Teoria dell'Informazione/RTI **Anno Accademico 2022–2023**

Lezione 15

Ugo Vaccaro

Nelle prime lezioni abbiamo provato il seguente risultato fondamentale:

Teorema 1 *Per ogni codifica binaria* $\mathsf{c} : \mathsf{X} \to \{0,1\}^*$ *UD dei simboli di una sorgente DSSM* X*, detto* $\ell(x)$ *il numero di bit della codifica* $\mathsf{c}(x)$ *del generico simbolo* $x \in \mathsf{X}$*, e denotato con* $P = \{P(x) : x \in \mathsf{X}\}$ *la distribuzione di probabilità in accordo alla quale la sorgente emette i simboli in* X*, vale che la lunghezza media della codifica* c *soddisfa la diseguaglianza*

$$\sum_{x \in \mathsf{X}} P(x)\ell(x) \geq H(P) = -\sum_{x \in \mathsf{X}} P(x) \log P(x).$$

Inoltre, esiste una codifica binaria $\mathsf{c} : \mathsf{X} \to \{0,1\}^*$ *prefisso (e quindi UD) dei simboli della sorgente la cui lunghezza media soddisfa la diseguaglianza*

$$\sum_{x \in \mathsf{X}} P(x)\ell(x) < H(P) + 1.$$

Il Teorema 1 è stato dimostrato sotto l'ipotesi che la distribuzione di probabilità $P = \{P(x) : x \in \mathsf{X}\}$, in accordo alla quale la sorgente X emette i simboli, sia nota. Cosa succede se della P ne abbiamo una "stima" Q? In altri termini, supponiamo di codificare la sorgente X sotto l'ipotesi che essa emetta simboli in accordo a $Q = \{Q(x) : x \in \mathsf{X}\}$ (che può essere anche diversa da P) e cerchiamo di capire quanto possiamo peggiorare rispetto al Teorema 1. A tal fine, ricordiamo (ancora una volta) la Diseguaglianza di Gibbs, che ci dice che la Divergenza Informazionale è sempre non negativa, ovvero che

$$D(P||Q) = \sum_{x \in X}^{n} P(x) \log \frac{P(x)}{Q(x)} \geq 0. \tag{1}$$

Supponiamo ora che codifichiamo la sorgente X, che emette i simboli in accordo a $P = \{P(x) : x \in \mathsf{X}\}$, immaginando (*erroneamente!*) che essa emetta invece i somboli in accordo a $Q = \{Q(x) : x \in \mathsf{X}\}$. In altri termini, invece di usare parole codice di lunghezza pari a $\left\lceil \log \frac{1}{P(x)} \right\rceil$, useremmo parole codice di lunghezza $\ell(x) = \left\lceil \log \frac{1}{Q(x)} \right\rceil$. Cosa accadrà al numero medio di simboli codice per simbolo sorgente $\sum_{x \in \mathsf{X}} P(x)\ell(x) = \sum_{x \in \mathsf{X}} P(x) \left\lceil \log \frac{1}{Q(x)} \right\rceil$ che useremo? Consideriamo per semplicità solo il caso in cui codifichiamo singoli simboli sorgente alla volta. Avremo che

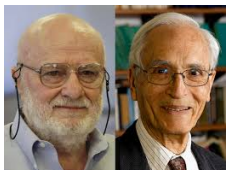
$$\begin{aligned}
\sum_{x \in \mathsf{X}} P(x)\ell(x) &= \sum_{x \in \mathsf{X}} P(x) \left\lceil \log \frac{1}{Q(x)} \right\rceil \\
&\geq \sum_{x \in \mathsf{X}} P(x) \log \frac{1}{Q(x)} \\
&= \sum_{x \in \mathsf{X}} P(x) \log \frac{P(x)}{Q(x)P(x)} \\
&= \sum_{x \in \mathsf{X}} P(x) \log \frac{1}{P(x)} + \sum_{x \in \mathsf{X}} P(x) \log \frac{P(x)}{Q(x)} \\
&= H(P) + D(P||Q).
\end{aligned}$$

D'altra parte

$$\begin{aligned}\sum_{x\in\mathsf{X}} P(x)\ell(x) &= \sum_{x\in\mathsf{X}} P(x)\left\lceil \log\frac{1}{Q(x)}\right\rceil \\ &< \sum_{x\in\mathsf{X}} P(x)\left(\log\frac{1}{Q(x)}+1\right) \\ &= \sum_{x\in\mathsf{X}} P(x)\log\frac{P(x)}{Q(x)P(x)}+1 \\ &= \sum_{x\in\mathsf{X}} P(x)\log\frac{1}{P(x)}+\sum_{x\in\mathsf{X}} P(x)\log\frac{P(x)}{Q(x)}+1 \\ &= H(P)+D(P||Q)+1.\end{aligned}$$

La morale è che migliore sarà la nostra stima $Q=\{Q(x):x\in\mathsf{X}\}$ di $P=\{P(x):x\in\mathsf{X}\}$ (ovvero, più vicina alla distribuzione P sarà Q, secondo la divergenza $D(P||Q)$) e più vicina all'entropia $H(P)$ sarà la lunghezza media del nostro codice, come era intuitivo aspettarci.

Molti algoritmi per la codifica di sorgenti che si usano in pratica richiedono la conoscenza *a priori* delle probabilità/frequenze con cui i caratteri sorgenti appaiono nel testo da codificare. Come appena osservato, se la conoscenza delle probabilità di emissione dei simboli, da parte della sorgente, non sono "corrette", potremmo non ottenere buoni risultati. Quando non si dispongono informazioni a priori sulla distribuzione di probabilità con cui la sorgente emette i simboli, in pratica si preferiscono usare altri algoritmi.



Uno di questi, che và sotto il nome di Algoritmo di Lempel-Ziv, dal nome degli scopritori, opera senza richiedere alcuna conoscenza *a priori* sulla statistica delle sorgente. Ciononostante, si può provare che, sotto ipotesi molto ragionevoli, esso produce una codifica per cui il numero di bit usati per ogni simbolo sorgente converge all'entropia della sorgente, al crescere della lunghezza della sequenza da codificare. In più, ammette un'implementazione molto efficiente, per cui esso è alla base di molti algoritmi di compressione dati commerciali. L'algoritmo procede costruendo un dizionario di sottostringhe, che chiameremo *frasi*, che appaiono nel testo da comprimere. La costruzione del dizionario avviene sequenzialmente, e non vi è bisogno di conoscere l'intero testo prima di procedere alla sua compressione. L'algoritmo suddivide la sequenza da comprimere in frasi distinte individuando (ed inserendo nel dizionario) sempre *la più corta frase non ancora incontrata*. Supponiamo ad esempio di avere la stringa sull'alfabeto $\{A,B\}$

$$AABABBBABAABABBBABBABB$$

da comprimere. Iniziamo ad individuare la più corta frase sulla sinistra che non abbiamo mai incontrato. Essa sarà sempre composta da una singola lettera, nel caso in questione sarà A. Per facilitare la comprensione dell'algoritmo, inseriamo il delimitatore | per individuare le frasi. Per cui avremo

$$A|ABABBBABAABABBBABBABB.$$

Adesso individuiamo la prossima frase non ancora incontrata. Poichè A è stata già incontrata, la nuova frase sarà AB, per cui la suddivisione sarà

$$A|AB|ABBBABAABABBBABBABB.$$

La nuova frase che non abbiamo mai ancora incontrato sarà ABB, dato che AB è stata già incontrata, e la prossima ancora sarà B, per cui avremo la suddivisione

$$A|AB|ABB|B|ABAABABBBABBABB.$$

Procedendo in questo modo la suddivisione della sequenza sull'alfabeto $\{A, B\}$ da comprimere sarà

$$A|AB|ABB|B|ABA|ABAB|BB|ABBA|BB.$$

L'ultima frase è non veramente nuova, ma poichè siamo arrivati alla fine della sequenza, ciò è inevitabile. L'algoritmo crea anche una tabella in cui vengono inserite le frasi, nell'ordine in cui esse vengono incontrate per la prima volta. Nel nostro caso la tabella sarebbe

0	λ
1	A
2	AB
3	ABB
4	B
5	ABA
6	ABAB
7	BB
8	ABBA

dove con λ intendiamo la sequenza vuota. La codifica di ciascuna frase α (tranne l'eventuale ultima, se essa non è nuova) avviene mediante la coppia (i, x), (la virgola nella coppia è solo per comodità di lettura nostra, nell'output dell'algoritmo non viene utilizzata) dove i è l'indice di riga della tabella in cui è contenuta la frase precedente β per cui $\alpha = \beta x$. Nell'esempio che stiamo sviluppando, avremmo la seguente sitazione

1	2	3	4	5	6	7	8	9
A	AB	ABB	B	ABA	ABAB	BB	ABBA	BB
A	1B	2B	0B	2A	5B	4B	3A	7

A questo punto codifichiamo il numero i in binario e la lettera A con 0 e la lettera B con 1 (se avessimo una stringa su di un alfabeto con k lettere, ogni lettera andrebbe ovviamente codificata con $\lceil \log k \rceil$ bit). Pertanto, la codifica che otterremmo sarebbe (di nuovo inseriamo delimitatori | e virgole ',' per comodità di lettura)

$$,0|1,1|10,1|00,1|010,0|101,1|100,1|011,0|0111$$

e la codifica binaria "vera" e finale sarà

$$011101001010010111001011001 11.$$

Notiamo che abbiamo codificato l'intero 2 (che indirizza alla seconda frase) una volta con 10, e la seconda volta con 010. In altri termini, appena vi è la necessità di usare $\log k$ bit per indirizzare una frase (ovvero, vi è la necessità di indirizzare una frase che è in una posizione compresa tra la riga $2^{k-1} + 1$ e 2^k della tabella, i successivi indirizzamenti dovranno usare almeno $\log k$ bit, anche se l'indice della riga della matrice che vogliamo indirizzare è $\leq 2^{k-1}$).

Questa crescita nel numero di bit è necessaria per permettere una decodifica non ambigua. Infatti, se ciò non fosse, nel leggere 10 non sapremmo se esso è un indirizzamento alla seconda riga della tabella o esso è l'inizio di 100, che rappresenta un indirizzamento alla quarta riga della tabella.

La decodifica (ovvero il passaggio dalla sequenza binaria alla sequenza di partenza) è altrettanto agevole. Il decodificatore inserisce il primo delimitatore | dopo un bit, il secondo delimitatore | dopo due bit, poi inserisce due delimitatori dopo due successive sequenze di 3 bit, poi 2^2 delimitatori dopo ciascuna sequenza di 4 bit, e

così via... Otterremo quindi, una sottosequenza di un bit, una di 2 bit, 2 sottosequenza lunghe 3 bit, 2^2 lunghe 4 bit, 2^3 lunghe 5 bit, 2^4 lunghe 6 bit, ed in generale 2^k di lunghezza $k+2$. Nel nostro esempio, partendo da 011101001010010111001011001111 otterremmo

$$0|11|101|001|0100|1011|1001|0110|0111.$$

A questo punto, ricordando che l'ultimo bit di ogni sottosequenza, ovvero quello immediatamente prima di un delimitatore | denota la codifica di A o B, e che la prima parte della sottosequenza codifica invece la riga della tabella, possiamo inserire virgole ",", ed ottenere

$$,0|1,1|10,1|00,1|010,0|101,1|100,1|011,0|0111.$$

Da questa separazione, ed usando la tabella, otteniamo

$$A|AB|ABB|B|ABA|ABAB|BB|ABBA|BB$$

ovvero la sequenza di partenza.

Vogliamo ora studiare l'efficienza della codifica. Sia n il numero di caratteri nella sequenza di partenza da codificare in binario, e denotiamo con $c(n)$ il numero di frasi distinte (ovvero il numero di righe della tabella) prodotte dall'algoritmo sulla sequenza in input. Ricordiamo che noi codifichiamo ciascuna frase α con la coppia (i,x), dove i è un indice di riga della tabella e x è un carattere della sequenza da codificare. Il numero di bit necessari per codificare in binario ciascuna coppia (i,x) sarà quindi pari a $\log c(n) + \log|\mathsf{X}|$, dove X è come al solito l'alfabeto sorgente. Supponiamo per semplicità che $|\mathsf{X}| = 2$, per cui il numero di bit necessari per codificare in binario ciascuna coppia (i,x) sarà pari a $\log c(n)+1$. Di conseguenza, il numero di bit *totale* per rappresentare la sequenza in input, composta da n caratteri, sarà al più

$$c(n)(\log c(n) + 1).$$

Il parametro cui siamo interessati è il solito, ovvero

$$\frac{\#\text{bit usati}}{\#\text{caratteri della sequenza input}} = \frac{c(n)(\log c(n)+1)}{n}.$$

Vogliamo provare che

$$\lim_{n\to\infty} \frac{c(n)(\log c(n)+1)}{n} = H(X),$$

dove $H(X)$ è la solita entropia della sorgente. Notiamo, ancora una volta, che l'algoritmo *non* conosce le probabilità con cui la sorgente emette A o B. Il primo risultato che proveremo è il seguente.

Lemma 1 *Sia $c(n)$ il numero di frasi distinte (ovvero il numero di righe della tabella) prodotte dall'algoritmo sulla sequenza in input. Vale che*

$$c(n) \leq \frac{n}{(1-\epsilon_n)\log n},$$

dove ϵ_n tende a 0 al crescere di n.

Dimostrazione. Fissato un intero k, calcoliamo la somma n_k delle lunghezze di tutte le possibili stringhe binarie di lunghezza al più k. Chiaramente, ci sono 2 stringhe binarie lunghe 1, 4 stringhe binarie lunghe 2, 8 stringhe binarie lunghe 3, ..., 2^j stringhe binarie lunghe j, per cui varrà

$$n_k = 1\times 2 + 2\times 2^2 + 3\times 2^3 + \ldots = \sum_{j=1}^{k} j2^j = (k-1)2^{k+1} + 2.$$

Il numero di frasi distinte in una sequenza di lunghezza n è massimo quando tutte le frasi sono le più corte possibili. Se la lunghezza n della sequenza è proprio uguale a n_k, ciò accadrà quando tutte le frasi sono lunghe al più k, per cui

$$c(n_k) \leq 2 + 2^2 + 2^3 + \ldots = \sum_{j=1}^{k} 2^j = 2^{k+1} - 2 < 2^{k+1} = \frac{n_k - 2}{k-1} \leq \frac{n_k}{k-1}.$$

Osserviamo ora che, per ogni valore di n esisterà sicuramente un valore di k per cui

$$n_k \leq n < n_{k+1}.$$

Poniamo $n = n_k + \Delta$. Poichè è facile verificare che $n_{k+1} - n_k = (k+1)2^{k+1}$, ne segue che $\Delta < (k+1)2^{k+1}$. L'algoritmo che suddivide la sequenza sorgente in frasi, sulla sequenza input lunga n, produrà quindi un numero di frasi distinte $c(n)$ pari al più a $c(n_k) + c(\Delta)$. Nella parte della sequenza input lunga Δ *tutte* le frasi sono lunghe almeno $k+1$, per cui $c(\Delta) \leq \Delta/(k+1)$. Mettendo tutto insieme, e ricordando che $c(n_k) \leq n_k/(k-1)$ otteniamo che

$$c(n) \leq \frac{n_k}{k-1} + \frac{\Delta}{k+1} \leq \frac{n_k + \Delta}{k-1} = \frac{n}{k-1}.$$

Dal fatto che $n_k \leq n < n_{k+1}$ otteniamo che

$$n \geq n_k = (k-1)2^{k+1} + 2 \geq 2^k \Rightarrow k \leq \log n.$$

Mentre

$$n < n_{k+1} = k2^{k+2} + 2 \leq (k+2)2^{k+2} \leq (\log n + 2)2^{k+2} \Rightarrow k + 2 > \log \frac{n}{\log n + 2}.$$

Sottraendo 3 ad entrambi i membri della precedente diseguaglianza, otteniamo che per ogni $n \geq 4$ vale

$$\begin{aligned} k - 1 &\geq \log n - \log(\log n + 2) - 3 \\ &= \big(1 - \frac{\log(\log n + 2) + 3}{\log n}\big) \log n \\ &= (1 - \epsilon_n) \log n \end{aligned}$$

avendo posto

$$\epsilon_n = \frac{\log(\log n + 2) + 3}{\log n},$$

che chiaramente và a zero al crescere di n. Ricordando infine che avevamo provato $c(n) \leq n/(k-1)$, dalla limitazione inferiore a $k-1$ appena trovata, ne deduciamo che

$$c(n) \leq \frac{n}{k-1} \leq \frac{n}{(1-\epsilon_n)\log n},$$

completando quindi la dimostrazione del Lemma. □

Passiamo ora alla prova del risultato principale di questa lezione, ovvero che

$$\lim_{n\to\infty} \frac{c(n)(\log c(n) + 1)}{n} = H(X). \tag{2}$$

Proveremo la (2) attraverso una serie di risultati preliminari.

Lemma 2 *Sia X una variabile casuale che assume valori in $\{1, \ldots, n\}$ con probabilità $p_1, \ldots, p_n$, rispettivamente. Sia $\mu = E[X] = \sum_{i\geq 1} ip_i$ il suo valor medio. Allora vale che*

$$H(X) = \sum_{i\geq 1} p_i \log \frac{1}{p_i} \leq \mu \log \mu - (\mu - 1)\log(\mu - 1).$$

Dimostrazione. Sia α un numero reale, arbitrario. Calcoliamo innanzitutto la differenza tra $H(X)$ e $\alpha\mu$. Si ha

$$\begin{aligned}
H(X) - \alpha\mu &= \sum_{i\geq 1} p_i \log \frac{1}{p_i} - \sum_{i\geq 1} \alpha i p_i \\
&= \sum_{i\geq 1} p_i \log \frac{1}{p_i} + \sum_{i\geq 1} (-\alpha i) p_i \\
&= \sum_{i\geq 1} p_i \log \frac{1}{p_i} + \sum_{i\geq 1} p_i \log 2^{-\alpha i} \\
&= \sum_{i\geq 1} p_i \log \frac{2^{-\alpha i}}{p_i} \leq \log \sum_{i\geq 1} 2^{-\alpha i} \qquad \text{(dalla diseguaglianza di Jensen)} \\
&= \log \sum_{i\geq 1} \left(\frac{1}{2^\alpha}\right)^i \leq \log\left(\frac{2^\alpha}{2^\alpha - 1} - 1\right) \qquad \text{(estendendo la somma per } i = 1, \ldots, \infty) \\
&= \log\left(\frac{1}{2^\alpha - 1}\right),
\end{aligned}$$

ovvero

$$H(X) \leq \alpha\mu + \log\left(\frac{1}{2^\alpha - 1}\right).$$

Poniamo ora $\alpha = \log \frac{\mu}{\mu-1}$. Otteniamo, sostituendo tale valore di α nella diseguaglianza di sopra,

$$\begin{aligned}
H(X) &\leq \mu \log \frac{\mu}{\mu - 1} + \log\left(\frac{1}{\frac{\mu}{\mu-1} - 1}\right) \\
&= \mu \log \frac{\mu}{\mu - 1} + \log \frac{1}{\frac{\mu - \mu + 1}{\mu - 1}} = \mu \log \frac{\mu}{\mu - 1} + \log(\mu - 1) \\
&= \mu \log \mu - (\mu - 1)\log(\mu - 1),
\end{aligned}$$

il che conclude la dimostrazione del Lemma.

□

Ricordiamo ora un'osservazione effettuata all'inizio del corso. Data una generica sequenza $\mathbf{x} = (x_1 \ldots x_n)$ emessa da una sorgente discreta, stazionaria e senza memoria, si ha che $p(\mathbf{x}) = p(x_1 \ldots x_n) = \prod_{i=1}^n p(x_i)$ e quindi $\log p(\mathbf{x}) = \sum_{i=1}^n \log p(x_i)$. D'altra parte, per la legge dei grandi numeri la media aritmetica $-\frac{1}{n}\sum_{i=1}^n \log p(x_i)$ converge (*in probabilità*) al valor medio della generica variabile casuale che assume valore $-\log p(x)$ con probabilità $p(x)$, e questo valor medio $\sum_x p(x)(-\log p(x)) = \sum_x p(x)\log(1/p(x))$ altro non è che l'entropia $H(X)$ della sorgente. Riassumendo, possiamo dire che vale il seguente risultato, che poniamo sotto forma di Lemma, per futura referenza.

Lemma 3 *Per ogni sequenza sorgente* $\mathbf{x} = (x_1 \ldots x_n)$ *vale che*

$$\lim_{n\to\infty} -\frac{1}{n} \log p(x_1 \ldots x_n) = H(X) \qquad \text{(in probabilità)}.$$

Un'altro risultato preliminare è il seguente.

Lemma 4 *Per ogni sequenza sorgente* $\mathbf{x} = (x_1 \ldots x_n)$ *vale che*

$$\log p(x_1 \ldots x_n) \leq c(n) H(Z) - c(n) \log c(n),$$

dove

1. *$c(n)$ è il numero di frasi distinte prodotte dall'algoritmo di Lempel e Ziv, quando esso ha $(x_1 \dots x_n)$ in input;*
2. *$\ell_{\max}$ è la lunghezza della più lunga frase prodotta dall'algoritmo di Lempel e Ziv, quando esso ha $(x_1 \dots x_n)$ in input, e c_i è il numero di frasi distinte di lunghezza i prodotte dall'algoritmo,*
3. *Z è la variabile casuale che assume valori $i = 1, \dots, \ell_{\max}$, con probabilità $P\{Z = i\} = \frac{c_i}{c(n)} = \pi_i$.*

Dimostrazione. Data la sequenza $(x_1 \dots x_n)$ emessa dalla sorgente, sia $s_1 \dots s_{c(n)}$ la sua suddivisione in frasi, così come prodotta dall'algoritmo. Denotiamo con $\ell(s)$ la lunghezza di una generica frase s e con c_ℓ il numero di frasi di una data lunghezza ℓ, con $\ell \in \{1, \dots, \ell_{max}\}$. Ovviamente, varà che

$$\sum_{s:\ell(s)=\ell} \frac{1}{c_\ell} = 1.$$

Ricordando che la sorgente è senza memoria, otteniamo

$$\begin{aligned}
\log p(x_1, \dots x_n) &= \log p(s_1 \dots s_{c(n)}) = \log \prod_{i=1}^{c(n)} p(s_i) \\
&= \sum_{i=1}^{c(n)} \log p(s_i) = \sum_{\ell=1}^{\ell_{\max}} \left[\sum_{s:\ell(s)=\ell} \log p(s) \right] && \text{(raggruppando le frasi della stessa lunghezza } \ell) \\
&= \sum_{\ell=1}^{\ell_{\max}} c_\ell \left[\sum_{s:\ell(s)=\ell} \frac{1}{c_\ell} \log p(s) \right] && \text{(moltiplicando e dividendo per } c_\ell) \\
&\le \sum_{\ell=1}^{\ell_{\max}} c_\ell \log \sum_{s:\ell(s)=\ell} \frac{1}{c_\ell} p(s) && \text{(applicando la dis. di Jensen alla funzione logaritmo)} \\
&= \sum_{\ell=1}^{\ell_{\max}} c_\ell \log \frac{1}{c_\ell} \sum_{s:\ell(s)=\ell} p(s) \le \sum_{\ell=1}^{\ell_{\max}} c_\ell \log \frac{1}{c_\ell} && \text{(poichè } \textstyle\sum_{s:\ell(s)=\ell} p(s) \le 1 \text{)} \\
&= \sum_{\ell=1}^{\ell_{\max}} c_\ell \log \frac{c(n)}{c(n) c_\ell} = \sum_{\ell=1}^{\ell_{\max}} c_\ell \log \frac{c(n)}{c_\ell} - \sum_{\ell=1}^{\ell_{\max}} c_\ell \log c(n) \\
&= \sum_{\ell=1}^{\ell_{\max}} c_\ell \log \frac{c(n)}{c_\ell} - c(n) \log c(n) && \text{(poichè } \quad \textstyle\sum_{\ell=1}^{\ell_{\max}} c_\ell = c(n))
\end{aligned}$$

Continuando, e ricordando la definizione della variabile casuale Z, otteniamo

$$\log p(x_1, \dots x_n) \le \sum_{\ell=1}^{\ell_{\max}} c_\ell \log \frac{c(n)}{c_\ell} - c(n) \log c(n) \tag{3}$$

$$= c(n) \sum_{\ell=1}^{\ell_{\max}} \frac{c_\ell}{c(n)} \log \frac{c(n)}{c_\ell} - c(n) \log c(n) \tag{4}$$

$$= c(n) \sum_{\ell=1}^{\ell_{\max}} \pi_\ell \log \frac{1}{\pi_\ell} - c(n) \log c(n) \tag{5}$$

$$= c(n) H(Z) - c(n) \log c(n), \tag{6}$$

il che conclude la dimostrazione del Lemma.

□

Lemma 5 *Con le stesse notazioni del Lemma 4, detta Z la variabile casuale che assume valori $i = 1, \ldots, \ell_{\max}$, con probabilità $P\{Z = i\} = \frac{c_i}{c(n)} = \pi_\ell$ vale che*

$$H(Z) \leq \log \frac{n}{c(n)} - \frac{n - c(n)}{c(n)} \log \frac{n - c(n)}{n}.$$

Dimostrazione. Osserviamo che, dalla definizione di Z, il suo valor medio $E[Z]$ è pari a

$$E[Z] = \sum_{\ell=1}^{\ell_{max}} \ell \pi_\ell = \sum_{\ell=1}^{\ell_{max}} \ell \frac{c_\ell}{c(n)} = \frac{n}{c(n)},$$

in quanto ovviamente vale che $\sum_{\ell=1}^{\ell_{max}} \ell c_\ell = n$. Poniamo $E[Z] = \mu$ ed applichiamo il Lemma 2 alla variabile casuale Z. Per comodità di notazione, poniamo $c = c(n)$ Otteniamo

$$\begin{aligned}
H(Z) &\leq \mu \log \mu - (\mu - 1) \log(\mu - 1) = \frac{n}{c} \log \frac{n}{c} - \left(\frac{n}{c} - 1\right) \log \left(\frac{n}{c} - 1\right) \\
&= \frac{n}{c} \log \frac{n}{c} - \left(\frac{n}{c} - 1\right) \log \left(\frac{n - c}{c}\right) = \frac{n}{c} \log \frac{n}{c} - \left(\frac{n}{c} - 1\right) \log \left(\frac{n}{n} \cdot \frac{n - c}{c}\right) \\
&= \frac{n}{c} \log \frac{n}{c} - \left(\frac{n}{c} - 1\right) \left[\log \frac{n}{c} + \log \frac{n - c}{n}\right] \\
&= \frac{n}{c} \log \frac{n}{c} - \left(\frac{n}{c} - 1\right) \log \frac{n}{c} - \left(\frac{n}{c} - 1\right) \log \frac{n - c}{n} \\
&\log \frac{n}{c} - \left(\frac{n}{c} - 1\right) \log \frac{n - c}{n} = \log \frac{n}{c} - \frac{n - c}{c} \log \frac{n - c}{n}.
\end{aligned}$$

□

Come conseguenza del Lemma 5, abbiamo che

$$\frac{c(n)}{n} H(Z) \leq \frac{c(n)}{n} \log \frac{n}{c(n)} - \frac{c(n)}{n} \frac{n - c(n)}{c(n)} \log \frac{n - c(n)}{n} \tag{7}$$

$$= \frac{c(n)}{n} \log \frac{n}{c(n)} + \frac{n - c(n)}{n} \log \frac{n}{n - c(n)}. \tag{8}$$

Ricordiamo che abbiamo già provato che

$$c(n) \leq \frac{n}{(1 - \epsilon_n) \log n},$$

dove ϵ_n tende a 0 al crescere di n. Ciò vuol dire essenzialmente che

$$c(n) = \frac{n}{\log n} + \alpha_n, \tag{9}$$

con α_n che tende a 0 al crescere di n. Per cui, sostituendo l'espressione (9) di $c(n)$ nel primo termine della (7) otteniamo che

$$\lim_{n \to \infty} \frac{c(n)}{n} \log \frac{n}{c(n)} = \lim_{n \to \infty} \frac{\log \log n}{\log n} = 0.$$

Analogamente, considerando il secondo termine della (7) otteniamo che

$$\frac{n - c(n)}{n} \log \frac{n}{n - c(n)} = -\left(1 - \frac{c(n)}{n}\right) \log \left(1 - \frac{c(n)}{n}\right),$$

e poichè evidentemente

$$\lim_{n \to \infty} \frac{c(n)}{n} = 0,$$

vale anche che

$$\lim_{n\to\infty} -\left(1-\frac{c(n)}{n}\right)\log\left(1-\frac{c(n)}{n}\right)=0.$$

Tutto ciò ci permette di concludere che

$$\lim_{n\to\infty}\frac{c(n)}{n}H(Z)=0. \tag{10}$$

Partiamo ora dal risultato del Lemma 4. Otteniamo che

$$-\frac{1}{n}\log p(x_1\dots x_n)\geq\frac{c(n)\log c(n)}{n}-\frac{c(n)}{n}H(Z). \tag{11}$$

Sappiamo che

$$\lim_{n\to\infty} -\frac{1}{n}\log p(x_1\dots x_n)=H(X), \qquad \text{(in probabilità)}$$

e che

$$\lim_{n\to\infty}\frac{c(n)}{n}H(Z)=0.$$

Otteniamo quindi

$$H(X)\geq\lim_{n\to\infty}\frac{c(n)(\log c(n)+1)}{n}.$$

Poichè il numero di bit per simbolo sorgente di una codifica U.D. (che compare al membro destro della precedente diseguaglianza) *non* può essere inferiore all'entropia $H(X)$ della sorgente otteniamo, *finalmente*, la (2).