Potrebbe aver senso, però se ho un canale che sbaglia più della metà delle volte, basta una porta not per ridurre questa quantità fino a una quantità minore di $\frac{1}{2}$
- $p = \frac{1}{2}$? Così ottengo un generatore di numeri casuali perfetto in quanto ogni bit viene trasformato in modo casuale in 0 o in 1 (cosa impossibile).

Possiamo concludere dicendo che p è sempre $0 < p < \frac{1}{2}$.

Tornando alla domanda di prima, quanto è la probabilità di avere k errori? Partiamo dalla probabilità di avere 1 errore, esso può avvenire in una posizione qualsiasi. Essendo eventi stocastici e statisticamente indipendenti, la probabilità di un errore è uguale a

$$p(1 \text{ errore}) = p(E_1 \vee E_2 \vee ... \vee E_n) \tag{7}$$

Con E_k s'intende l'evento che avvenga un errore in posizione k.
Quanto è la probabilità che avvenga un errore *solo* nella posizione 2?

$$\begin{aligned} p(\text{errore in pos 2}) &= (1-p)\ p\ (1-p)(1-p)\ ...\ (1-p) \\ &= p\ (1-p)(1-p)\ ...\ (1-p) \\ &= p\ (1-p)^{n-1} \end{aligned} \tag{8}$$

Ovviamente al posto della seconda posizione, questa regola vale per ogni posizione (proprietà commutativa della moltiplicazione).

Quindi la probabilità di avere un solo errore:

$$\begin{aligned} p(1 \text{ errore}) &= p(\text{errore in pos 1}) + p(\text{errore in pos 2}) + \ ...\ + p(\text{errore in pos n}) \\ &= np(1-p)^{n-1} \end{aligned} \tag{9}$$

E la probabilità di ottenere 2 errori?
Vediamo prima la probabilità di ottenere 2 errori in 2 posizioni fissate:

$$\begin{aligned} p(\text{errori in pos 1 e 2}) &= p \cdot p\ (1-p)(1-p)\ ...\ (1-p) \\ &= p^2\ (1-p)(1-p)\ ...\ (1-p) \\ &= p^2\ (1-p)^{n-2} \end{aligned} \tag{10}$$

Come posso creare le combinazioni di queste posizioni?

$$\binom{n}{2} = \frac{n(n-1)}{2} \tag{11}$$

Quindi questo è il numero di modi in cui posso disporre gli errori nel pacchetto

Da qui, la probabilità di ottenere 2 errori è:

$$p(2 \text{ errori}) = \binom{n}{2} p^2 (1-p)^{n-2} \tag{12}$$

Riprendendo la probabilità di avere un errore, essa si può riscrivere come:

$$p(1 \text{ errore}) = \binom{n}{1} p^1 (1-p)^{n-1} \tag{13}$$

Ora, generalizzando, si può rispondere alla domanda: qual è la probabilità che si verifichino k errori in un messaggio di n bit (errori disposti in qualsiasi posizione all'interno del pacchetto)?

$$p(\text{k errori}) = \binom{n}{k} p^k (1-p)^{n-k} \qquad \text{per } 0 \geq k \geq n \tag{14}$$

Vediamo ora come si comporta la funzione $p(\text{errori})$ dati un n e un p fissati.

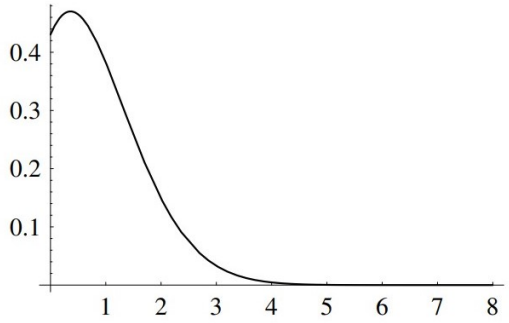


Figure 4: Grafico di $\binom{n}{k} p^k (1-p)^{n-k}$ con $n = 8$ e $p = 0.1$

Ma se all'interno del pacchetto avviene un numero pari di errori? Il controllo di parità fallisce in quanto il numero di 1 è pari (numero pari di 1 iniziale + numero pari di errori = numero pari di 1). Viceversa se il numero di errori è dispari, allora il controllo di parità se ne accorge.

Quanto è la probabilità che il controllo di parità fallisca? In altri termini: quanto è la probabilità che avvenga un numero pari di errori?

Procediamo per passi; recuperiamo la probabilità di commettere un errore dall'equazione (13):

$$\begin{aligned} p(1 \text{ errore}) &= \binom{n}{1} p^1 (1-p)^{n-1} \\ &= np(1-p)^{n-1} \end{aligned} \tag{15}$$

Ricordiamo un'approssimazione sfruttando le serie di Taylor:

$$(1+x)^\alpha \approx 1 + \alpha x \qquad \text{per } |x| < 1 \tag{16}$$

Possiamo considerare n come molto minore di $\frac{1}{p}$, quindi scrivere:

$$
\begin{aligned}
& np(1-p)^{n-1} \\
\approx\ & np[1-p(n-1)] \\
=\ & np[1-p(n-1)] \\
=\ & np[1-np+p] \\
=\ & np[1-np+p] \\
& np-n^2p^2+np^2
\end{aligned} \tag{17}
$$

Ora trascuriamo i termini con p^2 in quanto essendo minore di 1, dominerà p; quindi

$$
\begin{aligned}
p(1 \text{ errore}) &\approx np \\
&= \binom{n}{1} p
\end{aligned} \tag{18}
$$

Per due errori:

$$
\begin{aligned}
p(2 \text{ errori}) &\approx np\binom{n}{2} p^2 \\
&= \binom{n}{2} p^2
\end{aligned} \tag{19}
$$

Generalizzando per k errori:

$$
p(k \text{ errori}) \approx np\binom{n}{k} p^k \tag{20}
$$

Quindi, tornando al problema di prima, dobbiamo fare un'osservazione:

$$
\sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k} = 1 \tag{21}
$$

Se sommo tutte le probabilità, fare zero errori, fare un errore, fare due errori ecc. ovviamente ho come somma 1 (evento stocastico).

Sfrutto questa proprietà:

$$
(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k} \tag{22}
$$

... Lezione 02 parte 3 min 33 +- ...

Quindi la probabilità che il controllo di parità fallisca è

$$
p(\text{controllo parità fallisce}) = \frac{1+(1-2p)^n}{2} - (1-p)^n \tag{23}
$$

Burst di errori

Gli errori nella realtà avvengono in maniera non indipendente, di solito avvengono errori su sequenze di bit adiacenti. Definiamo L come lunghezza del **burst** di errori.

La parola `ciao` in ASCII è rappresentata in questo modo:

$$\texttt{C} = 1000011$$
$$\texttt{I} = 1001001$$
$$\texttt{A} = 1000001$$
$$\texttt{O} = 1001111$$

... fine lezione 02 parte 3

Lezione 3

Codici pesati

Dato un messaggio da spedire nel canale, si associa ad esso un insieme di valori. Ad esempio, dato un messaggio $m = m_1\ m_2\ ...\ m_n$, codifichiamo ogni m_i nel seguente modo:

$$\begin{gathered}\text{26 lettere A-Z}\\ \text{1 spazio } \textvisiblespace\\ \text{10 cifre 0-9}\\ \downarrow\\ \text{37 simboli}\end{gathered}$$

Successivamente si associa ad ogni messaggio un insieme di valori di *peso*:

$$\begin{gathered}\text{msg: } m_1,\ m_2,\ ...\ m_n,\ c\\ \text{pes: } n+1,\ n,...\ 2,\ 1\end{gathered}$$

Facciamo un esempio, mettiamo di voler spedire la parola `ciao`, utilizzando la codifica mostrata in precedenza.

	`C`	`I`	`A`	`O`	check
msg	2	8	0	14	check
pes	5	4	3	2	1

Il mittente pesa ogni carattere molitplicando la codifica per il peso:

$$2 \cdot 5 + 8 \cdot 4 + 0 \cdot 3 + 14 \cdot 2 + check \cdot 1$$

Succcessivamente si pone questa espressione uguale a $0\ mod\ 37$ (questo è il numero totale di caratteri).

$$2 \cdot 5 + 8 \cdot 4 + 0 \cdot 3 + 14 \cdot 2 + check \cdot 1 = 0\ mod\ 37$$

$$70 + check = 0\ mod\ 36$$

$$check = 4$$

Successivamente spedisce il messaggio con il vettore dei pesi.
Il ricevitore si calcola il peso (usando il valore di *check* spedito dal mittente) e controlla che il peso sia uguale a $0\ mod\ 37$.

Da notare che 37 è primo, quindi ci va bene perchè lavoriamo in un campo. Ad esempio i conti in modulo 6 hanno i divisori di zero ($3 \cdot 2 = 0 \bmod 6$). Se non è primo piuttosto aumentiamo il numero di caratteri (piuttosto non li usiamo, ma ci semplifica tutto avere un numero primo).

I codici pesati sono utili per individuare errori come ad esempio lo scambio di caratteri, l'inserimento di caratteri da parte del canale ecc...

Ad esempio, se all'interno di un messaggio al posto di `ab` viene inviato `ba`, allora la differenza fra le somme pesate è (fissato *k+1* come peso di `a`, k come peso di `b`).

$$a(k+1) + bk \to b(k+1) + ak$$
$$a(k+1) + bk - b(k+1) + ak =$$
$$= b(k+1) + ak - a(k+1) - bk$$
$$= bk + b + ak - ak - a - bk$$
$$= b - a$$

Quindi il messaggio viene accettato solo se la differenza è uguale a 0, quindi quando $b - a = 0 \ mod\ 37$, ma per essere 0 allora i due caratteri devono essere uguali. Se due simboli uguali vengono scambiati quindi, il messaggio viene accettato lo stesso.

Vediamo un algoritmo per calcolare i pesi:

```
1:  procedure CalcolaCheckDigit
2:      sum ← 0
3:      ssum ← 0
4:      while not EOF do
5:          read symbol
6:          sum ← sum + symbol mod 37
7:          ssum ← ssum + sum mod 37
8:      temp ← ssum + sum mod 37
9:      c ← 37 − temp mod 37
10:     return c
```

Una simulazione dell algoritmo con la parola `ciao` è la seguente:

msg	sum	ssum
`C` $= 2$	2	2
`I` $= 8$	10	12
`A` $= 0$	10	22
`O` $= 14$	24	$46 \equiv 9$

$$temp \leftarrow 9 + 24 \equiv 33$$
$$c \leftarrow 37 - 33 = 4 \equiv \texttt{E}$$

Quindi il messaggio spedito sarà `ciaoe`.

Il ricevente ricalcolerà la tabella usando il messaggio appena arrivato:

msg	sum	ssum
`C` $= 2$	2	2
`I` $= 8$	10	12
`A` $= 0$	10	22
`O` $= 14$	24	$46 \equiv 9$
`E` $= 4$	28	$37 \equiv \mathbf{0}$

La somma delle somme finale è 0, quindi il messaggio è ok. Se non fosse 0, allora il messaggio è stato modificato, ma non posso sapere dove.

Astraiamo il procedimento appena utilizzato:

msg	sum	ssum
`w`	w	2
`x`	$w + x$	$2w$+x
`y`	$w + x + y$	$3w + 2x + y$
`z`	$w + x + y + z$	$4w + 3x + 2y + z$
`c`	$w + x + y + z + c$	$5w + 4x + 3y + 2z + c \equiv 0 \ mod \ 37$

Si noti come la somma delle somme assegna un peso decrescente a tutti i caratteri: 5 a w, 4 a x e così via...

Il codice ISBN è un esempio di codice pesato su base 11; proviamo a verificare il seguente codice: 1-58488-508-4

msg	sum	ssum
`1`	1	1
`5`	6	7
`8`	$14 \equiv 3$	10
`4`	7	$17 \equiv 6$
`8`	$15 \equiv 4$	10
`8`	$12 \equiv 1$	$11 \equiv 0$
`5`	6	6
`0`	6	$12 \equiv 1$
`8`	$14 \equiv 3$	4
`4`	7	$11 \ mod \ 11 \equiv \mathbf{0}$

Quindi il codice è corretto. Nel caso l'ultima cifra fosse un 10 si inserisce una `x`.

Es. 2.4 - 1, pag.24

Caso: rumore bianco
$p = 0.001$ (probabilità errore in ogni posizione)
$n = 100$ (dimensione pacchetto)

Quanto è la probabilità che ci siano 0 errori?

$$\begin{aligned} p(\text{0 errori}) &= (1-p)^n \\ &= (1 - \frac{1}{1000})^{100} \\ &= e^{100 \cdot ln(1-\frac{1}{1000})} \\ &\approx e^{100 \cdot ln(-\frac{1}{1000})} \\ &= e^{-\frac{1}{10}} \end{aligned}$$

Es. 2.4 - 2, pag.25

Caso: rumore bianco
$p = 0.001$ (probabilità errore in ogni posizione)
$n = 100$ (dimensione pacchetto)

Quanto è la probabilità di non riuscire ad individuare un errore?

Per rispondere a questa domanda, possiamo rispondere a: quanto è la probabilità che il controllo di errore singolo sbagli?

Ricordando l'equazione (23):

$$\begin{aligned} p(\text{controllo parità fallisce}) &= \frac{1 + (1-2p)^n}{2} - (1-p)^n \\ &= \frac{1 + (1 - \frac{2}{1000})^{100}}{2} - (1 - \frac{1}{1000})^{100} \\ &\approx 0.0045 \end{aligned}$$

Codici a correzione d'errore

I codici a correzione d'errore, a differenza del controllo di parità, permette al ricevente di rilevare dove è avvenuto un errore ed eventualmente di correggerlo. Questo ovviamente richiede di aggiungere delle cifre di controllo ulteriori.

Codici rettangolari - 1 Errore

Il messaggio inserito nel canale viene rappresentato come un rettangolo (logicamente parlando, non fisicamente). Rappresentiamo i bit di messaggio in questo modo:

$$\begin{array}{cccccc} o & o & o & ... & o & x \\ o & o & o & ... & o & x \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ o & o & o & ... & o & x \\ x & x & x & ... & x & \end{array}$$

Nell'esempio il carattere o rappresenta un bit di messaggio, x rappresenta un carattere di controllo. Ogni cifra di controllo sulle righe codifica la riga corrispondente, quelle sulle colonne codificano la colonna.

In questo modo, se un bit di messaggio viene modificato, posso individuare subito l'errore in quanto verranno sbagliati i bit di controllo sulla relativa riga e colonna. Ad esempio, se un bit in posizione i, j viene alterato, allora i due controlli di parità sulla riga i e sulla colonna j falliranno, ma il ricevente intuisce subito quale bit ha subito modifiche.

Per capire quanto il metodo è efficiente, andiamo a calcolarne la ridondanza:

$$R = \frac{m \cdot n}{(m-1)(n-1)} \tag{24}$$
$$= 1 + \frac{1}{m-1} + \frac{1}{n-1} + \frac{1}{(m-1)(n-1)}$$

Quindi l'eccesso di ridondanza equivale a $\frac{1}{m-1} + \frac{1}{n-1} + \frac{1}{(m-1)(n-1)}$.
Quanto è l'eccesso di ridondanza più piccola possibile in questa classe di codici (codici rettangolari)? L'idea è quella di minimizzare la funzione

$$\frac{1}{m-1} + \frac{1}{n-1} + \frac{1}{(m-1)(n-1)}$$

Si dovrebbero quindi calcolare le derivate parziali rispetto a n e a m e vedere quando si annullano, ma è facilmente intuibile che questa quantità è la minore possibile quando $m = n$.

Per cui i più efficienti sono i codici **quadrati**, ovvero il sottoinsieme di codici rettangolari in cui il numero di righe è uguale al numero di colonne.

La loro ridondanza vale

$$R = \frac{n^2}{(n-1)^2} = 1 + \frac{2}{n-1} + \frac{1}{(n-1)^2}$$

I peggiori invece, sono i codici con una singola riga (i bit di controllo sono di più dei bit di messaggio!)

Lezione 4

Seguendo l'esempio dei codici rettangolari, proviamo a disporre ora i bit di messaggio in altre forme logiche, in modo da coprire il più gran numero di bit di messaggio con ogni bit di controllo.

Codici triangolari - 1 Errore

Proviamo a creare una forma logica di un codice triangolare:

$$\begin{array}{ccccc} o & o & o & o & x \\ o & o & o & x & \\ o & o & x & & \\ o & x & & & \\ x & & & & \end{array}$$

Ovviamente il numero di bit di messaggio deve consentire una costruzione di questo genere (dagli antichi greci: numeri triangolari).

Come prima, ogni bit di controllo copre la relativa colonna e riga.

Quanto vale la ridondanza di questo tipo di codici?

Il numero di bit totali è

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \tag{25}$$

mentre il numero di bit di messaggio è

$$\sum_{i=1}^{n-1} i = \frac{(n-1)(n-1+1)}{2} = \frac{(n-1)n}{2} \tag{26}$$

Per cui passiamo a calcolare la ridondanza:

$$R_{triangolare} = \frac{\frac{n(n+1)}{2}}{\frac{(n-1)n}{2}} \tag{27}$$

$$= \frac{\cancel{n}(n+1)}{\cancel{2}} \cdot \frac{\cancel{2}}{(n-1)\cancel{n}}$$

$$= \frac{(n+1)}{(n-1)}$$

$$= \frac{(n-1)+2}{(n-1)}$$

$$= 1 + \frac{2}{(n-1)}$$

Da qui, l'eccesso di ridondanza vale

$$\frac{2}{(n-1)} \tag{28}$$

Confrontiamolo con l'eccesso di ridondanza dei codici quadrati:

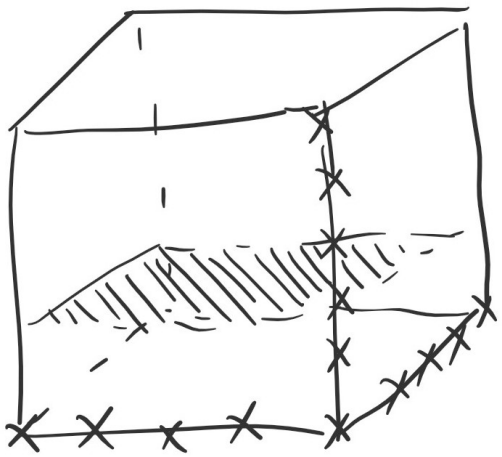
$$ecc_{quadrato} = \frac{2}{n-1} + \frac{1}{(n-1)^2}$$

$$ecc_{triangolo} = \frac{2}{n-1}$$

Si nota ovviamente come i codici triangolari siano sicuramente più efficienti.

Codici cubici - 1 Errore

Un'altra disposizione che si può seguire per ordinare i bit di messaggio è a forma cubica.



Facendo dei conti *a spanne*, ci sono n^3 posizioni, le cifre di controllo sono $3(n-1)+1 = 3n-2$.

La ridondanza più o meno quindi vale

$$R = \frac{(n-1)^3 + 3n - 2}{(n-1)^3} \approx 1 + \frac{3}{n^2} \tag{29}$$

Seguendo questo ragionamento, potrei pensare di aumentare le dimensioni

Codici ipercubici

Seguendo il modello dei *conti a spanne*, diciamo più o meno che le cifre di messaggi è $(n-1)^4$, mentre le check digit saranno $4(n-1)+1 = 4n-3$.

La ridondanza quindi varrà:

$$R = 1 + \frac{4n-3}{(n-1)^4} \approx 1 + \frac{4}{n^3} \tag{30}$$

Da qui si nota come si può passare a ragionare in k dimensioni, e si possono tirare delle conclusioni:

$$\begin{gathered}(n-1)^k \quad \text{bit di messaggio} \\ k(n-1)+1 = kn-k+1 \quad \text{bit di controllo} \\ R = 1 + \frac{kn-k+1}{(n-1)^k} \approx 1 + \frac{k}{n^{k-1}} \quad \text{ridondanza}\end{gathered} \tag{31}$$

Da qui si nota come la ridondanza decresce in maniera polinomiale, possiamo fare di meglio?

Codici di Hamming

Codici ottimali per individuare e correggere 1 errore (al massimo) in un canale caratterizzato da rumore bianco. Cosa significa codici ottimali? Il rapporto fra cifre di controllo e cifre di messaggio è esponenziale, meglio di così non si può fare.

Da ora in poi ci riferiremo con k alle cifre di messaggio, m alle cifre di controllo. n sarà la loro somma.

Le combinazioni (configurazioni) di cifre di controllo sono 2^m. Da notare come deve sempre valere $2^m \geq n+1$, questo è necessario in quanto ci dev'essere una configurazione per ogni posizione di errore, più uno (configurazione per rappresentare zero errori). Altrimenti non abbiamo abbastanza configurazioni per rappresentare tutte le situazioni (errore in posizione 3, errore in posizione 8, ecc).

Supponiamo di voler mandare un messaggio di 7 bit, ci chiediamo quanti di questi bit possono essere di messaggio e quanti di controllo?

$$\begin{gathered}2^m \geq 8 \to m \geq 3 \to m = 3 \\ k = 7 - 3 = 4\end{gathered}$$

Oppure posso chiedermi: devo inviare 4 cifre di messaggiosul canale, di quante cifre di controllo ho bisogno?

$$2^m \geq n+1 \rightarrow 2^m \geq m+k+1 \rightarrow 2^m \geq m+5$$

Quando è vero $2^m \geq m+5$? Proviamo ad assegnare dei valori a m:

m	2^m	$m+5$
1	2	6
2	4	7
3	8	8
4	16	9

Il valore più piccolo che ci va bene è proprio 3, quindi l'ideale è utiliazzare 3 cifre di controllo.

Ciascuna cifra di controllo fa il controllo di parità su diverse cifre di messaggio. Facciamo un esempio:
Ho 9 cifre di messaggio, avrò quindi 3 cifre di controllo, come si suddividono le cifre per il calcolo delle parità?

$$\begin{aligned} c_1 &= x_1 \oplus x_2 \oplus x_5 \oplus x_7 \\ c_2 &= x_5 \oplus x_7 \oplus x_8 \oplus x_9 \\ c_3 &= x_1 \oplus x_2 \oplus x_8 \oplus x_9 \end{aligned} \tag{32}$$

Ma quanto vale la somma di c_1 e c_2 (ricordando che $x \oplus 0 = x$)?

$$c_1 \oplus c_2 = x_1 \oplus x_2 \oplus x_8 \oplus x_9$$

che è esattamente uguale a c_3!

Allo stesso modo considerando vettori binari di n elementi, una somma in modulo due e un prodotto modulo due, posso effettuare il calcolo in questo modo:

$$\begin{aligned} v_1 &= (1,1,0,0,1,0,1,0,0) \\ v_2 &= (0,0,0,0,1,0,1,1,1) \\ &\downarrow \\ v_1 + v_2 &= (1,1,0,0,0,0,0,1,1) \end{aligned} \tag{33}$$

Questo insieme, composto da $(\{0,1\}^n, +_2, \cdot)$ è uno spazio vettoriale, quindi le tre equazioni di parità considerate sono linearmente dipendenti, e questo ovviamente è da evitare visto che la terza equazione non da informazione in quanto è ricavabile dalle altre due.

Ripendendo l'esempio del pacchetto di 7 bit, scriviamo questa tabella che rappresenta le posizioni:

pos	pos in binario
1	001
2	010
3	011
4	100
5	101
6	110
7	111
0 (nessun errore)	000

Da notare come i bit necessari per coprire tutte le posizioni sono 3, esattamente come il numero di bit di controllo. L'insieme delle posizioni in valore binario assume il nome di *sindrome*. Se la sindrome vale 101 allora è avvenuto un errore in posizione 5. Non ci resta che capire come assegnare le posizioni per ogni bit.

Definiamo le cifre di controllo come le colonne della tabella precedente, quindi:

1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
	c_3	c_2	c_1

Supponiamo che sia avvenuto un errore in posizione 5, quindi la terza e la prima cifra di controllo devono 'attivarsi', quindi valere 1, mentre la seconda deve rimanere a 0 (in questo modo il valore è 101 che rappresenta la posizione 5).
Quindi il bit numero 5 deve far parte delle equazioni che calcolano c_3 e c_1, e non deve far parte dell'equazione che calcola c_2. Per calcolare c_1 quindi, mi basta vedere su che posizioni ci sono degli 1, nel nostro esempio calcoleremo:

- c_1 sfruttando le posizioni 1, 3, 5 e 7
- c_2 sfruttando le posizioni 2, 3, 6 e 7
- c_3 sfruttando le posizioni 4, 5, 6 e 7

Quindi sfruttando queste c, se avviene un errore in posizione 5, la tripla varrà 101, visto che la posizione 5 è inclusa in c_1 e in c_3.

Le equazioni così formate saranno linearmente indipendenti (dimostrazione balza).

In generale le cifre di controllo vengono posizionate seguendo il primo numero che codificano, quindi:

- c_1 in posizione 1
- c_2 in posizione 2
- c_3 in posizione 4

Quest'ordine, al crescere delle cifre di messaggio, crescerà seguendo l'ordine di 2^n (quindi posizione 1, 2, 4, 8, 16...).
Da notare come la distanza fra una cifra di controllo e l'altra cresce in maniera esponenziale, questo a confermare il fatto che i codici di hamming sono ottimali.

Vediamo un esempio:

- Messaggio di 4 bit: `0110`
- $k = 4$

Sfruttando la disequazione $2^m \geq \underset{n}{m} + k + 1$, otteniamo che $m = 3$, quindi il numero totale di bit sarà 4+3.
Ora disponiamo le cifre di controllo, ricordando che la loro posizione vale 2^i con i che parte a 0 a m.

$$\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ c_1 & c_2 & & c_3 & & & \end{array}$$

Quindi le 4 cifre di messaggio verranno distribuite lungo le posizioni rimanenti:

$$\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ c_1 & c_2 & m_1 & c_3 & m_2 & m_3 & m_4 \end{array}$$

A questo punto si passa a calcolare il valore delle varie c_i in questo modo:

$$c_1 \oplus m_1 \oplus m_2 \oplus m_4 = 0$$
$$c_2 \oplus m_1 \oplus m_3 \oplus m_4 = 0$$
$$c_3 \oplus m_2 \oplus m_3 \oplus m_4 = 0$$

In ogni riga il c_i viene calcolato in modo da avere un numero pari di 1.
Il messaggio per ora è uguale a:

$$c_1\ c_2\ 0\ c_3\ 1\ 1\ 0$$

Quindi passo al calcolo di c_1:

$$\text{parity}(c_1\ 0\ 1\ 0) = 0$$

Per avere parità pari c_1 deve valere 1, quindi il messaggio diventa:

$$1\ c_2\ 0\ c_3\ 1\ 1\ 0$$

E così via per le altre due cifre di controllo:

$$\text{parity}(c_2\ 0\ 1\ 0) = 0$$
$$\downarrow$$
$$c_2 = 1$$

$$\text{parity}(c_3\ 1\ 1\ 0) = 0$$
$$\downarrow$$
$$c_3 = 0$$

Il messaggio che verrà inviato nel canale sarà quindi:

$$1100110$$

Quando il mittente riceve il messaggio, dovrà controllare se sono avvenuti degli errori. Il messaggio, abbiamo detto, è così composto:

$$c_1\ c_2\ 0\ c_3\ 1\ 1\ 0$$

Il mittente calcolerà il valore della sindrome controllando le parità delle varie posizioni, un trucco per vedere visivamente in che modo vengono effettuati i calcoli è il seguente:

Quindi partendo dalla fine, il primo bit della sindrome viene calcolando facendo la parità sull'ultima cifra, terz'ultima e così via scalando sempre di 1, il secondo bit invece ne prende due alla volta e scala di due, il terzo ne prende quattro e scala di quattro. Seguendo sempre quindi lo schema di 2^n.

Se capitasse un errore? Mettiamo che avvenga un errore in posizione 5, il messaggio passa da:

$$1\ 1\ 0\ 0\ 1\ 1\ 0$$

a

$$1\ 1\ 0\ 0\ \mathbf{0}\ 1\ 0$$

La cosa che farà il mittente sarà quindi quello di calcolare la sindrome.

$$s_1 = \text{parity}(1000) = 1$$

$$s_2 = \text{parity}(1010) = 0$$

$$s_3 = \text{parity}(0010) = 1$$

Il valore finale sarà quindi $s_1\ s_2\ s_3$ uguale a 101, quindi il mittente riconosce che è avvenuto un errore in posizione 5.
E se la cifra modificata fosse di controllo? Proviamo con la seconda cifra:

$$1\ \mathbf{0}\ 0\ 0\ 1\ 1\ 0$$

Calcolo della sindrome:

$$s_1 = \text{parity}(1010) = 0$$

$$s_2 = \text{parity}(0010) = 1$$

$$s_3 = \text{parity}(0110) = 0$$

Quindi questo tipo di codici riconosce errori anche sulle cifre di controllo! Ma nel caso di due errori nel messaggio? Proviamo a modificare due cifre del pacchetto:

$$1\ \mathbf{0}\ 0\ 0\ \mathbf{0}\ 1\ 0$$

Calcolo della sindrome:

$$s_1 = \text{parity}(1000) = 1$$

$$s_2 = \text{parity}(0010) = 1$$

$$s_3 = \text{parity}(0010) = 1$$

Essa dice che è avvenuto un errore in posizione 7, cosa falsa. Si noti quindi come il codice di Hamming copre un singolo errore, nel caso di errori multipli essi non sono riconoscibili.

Come si gestisce il caso di 4 cifre di controllo?

$$m = 4$$
$$2^m \geq n + 1$$
$$16 \geq n + 1$$
$$n \leq 15$$
$$n = 15 \text{ (ottimale)}$$
$$k = n - m = 15 - 4 = 11$$

Vediamo come si distribuiscono le cifre di controllo nel messaggio:

c_1	c_2	m_1	c_3	m_2	m_3	m_4	c_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Come si calcolano i bit di controllo? Come prima:

$$c_1 \oplus m_1 \oplus m_2 \oplus m_4 \oplus m_5 \oplus m_7 \oplus m_9 \oplus m_{11} = 0$$
$$c_2 \oplus m_1 \oplus m_3 \oplus m_4 \oplus m_6 \oplus m_7 \oplus m_{10} \oplus m_{11} = 0$$
$$c_3 \oplus m_2 \oplus m_3 \oplus m_4 \oplus m_8 \oplus m_9 \oplus m_{10} \oplus m_{11} = 0$$
$$c_4 \oplus m_5 \oplus m_6 \oplus m_7 \oplus m_8 \oplus m_9 \oplus m_{10} \oplus m_{11} = 0$$

Codici di Hamming ottimali

Come si può dedurre da quello spiegato sopra, i codici di Hamming definiti "ottimali" sono quelli che utilizzano tutte le configurazioni della sindrome sono utilizzate. Questo equivale a dire che:

$$2^m = n + 1 \tag{34}$$

Isolando la n, si ottiene che $n = 2^m - 1$. Questo per mostrare che i codici ottimali hanno sempre un numero di bit totali uguale a una potenza di 2 meno uno (3, 7, 15, 31, 63, 127...). Quando vale la ridondanda dei codici di Hamming ottimali?

$$R = \frac{n}{k} \tag{35}$$
$$R = \frac{k + m}{k}$$
$$R = 1 + \frac{m}{k}$$

Qual è la relazione fra k e m? Per i codici quadrati, cubici, ecc abbiamo notato un rapporto polinomiale.

$$2^m = m + k + 1$$
$$2^m - m - 1 = k$$

A sinistra prevale 2^m, quindi:

$$2^m \approx k$$

Per cui possiamo riscrivere la ridondanza in questo modo:

$$R = 1 + \frac{m}{2^m}$$

oppure

$$R = 1 + \frac{log_2 k}{k}$$

Che decresce esponenzialmente.

I casi limite di questo tipo di codici sono $m = 2$ (2 cifre di controllo e 1 di messaggio) e $m = 1$ (1 cifra di controllo), cosa che deriva dal fatto che le funzioni esponenziali crescono inizialmente in modo lento, e poi esplodono. Più sono lunghi i pacchetti, più le cifre di controllo sono "dilatate" nel messaggio (all'inizio adiacenti $[2^0 - 1]$, poi staccate di 1 $[2^1 - 1]$, poi di 3 $[2^2 - 1]$, poi di 7 $[2^3 - 1]$ e così via).

Caso particolare - Codici di Hamming a 2 cifre

Nonostante possano sembrare molto ridondanti, i codici di Hamming a due cifre di controllo possono avere un'applicazione efficace. In questo caso la banda del canale viene utilizzata solo a $\frac{1}{3}$ della sua disponibilità in quanto per mandare un bit ne invio un totale di 3, in particolare invio 000 per inviare 0, 111 per inviare 1.

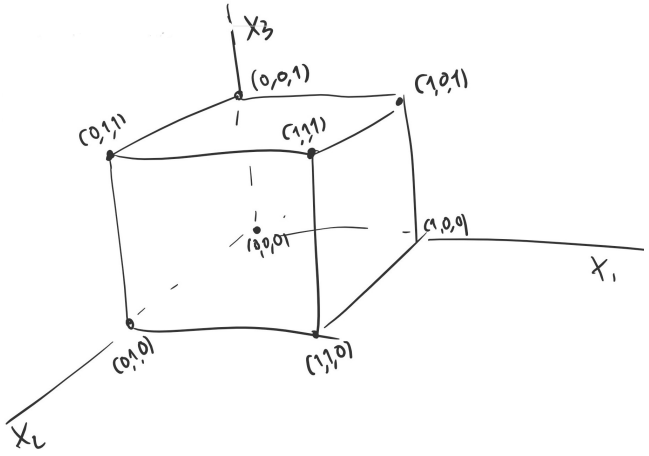
Su uno Space Shuttle, le possibilità di errore sono tante (raggi solari, temperature ecc), tutti i componenti sono triplicati. Se un pacchetto arriva con 0 e con 1 (esempio, arriva 001) allora si ragiona in probabilità, come si è visto in precedenza la probabilità che avvengano due errori (nel modello di rumore bianco), è moolto più bassa di quella che avvenga un errore solo. Quindi per decodificare si fa osserva quale bit è più "usato", ad esempio se arriva 101 allora viene decodificato come 1, 001 come 0, 100 come 0 e così via.

Lezione 5

Intepretazione geometrica

Cubi

Abbiamo detto che l'insieme $< \{0,1\}^n, +_2, \cdot >$ è uno spazio vettoriale. Supponiamo di avere $n = 3$, quindi dal canale entrano ed escono vettori di 3 bit. Diciamo che ognuno di questi bit rappresenta un lato di un cubo in tre dimensioni, successivamente "appoggiamo" questo cubo all'interno di uno spazio come $\mathbb{R}^3$.



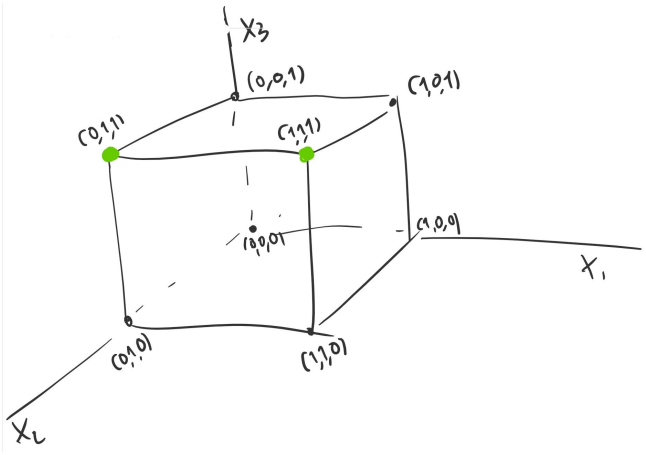
L'origine corrisponderà al messaggio (0, 0, 0), ogni asse rappresenterà quindi un bit.

L'idea è ovviamente utilizzabile in n dimensione, usando quindi dei cubi di n dimensioni.

I vertici del cubo rappresentano quindi tutte le possibile sequenze di messaggi. Un sottoinsieme di questi vettori, o vertici, rappresenta i messaggi "validi", quindi senza errore. Ma cosa vuol dire dal punto di vista geometrico che un messaggio non è valido? Supponiamo di mandare un messaggio 010, e che il rumore lo trasformi in 011. Un errore significa che ci siamo "mossi" da un vertice ad un altro del cubo. Un errore alla prima posizione ci fa muovere lungo l'asse x_1, un errore alla seconda posizione lungo l'asse x_2 e così via.

Il mittente e il destinatario devono quindi mettersi d'accordo su quali vertici siano "accettabili".

Mettiamo che i vertici accettabili siano 011 e 111



Ora mettiamo che nel canale il messaggio da 011 si sposti lungo la direzione x_1, quindi il messaggio diventa 111. Esso è accettabile però!

Definiamo la distanza di Hamming: dati due vettori u e v come il numero che bit che bisogna cambiare in u per ottenere v (e viceversa).

Ad esempio la distanza di Hamming fra 011 e 111 è 1.

Possiamo definire questa distanza anche usando la somma bit a bit. $(0,1,1) \oplus (1,1,1) = (1,0,0)$ La distanza di Hamming è uguale a:

$$d_H(u) = \sum_{i=1}^{n} u_i \tag{36}$$

Che rappresenta il numero di 1 all'interno dello xor bit a bit fra i due vettori.

Tornando al nostro cubo, diciamo che due vertici adiacenti non possono essere entrambi validi visto che, a causa di un errore, ci si sposta da uno all'altro; il mittente non rileverà l'errore!

Se nel mio codice la distanza di Hamming fra codici valdi è uguale a 1, non riuscirò mai a riconoscere errori.

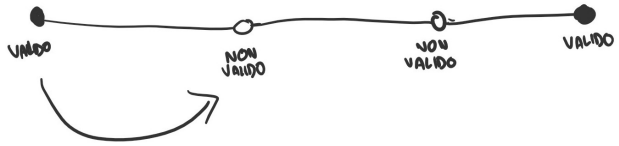
Il numero di "passi" che devo fare da un vertice all'altro del cubo è uguale alla distanza di Hamming che c'è fra i due valori rappresentati dai vertici.

Se selezionassi le parole valide usando distanza di Hamming = 2? Posso vedere la distanza fra i vertici in questo modo:

Nel caso di errore singolo, il messaggio inviato si "sposterebbe" in un vertice adiacente (quindi con distanza di Hamming = 1) che però non verrà accettato dal mittente (visto che i validi sono a distanza 2).

E per quanto riguarda distanza di Hamming = 3?

Nel caso di errore singolo, ci si sposterebbe in questo modo:



A questo punto il mittente si chiede: è più probabile che il messaggio arrivi dal vertice a sinistra (con un errore) o dal vertice di destra (con due errori)? Ovviamente è più probabile il vertice sinistro, quindi con questa distanza posso rilevare l'errore e anche correggerlo. Nel caso di due errori, però, il ricevente sbaglierebbe a correggere l'errore; seguendo l'esempio precedente andrebbe al vertice destro.

Vediamo quindi anche il caso di distanza di Hamming = 4:

É facile notare come si può correggere facilmente un errore, ma in caso di errore doppio ci si troverà ad un punto equidistante dai due estremi (ovviamente nel caso di tre errori se ne introdurrà un quarto correggendo in maniera sbagliata).

E così via, riassumiamo il tutto con questa tabella:

d_H	errori rilevati	errori corretti
1	0	0
2	1	0
3	1	1
4	2	1
5	2	2
$\vdots$	$\vdots$	$\vdots$

Sfere

Proviamo a cambiare punto di vista, riprendiamo la rappresentazione su segmenti:

Immaginiamo che il messaggio valido inserito nel canale sia il centro di una sfera. Se il messaggio che esce fa parte di un unica sfera, lo si può correggere "riportandolo" al centro della sfera. Se invece il messaggio fa parte di più sfere, rilevo l'errore ma non so a quale centro appartenga.

Ma come definisco una sfera nello spazio vettoriale? Dico che la sfera S è

$$S(x,r) = \{y \in \{0,1\}^n | d_H(x,y) \leq r\} \tag{37}$$

Possiamo vedere l'effetto degli errori come una somma xor:

$$(1,1,0) \oplus (0,1,0) = (1,0,0)$$

In questo caso è avvenuto un posizione 2, quindi è come sommare un vettore con un 1 in posizione 2.

Se capissi qual è il vettore errore mi basterebbe ri-sommarla, per ottenere il messaggio di partenza:

$$(1,1,0) \oplus (0,1,0) = (1,0,0)$$
$$(1,0,0) \oplus (0,1,0) = (1,1,0)$$

Riconsideriamo i codici di Hamming con distanza 3, quindi a rilevamento e correzione d'errore singolo. Ci chiediamo quanto vale il volume della sfera centrata nel vettore che inviamo di raggio 1.

$$V(S(x,1)) = 1 + n \tag{38}$$

In generale:

$$V(S(x,r)) = 1 + n + \binom{n}{2} + \binom{n}{3} + \dots \binom{n}{r}$$
$$V(S(x,r)) = \sum_{i=0}^{r} \binom{n}{i} \tag{39}$$

Quanti messaggi validi posso creare in questo modo? Data distanza di Hamming $d_H = 3$, messaggi di n bit, quindi uno spazio grande 2^n e n sfere grandi $(n+1)$, dico che:

$$\frac{\text{volume spazio}}{\text{volume sfera}} \geq \text{numero massimo di sfere / messaggi} \tag{40}$$

Applicato al nostro esempio:

$$\frac{2^n}{n+1} \geq 2^k \qquad n = k + m$$
$$2^n \geq 2^k(n+1)$$
$$\cancel{2^k} 2^m \geq \cancel{2^k}(n+1)$$
$$2^m \geq n+1$$

Che è la stessa conclusione a cui è arrivato Hamming per i codici ottimali!

Codifica di sorgente

L'obiettivo della codifica di sorgente è quello di creare delle codeword ottimali, ad esempio codificando con codeword più piccole i caratteri più utilizzati (codice morse).

Ci sono diversi codici per la codifica di sorgente, distinguibili in due:

- Codici a blocchi: tutte le codeword hanno la stessa lunghezza ($\ell_1 = \ell_2 = ... = \ell_q$)
  - Vantaggi: il ricevente ha 'vita facile' per decodificare in quanto sa subito quando inizia e quando finisce la codeword (sa già la lunghezza)
  - Svantaggi: se un simbolo esce con probabilità alta, allora abbiamo un grande spreco (uso gli stessi bit che userei con un simbolo usato raramente)

  La lunghezza media delle codeword ($\sum_{i=1}^{q} p_i = 1$) è ovviamente:

$$L = \sum_{i=1}^{q} + p_i \ell_i = \sum_{i=1}^{q} p_i \ell = \ell \sum_{i=1}^{q} p_i = \ell \tag{41}$$

  uguale per tutti. Questo tipo di codice conviene quando la distribuzione delle probabilità per i simboli di uscire è uniforme.

- Lunghezza variabile: le codeword hanno lunghezze diverse, di solito i caratteri che escono con probabilità p_i maggiori sono codificati in codeword più brevi.
  - Vantaggi: C'è meno spreco di spazio, le codeword si adattano alle probabilità dei simboli
  - Svantaggi: Per il ricevente non è semplice capire quando una codeword finisce.

  A causa di questo svantaggio si introducono due sottoclassi dei codici a lunghezza variabile:

- Univocamente decodificabili: quando il ricevente riceve una sequenza di codeword (es $w_1, w_2 \dots w_k$) di lunghezza diversa, allora esiste un unico modo di spezzare la sequenza di codeword
- Non univocamente decodificabili: al contrario, non esiste un unico modo di spezzare la sequenza ricevuta (ad esempio posso spezzare una sequenza sia in $s_1 s_1 s_2$ che in $s_3 s_2$).

Nel corso useremo **solo** codici univocamente decodificabili.

Lezione 6

Cominciamo con un esempio: supponiamo di avere una sorgente caratterizzata da quattro simboli, codificati in questo modo:

$$s_1 \to 0$$

$$s_2 \to 01$$

$$s_3 \to 11$$

$$s_4 \to 00$$

A questo punto supponiamo che il ricevente riceva la stringa 0011, qual è la sequenza di codeword che ho ricevuto? Una soluzione potrebbe essere s_1, s_1, s_3, ma anche s_4, s_3. Questa codifica e questa sequenza mette in crisi il ricevitore, ed è uno dei casi da evitare (non univocamente decodificabile).

Tra i codici **univocamente decodificabili** possiamo distinguere due ulteriori sottocasi:

- Istantanei: il ricevente appena finito di ricevere una codeword la decodifica senza ambiguità.
- Non istantantei: non vale la regola precedente.

Vediamo un esempio di codice univocamente decodificabile non istantaneo:

$$s_1 \to 0$$

$$s_2 \to 01$$

$$s_3 \to 011$$

$$s_4 \to 111$$

Immaginiamo di dover decodificare la sequenza 011111111, all'inizio potrei star leggendo s_1, s_2, s_3, dipende da cosa arriva dopo. Dopo arriva 1, e anche qui c'è ambiguità fra s_2 e s_3, e così via.

Partendo dal fondo però, quando la trasmissione è finita, si può decodificare il messaggio partendo dalla fine:

$$011 \;\; 111 \;\; 111$$

$$s_3 \;\; s_4 \;\; s_4$$

Il lato negativo è che devo attendere la fine della trasmissione per decodificare il messaggio. I codici istantanei invece, non sono caratterizzati da questa pecca. Un codice istantaneo implica che **nessuna codeword è prefissa di un'altra**. Nel codice di prima, s_1 è prefisso di s_2, ecc.

Vediamo questo esempio:

$$s_1 \to 0$$
$$s_2 \to 10$$
$$s_3 \to 110$$
$$s_4 \to 111$$

E decodifichiamo 1011001110 $(s_2, s_3, s_1, s_4, s_1)$ si noti come partendo dall'inizio si può riconoscere immediatamente le codeword, senza attendere il ricevimento della sequenza completa.

Albero di decodifica

Il lavoro del ricevente per codificare si può rappresentare con un albero di decodifica. All'inizio egli si trova alla radice dell'albero, poi si sposta all'interno di esso seguendo i simboli sulle freccie:

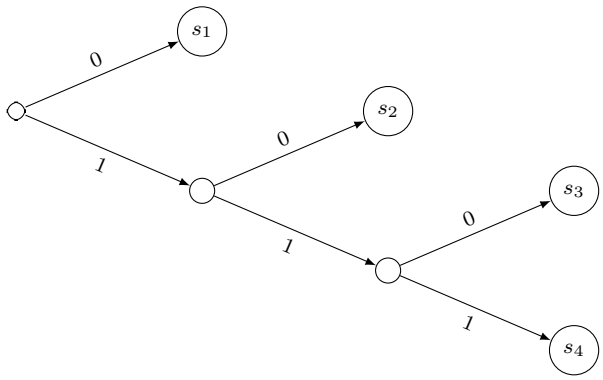


Figure 5: Albero di decodifica per $s_1 \to 0$, $s_2 \to 10$, $s_3 \to 110$, $s_4 \to 111$

Questa rappresentazione è molto utile in quanto si riconosce immediatamente l'istantaneità dei codici istantanei, inoltre è possibile passare dall'albero di decodifica al codice e viceversa.

Un albero è definito vuoto oppure un nodo che punta ad altri alberi (ovviamente in termini ricorsivi).

Un'altra cosa da notare è che gli alberi di decodifica sono utilizzabili solo per codici istantanei, cosa succede se non lo sono? Proviamo con:

$$s_1 \to 0$$
$$s_2 \to 01$$
$$s_3 \to 011$$
$$s_4 \to 111$$

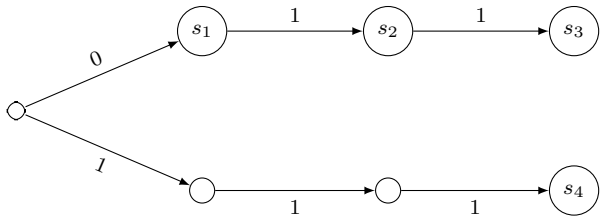


Figure 6: Albero di decodifica per $s_1 \to 0$, $s_2 \to 01$, $s_3 \to 011$, $s_4 \to 111$, codice non istantaneo

Si nota facilmente come durante il ricevimento non si può stabilire univocamente a che s_i si sta facendo riferimento (una codifica non deterministica, o megllio, lo diventa solo alla fine).

L'esempio di codice istantaneo di prima, $s_1 \to 0$, $s_2 \to 10$, $s_3 \to 110$, $s_4 \to 111$ è chiamato **comma code**, in quanto il messaggio termina o quanto arrivano tre 1 oppure quando arriva uno 0. Il relativo albero verrà disegnato "a pettine".

Vediamo un altro esempio di comma code a cinque codeword:

$$s_1 \to 0$$
$$s_2 \to 10$$
$$s_3 \to 110$$
$$s_4 \to 1110$$
$$s_5 \to 1111$$

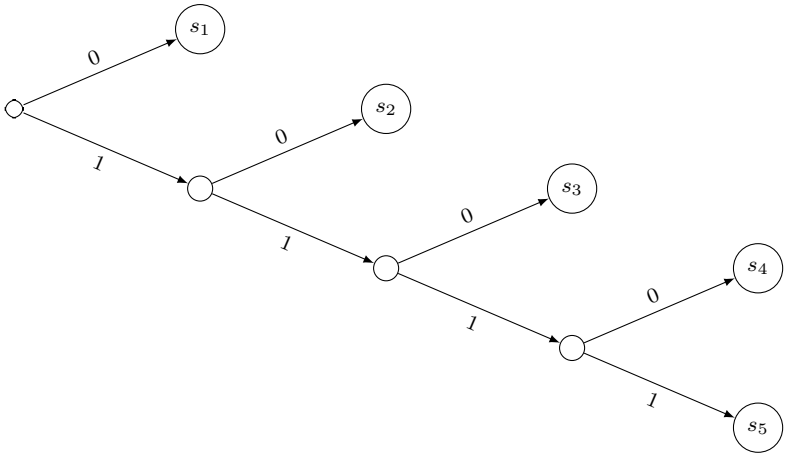


Figure 7: Albero di decodifica per $s_1 \to 0$, $s_2 \to 10$, $s_3 \to 110$, $s_4 \to 1110$, $s_5 \to 1111$

Vediamo ora un altro esempio (non comma code):

$$s_1 \to 00$$
$$s_2 \to 01$$
$$s_3 \to 10$$
$$s_4 \to 110$$
$$s_5 \to 111$$

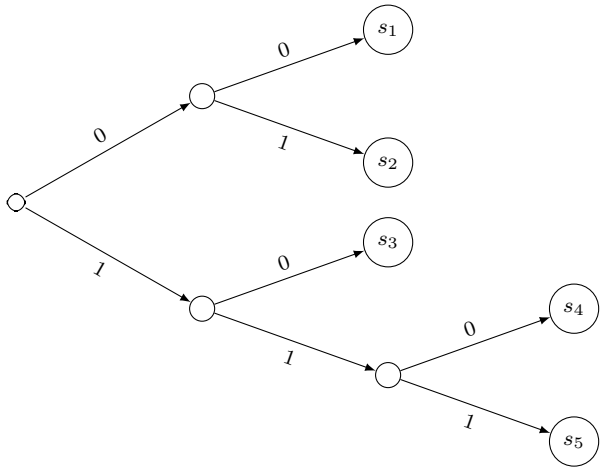


Figure 8: Albero di decodifica per $s_1 \to 00$, $s_2 \to 01$, $s_3 \to 10$, $s_4 \to 110$, $s_5 \to 111$

Dopo aver definito queste due codifiche a cinque simboli, quale sarebbe da preferire? Il criterio che utilizziamo è la lunghezza media, quindi senza sapere le probabilità dei simboli della sorgente non possiamo decidere a priori qual è più vantaggioso.
Supponiamo di avere queste probabilità: $p_1 = 0.9,\ p_2 = p_3 = p_4 = p_5 = 0.025$. Andiamo quindi a calcolare le due lunghezze medie:

$$L_{pettine} = 0.9 \cdot 1 + 0.025 \cdot 2 + 0.025 \cdot 3 + 0.025 \cdot 4 + 0.025 \cdot 4 = 1.225 \frac{bit}{simbolo}$$

$$L_{altroalbero} = 0.9 \cdot 2 + 0.025 \cdot 2 + 0.025 \cdot 2 + 0.025 \cdot 3 + 0.025 \cdot 3 = 2.05 \frac{bit}{simbolo}$$

Quindi utilizzando questa distribuzione di probabilità il comma code è da preferire. Usando la distribizione uniforme: $p_1 = p_2 = p_3 = p_4 = p_5 = 0.2$ avremo che:

$$L_{pettine} = 0.9 \cdot 2 + 0.2 \cdot 2 + 0.2 \cdot 3 + 0.2 \cdot 4 + 0.2 \cdot 4 = 2.4 \frac{bit}{simbolo}$$

$$L_{altroalbero} = 0.2 \cdot 2 + 0.2 \cdot 2 + 0.2 \cdot 2 + 0.2 \cdot 3 + 0.2 \cdot 3 = 2.05 \frac{bit}{simbolo}$$

La conclusione è che senza probabilità non possiamo trarre conclusioni sul tipo di albero da preferire per ottimizzare la lunghezza media delle codeword.

Disuguaglianza di Kraft

Una domanda che potremmo porci è: quali sono le condizioni per cui posso costruire un codice istantaneo?

Teorema 1 *Condizione necessaria e sufficiente affichè esista un codice istantaneo r-ario per una sorgente S di q simboli con codeword di lunghezze $\ell_1, \ell_2, ..., \ell_q$ è che valga:*

$$K = \sum_{i=1}^{q} \frac{1}{r^{\ell_i}} \leq 1 \tag{42}$$

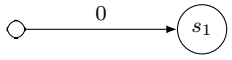
Questo teorema non dice: questo è un codice istantaneo se e solo se..., ma invece dice che esiste un codice istantaneo se e solo se...

L'utilità è la seguente: ho una sorgente con q simboli, voglio costruire un codice istantaneo r-ario con codeword di lunghezze $\ell_1, \ell_2, ..., \ell_q$, posso? Allora calcolo K e vedo se il valore viene ≤ 1

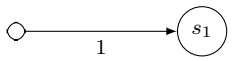
Dimostrazione 1 *La condizione necessaria e sufficiente richiede di dover dimostrare queste due cose:*

- *Per ogni codice istantaneo r-ario per q simboli e codeword di lunghezze* $\ell_1, \ell_2, ..., \ell_q \rightarrow k \leq 1$
- $k \leq 1 \rightarrow$ *esiste codice istantaneo r-ario per q simboli e codeword di lunghezze* $\ell_1, \ell_2, ..., \ell_q$

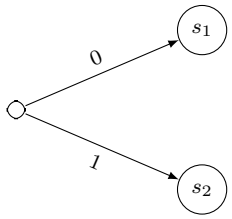
Partiamo dalla prima:
Utilizziamo come valore di $r = 2$*. Parlare di codici istantanei è uguale al considerare i relativi alberi di decodifica. Possiamo quindi dimostrare per induzione sulla profondità dell'albero.*
Caso base *(albero di decodifica di profondità 1) I tre casi possibili sono i seguenti:*



oppure



oppure

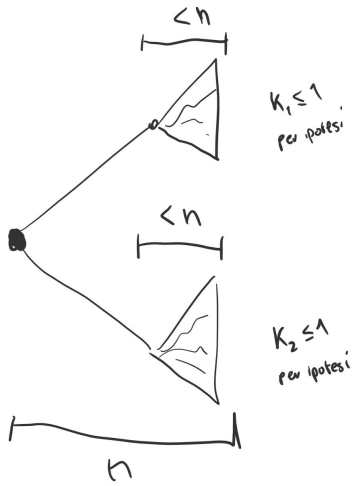


Andiamo quindi a calcolare i $K = \sum_{i=1}^{q} \frac{1}{r^{\ell_i}}$ *dei tre alberi appena costruiti:*

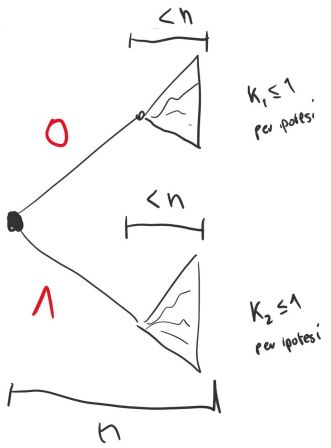
1. $K = \frac{1}{2^1} = \frac{1}{2} \leq 1$
2. $K = \frac{1}{2^1} = \frac{1}{2} \leq 1$
3. $K = \frac{1}{2^1} + \frac{1}{2^1} = 1 \leq 1$

Quindi per il passo base abbiamo dimostrato.

***Passo induttivo**:*
Supponiamo che il teorema valga per alberi di profondità $< n$ e dimostriamo che valga per alberi di profondità n (dimostro per $< n$ e non $n-1$ perchè magari il sottoalbero sopra ha profondità 1, l'altro $n-1$, ma deve valere per entrambi). La situazione è quindi la seguente:



Quindi per ipotesi i due sottoalberi profondi al massimo $n-1$ hanno $K \leq 1$; supponendo di 'attaccarli' ad un nuovo nodo significa appendere come prefisso di tutte le codeword un altro simbolo.



Come cambiano le K quando allungo di uno le lunghezze delle codeword?

Tutte le ℓ_i diventano $\ell_i + 1$:

$$K_1 = \sum_{i=1}^{q} \frac{1}{r^{\ell_i+1}} = \sum_{i=1}^{q} (\frac{1}{r^{\ell_i}} \frac{1}{r}) = \frac{1}{r} \sum_{i=1}^{q} \frac{1}{r^{\ell_i}}$$

Quindi il contributo dato da ognuno dei due sottoalberi al calcolo di K totale è dato da $\frac{1}{2}K_i$ (per $r = 2$). Il K relativo all'albero totale quindi diventa:

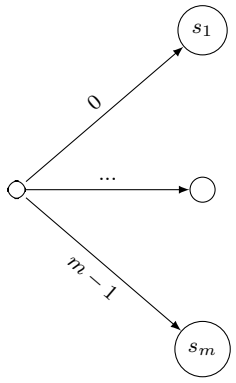
$$K = \frac{1}{2}K_1 + \frac{1}{2}K_2 = \frac{1}{2}(K_1 + K_2)$$

Sappiamo che $K_1 \leq 1$ e che $K_2 \leq 1$, quindi la loro somma sarà sicuramente ≤ 2. La somma, moltiplicata per $\frac{1}{2}$ è ≤ 1; quindi possiamo scrivere:

$$K \leq 1$$

che è quello che volevamo dimostrare.

In caso n non fosse $n = 2$ bisogna solo creare più alberi di profondità 1 nel caso base:



Dato che $m \leq r$ (delle r foglie possibili ne ha solo m)

$$K = m(\frac{1}{r}) = \frac{m}{r} \leq 1$$

Nel passo induttivo il numero di sottoalberi sarà m, e quando attacchiamo i sottoalberi alla nuova radice il nuovo K verrà

$$K = \frac{1}{r}K_1 + \frac{1}{r}K_2 + ... + \frac{1}{r}K_m$$

Dimostriamo ora la seconda parte, ovvero "$k \leq 1 \rightarrow$ esiste codice istantaneo r-ario per q simboli e codeword di lunghezze $\ell_1, \ell_2, ..., \ell_q$ ".

Partiamo dall'ipotesi che $K \leq 1$, successivamente definiamo ℓ come:

$$\ell = max\{\ell_1, \ell_2, ..., \ell_q\}$$

Inoltre chiamo t_j il numero di codeword di lunghezza j.

Riscriviamo K in questo modo:

$$K = \sum_{i=1}^{q} \frac{1}{r^{\ell_i}} = \sum_{i=1}^{\ell} t_i \cdot \frac{1}{r^i} \leq 1$$

Posso riscriverlo in questo modo perchè è come se stessi raccogliendo le codeword di lunghezza uguale t_i. Moltiplico per r^ℓ entrambi i membri:

$$\sum_{i=1}^{\ell} t_i r^{\ell-1} \leq r^\ell$$

Questo equivale a:

$$t_1 r^{\ell-1} + t_2 r^{\ell-2} + .. + t_{\ell-1} r + t_\ell \leq r^\ell$$

Isolo t_ℓ:

$$t_\ell \leq r^\ell - t_1 r^{\ell-1} - t_2 r^{\ell-2} - ... - t_{\ell-1} r$$

So che t_ℓ è maggiore o uguale a zero (numero di codeword di lunghezza ℓ)

$$0 \leq t_\ell \leq r^\ell - t_1 r^{\ell-1} - t_2 r^{\ell-2} - ... - t_{\ell-1} r$$

Quindi ho posto un vincolo al numero massimo di codeword di lunghezza ℓ. Ora considero solo questo pezzo:

$$0 \leq r^\ell - t_1 r^{\ell-1} - t_2 r^{\ell-2} - ... - t_{\ell-1} r$$

Divido tutto per r e porto a sinistra l'ultimo termine $t_{\ell-1} r$:

$$0 \leq t_{\ell-1} \leq r^{\ell-1} - t_1 r^{\ell-2} - t_2 r^{\ell-3} - ... - t_{\ell-2} r$$

Posso rifarlo per tutti i t:

$$0 \leq t_{\ell-2} \leq ...$$

E mi fermo a

$$0 \leq t_2 \leq r^2 - t_1 r \tag{43}$$

$$0 \leq t_1 \leq r \tag{44}$$

Quindi sto ponendo dei limiti alle codeword di lunghezza 1, di lunghezza 2, ecc...

A questo punto, partendo dall'ultima equazione, risalgo.

Quante codeword di lunghezza 1 posso creare? (Equazione 44) Boh, di sicuro un numero compreso fra 0 e r.

Poi mi chiedo quante di lunghezza 2 (Equazione 43)? Boh quelle rimaste sono $r - t_1$, da queste posso appendere un carattere qualsiasi, quindi r. Per cui ne posso creare al masssimo $r^2 - t_1 r$

Proseguendo in questo modo verifico tutte le disuguaglianze, partendo dalla sola ipotesi che $K \leq 1$

Appunti "Disugliaglianza di Kraft MacMillan.pdf"

Lezione 7

Kraft da condizione necessaria e sufficiente per fare in modo che esista un codice istantaneo r-ario di q simboli avente codeword di lunghezza $\ell_1, \ell_2, ...\ell_q$.

MacMillan dimostra che lo **stesso** teorema è estendibile ai codici univocamente decodificabili.

Figure 9: I codici istantanei sono un sottoinsieme dei codici univocamente decodificabili

Importante

↓

Non è vero che il codice è istantaneo/univocamente decodificabile se $K \leq 1$
ma è vero che *esiste* un codice istantaneo con quelle caratteristiche
(r-ario con q caratteri con lunghezze $\ell_1, \ell_2, ..., \ell_q$) se e solo se $K \leq 1$

.

Riprendiamo come esempio questo codice non istantaneo:

$$s_1 \to 0$$
$$s_2 \to 01$$
$$s_3 \to 011$$
$$s_4 \to 111$$

Rigirando le cifre codificate:

$$s_1 \to 0$$

$$s_2 \to 10$$

$$s_3 \to 110$$

$$s_4 \to 111$$

Otteniamo un codice istantaneo!

Effettivamente la K di Kraft accede solo alle lunghezze delle codeword (ℓ_i) e il numero di simboli con cui scriviamo le codeword (r).
Questo per dire che le caratteristiche del codice possono renderlo istantaneo, ma sta poi a chi codifica farlo diventare istantaneo.

Disuguaglianza di McMillan

Teorema 2 *Condizione necessaria e sufficiente affinchè* ***esista*** *un codice univocamente decodificabile r-ario per una sorgente di q simboli con codeword di lunghezze $\ell_1, \ell_2, ..., \ell_q$ è che valga:*

$$K = \sum_{i=1}^{q} \frac{1}{r^{\ell_i}} \leq 1 \tag{45}$$

1. Per ogni codice univocamente decodificabile r-ario per una sorgente di q simboli con codeword di lunghezze $\ell_1, \ell_2, ..., \ell_q \to K \leq 1$.

2. Se $K \leq 1$ allora esiste un codice univocamente decodificabile r-ario con lunghezze $\ell_1, \ell_2, ..., \ell_q$.

Dimostrazione 2 *Dividiamo l'enunciato del teorema nelle due considerazioni fatte sopra.*

Parte 2:
Se $K \leq 1 \to$ (per Kraft) esiste un codice istantaneo r-ario con lunghezze $\ell_1, \ell_2, ..., \ell_q$. Lo stesso codice è univocamente decodificabile (istantaei sono un sottoinsieme di univocamente decodificabili).

Parte 1:
Eleviamo K^n, dove $n > 1$ intero. Quindi avremo che:

$$K^n = [\sum_{i=1}^{q} \frac{1}{r^{\ell_i}}]^n$$

Definiamo $\ell = max\{\ell_1, \ell_2, ..., \ell_q\}$, quindi riscriviamo la sommatoria come:

$$K^n = \sum_{t=n}^{n\ell} \frac{Nt}{r^t}$$

N.B. passaggio simile a quello effettuato nella dimostazione della seconda parte di Kraft

Da qui noto che il numeratore Nt è il numero di codeword di lunghezza t, ma questo numero non può essere maggiore di r^t visto che rappresenta tutti i possibili vettori lunghi t dove ogni elemento è un carattere dell'alfabeto r-ario.

$$Nt \leq r^t$$

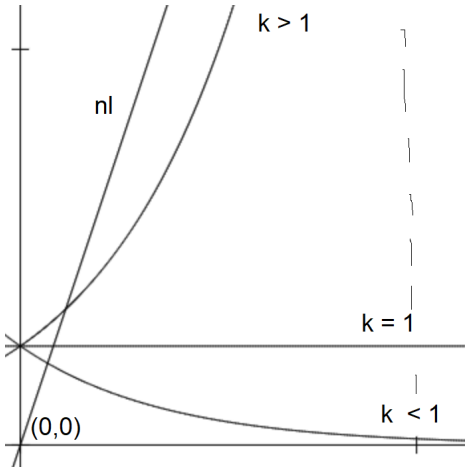
Quindi posso maggiorare la sommatoria:

$$K^n \leq \sum_{t=n}^{n\ell} \frac{Nt}{r^t} = \sum_{t=n}^{n\ell} 1 = n\ell - n + 1 = n\ell - (n-1)$$

$(n-1)$ *è negativo visto che per ipotesi* $n > 0$*, quindi:*

$$K^n \leq n\ell$$

Analizziamo quindi i diversi casi:



In maniera asintotica la disuguaglianza è rispettata solo da $k = 1$ e $k < 1$, quindi questo conclude la dimostazione.

Una domanda che potrebbe sorgere è: ha senso utilizzare codici univocamente decodificabili ma non istantanei? No perchè lo sforzo per trovarli è uguali, quindi inutile perderere tempo su codici non istantanei.

Date le probabilità dei simboli qual è il modo più veloce per creare un codice istantaneo che minimizzi $L = \sum_{i=1}^{q} p_i \ell_i$?

Codici a blocchi accorciati

Si parte da codeword di lunghezza uguale per poi accorciare.Facciamo un esempio:
$r = 2$
$q = 5$
Per rappresentare 5 simboli ci servono 3 bit; tutte le possibili combinazioni sono:

000
001
010
011
100
101
110
111

Dobbiamo utilizzarne solo 5, quindi scartiamone 3

000 s_1
~~001~~
010 s_2
~~011~~
100 s_3
~~101~~
110 s_4
111 s_5

A questo punto creiamo un albero di decodifica:

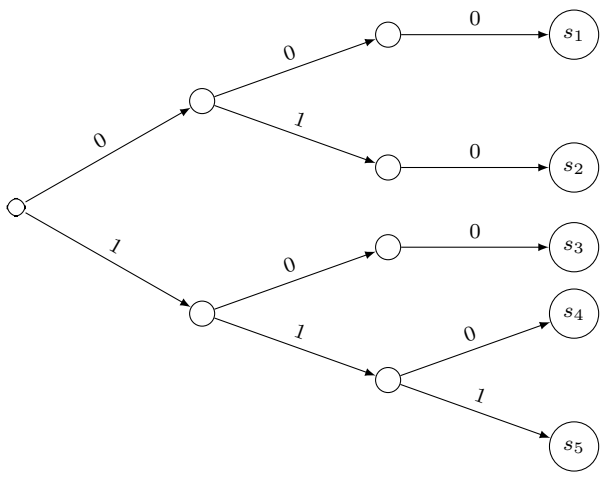


Figure 10: Albero di decodifica codice a blocchi

Partendo dall'albero di questo codice a blocchi, posso effettuare un processo di "sfoltimento" sui nodi di decisione con solo un figlio:

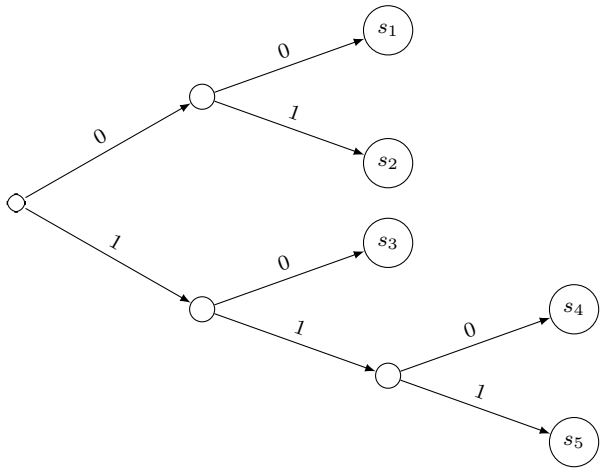


Figure 11: Albero sfoltito a partire dal precedente

Ma come facciamo ad assicurarci che questa sia la migliore codifica per questo problema?

Codici di Huffman

I codici restituiti dall'algoritmo di Huffman sono **ottimali**, quindi il valore di $L = \sum_{i=1}^{q} p_i \ell_i$ è minimo.

Ordiniamo le probabilità dalla più grande alla più piccola (ordine non crescente):

$$p_1 \geq p_2 \geq ... \geq p_q$$

allora un codice ottimale deve associare queste probabilità alle lunghezze poste in ordine crescente:

$$\ell_1 \leq \ell_2 \leq ... \leq \ell_q$$

Se questo non è verificato allora il codice non è ottimale.

Dimostrazione 3 *Assumiamo che le probabiltà siano ordinate in ordine non crescente.*
Esistono due indici m e n con $m < n$ tali per cui $p_m > p_n$ e $\ell_m > \ell_n$ (quindi viola la regola di prima). Se questo è vero, posso scambiare le codeword e ottenere una lunghezza media più piccola, quindi il codice iniziale da cui siamo partiti

non era ottimale. Il contributo nella L prima dello scambio era $p_m\ell_m + p_n\ell_n$, questo contributo diventerà $p_m\ell_n + p_n\ell_m$. La differenza fra queste quantità vale

$$p_m\ell_n + p_n\ell_m - p_m\ell_m + p_n\ell_n$$

$$= p_m(\ell_n - \ell_m) - p_n(\ell_n - \ell_m)$$

$$= (p_m - p_n)(\ell_n - \ell_m)$$

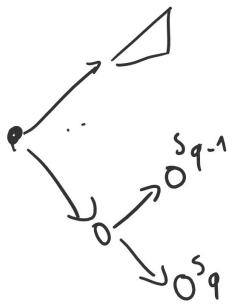
Queste due quantità sono rispettivamente > 0 e < 0, quindi la differenza è negativa, quindi scambiando le codeword avrò una lunghezza media minore. Quindi questo dimostra che sono partito da un codice non ottimale in quanto le lunghezze delle codeword non sono in ordine non decrescente.

Ora riconsideriamo

$$p_1 \geq p_2 \geq ... \geq p_{q-1} \geq p_q$$

$$\ell_1 \leq \ell_2 \leq ... \leq \ell_{q-1} \leq \ell_q$$

I due simboli meno probabili avranno codeword di uguale lunghezza, per costruzione dell'albero (si apre sempre in due, quindi $\downarrow$)



Quindi per concludere: se le probabilità e le lunghezze seguono la regola precedente e questa regola appena spiegata è rispettata, allora il codice è ottimale. Ma come creo un codice ottimale?

Facciamo un esempio di esecuzione di Algoritmo di Huffman.
$r = 2$, Abbiamo cinque simboli che escono con le seguenti probabilità:

$$p_1 = 0.4$$
$$p_2 = 0.2$$
$$p_3 = 0.2$$
$$p_4 = 0.1$$
$$p_5 = 0.1$$

L'algoritmo di Huffman ha un esecuzione di tipo greedy e funziona così:

1. Prendo i due simboli meno probabili e creo un simbolo 'virtuale' combinandoli, quindi dall'esempio prima otterrò:

$$p_1 = 0.4$$
$$p_2 = 0.2$$
$$p_3 = 0.2$$
$$p_{4,5} = 0.1 + 0.1 = 0.2$$

2. Itero il passaggio precedente mantenendo traccia delle varie unioni:

$p_1 = 0.4$	$p_1 = 0.4$	$p_1 = 0.4$
$p_2 = 0.2$	$p_{3,4,5} = 0.4$	$p_{2,3,4,5} = 0.6$
$p_3 = 0.2$	$p_2 = 0.2$	
$p_{4,5} = 0.2$		

3. Mi fermo quando ottengo r probabilità e assegno un carattere ciascuno:

$$p_1 = 0.4 \to \texttt{0}$$
$$p_{2,3,4,5} = 0.6 \to \texttt{1}$$

4. Ora itero all'indietro, appendendo un carattere alla volta nelle probabilità virtuali (appendo alla fine, se no creo prefissi e il codice non è più istantaneo), quando una virtuale ha lunghezza uguale a un altro simbolo la metto sotto.

$p_{2,3,4,5} = 0.6 \to \texttt{0}$	$p_1 = 0.4 \to \texttt{1}$	$p_1 = 0.4 \to \texttt{1}$	$p_1 = 0.4 \to \texttt{1}$
$p_1 = 0.4 \to \texttt{1}$	$p_{3,4,5} = 0.4 \to \texttt{00}$	$p_2 = 0.2 \to \texttt{01}$	$p_2 = 0.2 \to \texttt{01}$
	$p_2 = 0.2 \to \texttt{01}$	$p_3 = 0.2 \to \texttt{000}$	$p_3 = 0.2 \to \texttt{000}$
		$p_{4,5} = 0.2 \to \texttt{001}$	$p_4 = 0.1 \to \texttt{0010}$
			$p_5 = 0.1 \to \texttt{0011}$

Si noti come la regola delle probabilità in ordine inverso rispetto alle lunghezze sia rispettata in tutti i passaggi.

Quanto vale la lunghezza media del codice appena calcolato?

$$L = \sum_{i=1}^{q} p_i \ell_i = 0.4 \cdot 1 + 0.2(2+3) + 0.1 \cdot (4+4) =$$

$$= 0.4 + 0.2 \cdot 5 + 0.1 \cdot 8 = 0.4 + 1 + 0.8 = 2.2 \frac{bit}{simbolo}$$

Proviamo a effettuare un altra esecuzione variando una regola, nel caso due simboli abbiano stessa probabilità, metto sopra quello virtuale (prima era il contrario):

$p_1 = 0.4 \rightarrow \texttt{00}$	$p_1 = 0.4 \rightarrow \texttt{00}$	$p_{2,3} = 0.4 \rightarrow \texttt{1}$	$p_{1,4,5} = 0.6 \rightarrow \texttt{0}$
$p_2 = 0.2 \rightarrow \texttt{10}$	$p_{4,5} = 0.2 \rightarrow \texttt{01}$	$p_1 = 0.4 \rightarrow \texttt{00}$	$p_{2,3} = 0.4 \rightarrow \texttt{1}$
$p_3 = 0.2 \rightarrow \texttt{11}$	$p_2 = 0.2 \rightarrow \texttt{10}$	$p_{4,5} = 0.2 \rightarrow \texttt{01}$	
$p_4 = 0.1 \rightarrow \texttt{010}$	$p_3 = 0.2 \rightarrow \texttt{11}$		
$p_5 = 0.1 \rightarrow \texttt{011}$			

Le codeword ottenute sono ovviamente diverse, ma la lunghezza media rimarrà uguale:

$$L = \sum_{i=1}^{q} p_i \ell_i = 0.4 \cdot 2 + 0.2(2 + 2) + 0.1 \cdot (3 + 3) =$$

$$= 0.8 + 0.2 \cdot 4 + 0.1 \cdot 6 = 0.8 + 0.8 + 0.6 = 2.2 \frac{bit}{simbolo}$$

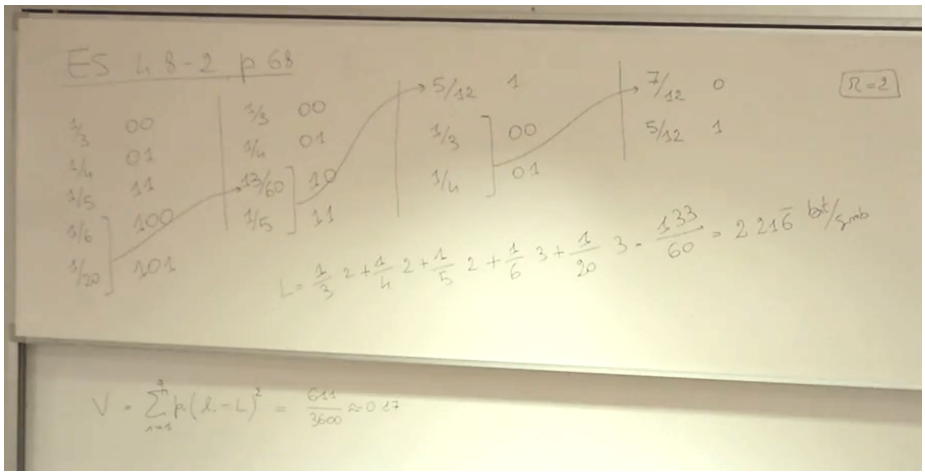
Quello che cambia sarà la varianza.

Lezione 8

Portando le probabilità virtuali uguali sopra quindi abbiamo detto si ridurrà la varianza (scelta che preferiamo per robustezza ecc).

Es.4.8-2 pag.68 Creare codice ottimale per le seguenti probabilità:

$$\frac{1}{3} \mid \frac{1}{4} \mid \frac{1}{5} \mid \frac{1}{6} \mid \frac{1}{20}$$

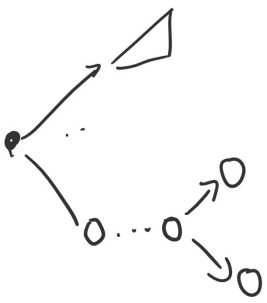


Non è possibile costruire codice binario istantaneo migliore (in termini di lunghezza media) di quello prodotto dall'algoritmo di Huffman

Dimostrazione 4 *Per assurdo:*
Consideriamo delle probabilità $p_1, p_2, ..., p_q$ e una L dell'algoritmo di Huffman. Succcessiamente consideriamo una L' ipoteticamente migliore, quindi diciamo che $L' < L$. Consideriamo l'albero i due alberi costruiti:

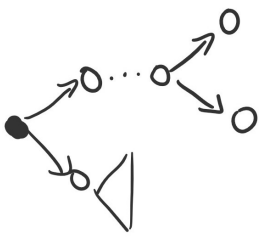
Albero algoritmo Huffman:

L



Albero algoritmo "migliore":

$L' < L$



Ora soffermiamoci sui due codici più lunghi. Ricordiamo che in un codice ottimale, le probabilità sono ordinate in modo decrescente e le lunghezze in modo crescente. In altri termini:

$$p_1 \geq p_2 \geq ... \geq p_{q-1} \geq p_q$$
$$\ell_1 \leq \ell_2 \leq ... \leq \ell_{q-1} \leq \ell_q$$

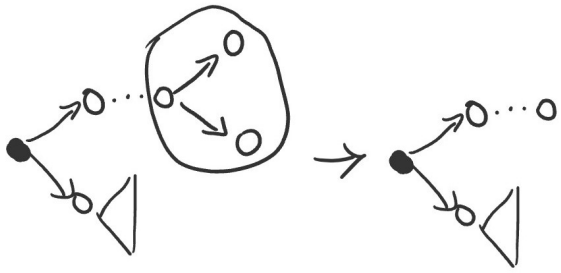
Ricordiamoci che le ultime due codeword in ordine di lunghezza devono essere uguali (pensa all'albero, se non sono uguali vuol dire che ho un nodo di decisione con un solo figlio), quindi:

$$\ell_{q-1} = \ell_q$$

Nel calcolo di L avrò che negli ultimi due termini andrò a sommare $(\ell_{q-1} \cdot p_{q-1}) + (\ell_q \cdot p_q)$*, ma visto che* $\ell_{q-1} = \ell_q$ *è come se stessi facendo*

$$\ell_q(p_{q-1} + p_q)$$

Soffermiamoci su questi due caratteri e consideriamo l'ultimo sottoalbero; sostituiamolo con un singolo nodo "virtuale" (come accade nell algoritmo di huffman):



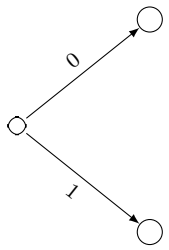
Avendo effettuato questa operazione, nella lunghezza media il contributo diventa:

$$\ell_q - 1(p_{q-1} + p_q)$$

N.B. sottraggo uno in quando scendo di un livello nell'albero, quindi le codeword avranno un carattere in meno

Questa operazione accorcia le lunghezze medie dei due algoritmi in maniera uguale. A questo punto avrò un albero che rappresenta una sorgente di $q - 1$ *simboli. Continuo iterativamente ad accorciare gli alberi risultanti, facendo ridurre la lunghezza media in modo uguale nei due algoritmi. Alla fine per Huffman arriverò all'albero con due soli simboli, la cui lunghezza media è 1, a questo punto l'altro algoritmo dovrebbe avere un'albero con lunghezza media di lunghezza* < 1*, ma che non ha senso!*

Albero minimale di Huffman con $L = 1$



Albero minimale dell'algoritmo migliore con $L < 1$

?

Quindi l'esistenza di tale algoritmo è assurda

Algoritmo di Huffman con $r > 2$

Ora proviamo a lavorare con $r = 3$, con cifre 0, 1, 2.

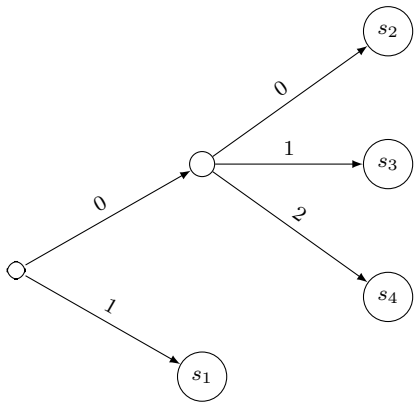
Supponiamo di avere una sorgente di quattro simboli con probabilità:

$$p_1 = 0.4$$
$$p_2 = 0.3$$
$$p_3 = 0.2$$
$$p_4 = 0.1$$

A questo punto lanciamo l'algoritmo di Huffman:

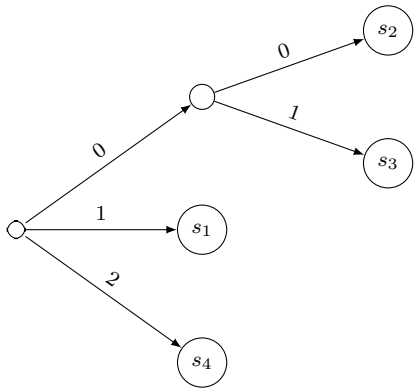
$p_1 = 0.4 \to \texttt{1}$
$p_2 = 0.3 \to \texttt{00}$
$p_3 = 0.2 \to \texttt{01}$
$p_4 = 0.1 \to \texttt{02}$

$p_{2,3,4} = 0.6 \to \texttt{0}$
$p_1 = 0.4 \to \texttt{1}$



$$L = 1 \cdot 0.4 + 2 \cdot (0.3 + 0.2 + 0.1) = 1.6 \ \frac{\text{cifre ternarie}}{\text{simbolo}}$$

Però questo albero non è ottimale! Lo si può correggere in questo modo:



$$L = 1 \cdot (0.4 + 0.1) + 2 \cdot (0.3 + 0.2) = 1.5 \ \frac{\text{cifre ternarie}}{\text{simbolo}}$$

C'è qualche problema quindi con l'algoritmo applicato ad un $r \neq 2$.In particolare vorrei che dalla radice uscissero tutti i simboli di gamma.

Nell'esempio precedente, potrei lavorare in questo modo:

$p_1 = 0.4 \to \texttt{0}$	$p_1 = 0.4 \to \texttt{0}$
$p_2 = 0.3 \to \texttt{2}$	$p_{3,4} = 0.3 \to \texttt{1}$
$p_3 = 0.2 \to \texttt{10}$	$p_2 = 0.3 \to \texttt{2}$
$p_4 = 0.1 \to \texttt{11}$	

Ma quindi quanti simboli devo raccogliere ad ogni iterazione?

L'operazione appena effettuata sopra può essere vista come un raccoglimento triplo, includendo un simbolo fittizio con probabilità 0:

$p_1 = 0.4 \to \texttt{0}$	$p_1 = 0.4 \to \texttt{0}$
$p_2 = 0.3 \to \texttt{2}$	$p_{3,4,5} = 0.3 \to \texttt{1}$
$p_3 = 0.2 \to \texttt{10}$	$p_2 = 0.3 \to \texttt{2}$
$p_4 = 0.1 \to \texttt{11}$	
~~$p_5 = 0.0 \to \texttt{12}$~~	

Quindi posso definire un'altro insieme $q' \geq q$ che include anche i simboli fittizi, per un totale di $q' - q$ simboli fittizi.

$$q' = k(r-1) + r$$

con k che rappresenta il numero di passi (iterazioni) dell'algoritmo, ma non conosco questa quantità!

$$q' = k(r-1) + r = (k+1)(r-1) + 1$$
$$\equiv 1 \; mod \; (r-1)$$

Quindi devo trovare il più piccolo $q' \geq q$ tale che $q' \equiv 1 \; mod \; (r-1)$.

Proviamo con l'esempio di prima: devo trovare il più piccolo $q' \geq 4$ tale che $q' \equiv 1 \; mod \; 2$, quindi $q' = 5$.
Il numero di simboli fittizi è quindi $q' - q = 1$.

Vediamo un esempio con una sorgente un po' più grossa con $r = 4$:

$$p_1 = 0.22$$
$$p_2 = 0.20$$
$$p_3 = 0.18$$
$$p_4 = 0.15$$
$$p_5 = 0.10$$
$$p_6 = 0.08$$
$$p_7 = 0.05$$
$$p_8 = 0.02$$

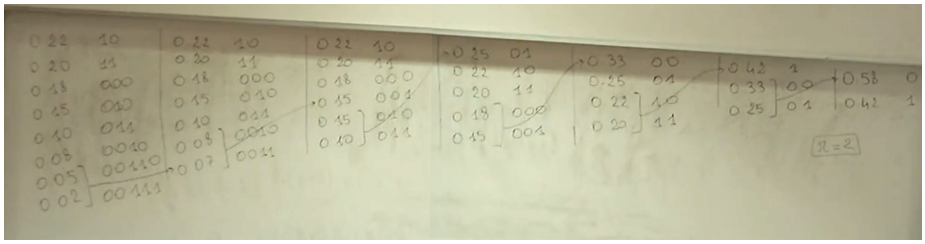
Secondo la regola di prima dobbiamo trovare il più piccolo $q' \geq 8$ tale che $q' \equiv 1 \; mod \; (3)$. Il q' cercato è 10 in quanto è ≥ 8 e diviso 3 ha resto 1; in conclusione il numero di simboli fittizi sarà $q' - q = 10 - 8 = 2$.
I simboli finali saranno:

$$p_1 = 0.22$$
$$p_2 = 0.20$$
$$p_3 = 0.18$$
$$p_4 = 0.15$$
$$p_5 = 0.10$$
$$p_6 = 0.08$$
$$p_7 = 0.05$$
$$p_8 = 0.02$$
$$p_9 = 0.00$$
$$p_{10} = 0.00$$

Applichiamo quindi l'algoritmo di Huffman a questo insieme di probabilità:

$p_1 = 0.22 \to 1$	$p_1 = 0.22 \to 1$	$p_{4,5,6,7,8,9,10} = 0.4 \to 0$
$p_2 = 0.20 \to 2$	$p_2 = 0.20 \to 2$	$p_1 = 0.22 \to 1$
$p_3 = 0.18 \to 3$	$p_3 = 0.18 \to 3$	$p_2 = 0.20 \to 2$
$p_4 = 0.15 \to 00$	$p_4 = 0.15 \to 00$	$p_3 = 0.18 \to 3$
$p_5 = 0.10 \to 01$	$p_5 = 0.10 \to 01$	
$p_6 = 0.08 \to 02$	$p_6 = 0.08 \to 02$	
$p_7 = 0.05 \to 030$	$p_{7,8,9,10} = 0.07 \to 03$	
$p_8 = 0.02 \to 031$		
~~$p_9 = 0.00 \to 032$~~		
~~$p_{10} = 0.00 \to 033$~~		

Ora proviamo ad utilizzare $r = 2$ cifre sugli stessi simboli, e poi confrontiamo le lunghezze medie dei codici:



$$L_{binario} = 0.42 \cdot 2 + 0.43 \cdot 3 + 0.08 \cdot 4 + 0.07 \cdot 4 =$$

$$= 0.84 + 1.29 + 0.32 + 0.35 = 2.13 + 0.67 = 2.8 \ \frac{\text{bit}}{\text{simbolo}}$$

Mentre per $r = 4$:

$$L_{quattro} = 0.60 \cdot 1 + 0.33 \cdot 2 + 0.07 \cdot 3 =$$

$$= 0.60 + 0.66 + 0.21 = 1.26 + 0.21 = 1.47 \ \frac{\text{cifre quaternarie}}{\text{simbolo}}$$

Effettivamente potremmo passare da un codice all'altro utilizzando una tabella per la conversione dei codici:

Γ	binario
0	00
1	01
2	10
3	11

$p_1 = 0.22 \to \texttt{1}$	$p_1 = 0.22 \to \texttt{01}$
$p_2 = 0.20 \to \texttt{2}$	$p_2 = 0.20 \to \texttt{11}$
$p_3 = 0.18 \to \texttt{3}$	$p_3 = 0.18 \to \texttt{11}$
$p_4 = 0.15 \to \texttt{00}$	$p_4 = 0.15 \to \texttt{0000}$
$p_5 = 0.10 \to \texttt{01}$	$p_5 = 0.10 \to \texttt{0001}$
$p_6 = 0.08 \to \texttt{02}$	$p_6 = 0.08 \to \texttt{0010}$
$p_7 = 0.05 \to \texttt{030}$	$p_7 = 0.05 \to \texttt{001100}$
$p_8 = 0.02 \to \texttt{031}$	$p_8 = 0.02 \to \texttt{001101}$

E viene un codice diverso da quello calcolato prima, la lunghezza media sarà il doppio di quella precedente (ogni simbolo di Σ viene convertito in due simboli binari), quindi avrò che $L = 2 \cdot 1.47 = 2.94 \ \frac{\text{bit}}{\text{simbolo}}$, che è maggiore di $2.8 \ \frac{\text{bit}}{\text{simbolo}}$ calcolato applicando Huffman con $r = 2$.
Possiamo quindi affermare che questo tipo di "scorciatoia" non produce un codice ottimale.

Come si dimostra che il codice prodotto sia ottimale (con $r = 4$)? Allo stesso modo, si individuano le r codeword più piccole e le si raggruppano ottenendo un albero r-ario (aggiungendo il giusto numero di simboli fittizi).

Robustezza codici di Huffman

Come mai si preferiscono codici con meno varianza (quindi a parità di probabilità metto il simbolo virtuale sopra)?

Data una sorgente S noi stimiamo le probabilità di uscita dei simboli s_i osservando in un lasso di tempo più o meno lungo. In pratica calcolo dei p'_i che sarebbero delle p_i sommate ad un errore. Ovviamente deve valere che

$$\sum_{i=1}^{q} p'_i = 1$$

Ma dato che $p'_i = p_i + e_i$:

$$\sum_{i=1}^{q} p'_i = \sum_{i=1}^{q} p_i + \sum_{i=1}^{q} e_i = 1$$

Quindi

$$1 = 1 + \sum_{i=1}^{q} e_i$$

$$\sum_{i=1}^{q} e_i = 0 \tag{46}$$

Quindi gli errori in media si "compensano" a vicenda (la coperta è corta: se la tiro da una parte si scopre l'altra ;)) La media degli errori è uguale a 0:

$$\frac{1}{q} \sum_{i=1}^{q} e_i = 0 \tag{47}$$

La varianza di questi errori vale:

$$\sigma^2 = \frac{1}{q}\sum_{i=1}^{q} e_i^2 \tag{48}$$

E la lunghezza calcolata con le probabilità stimate?

$$L' = \sum_{i=1}^{q} p_i' \ell_i = \sum_{i=1}^{q} p_i \ell_i + \sum_{i=1}^{q} e_i \ell_I = L + \sum_{i=1}^{q} e_i \ell_i$$

Quindi l'obiettivo è quello di minimizzare $\sum_{i=1}^{q} e_i \ell_i$. Consideriamola come una funzione f di q variabili:

$$minf(e_1, e_2, ..., e_q) = \sum_{i=1}^{q} e_i \ell_i$$

$$\text{t.c.} \sum_{i=1}^{q} e_i \ell_i = 0 \qquad \wedge$$

$$\frac{1}{q}\sum_{i=1}^{q} e_i^2 - \sigma^2 = 0$$

Per risolvere questo problema di minimo vincolato si usano i moltiplicatori di Lagrange:

$$\mathscr{L}(l_1, ..., l_q) = \sum_{i=1}^{q} e_i \ell_i - \lambda \sum_{i=1}^{q} e_i - \mu(\frac{1}{q}\sum_{i=1}^{q} e_i^2 - \sigma^2)$$

Per risolvere questo problema uso le derivate:

$$\frac{\partial \mathscr{L}}{\partial e_i} = \ell_i - \lambda - \frac{2\mu}{q} e_i = 0 \qquad \forall i \in \{1, 2, ..., q\}$$

Derivo rispetto a $\mathscr{L}$:

$$\sum_{i=1}^{q} \ell_i - \lambda q - \frac{2\mu}{q}\sum_{i=1}^{q} e_i = 0$$

Ma ricordiamo che $\sum_{i=1}^{q} e_i = 0$, quindi

$$\boxed{\lambda = \frac{1}{q}\sum_{i=1}^{q} \ell_i}$$

Derivo rispetto a e_i:

$$\sum_{i=1}^{q} e_i \ell_i - \lambda \sum_{i=1}^{q} e_i - \frac{2\mu}{q}\sum_{i=1}^{q} e_i^2 = 0$$

Ma ricordiamo che $\sum_{i=1}^{q} e_i = 0$ e che $\frac{1}{q}\sum_{i=1}^{q} e_i^2 = 0$, quindi

$$\sum_{i=1}^{q} \ell_i - 2\mu\sigma^2 = 0$$

$$\boxed{\mu = \frac{1}{2\sigma^2}\sum_{i=1}^{q} e_i\ell_i}$$

Ora sostituiamo i moltiplicatori appena trovati all'interno di

$$\frac{\partial \mathscr{L}}{\partial e_i} = \ell_i - \lambda - \frac{2\mu}{q}e_i = 0$$

e otterremo:

$$\sum_{i=1}^{q} \ell_i^2 - \frac{1}{q}(\sum_{i=1}^{q} \ell_i)^2 - \frac{1}{q\sigma^2}(\sum_{i=1}^{q} e_i\ell_i)^2 = 0$$

$$(\sum_{i=1}^{q} e_i\ell_i)^2 = \sigma^2[q\sum_{i=1}^{q} \ell_i - (\sum_{i=1}^{q} \ell_i)^2]$$

$$\sigma^2 q^2[\frac{1}{q}\sum_{i=1}^{q} \ell_i^2 - (\frac{1}{q}\sum_{i=1}^{q} \ell_{-i})^2]$$

$$= \sigma^2 q^2[var(\ell_i)]$$

Quindi otteniamo che la quantità che vogliamo minimizzare è proporzionale alla varianza delle ℓ_i moltiplicata per il numero di simboli della sorgente al quadrato e il quadrato di sigma. Quindi più piccola è la varianza di ℓ_i più è piccola la sommatoria degli errori che vogliamo minimizzare (che rappresenta l'errore con cui conosco le probabilità p_i').
Quindi varianza minore rappresenta una minor differenza $L - L'$. In questo modo rendiamo 'robusto' il codice di Huffman: rendiamo la lunghezza media meno sensibile agli errori. All'orale interessano i concetti, non saper rifare tutto il calcolo (però sapere il procedimento e le conclusioni)

Lezione 9

Teoria dell'informazione - Entropia

Prendiamo una sorgente $S = s_1, s_2, ..., s_q$ caratterizzato dalle probabilità $p_1, p_2, ..., p_q$. Definiamo una funzione $I(s_i)/I(p_i)$ come la quantità di informazione data da un simbolo. La quantità di informazione è proporzionale alla "sorpresa" che abbiamo nel leggere il simbolo.

Dominio:

$$I : S \to \mathbb{R}$$

Inversamente proporzionale alla probabilità:

$$I(p_i) = \frac{1}{p_i}$$

Abbiamo bisogno inoltre della proprietà additiva: l'informazione di due eventi stocastici indipendenti si sommano:

$$I(p_i) + I(p_j) = \frac{1}{p_i} + \frac{1}{p_j} = \frac{p_j + p_i}{p_i p_j}$$

Ma quello che accade nella realtà è che la probabilità che avvengano due eventi è

$$I(p_i p_j) = \frac{1}{p_i p_j}$$

Quindi la funzione $I = \frac{1}{p_i}$ non gode della proprietà additiva, quindi la scartiamo.

Cerchiamo di capire come si comporta la funzione che stiamo cercando:

1. $I(p) \geq 0 \quad (positivita)$
2. $I(p_1 p_2) = I(p_1) + I(p_2) \quad (additivita)$
   se gli eventi sono indipendenti
3. I è una funzione continua di p

Notiamo che $I(p^n) = nI(p)$

Base induzione:

$I(p) = I(p)$

Passo induttivo:

Supponiamo che valga $I(p^{n-1}) = (n-1)I(p)$,

$I(p^n) = I(pp^{n-1}) = I(p) + I(p^{n-1}) = I(p) + (n-1)I(p) = nI(p)$

Ora definiamo $y = p^n$, da cui ricaviamo che $p = y^{\frac{1}{n}}$. Poi calcoliamo $I(y) = I(p^n) = nI(p) = nI(y^{\frac{1}{n}})$.

$$I(y^{\frac{1}{n}}) = \frac{1}{n} I(Y) \qquad \forall n \in N$$

Ora estendiamo per tutti i numeri razionali:

$$I(y^{\frac{m}{n}}) = I((y^{\frac{1}{n}})^m) = mI(y^{\frac{1}{n}}) = \frac{m}{n} I(y)$$

Osserviamo queste proprietà, c'è una funzione che le rispetta tutte: il logaritmo.

$$I(p) = k \cdot \log p$$

per $k = -1$ ottengo:

$$\boxed{I(p) = \log \frac{1}{p}}$$

La proprietà di additività è rispettata:

$$I(p_1) + I(p_2) = \log \frac{1}{p_1} + \log \frac{1}{p_2} = \log \frac{1}{p_1 p_2} = I(p_1 p_2)$$

Quale base utilizziamo per il logaritmo? Essa ci fornisce l'unità di misura con cui calcoliamo la quantità di informazione. La base più utilizzata in natura è e, mentre nel mondo digitale si utilizza molto la base 2.

Quanto si calcola in ln l'unità di misura dell'informazione è *nat*, in base 2 l'unità di misura è il *bit*, in base 10 si misura in *Hartley*. In ogni caso posso cambiare base velocemente moltiplicando per una costante C.

Ma il logaritmo è l'unica funzione che va bene per rappresentare la funzione I?

Dimostrazione 5 *Fissiamo* $I(p) = \log_2 \frac{1}{p}$.
Abbiamo già visto questa proprietà:

$$I(p^n) = nI(p)$$

Supponiamo che esista una funzione g *per cui* $g(p^n) = ng(p)$ *che sia continua, e che restituisca valori* ≥ 0.

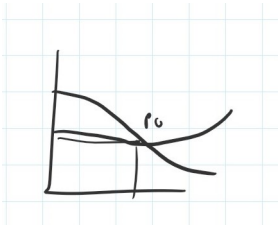
Proviamo a calcolare la differenza fra il logaritmo e questa ipotetica funzione:

$$g(p^n) - C \log_2 \frac{1}{p^n} = n[g(p) - C \log_2 \frac{1}{p}]$$

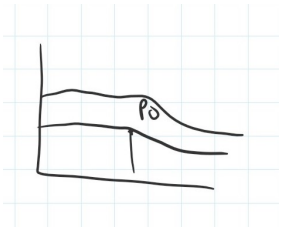
Poniamo $C = \frac{g(p_0)}{\log_2 \frac{1}{p_0}}$, *con* $p_0 \neq 0, 1$, *quindi diverso dagli estremi. In questo modo la differenza fra le due diventa zero.*

Ora possono succedere due cose:

- *Due funzioni completamente diverse:*



- *Due funzioni che sembrano diverse ma sono la stessa funzione:*



Sfrutto una proprietà dall'analisi matematica per cui ogni numero reale può essere scritto come:

$$\forall\ z\ \exists\ n\ \mid\ z = p_0^n$$

Quindi per $p = p_0$ posso scrivere:

$$g(z) - C\ log_2 \frac{1}{z} = 0$$

Uguale a 0 in quanto $C = \frac{g(p_0)}{\log_2 \frac{1}{p_0}}$ annulla l'equazione sopra

$$g(z) = C\ log_2 \frac{1}{z}$$

Quindi questa funzione si dimostra essere un logaritmo

Quindi abbiamo definito l'unica funzione possibile per definire la quantità di informazione per un evento.

Consideriamo una sorgente S che emette q simboli con $p_1, p_2, ..., p_q$. Abbiamo quindi detto che:

$$I(s_i) = I(p_i) = log_r \frac{1}{p_i}$$

Quanto è la quantità di informazione media data una sorgente? Chiamo questo valore come $H(s)$. Per calcolare questo valore peso ogni quantità di informazione:

$$H(s) = p_1 I(s_1) + p_2 I(s_2) + ... + p_q I(s_q)$$

Riscriviamo in forma compatta:

$$H(s) = \sum_{i=1}^{q} p_i I(s_i)$$

$$H(s) = \sum_{i=1}^{q} p_i \log \frac{1}{p_i}$$

oppure scritto come

$$H(s) = -\sum_{i=1}^{q} p_i \log p_i$$

Questa formula appena scritta è l'**entropia** di una sorgente ed è definita come:

$$H_r(s) = \sum_{i=1}^{q} p_i \log_r \frac{1}{p_i}$$

L'unità di misura dell'entropia è data dalla base del logaritmo,

$$H_r(s) = H_2(s) \log_r 2$$

Ma posso cambiare velocemente base tramite una costante.

Se una sorgente fa uscire messaggi lunghi N con simboli s_q, mi aspetterò di avere

- $N \cdot p_1$ occorrenze di s_1
- $N \cdot p_2$ occorrenze di s_2
- ..
- $N \cdot p_q$ occorrenze di s_q

La probabilità P di avere un messaggio fatto in questo modo sarà:

$$P = p_1^{Np_1} \cdot p_2^{Np_2} \cdot ... \cdot p_q^{Np_q} = [p_1^{p_1} p_2^{p_2} ... p_q^{p_q}]^N$$

Quando esce questo messaggio, la quantità di informazione che otteniamo è $\log \frac{1}{p}$, che è quindi:

$$\log \frac{1}{p} = \log \frac{1}{[p_1^{p_1} p_2^{p_2} ... p_q^{p_q}]^N} = \log[\frac{1}{p_1^{p_1} p_2^{p_2} ... p_q^{p_q}}]^N = N \log[\frac{1}{p_1^{p_1}} \cdot \frac{1}{p_2^{p_2}} \cdot ... \cdot \frac{1}{p_q^{p_q}}]$$

$$= N \sum_{i=1}^{q} \log[\frac{1}{p_i}]^{p_i} = N \sum_{i=1}^{q} p_i \log \frac{1}{p_i}$$

Quindi l'entropia possiamo vederla come la quantità di informazione mediamente emessa dalla sorgente:

$$H(s) = \frac{\log \frac{1}{p}}{N}$$

Vediamo ora come è fatta questa funzione entropia:

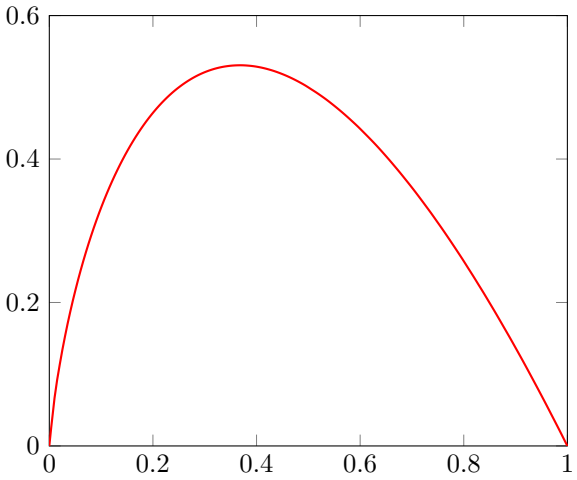


Figure 12: Grafico funzione $H_2(s)$

In $p = 0$ abbiamo un punto di discontinuità a cui ci si avvicina con pendenza infinita, quindi definiamo formalmente un'altra funzione per cui se $p = 0$ allora la funzione vale 0 (un evento impossibile non ci da informazione).

Il punto massimo vale $\frac{1}{e}$.

L'entropia $H(s)$ è sempre un valore ≥ 0.

Quando vale 0? Essendo una somma di termini ≥ 0, allora essa si annulla solo se tutti i termini sono uguali a 0, che è il caso in cui la sorgente ha come probabilità $p_1 = 1, p_2 = 0, p_3 = 0, ...p_q = 0$, quindi quando esce sempre lo stesso simbolo.

Lezione 10