Appunti di Teoria dell'Informazione

Filippo Mineo

9 maggio 2012

Indice

Per eventuali suggerimenti o correzioni scrivetemi a fsttesla@hotmail.it.

Dati tecnici

Corso di laurea:	Informatica magistrale
Nome completo:	*Teoria dell'Informazione, Crittografia e Complessità*
Docente:	Prof. Agostino Dovier
CFU:	9
Modalità di esame:	Orale, su appuntamento

1 Codici di sorgente

L'obiettivo è codificare un'informazione generata da un testo (sorgente) nel modo più "compatto" possibile, senza preoccuparci di errori o nemici, quindi lo scopo è la *compressione*.

Avremo due alfabeti: $\mathcal{A} = \{a_1, \dots, a_k\}$ primario (del testo) con k lettere e $\mathcal{B} = \{b_1, \dots, b_D\}$ secondario (del codice di sorgente) con D lettere.

Le famiglie di codici di sorgente sono 4: B-LV, LV-B, B-B, LV-LV. «B» sta per «blocco» e «LV» sta per «lunghezza variabile». Le sigle della coppia indicano la corrispondenza tra elementi del testo ed elementi del codice di sorgente. Ad esempio l'ASCII è B-B, mentre il codice Morse è B-LV.

Un'idea chiave per comprimere è sfruttare la probabilità di emissione delle lettere da parte della sorgente.

1.1 B-LV

Definizione 1.1.

- Un codice B-LV è una funzione $\varphi\colon \mathcal{A} \to \mathcal{B}^+$. Gli elementi di $\varphi(\mathcal{A})$ si dicono PAROLE DI CODICE.
- Un codice φ può essere esteso alle stringhe con la $\hat{\varphi}\colon \mathcal{A}^* \to \mathcal{B}^*$, $\hat{\varphi}(a_1 \cdots a_n) \coloneqq \varphi(a_1) \cdots \varphi(a_n)$.
- Un codice φ si dice UNIVOCAMENTE DECODIFICABILE (UD) se e solo se $\hat{\varphi}$ è iniettiva.
- Un codice si dice A PREFISSO se nessuna sua parola di codice è prefisso di un'altra.
- Il RITARDO DI DECODIFICA è il numero massimo di caratteri aggiuntivi del codice di sorgente da leggere per compiere un "passo di decodifica". Un codice si dice ISTANTANEAMENTE DECODIFICABILE se il ritardo di decodifica è nullo.

Proposizione 1.2. Se un codice è a prefisso (dunque anche UD) allora è istantaneamente decodificabile.

Esistono i codici UD non a prefisso in cui il ritardo di decodifica non è troppo svantaggioso rispetto alla maggiore compressione che offrono, se confrontati con codici UD a prefisso equivalenti.

D'ora in poi indichiamo con «ℓ_i» il valore $|\varphi(a_i)|$, con «ℓ» la massima tra tali lunghezze e con «α» il valore $\sum_{i=1}^{k} D^{-\ell_i}$.

Teorema 1.3: Disuguaglianza di Kraft-McMillan. Se un codice è UD allora $\sum_{i=1}^{k} D^{-\ell_i} \leq 1$.

Dimostrazione. Sia «$N(n,h)$» il numero di stringhe di $\mathcal{A}^*$ di lunghezza n a cui corrisponde una stringa di $\mathcal{B}^*$ di lunghezza h. Se un codice è UD allora $N(n,h) \leq D^h$.

α^n è formata da n^k addendi, in cui il generico addendo è nella forma $D^{-(\ell_{j_1}+\cdots+\ell_{j_n})}$, con $j_i \in \{1, \dots, k\}$ indici. In α ci possono essere addendi di valore uguale, infatti basta che abbiano la stessa somma come esponente. È come se con α osservassimo tutte le stringhe di $\mathcal{A}^n$. Allora possiamo scrivere che

$\forall n\colon \alpha^n = \sum_{i=1}^{n\ell} \underbrace{N(n,i)}_{\leq D^i} D^{-i} \leq \sum_{i=1}^{n\ell} D^i D^{-i} \Rightarrow \forall n\colon \alpha^n \leq n\ell \Rightarrow \alpha \leq 1$, in cui l'ultimo passaggio discende dalle proprietà dell'esponenziale.

Questo teorema ha come conseguenze la sufficienza dei codici a prefisso (Lemma 1.4) e il Teorema di Shannon (Teorema 1.9).

Lemma 1.4. Dato un codice UD, ne esiste uno a prefisso con la stessa lunghezza.

Dimostrazione. Per semplicità assumiamo che $\ell_1 \leq \cdots \leq \ell_n$, senza perdita di generalità.

Consideriamo l'albero D-ario di lunghezza ℓ completo. Partendo da $i = 1$ e proseguendo fino a $i = n$, seguo l'albero per ℓ_i passi scegliendo sempre il ramo libero più alto e al termine del cammino pongo l'etichetta «a_i».

Se il codice era UD lo spazio è sufficiente, infatti per ogni a_i si usano $D^{\ell-\ell_i}$ nodi, per un totale di $D^\ell \cdot \underbrace{\sum_{i=1}^{n} D^{-\ell_i}}_{\leq 1}$ nodi.

Siano φ_1 e φ_2 codici a prefisso per lo stesso alfabeto primario $\mathcal{A}$. Qual è il migliore? Qui entra in gioco la PROBABILITÀ DI EMISSIONE di un simbolo, che indicheremo con «p_i». A ogni alfabeto è quindi associata una DISTRIBUZIONE DI PROBABILITÀ $P \coloneqq (p_1, \ldots, p_k)$.

Definizione 1.5. La LUNGHEZZA MEDIA di un codice è $\mathrm{EL} \coloneqq \sum_{i=1}^{k} p_i \ell_i$.

Naturalmente vogliamo che la lunghezza media sia minore possibile.

Definizione 1.6. L'ENTROPIA di una distribuzione probabilità P è $H(P) \coloneqq -\sum_{i=1}^{k} p_i \log p_i$.

Se una p_i è nulla, per continuità si pone $p_i \log p_i = 0$. La base del logaritmo è indifferente (purché coerente col resto dei calcoli), ma la scelta tipica è D.

L'entropia è una misura della "*confusione*" nella sorgente. In generale l'entropia è alta quando le probabilità sono simili, mentre è bassa quando sono dissimili, ossia uno o più caratteri prevalgono. Il valore massimo è $H\left(\underbrace{\frac{1}{k}, \ldots, \frac{1}{k}}_{k \text{ volte}}\right) = \log_D k$.

Definizione 1.7. Siano P e Q due distribuzioni di probabilità sullo stesso alfabeto. La DIVERGENZA tra P e Q è $D(P \mathbin{/\!/} Q) \coloneqq \sum_{i=1}^{k} p_i \log \frac{p_i}{q_i}$.

Considerando $D(P \mathbin{/\!/} Q)$ notiamo che:

- la divergenza è asimmetrica rispetto ai suoi argomenti;
- se $P = Q$ è nulla;
- se esistono p_i e q_i tali che p_i tende a 1 e q_i tende a 0, il loro addendo è un numero molto grande positivo;
- viceversa se p_i tende a 0 e q_i tende a 1, il loro addendo è un numero molto grande negativo.

Estendiamo nei casi limite ponendo:

- $0 \log \frac{0}{\neq 0} \rightsquigarrow 0$ per continuità;
- $(\neq 0) \log \frac{\neq 0}{0} \rightsquigarrow +\infty$ per continuità;
- $0 \log \frac{0}{0} \rightsquigarrow 0$ per buon senso (non ha senso che accada).

✎ .....

Proposizione 1.8. $D(P \mathbin{/\!/} Q) \geq 0$.

Dimostrazione. Consideriamo e come base del logaritmo, per cui è facile seguire la via analitica. Si può dimostrare che vale per qualsiasi base attraverso una dimostrazione combinatoria più difficile.

Confrontando le funzioni $\log z$ e $1 - \frac{1}{z}$ abbiamo che $\forall z \geq 0\colon \log z \geq 1 - \frac{1}{z}$. Allora

$$D(P \mathbin{/\!/} Q) = \sum_{i=1}^{k} p_i \log \frac{p_i}{q_i} \geq \sum_{i=1}^{k} p_i \left(1 - \frac{q_i}{p_i}\right) = \underbrace{\sum_{i=1}^{k} p_i}_{=1} - \underbrace{\sum_{i=1}^{k} q_i}_{=1} = 0 \,.$$

..... ✎

✎ .....

Teorema 1.9: Teorema di Shannon. Se un codice è UD allora $\mathrm{EL} \geq H(P)$.

Dimostrazione. Codice UD $\Rightarrow \alpha \leq 1 \Rightarrow \log \alpha \leq 0$.

Definisco una distribuzione "artificiale" $Q = \left(\frac{D^{-\ell_1}}{\alpha}, \ldots, \frac{D^{-\ell_k}}{\alpha}\right)$. Allora

$$\begin{aligned} D(P \mathbin{/\!/} Q) &= \sum_{i=1}^{k} p_i \log \frac{p_i \alpha}{D^{-\ell_i}} = \sum_{i=1}^{k} p_i (\log p_i + \log \alpha + \ell_i) = \\ &= \sum_{i=1}^{k} p_i \log p_i + \log \alpha \sum_{i=1}^{k} p_i + \sum_{i=1}^{k} p_i \ell_i = -H(P) + \log \alpha + \mathrm{EL} \Rightarrow \\ &\Rightarrow -H(P) + \log \alpha + \mathrm{EL} \geq 0 \Rightarrow \mathrm{EL} \geq H(P) - \underbrace{\log \alpha}_{\leq 0} \geq H(P) \,. \end{aligned}$$

..... ✎

Esistono casi in cui $\mathrm{EL} = H(P)$, ossia quando $\alpha = 1$, anche se comunque non si può fare di meglio.

1.1.1 Codice di Shannon-Fano

Il codice di Shannon-Fano parte dal calcolo della lunghezza delle parole di codice, fissandola pari a $\ell_i = \lceil -\log_D p_i \rceil$. Poi con un algoritmo *greedy* viene popolato l'albero, sapendo che il posto si trova sempre perché

$$\begin{aligned} \lceil -\log_D p_i \rceil < -\log_D p_i + 1 \Rightarrow \lceil -\log_D p_i \rceil = -\log_D p_i + \xi_i \text{ con } 0 \leq \xi_i < 1 \Rightarrow \\ \Rightarrow \sum_{i=1}^{k} D^{-\ell_i} = \sum_{i=1}^{k} D^{\log_D p_i - \xi_i} = \sum_{i=1}^{k} p_i \underbrace{D^{-\xi_i}}_{<1} \leq 1 \,. \end{aligned}$$

$$\mathrm{EL} = \sum_{i=1}^{k} p_i \lceil -\log_D p_i \rceil = \sum_{i=1}^{k} p_i (-\log_D p_i + \xi_i) = -\sum_{i=1}^{k} p_i \log_D p_i + \underbrace{\sum_{i=1}^{k} p_i \xi_i}_{<1} < H(P) + 1 \,.$$

Questo ci dice che il codice ottimo si può cercare tra $H(P)$ e $H(P)+1$, infatti in generale Shannon-Fano non è ottimo.

✎ .....

Definizione 1.10.

- Il rapporto $R_n \coloneqq \frac{\mathrm{EL}_n}{n}$ si dice TASSO DI COMPRESSIONE e corrisponde alla lunghezza media per carattere di un codice su un alfabeto $\mathcal{A}^n$.
- Un codice il cui tasso tende a $H(P)$ per n che tende a ∞ si dice ASINTOTICAMENTE OTTIMO.

Esempio 1.11. Sia $\mathcal{A} = \{a, b\}$ con $P = \left(\frac{3}{4}, \frac{1}{4}\right)$, da cui $H(P) = \sim 0{,}81$. Applicando Shannon-Fano otteniamo $\ell_a = 1$, $\ell_b = 2$ ed $\mathrm{EL} = 1{,}25$.

Invece di considerare $\mathcal{A}$ prendiamo $\mathcal{A}^2 = \{aa, ab, ba, bb\}$ e calcoliamo P_2 in un ottica di processo *bernoulliano* (la lettera appena uscita non influenza la successiva): $P_2 = \left(\frac{9}{16}, \frac{3}{16}, \frac{3}{16}, \frac{1}{16}\right)$. Applicando di nuovo Shannon-Fano otteniamo $\mathrm{EL}_2 = 1{,}93$. Nonostante l'apparente peggioramento, ora il tasso è $R_2 = 0{,}97$. Più avanti dimostreremo che Shannon-Fano è asintoticamente ottimo.

Siano X e Y variabili aleatorie rispettivamente con valori in $\{x_1, \dots, x_k\}$ e $\{y_1, \dots, y_h\}$ e distribuzione di probabilità $P = (p_1, \dots, p_k)$ e $Q = (q_1, \dots, q_h)$. Indichiamo con «p_{ij}» la probabilità congiunta $\mathcal{P}(X = x_i \wedge Y = y_j)$. Avremo kh eventi di questo tipo, la cui sommatoria è ancora 1, quindi possiamo considerare la distribuzione di probabilità congiunta «$X \wedge Y$». La sua entropia è $H(X \wedge Y) = -\sum_{i=1}^{k}\sum_{j=1}^{h} p_{ij} \log p_{ij}$. Ipotizziamo quindi che X e Y siano indipendenti, ossia $p_{ij} = p_i q_j$, allora

$$H(X \wedge Y) = -\sum_{i=1}^{k}\sum_{j=1}^{h} p_i q_j \log(p_i q_j) = -\sum_{i=1}^{k}\sum_{j=1}^{h} p_j q_j \log p_i - \sum_{i=1}^{k}\sum_{j=1}^{h} p_i q_j \log q_j =$$
$$= \underbrace{\sum_{j=1}^{h} q_j}_{=1}\underbrace{\left(-\sum_{i=1}^{k} p_i \log p_i\right)}_{=H(P)} + \underbrace{\sum_{i=1}^{k} p_i}_{=1}\underbrace{\left(-\sum_{j=1}^{h} q_j \log q_j\right)}_{=H(Q)} = H(P) + H(Q)\,,$$

oppure, con un leggero abuso di notazione, $H(X \wedge Y) = H(X) + H(Y)$.

In generale siano $X_1, \dots, X_n$ v.a. indipendenti sullo stesso alfabeto e con la stessa distribuzione di probabilità P. Allora $H(X_1 \wedge \cdots \wedge X_n) = nH(P)$.

Definizione 1.12. Una SORGENTE DI INFORMAZIONE può essere descritta da una sequenza di v.a. $X_1, X_2, \dots$ a valori su uno stesso alfabeto. Si dice STAZIONARIA se tutte le distribuzioni di probabilità sono uguali e si dice SENZA MEMORIA se un simbolo non dipende in nessun modo dal precedente. Se una sorgente di informazione possiede entrambe queste proprietà, le sue v.a. sono indipendenti e si dice anche BERNOULLIANA.

Codifichiamo con un codice UD n-uple provenienti da una sorgente di informazione stazionaria e senza memoria, quindi abbiamo un alfabeto $\mathcal{A}$ e una distribuzione di probabilità P^n. $\mathrm{EL}_n \geq H(P^n) = nH(P) \Rightarrow$ $\Rightarrow R_n \geq H(P)$, che è la versione sulle n-uple del Teorema di Shannon.

Applicando Shannon-Fano otteniamo $\mathrm{EL}_n < H(P^n) + 1 = nH(P) + 1 \Rightarrow R_n < H(P) + n^{-1}$. Da questa disequazione si vede che Shannon-Fano tende asintoticamente all'entropia.

1.1.2 Ricerca del codice ottimo – Codice di Huffman

Per la ricerca del codice ottimo, ossia con EL minima, assumiamo che $p_1 \geq \cdots \geq p_k$, senza perdita di generalità.

Definizione 1.13.

- Sia φ un codice su un alfabeto $\mathcal{A}$ con distribuzione di probabilità P. La GEMINAZIONE di φ consiste sostituire nell'alfabeto un simbolo a_i con due nuovi simboli a'_i e a''_i tali che $p'_i + p''_j = p_i$. Nell'albero di φ, la foglia a_i diventa il nodo padre delle nuove foglie a'_i e a''_i. Il nuovo alfabeto $\tilde{\mathcal{A}} = \{a_1, \dots, a'_i, a''_i, \dots, a_k\}$ viene chiamato SORGENTE ESTESA.
- L'operazione inversa della geminazione viene detta TAGLIO.

..... ✎

Siccome $\mathrm{EL} = c + \ell_i p_i$ e $\widetilde{\mathrm{EL}} = c + (\ell_i + 1)(p'_i + p''_i)$, con c costante che raggruppa gli addendi non coinvolti, allora in seguito alla geminazione la lunghezza media è aumentata di $\widetilde{\mathrm{EL}} - \mathrm{EL} = p_i$.

Per costruire il codice ottimo, osserviamo alcuni fatti:

✎ .....

Lemma 1.14.

0. Un codice ottimo esiste.

   Dimostrazione. Per individuarlo basta un algoritmo esponenziale che costruisce tutti i codici sull'albero D-ario di lunghezza $k - 1$.

1. Non esiste codice ottimo in cui $p_i > p_j \wedge \ell_i > \ell_j$.

   Dimostrazione. Supponiamo per assurdo che un simile codice ottimo φ esista, per qualche i e j. Scambiamo nel suo albero a_i e a_j, ottenendo così un nuovo codice φ'.
   $\mathrm{EL} = c + p_j \ell_j + p_i \ell_i$ e $\mathrm{EL}' = c + p_i \ell_j + p_j \ell_i$ (c costante che raggruppa gli addendi non coinvolti), quindi $\mathrm{EL} - \mathrm{EL}' = (\ell_j - \ell_i)(p_j - p_i) > 0$ ↯ con la minimalità di EL.

2. Esiste un codice ottimo in cui $\ell_1 \leq \cdots \leq \ell_k$.

   Dimostrazione. Supponiamo che $\ell_i > \ell_j$ per qualche i, j con $i < j$ e che il codice sia ottimo. Scambiamo nel suo albero a_i e a_j e vediamo che:

   - per $p_i > p_j$ la lunghezza media cala ↯;
   - quindi l'unica soluzione è che $p_i = p_j$.

3. Esiste un codice ottimo tale che nel suo albero a_{k-1} e a_k sono foglie sorelle al livello più basso (formano una "forchetta").

   Dimostrazione. Consideriamo varie casistiche:

   - a_k è figlio unico al livello più basso: ↯ perché non sarebbe un codice ottimo;
   - a_k è foglia sorella al livello più basso con a_i, per $i \neq k - 1$, e a_{k-1} è a un livello superiore: o ↯ perché il codice non era ottimo, oppure $p_i = p_{k-1}$ e quindi basta scambiare a_i con a_{k-1} per ottenere un codice ottimo con la proprietà desiderata;
   - come il caso precedente, solo che a_{k-1} è allo stesso livello: basta scambiare a_i con a_{k-1} per ottenere un codice ottimo con la proprietà desiderata.

4. Se gemino un codice ottimo su un nodo associato a un simbolo che genera due caratteri con probabilità minimale nella sorgente estesa, allora il codice per la sorgente estesa così ottenuto è ottimo.

   Dimostrazione. Sia ① l'albero ottimo di partenza. A un simbolo a_i sostituisco a'_i e a''_i, con probabilità minimale nel nuovo alfabeto, ottenendo ②. Supponiamo per assurdo che non sia ottimo, ma allora esiste un ottimo ③ che, tagliando a'_i e a''_i, mi porta in un albero ④ che ha di nuovo a_i. Vale che
   $$\left.\begin{array}{r} \mathrm{EL}② - \mathrm{EL}① = \mathrm{EL}③ - \mathrm{EL}④ = p_i \\ ③ \text{ ottimo} \end{array}\right\} \Rightarrow \mathrm{EL}④ < \mathrm{EL}① \text{ ↯ con la minimalità di ①.}$$

Corollario 1.15. Se $\mathcal{A} = \{a_1, a_2\}$ il codice ottimo è $\varphi(a_1) = \langle 0 \rangle$, $\varphi(a_2) = \langle 1 \rangle$.

Dimostrazione. Discende in particolare dal punto 3. dei Lemmi precedenti.

Pseudocodice 1.16: Huffman – Codifica. Input: $\mathcal{A}_k = \{a_1, \dots, a_k\}$, $P_k = \{p_1, \dots, p_k\}$.

```
i := k;
F := new Stack;
while (i >= 2) do
  Scegli a, b in A_i tali che p_i(a) e p_i(b) siano minime in P_i;
  A_{i-1} := (A_i \ {a,b}) ∪ {α_i}; // α_i nuovo simbolo
  P_{i-1}(s) := { p_i(s)          s ∈ A_i \ {a,b}
                { p_i(a) + p_i(b) s = α_i ; // per ogni s ∈ A_{i-1}
  F.push(α_i, a, b);
  i--;
od
φ(α_2) := "";
while (!F.empty()) do
  (α, β, γ) := F.pop();
  φ(β) := φ(α)·"0";
  φ(γ) := φ(α)·"1";
od
```

Esempio 1.17. Prima fase:

$\mathcal{A}_5 = \{a, b, c, d, e\}$ $P_5 = (20, 15, 10, 5, 3)$
$\mathcal{A}_4 = \{a, b, c, \alpha_5\}$ $P_4 = (20, 15, 10, 8)$
$\mathcal{A}_3 = \{a, b, \alpha_4\}$ $P_3 = (20, 15, 18)$
$\mathcal{A}_2 = \{a, \alpha_3\}$ $P_2 = (20, 33)$
$\mathcal{A}_1 = \{\alpha_2\}$ $P_1 = (53)$

α_5	d	e
α_4	c	α_5
α_3	b	α_4
α_2	a	α_3

Seconda fase:

$\varphi(\alpha_2) = \varepsilon$ $\varphi(a) = \langle 0 \rangle$ $\varphi(\alpha_3) = \langle 1 \rangle$ $\varphi(b) = \langle 10 \rangle$ $\varphi(\alpha_4) = \langle 11 \rangle$
$\varphi(c) = \langle 110 \rangle$ $\varphi(\alpha_5) = \langle 111 \rangle$ $\varphi(d) = \langle 1110 \rangle$ $\varphi(e) = \langle 1111 \rangle$.

Ci sono due modi per impiegare il codice di Huffman:

- Scandisco il *file*, calcolo P, costruisco l'albero e lo memorizzo, comprimo il *file*. La dimensione finale potrebbe essere penalizzata dall'albero non compresso contenuto nel *file*.
- Fisso un albero e uso sempre quello, sconfinando nella semantica. Si comprimono ottimalmente solo i *files* che rispettano la frequenza.

1.1.3 Ricerca del codice universale – Codice multinomiale

Definizione 1.18. Siano X una v.a. con valori in $\{x_1, \dots, x_k\}$ e distribuzione di probabilità $P = (p_1, \dots, p_k)$ e Y una v.a. generica. Allora l'ENTROPIA CONDIZIONATA di $Y \mid X$ (leggi «Y data X») viene definita come

$$H(Y \mid X) \coloneqq \sum_{i=1}^{k} \big(\mathcal{P}(X = x_i) \cdot H(Y \mid X = x_i)\big).$$

Se Y è una v.a. con valori in $\{y_1, \dots, y_h\}$ e distribuzione di probabilità $Q = (q_1, \dots, q_h)$, l'entropia condizionata viene anche scritta come $H(Y \mid X) = \sum_{i=1}^{k} p_i \left(- \sum_{j=1}^{h} q_{j|i} \log q_{j|i} \right)$, dove $q_{j|i} = \frac{p_{ij}}{p_i}$, quindi

$$H(Y \mid X) = -\sum_{i=1}^{k} \sum_{j=1}^{h} \cancel{p_i} \frac{p_{ij}}{\cancel{p_i}} \log \frac{p_{ij}}{p_i} = -\sum_{i=1}^{k} \sum_{j=1}^{h} p_{ij} \log p_{ij} - \left(-\sum_{i=1}^{k} \sum_{j=1}^{h} p_{ij} \log p_i \right) =$$

$$= H(X \wedge Y) - \left(-\sum_{i=1}^{k} \log p_i \underbrace{\sum_{j=1}^{h} p_{ij}}_{=p_i} \right) = H(X \wedge Y) - H(X) \,,$$

e similmente $H(X \mid Y) = H(X \wedge Y) - H(Y)$.

✎ .....

Definizione 1.19. La MUTUA INFORMAZIONE di due v.a. X e Y viene definita come

$$I(X \wedge Y) \coloneqq D(P_{XY} \mathbin{/\!/} P_X P_Y) = \sum_{i=1}^{k} \sum_{j=1}^{h} p_{ij} \log \frac{p_{ij}}{p_i q_j} \,.$$

..... ✎

Essendo una divergenza, varrà sempre $I(X \wedge Y) \geq 0$. Consideriamo i due casi estremi:

- X e Y indipendenti $\Rightarrow p_{ij} = p_i q_j \Rightarrow I(X \wedge Y) = 0$;
- $X = Y \Rightarrow p_{ij} = \begin{cases} 0 & i \neq j \\ p_i & i = j \end{cases} \Rightarrow I(X \wedge Y) = \sum_{i=1}^{k} p_i \log \frac{p_i}{p_i^2} = -\sum_{i=1}^{k} p_i \log p_i = H(X)$.

Per il secondo punto, l'entropia viene anche chiamata AUTOINFORMAZIONE.

✎ .....

Esercizio 1.20. $I(X \wedge Y) = H(Y) - H(Y \mid X) = H(X) - H(X \mid Y)$.

Soluzione. Ricordando che $q_{i|j} = \frac{p_{ij}}{p_i}$,

$$I(X \wedge Y) = \sum_{i=1}^{k} \sum_{j=1}^{h} p_{ij} \log \frac{p_{ij}}{p_i q_j} = \sum_{i=1}^{k} \sum_{j=1}^{h} p_{ij} \log \frac{q_{i|j}}{q_j} =$$

$$= \sum_{i=1}^{k} \sum_{j=1}^{h} p_{ij} \log q_{i|j} - \sum_{j=1}^{h} \log q_j \underbrace{\sum_{i=1}^{k} p_{ij}}_{=q_j} = H(Y) - H(Y \mid X) \,.$$

La seconda uguaglianza si ottiene per simmetria.

..... ✎

Con il solito $\mathcal{A} = \{a_1, \dots, a_k\}$, consideriamo n-uple e le raggruppiamo in classi di permutazioni dette TIPI. Ogni tipo j è identificato univocamente da una k-upla $(n_{j_1}, \dots, n_{j_k})$, dove $\sum_{i=1}^{k} n_{j_i} = n$ e gli n_{j_i} sono le occorrenze del simbolo a_i nelle n-uple del tipo. Se $x \in \mathcal{A}^n$ possiamo anche denotare il suo tipo con «$[x]$».

I tipi sono $\binom{n+k-1}{n}$, abbreviato con «γ_n» (benché in effetti dipenda anche da k, che però è fissato). La cardinalità del tipo j è $\binom{n}{n_{j_1}, \dots, n_{j_k}} \coloneqq \frac{n!}{n_{j_1}! \cdots n_{j_k}!}$ (COEFFICIENTE MULTINOMIALE).

Osserviamo che, nonostante il numero di n-uple in $\mathcal{A}_n$ sia k^n, esponenziale rispetto a n, si dimostra (Esercizio 1.21) che $\gamma_n \leq (n+1)^k$, quindi invece il numero di tipi cresce polinomialmente. Come bilanciamento, qualche tipo dovrà crescere esponenzialmente.

✎ .....

Esercizio 1.21. $\gamma_n \leq (n+1)^k$.

Soluzione. Per induzione su k:

- $(\!|k=1|\!)$ $\binom{n}{n} = 1 \le n+1$
- $(\!|k>1|\!)$ $\binom{n+k}{n} = \frac{(n+k)!}{n!k!} = \frac{n+k}{k}\binom{n+k-1}{n} \overset{\text{ind.}}{\le} \left(\frac{n}{k}+1\right)(n+1)^k < (n+1)^{k+1}$.

.....

Il CODICE MULTINOMIALE si basa su questa idea: data $x \in \mathcal{A}_n$, $\varphi(x) = \varphi_{\mathrm{p}}(x)\varphi_{\mathrm{s}}(x)$. φ_{p} è il PREFISSO, di lunghezza fissa, che individua il tipo; φ_{s} è il SUFFISSO, di lunghezza variabile a seconda della cardinalità del tipo, che identifica la stringa all'interno del tipo selezionato.

Sapendo che $|\varphi_{\mathrm{p}}(x)| = \lceil \log_D \gamma_n \rceil$ e $|\varphi_{\mathrm{s}}(x)| = \left\lceil \log_D \binom{n}{n_{j_1},\dots,n_{j_k}} \right\rceil$ possiamo calcolare

$$R_n = \frac{\mathrm{EL}_n}{n} = \frac{\sum_{x\in\mathcal{A}^n} P^n(x)|\varphi(x)|}{n} = \frac{\sum_{x\in\mathcal{A}^n} P^n(x)|\varphi_{\mathrm{p}}(x)|}{n} + \frac{\sum_{x\in\mathcal{A}^n} P^n(x)|\varphi_{\mathrm{s}}(x)|}{n}\,,$$

quindi esaminiamo separatamente

- il prefisso

$$\frac{\lceil \log_D \gamma_n \rceil}{n} \underbrace{\sum_{x\in\mathcal{A}^n} P^n(x)}_{=1} < \frac{\log_D \gamma_n + 1}{n} \le \frac{\log_D (n+1)^k + 1}{n} = k\,\frac{\log_D(n+1)}{n} + \frac{1}{n}\,,$$

  che è un infinitesimo;

- il suffisso, che dipende dal tipo. All'interno di un tipo le n-uple hanno la stessa probabilità (nell'ipotesi di una sorgente bernoulliana) e la codifica ha la stessa lunghezza. Indicheremo con «$|T_j|$» la lunghezza delle n-uple di T_j e con «$\mathcal{P}(T_j)$» il valore $\sum_{x\in T_j} P^n(x) = P^n(x)|T_j|$.

$$\frac{\sum_{x\in\mathcal{A}^n} P^n(x)|\varphi_{\mathrm{s}}(x)|}{n} = \frac{\sum_{j=1}^{\gamma_n} \mathcal{P}(T_j)\lceil \log_D |T_j| \rceil}{n} < \frac{\sum_{j=1}^{\gamma_n} \mathcal{P}(T_j)\log_D |T_j|}{n} + \frac{1}{n}\,, \tag{1.1}$$

  dove il secondo addendo è un infinitesimo.

  Introduciamo una v.a. "fittizia" J a valori in $\{1,\dots,\gamma_n\}$ con probabilità $\mathcal{P}(J=j) = \mathcal{P}(T_j)$. Allora

$$H(X^n \mid J) = \sum_{j=1}^{\gamma_n} \underbrace{\mathcal{P}(J=j)}_{=\mathcal{P}(T_j)} \cdot H(X^n \mid J=j)\,.$$

  Se $J=j$, le n-uple sono "determinate", tutte dello stesso tipo, ognuna permutazione delle altre ed equiprobabili, quindi la distribuzione è uniforme e la sua entropia è massima, pari a $\log_D |T_j|$, da cui $H(X^n \mid J) = \sum_{j=1}^{\gamma_n} P^n(x)|T_j|\log_D |T_j|$, uguale al numeratore del primo addendo della (1.1). Sappiamo anche che

$$H(X^n \mid J) = \underbrace{H(X^n)}_{=nH(X)} + \underbrace{H(J \mid X^n)}_{=0} - \underbrace{H(J)}_{\ge 0} \le nH(X)\,,$$

  dove il secondo addendo è nullo perché, fissata la n-upla, J è determinata. Allora ripartendo dalla fine della (1.1) si ottiene

$$\frac{\sum_{j=1}^{\gamma_n} \mathcal{P}(T_j)\log_D |T_j|}{n} + \frac{1}{n} \le H(X) + \frac{1}{n}\,.$$

Rimontiamo prefisso e suffisso e avremo $R_n = \frac{\mathrm{EL}_n}{n} < \underbrace{\frac{2}{n} + k\frac{\log_D(n+1)}{n}}_{\text{infinitesimi}} + H(P)$, quindi questo è un codice asintoticamente ottimo e UNIVERSALE.

1.2 LV-B

L'idea è quella di avere un vocabolario di parole emesse dalla sorgente.

✎ .....

Definizione 1.22.

- Una FAMIGLIA DI MESSAGGI $\mathcal{M} \subseteq \mathcal{A}^+$ è un insieme $\mathcal{M} \coloneqq \{m_1, \dots, m_t\}$.
- La LUNGHEZZA MEDIA dell'*input* è $\mathrm{EN} \coloneqq \sum_{i=1}^{t} \mathcal{P}(m_i)|m_i|$.
- $\mathcal{M}$ si dice ESAURIENTE se ogni sequenza "opportunamente lunga" di caratteri di $\mathcal{A}$ ha almeno un prefisso in $\mathcal{M}$.
- $\mathcal{M}$ si dice A PREFISSO se nessun suo messaggio è prefisso di un'altro.
- $\mathcal{M}$ si dice COMPLETA se è esauriente e a prefisso.

..... ✎

La codifica per una parola del vocabolario è una parola di codice di lunghezza fissa $\ell \coloneqq \lceil \log_D t \rceil$. Il tasso di compressione diventa $R \coloneqq \frac{\ell}{\mathrm{EN}}$. Per questo cercheremo di costruire codici con EN grande.

Nel caso di una famiglia non a prefisso, si cerca di codificare la parola più lunga, introducendo un ritardo di codifica. Tuttavia rispetto ai codici B-LV l'assenza di questa proprietà non è così grave.

D'ora in poi indichiamo con «n_i» la lunghezza del messaggio m_i e con «n» la massima tra tali lunghezze.

✎ .....

Teorema 1.23: Controdisuguaglianza di Kraft. Se una famiglia $\mathcal{M}$ è esauriente allora $\sum_{i=1}^{t} k^{-n_i} \geq 1$. Se inoltre in particolare è completa allora vale l'uguaglianza.

Dimostrazione. Consideriamo l'albero k-ario completo di lunghezza n, le cui foglie sono stringhe di $\mathcal{A}^n$, ognuna delle quali avente un prefisso in $\mathcal{M}$. m_i è prefisso di k^{n-n_i} stringhe di $\mathcal{A}$, eventualmente condivise con altri messaggi. Quindi $\sum_{i=1}^{t} k^{n-n_i} \geq k^n \Rightarrow \sum_{i=1}^{t} k^{-n_i} \geq 1$.

Se inoltre $\mathcal{M}$ è a prefisso allora non ci sono stringhe di $\mathcal{A}$ condivise tra i messaggi, quindi vale l'uguaglianza.

..... ✎

1.2.1 Famiglie complete – Codice di Tunstall

Considero uno tra i messaggi più lunghi e lo scompongo in ma_i. Allora anche $ma_1, \dots, ma_k \in \mathcal{M}$ perché $\mathcal{M}$ è esauriente e perché m non può contenere messaggi ($\mathcal{M}$ è a prefisso). Tagliando in corrispondenza del nodo m ottengo un nuovo codice completo, ma quindi posso ricominciare, finché arrivo al codice con $\mathcal{M}_0 = \{a_1, \dots, a_k\}$ e albero $\mathcal{T}_0$.

Questo fa capire che tutte le famiglie complete si ottengono da $\mathcal{T}_0$ mediante una geminazione che rimpiazza un nodo m con $ma_1, \dots, ma_k$ (chiamata regola «$\mathcal{R}$»). Tutti gli alberi $\mathcal{T}_j$ ottenuti con j applicazioni della regola $\mathcal{R}$ hanno $k + j(k-1)$ foglie.

Una sorgente può essere vista sia come sorgente di $\mathcal{A}$ sia di $\mathcal{M}$. Allora a una famiglia di messaggi generata con j applicazioni di $\mathcal{R}$ associamo una distribuzione di probabilità S_j, in cui ogni s_i è il prodotto delle probabilità dei caratteri che compongono m_i. Calcoliamo la differenza tra due lunghezze medie consecutive, isolando il messaggio m su cui avviene la geminazione:

$$\mathrm{EN}_j = \sum_{\substack{\mu \in \mathcal{M} \\ \mu \neq m}} \mathcal{P}(\mu)|\mu| + \mathcal{P}(m)|m|\,,$$

$$\begin{aligned}\mathrm{EN}_{j+1} &= \sum_{\substack{\mu\in\mathcal{M}\\ \mu\neq m}} \mathcal{P}(\mu)|\mu| + \sum_{i=1}^{k}\mathcal{P}(ma_i)|ma_i| = \\ &= \sum_{\substack{\mu\in\mathcal{M}\\ \mu\neq m}} \mathcal{P}(\mu)|\mu| + \sum_{i=1}^{k}\mathcal{P}(m)\mathcal{P}(a_i)(|m|+1) = \\ &= \sum_{\substack{\mu\in\mathcal{M}\\ \mu\neq m}} \mathcal{P}(\mu)|\mu| + \mathcal{P}(m)(|m|+1)\,,\end{aligned}$$

da cui $\Delta\mathrm{EN} = \mathcal{P}(m)$.

✎ .....

Lemma 1.24. $H(S_j) = H(P)\cdot \mathrm{EN}_j$.

Dimostrazione. Per induzione su j.

(B) $S_0 = P$ e $\mathrm{EN}_0 = 1$.

(P) $H(S_{j+1}) = -\sum_{\substack{\mu\in\mathcal{M}_{j+1}\\ \mu\neq ma_i}} \mathcal{P}(\mu)\log\mathcal{P}(\mu) - \underbrace{\sum_{i=1}^{k}\mathcal{P}(ma_i)\log\mathcal{P}(ma_i)}_{\text{espanso di seguito}}$

$$\sum_{i=1}^{k}\mathcal{P}(m)\mathcal{P}(a_i)\big(\log\mathcal{P}(m)+\log\mathcal{P}(a_i)\big) = \sum_{i=1}^{k}\mathcal{P}(m)\mathcal{P}(a_i)\log\mathcal{P}(m)\sum_{i=1}^{k}$$

$$\sum_{i=1}^{k}\mathcal{P}(m)\mathcal{P}(a_i)\log\mathcal{P}(a_i) = \mathcal{P}(m)\log\mathcal{P}(m)\underbrace{\sum_{i=1}^{k}\mathcal{P}(a_i)}_{=1} - \mathcal{P}(m)H(P)$$

$$H(S_{j+1}) = \underbrace{-\sum_{\substack{\mu\in\mathcal{M}_{j+1}\\ \mu\neq ma_i}} \mathcal{P}(\mu)\log\mathcal{P}(\mu)}_{=H(S_j)} - \mathcal{P}(m)\log\mathcal{P}(m) + \mathcal{P}(m)H(P) =$$

$$\begin{aligned}&= H(S_j) + \mathcal{P}(m)H(P) \overset{\text{ind.}}{=} H(P)\cdot\mathrm{EN}_j + \mathcal{P}(m)H(P) = \\ &= H(P)\big(\mathrm{EN}_j + \mathcal{P}(m)\big) = H(P)\cdot\mathrm{EN}_{j+1}\,.\end{aligned}$$

..... ✎

Tornando a esaminare il tasso, abbiamo che $R_j = \frac{\lceil \log_D|\mathcal{M}_j|\rceil H(P)}{H(S_j)}$. In generale $0 \le H(S_j) \le \log|\mathcal{M}_j|$, quindi $R_j \ge \frac{\lceil \log_D|\mathcal{M}_j|\rceil}{\log_D|\mathcal{M}_j|}H(P) \ge H(P)$, che è la forma LV-B del teorema di Shannon.

Se la regola $\mathcal{R}$ è applicata a un nodo con probabilità massima, la chiameremo REGOLA DI TUNSTALL e la scriveremo come «$\mathcal{R}_T$».

✎ .....

Lemma 1.25. Sia $\mathcal{T}_j$ un albero ottenuto con j applicazioni di $\mathcal{R}$. Allora esiste una sequenza di j applicazioni di $\mathcal{R}$ sui messaggi $m_1,\ldots,m_j$ tale che $\mathcal{P}(m_1)\ge\cdots\ge\mathcal{P}(m_j)$.

Dimostrazione. Una volta costruito $\mathcal{T}_j$ basta riordinare le applicazioni di $\mathcal{R}$ in modo tale da rispettare la proprietà. Si può sempre fare perché la probabilità di un sottoalbero è strettamente minore di quella del nodo a cui appartiene.

..... ✎

Una sequenza con questa proprietà è detta REGOLARE. Possiamo quindi concentrarci sulle famiglie generate da sequenze regolari.

✎ .....

Lemma 1.26. Sia $\mathcal{T}$ un albero ottenuto da $\mathcal{T}_0$ con una sequenza regolare di j applicazioni di $\mathcal{R}$. Se in un

passo non è stata usata $\mathcal{R}_T$ allora esiste un albero $\mathcal{T}'$ ottenuto da $\mathcal{T}_0$ con j applicazioni regolari tale che $\mathrm{EN}(\mathcal{T}') > \mathrm{EN}(\mathcal{T})$.

Dimostrazione. Sia $u+1$ il primo passo in cui non ho usato $\mathcal{R}_T$. Siano m ed n messaggi in $\mathcal{T}_u$ tali che $\mathcal{P}(m) > \mathcal{P}(n)$, con n nodo geminato. Poiché $\mathcal{T}$ deriva da una sequenza regolare, m è ancora una foglia in $\mathcal{T}$. Al passo $u+1$ gemino su m e ripeto nel sottoalbero di m le applicazioni di $\mathcal{R}$ che prima avevo fatto nel sottoalbero di n. Allora $\mathrm{EN}(\mathcal{T}') - \mathrm{EN}(\mathcal{T}) = \mathcal{P}(m) - \mathcal{P}(n) > 0 \Rightarrow \mathrm{EN}(\mathcal{T}') > \mathrm{EN}(\mathcal{T})$, infatti l'unica differenza non viene "recuperata" nei passi successivi, che sono identici in entrambe le costruzioni.

✎

Per il prossimo lemma introduciamo alcune notazioni: «Π_j» sarà $\max_{m\in\mathcal{M}_j} \mathcal{P}(m)$, «$\pi_j$» sarà $\min_{m\in\mathcal{M}_j} \mathcal{P}(m)$ e infine «p^*» è $\pi_0 = \min_{a\in\mathcal{A}} \mathcal{P}(a)$. L'idea è mostrare che per i codici di Tunstall Π_j e π_j sono "simili":

✎

Lemma 1.27. $\dfrac{\Pi_j}{\pi_j} \leq \frac{1}{p^*}$.

Dimostrazione. Per induzione su j.

(B) $\Pi_0 \leq 1$ e $\pi_0 = p^*$.

(P) Se in $\mathcal{T}_j$ c'era un solo massimo allora $\Pi_{j+1} < \Pi_j$, altrimenti $\Pi_{j+1} = \Pi_j$. In generale $\Pi_{j+1} \leq \Pi_j$.

$\pi_{j+1} = \min\{\pi_j, \Pi_j p_1, \ldots, \Pi_j p_k\} = \min\{\pi_j, \Pi_j p^*\}$. Se $\pi_{j+1} = \pi_j$ allora $\frac{\Pi_{j+1}}{\pi_{j+1}} \leq \frac{\Pi_j}{\pi_j} \overset{\text{ind.}}{\leq} \frac{1}{p^*}$. Altrimenti se $\pi_{j+1} = \Pi_j p^*$ allora $\frac{\Pi_{j+1}}{\pi_{j+1}} \leq \frac{\cancel{\Pi_j}}{\cancel{\Pi_j} p^*} = \frac{1}{p^*}$.

✎

✎

Corollario 1.28. Il CODICE DI TUNSTALL, che si ottiene applicando sempre $\mathcal{R}_T$, è asintoticamente ottimo.

Dimostrazione.

$$H(S_j) = -\sum_{i=1}^{|\mathcal{M}_j|} \mathcal{P}(m_i) \log \underbrace{\mathcal{P}(m_i)}_{\leq \Pi_j} \geq -\sum_{i=1}^{|\mathcal{M}_j|} \mathcal{P}(m_i) \log \Pi_j = -\log \Pi_j \geq \log p^* - \log \pi_j \,.$$

Siccome $\pi_j \leq \frac{1}{|\mathcal{M}_j|} \Rightarrow \log \pi_j \leq -\log|\mathcal{M}_j|$ allora $H(S_j) \geq \log p^* + \log|\mathcal{M}_j| \Rightarrow R_j = \frac{\log|\mathcal{M}_j|}{\log p^* + \log|\mathcal{M}_j|} H(P)$, quindi all'aumentare di j il tasso tende all'entropia.

✎

✎

Esempio 1.29. Dato un alfabeto $\mathcal{A} = \{a, b, c\}$ e la sua distribuzione di probabilità $P = (p_a, p_b, p_c)$, con $p_a > p_b \wedge p_a > p_c$, costruiamo

- il codice di Tunstall: $\mathcal{M} = \{aa, ab, ac, b, c\}$ con $\mathrm{EN} = 1 + p_a$;
- un codice esauriente ma non a prefisso: $\mathcal{M} = \{a, aa, aaa, b, c\}$ con $\mathrm{EN} = 1 + p_a^2 + p_a^3$, infatti $S = \big(p_a(1-p_a), p_a^2(1-p_a), p_a^3(1-p_a), p_b, p_c\big)$.

Poiché $1 + p_a^2 + p_a^3 > 1 + p_a$ per $p_a > \frac{-1+\sqrt{5}}{2}$, in tal caso Tunstall non è ottimo. Ciò significa che rinunciando alla proprietà del prefisso di può fare meglio di Tunstall.

✎

1.2.2 Codice di Lempel-Ziv (LZ)

Abbiamo un *buffer* V diviso in due parti $[1..M-1]$ e $[M..N]$. Dato $i \in \{1, \ldots, M-1\}$, definiamo «$L(i)$» come il massimo intero tale che $V[i..i+L(i)-1] = V[M..M+L(i)-1]$. Questo è un banale algoritmo di ricerca di sottostringhe quadratico sulla lunghezza di mezzo *buffer* (ma si può anche fare lineare). Inoltre definiamo «p» come un indice tale che $L(p) = \max\{L(1), \ldots, L(M-1)\}$ e «$\tilde{L}$» come $L(p)$.

L'algoritmo di compressione consiste nelle seguenti operazioni:

- predispongo il *buffer* con zeri nella prima parte e l'inizio del *file* nella seconda;
- finché il *file* non è finito:
  - calcolo p e $\tilde{L}$;
  - guardo il simbolo $x \coloneqq V[M + \tilde{L}]$;
  - memorizzo la tripla $\langle \tilde{L}, p, x \rangle$;
  - faccio uno *shift* a sinistra di $\tilde{L} + 1$ posizioni, facendo entrare un nuovo pezzo di *file* nel *buffer*.

Anche se non abbiamo una famiglia di messaggi, LZ è LV-B. Misurando lo spazio in *bits*, le componenti delle triple occupano:

- $\tilde{L} \in \{0, \dots, N - M + 1\} \Rightarrow |\tilde{L}| = \lceil \log_2(2 + N - M) \rceil$ bit (che denotiamo con «ℓ_1»);
- $p \in \{1, \dots, M - 1\} \Rightarrow |p| = \lceil \log_2(M - 1) \rceil$ bit («ℓ_2»);
- $|x| = 1$ bit.

Riusciremo quindi a comprimere se "mediamente" $\tilde{L} + 1 > \ell_1 + \ell_2 + 1 \Rightarrow \tilde{L} > \ell_1 + \ell_2$.

✎ .....

Esempio 1.30: LZ – Codifica. Codifichiamo la stringa ‹0011010010000111› con un *buffer* da $4 + 4$ bit.

buffer	$\langle \tilde{L}, p, x \rangle$	***shift***
0 0 0 0 | 0 0 1 1	$\langle 2, 1, 1 \rangle$	3
2 2 2 2		
0 0 0 1 | 1 0 1 0	$\langle 1, 4, 0 \rangle$	2
0 0 0 1		
0 1 1 0 | 1 0 0 1	$\langle 2, 3, 0 \rangle$	3
0 1 2 0		
0 1 0 0 | 1 0 0 0	$\langle 3, 2, 0 \rangle$	4
0 3 0 0		
1 0 0 0 | 0 1 1 1	$\langle 1, 2, 1 \rangle$	2
0 1 1 1		
0 0 0 1 | 1 1	$\langle 1, 4, 1 \rangle$	2
0 0 0 1		

Quando mi fermo alla fine del *file* ho una configurazione particolare.

La stringa codificata è ‹010001 001110 010100 011010 001011 001111›, quindi notiamo che in questo esempio la stringa iniziale era troppo breve per poter essere compressa.

..... ✎

✎ .....

Esempio 1.31: LZ – Decodifica. I simboli che escono dal *buffer* con lo *shift* vengono aggiunti alla stringa decodificata.

buffer	$\langle \tilde{L}, p, x \rangle$	***shift***
0 0 0 0 | 0 0 1	$\langle 2, 1, 1 \rangle$	3
0 0 0 1 | 1 0	$\langle 1, 4, 0 \rangle$	2
0 1 1 0 | 1 0 0	$\langle 2, 3, 0 \rangle$	3
0 1 0 0 | 1 0 0 0	$\langle 3, 2, 0 \rangle$	4
1 0 0 0 | 0 1	$\langle 1, 2, 1 \rangle$	2
0 0 0 1 | 1 1	$\langle 1, 4, 1 \rangle$	2

Ciò che resta alla fine nel *buffer* viene riversato direttamente nella stringa.

Quanto posso comprimere nel caso migliore possibile (*file* di soli zeri)? Se prendiamo un *buffer* con due parti di pari lunghezza uguale a ℓ e un file lungo f, la dimensione compressa è circa $f\frac{2\log \ell+1}{\ell}$, dove la frazione rappresenta il rapporto di compressione.

LZ è universale e l'idea è che le stringhe frequenti vengono catturate con un'unica tripla. Si dimostra infatti che LZ tende all'entropia, ossia è ottimo.

Di questo algoritmo esiste la variante DEFLATE, che prevede di applicare Huffman all'*output* a blocchi di LZ, focalizzandosi sulla componente $\tilde{L}$ delle triple.

1.2.3 Codice di Lempel-Ziv-Welch (LZW)

Assumiamo di avere una struttura dati STRING TABLE (un vettore di stringhe).

Pseudocodice 1.32: LZW – Codifica. Partiamo con una *string table* `table` vuota.

```
string := read(); // Supponiamo di lavorare con files non vuoti.
while (ci sono caratteri) do
  char := read();
  if (string·char ∈ table) then
    string := string·char;
  else
    write(il codice di string);
    table.add(string·char);
    string := char;
  fi
od
write(il codice di string);
```

Supponendo di avere un alfabeto primario a 8 bit, possiamo decidere di costruire una *string table* da 10 bit. I caratteri mantengono il loro codice, mentre le coppie, triple, ... assumono un codice man mano che vengono scoperte.

Esempio 1.33: LZW – Codifica. Codifichiamo la stringa ‹ABAABAAABA›.

`char`	`string·char ∈ table`	`write`	`table.add`	`string`
				‹A›
‹B›	✗	65	$256 \rightsquigarrow$ ‹AB›	‹B›
‹A›	✗	66	$257 \rightsquigarrow$ ‹BA›	‹A›
‹A›	✗	65	$258 \rightsquigarrow$ ‹AA›	‹A›
‹B›	✓			‹AB›
‹A›	✗	256	$259 \rightsquigarrow$ ‹ABA›	‹A›
‹A›	✓			‹AA›
‹A›	✗	258	$260 \rightsquigarrow$ ‹AAA›	‹A›
‹B›	✓			‹AB›
‹A›	✓			‹ABA›
		259		

Il codice LZW è un codice LV-B non a prefisso in cui il vocabolario viene costruito passo per passo. Fatto interessante è che non serve memorizzare la *string table*.

Pseudocodice 1.34: LZW – Decodifica. Partiamo con la *string table* `table` con le prime 256 celle riempite con i caratteri ASCII.

```
oc := read();
string := table[oc];
write(string);
while (ci sono caratteri) do
  nc := read();
  string := table[nc];
  write(string);
  ch := string[0];
  table.add(oc·ch);
  oc := nc;
od
```

Esempio 1.35: LZW – Decodifica. Decodifichiamo la sequenza di numeri (stringa) $65, 66, 65, 256, 258, 259$.

`nc` & `oc`	`string` & `write`	`ch`	`table.add`
65	‹A›		
66	‹B›	‹B›	$256 \rightsquigarrow$ ‹AB›
65	‹A›	‹A›	$257 \rightsquigarrow$ ‹BA›
256	‹AB›	‹A›	$258 \rightsquigarrow$ ‹AA›
258	‹AA›	‹A›	$259 \rightsquigarrow$ ‹ABA›
259	‹ABA›	‹A›	$260 \rightsquigarrow$ ‹AAA›

1.2.4 Codice di Burrows-Wheeler

Il *file* viene diviso in blocchi (anche grandi), ognuno dei quali costituirà una stringa «x». Per ogni x separatamente, si scrivono le permutazioni cicliche, le si ordina lessicograficamente e le si mette in una matrice quadrata di lato $|x| \times |x|$.

Data una colonna, posso ricostruire l'intera matrice? Con una qualsiasi colonna diversa dalla prima, posso ottenere in modo deterministico la prima, che è la versione ordinata di una qualunque colonna. L'ultima colonna, in particolare, è sia la più lontana dalla prima, sia quella che la precede. Quindi posso calcolare le coppie (ultima, prima), che concatenate alfabeticamente alla prima colonna mi forniscono la seconda. Allora posso iterare il ragionamento per ottenere le triple (ultima, prima, seconda) con cui ricostruire la terza, e così via. In fase di decodifica quindi ci bastano l'ultima colonna (che scriveremo come L) e l'indice di riga in cui si trova la stringa originale.

Esempio 1.36. Sia $x =$ ‹pippi›. La sua matrice è $\begin{bmatrix} \mathrm{i}\,\mathrm{p}\,\mathrm{i}\,\mathrm{p}\,\mathrm{p} \\ \mathrm{i}\,\mathrm{p}\,\mathrm{p}\,\mathrm{i}\,\mathrm{p} \\ \mathrm{p}\,\mathrm{i}\,\mathrm{p}\,\mathrm{i}\,\mathrm{p} \\ \mathrm{p}\,\mathrm{i}\,\mathrm{p}\,\mathrm{p}\,\mathrm{i} \\ \mathrm{p}\,\mathrm{p}\,\mathrm{i}\,\mathrm{p}\,\mathrm{i} \end{bmatrix}$.

Partendo dall'ultima colonna $L = \begin{bmatrix} \mathrm{p} \\ \mathrm{p} \\ \mathrm{p} \\ \mathrm{i} \\ \mathrm{i} \end{bmatrix}$ otteniamo subito la prima: $\begin{bmatrix} \mathrm{i} \\ \mathrm{i} \\ \mathrm{p} \\ \mathrm{p} \\ \mathrm{p} \end{bmatrix}$. Le coppie che consentono di calcolare la seconda colonna sono ‹pi›, ‹pi›, ‹pp›, ‹ip›, ‹ip›. Quando vengono applicate in ordine alfabetico alla prima colonna, formano la matrice incompleta $\begin{bmatrix} \mathrm{i}\,\mathrm{p}\,?\,?\,\mathrm{p} \\ \mathrm{i}\,\mathrm{p}\,?\,?\,\mathrm{p} \\ \mathrm{p}\,\mathrm{i}\,?\,?\,\mathrm{p} \\ \mathrm{p}\,\mathrm{i}\,?\,?\,\mathrm{i} \\ \mathrm{p}\,\mathrm{p}\,?\,?\,\mathrm{i} \end{bmatrix}$. Iterando con le triple e le quadruple si ottiene la matrice completa.

Notiamo che nell'ultima colonna ci sono molte ripetizioni contigue, fatto che viene sfruttato nell'algoritmo di codifica.

✎ .....

Pseudocodice 1.37: Burrows-Wheeler – Codifica. Sia $\mathcal{A}$ l'alfabeto primario. Abbiamo l'ultima colonna `L` come *input*. Il seguente frammento è solo la preparazione per la codifica.

```
for i := 1 to |L| do
  R[i] := indice di L[i] in A;
  Porta l'R[i]-esimo carattere di A in posizione 0 e sposta a destra tutti gli altri;
od
```

..... ✎

Grazie al frammento precedente, le lettere probabili staranno sempre "all'inizio" di $\mathcal{A}$, quindi si può creare un codice di Huffman "fisso".

1.2.5 LV-LV

Alcuni dei codici reali che abbiamo incontrato coniugano una prima parte LV-B con un *postprocessing* B-LV. Allora possiamo costruire un codice

$$\xrightarrow{\text{LV}} \boxed{\text{Tunstall}} \xrightarrow{\text{B}} \boxed{\text{Huffman}} \xrightarrow{\text{LV}}$$

sfruttando i codici ottimi delle due tipologie. Il tasso sarà $R = \frac{\text{EL}}{\text{EN}}$, che è la forma più generale.

Partendo da un $\mathcal{A}$ con distribuzione di probabilità P, applicando j volte Tunstall, otterremo un $\mathcal{M}_j$ con distribuzione S_j. Allora

$$\begin{aligned} R_j = \frac{\text{EL}_j}{\text{EN}_j} = \frac{H(S_j)+\xi}{\text{EN}_j} \text{ per qualche } 0 \le \xi < 1 \Rightarrow \\ \Rightarrow R_j = \frac{\text{EN}_j \cdot H(P)+\xi}{\text{EN}_j} = H(P) + \frac{\xi}{\text{EN}_j} < H(P) + \frac{1}{\text{EN}_j}\,, \end{aligned}$$

quindi questo codice tende all'entropia.

1.3 B-B

Il codice sarà una funzione $\varphi\colon \mathcal{A}^n \to \mathcal{B}^\ell$, quindi il tasso è $R = \frac{\ell}{n}$. Se n è fissato, affinché tutte le n-uple possano essere codificate, deve valere $D^\ell \ge k^n \Rightarrow \ell \ge n \log_D k \Rightarrow \ell = \lceil n \log_D k \rceil$ (perché lo vogliamo minimo). Un codice B-B è solo una *riscrittura*, non una compressione.

Shannon non si fermò qui e andò oltre: se accettassi qualche errore? Ammetto che φ non sia iniettiva. Sia ψ l'inversa di φ, definita "bene" dove possibile.

✎ .....

Definizione 1.38. La PROBABILITÀ DI ERRORE «p_{err}» è definita come $p_{\text{err}} \coloneqq \mathcal{P}\big(\psi \circ \varphi(X) \ne X\big)$.

..... ✎

Ammettendo la probabilità di errore, introduciamo «$\mathcal{C}$», sottoinsieme di $\mathcal{A}^n$ contenente le n-uple correttamente codificate. Inoltre passiamo a considerare un tasso semplificato $R \coloneqq \frac{\log|\mathcal{C}|}{n}$, in cui rispetto a quello normale si ignora la base del logaritmo e si elimina la parte estremo superiore.

Cerco allora il più piccolo $\mathcal{C}$ tale $\mathcal{P}(\mathcal{C})$ sia alta. La soluzione si ottiene ordinando le n-uple di $\mathcal{A}^n$ per probabilità decrescente e scegliendo il minimo numero di n-uple altamente probabili tali che la somma delle loro probabilità sia maggiore o uguale a $1 - p_{\text{err}}$, con p_{err} fissata in partenza.

Sia $F = (f_1, \dots, f_k)$ una distribuzione di probabilità. Partizioniamo $\mathcal{A}^n$ in tipi. Fissiamo un tipo T caratterizzato dalle probabilità $(n_1, \dots, n_k)$.

$$F^n(T) = \sum_{x \in T} F^n(x) = F^n(\overset{\in T}{x})|T|. \qquad F^n(x) = f_1^{n_1} \cdots f_k^{n_k} = \exp_2\left(\sum_{i=1}^{k} n_i \log f_i\right)\,.$$

Ciò vale per ogni F, in particolare per $F = \left(\frac{n_1}{n}, \dots, \frac{n_k}{n}\right)$, ossia la frequenza del tipo T. Allora

$$F^n(x) = \exp_2\left(n \sum_{i=1}^{k} \frac{n_i}{n} \log f_i\right) = \exp_2\left(n \sum_{i=1}^{k} f_i \log f_i\right) = 2^{-nH(F)} \Rightarrow$$
$$\Rightarrow F^n(T) = 2^{-nH(F)}|T| \leq 1 \Rightarrow |T| \leq 2^{nH(F)} .$$

✎ .....

Lemma 1.39. $\dfrac{n!}{m!} \leq n^{n-m}$.

Dimostrazione.

- $(\!|n \geq m|\!)$ $\dfrac{n!}{m!} = \dfrac{\overbrace{n\,(n-1)\cdots(m+1)}^{n-m \text{ fattori}}\cancel{m!}}{\cancel{m!}} \leq n^{n-m}$;
- $(\!|n < m|\!)$ $\dfrac{n!}{m!} = \dfrac{\cancel{n!}}{\underbrace{m\,(m-1)\cdots(n+1)}_{m-n \text{ fattori}}\cancel{n!}} \leq n^{n-m}$.

..... ✎

Consideriamo il tipo T caratterizzato da $(n_1, \dots, n_k)$, il tipo T_j caratterizzato da $(m_1, \dots, m_k)$ e la distribuzione $F = (f_1, \dots, f_k) = \left(\frac{n_1}{n}, \dots, \frac{n_k}{n}\right)$.

$$\frac{F^n(T_j)}{F^n(T)} = \frac{|T_j| \cdot F^n\left(\underset{\in T_j}{y}\right)}{|T| \cdot F^n\left(\underset{\in T}{x}\right)} = \frac{\dfrac{\cancel{n!}}{m_1! \cdots m_k!} \cdot f_1^{m_1} \cdots f_k^{m_k}}{\dfrac{\cancel{n!}}{n_1! \cdots n_k!} \cdot f_1^{n_1} \cdots f_k^{n_k}} = \frac{n_1! \cdots n_k!}{m_1! \cdots m_k!} \cdot \frac{\left(\frac{n_1}{n}\right)^{m_1} \cdots \left(\frac{n_k}{n}\right)^{m_k}}{\left(\frac{n_1}{n}\right)^{n_1} \cdots \left(\frac{n_k}{n}\right)^{n_k}} =$$
$$= \underbrace{\left(\frac{n_1!}{m_1!} \cdot n_1^{(m_1-n_1)}\right)}_{\leq 1} \cdots \underbrace{\left(\frac{n_k!}{m_k!} \cdot n_k^{(m_k-n_k)}\right)}_{\leq 1} \cdot \underbrace{\frac{n^{(n_1+\cdots+n_k)}}{n^{(m_1+\cdots+m_k)}}}_{=1} \leq 1 \Rightarrow F^n(T_j) \leq F^n(T) ,$$

ossia se uso la probabilità uguale alla frequenza di un tipo T, allora T ha probabilità massima. Allora

$$1 = \sum_{j=1}^{\gamma_n} F^n(T_j) \leq \gamma_n F^n(T) \leq (n+1)^k \cdot 2^{-nH(F)}|T| =$$
$$= \exp_2\Big(-n\big(H(F) - \tfrac{k}{n}\log(n+1)\big)\Big)|T| \Rightarrow |T| \geq 2^{n(H(F)-\xi)} ,$$

con ξ infinitesimo. Complessivamente si ha $|T| \approx 2^{nH(F)}$.

Rammentiamo che una v.a. X ha distribuzione BINOMIALE se $\mathcal{P}(X = x) = \binom{n}{x} p^x (1-p)^{n-x}$, con $p \in]0, 1[$. Il VALORE ATTESO «μ_X» vale np e la VARIANZA «σ_X^2» vale $np(1-p)$.

✎ .....

Lemma 1.40: Disuguaglianza di Čebyšëv. $\forall \varepsilon > 0 \colon \mathcal{P}\big(|X - \mu_X| \geq \varepsilon\big) \leq \dfrac{\sigma_X^2}{\varepsilon^2}$.

..... ✎

Indichiamo con «$N(a_i \parallel x)$» la v.a. che rappresenta il numero di occorrenze di a_i nella n-upla x. Essa avrà valore atteso np_i e varianza $np_i(1-p_i)$. La v.a. derivata $\frac{N(a_i \| x)}{n}$ avrà di conseguenza valore atteso p_i e varianza $\frac{p_i(1-p_i)}{n}$.

✎ .....

Definizione 1.41. Sia «δ» una successione di numeri reali positivi $\delta_1, \delta_2, \delta_3, \dots$ tale che $\lim\limits_{n\to+\infty} \delta_n = 0$ e che $\lim\limits_{n\to+\infty} \delta_n \sqrt{n} = +\infty$. Definiamo l'insieme «$\mathcal{T}_n$» delle n-uple di δ-TIPICHE nel seguente modo:

$$\mathcal{T}_n = \left\{ x \in \mathcal{A}^n \colon \forall i \colon \left| \frac{N(a_i \parallel x)}{n} - p_i \right| \leq \delta_n \right\} .$$

Quando δ_n è piccolo (n è grande), essere δ-tipico significa che la frequenza dei simboli è circa uguale alla probabilità.

$$\mathcal{P}\left(\left|\frac{N(a_i \parallel x)}{n} - p_i\right| > \delta_n\right) \overset{\text{Čeb.}}{\leq} \frac{p_i(1-p_i)}{n\,\delta_n^2} = \frac{\overbrace{p_i(1-p_i)}^{\text{parabola}}}{(\sqrt{n}\,\delta_n)^2} \leq \frac{1}{4}\frac{1}{(\sqrt{n}\,\delta_n)^2}\,,$$

che tende a 0 per n che tende a $+\infty$. Allora $\mathcal{P}(x \notin \mathcal{T}_n) \leq \frac{k}{4}\frac{1}{(\sqrt{n}\,\delta_n)^2}$ e $\mathcal{P}(x \in \mathcal{T}_n) \geq 1 - \frac{k}{4}\frac{1}{(\sqrt{n}\,\delta_n)^2}$.

Preso un $\mathcal{A}^n$ e il suo $\mathcal{T}_n$, un tipo T_j sta o tutto dentro o tutto fuori a $\mathcal{T}_n$. Sia «$\mathcal{I}_n$» l'insieme degli indici dei tipi inclusi in $\mathcal{T}_n$. $|\mathcal{T}_n| = \left|\bigcup_{j\in\mathcal{I}_n} T_j\right| \overset{\text{disg.}}{=} \sum_{j\in\mathcal{I}_n} |T_j| \leq \sum_{j\in\mathcal{I}_n} 2^{nH(F_j)}$. Poiché se $T_j \subseteq \mathcal{T}_n \Rightarrow F_j \approx P$ allora $|\mathcal{T}_n| \leq (n+1)^k\, 2^{nH(P)} = 2^{n(H(P)+\xi)}$, con ξ infinitesimo. Inoltre vale $2^{n(H(P)-\xi')} \leq |\mathcal{T}_n|$, con ξ' infinitesimo. Allora complessivamente $R = \frac{\log|\mathcal{T}_n|}{n} \approx \frac{\not{n}\,H(P)}{\not{n}} = H(P)$.

Questa è la parte diretta del 2º Teorema di Shannon. Anche usando un codice B-B con errore non possiamo fare meglio dell'entropia.

1.4 Complessità di Kolmogorov

Definizione 1.42. Sia U un calcolatore universale (o l'interprete di un linguaggio di programmazione). Se P è un programma per U, denotiamo con «$[\![P]\!]^U()$» l'esecuzione, con eventuale *output*, di P su U. La COMPLESSITÀ DI KOLMOGOROV di una stringa x rispetto a U è $K_U(x) \coloneqq \min\{|P|\colon [\![P]\!]^U() = x\}$.

Supponiamo che P sia un programma che genera x, ossia $[\![P]\!]() = x$. Come faccio a capire se è il più piccolo? Dovrei provare $[\![Q]\!]()$ per ogni Q di lunghezza inferiore a P, ma non lo posso fare, a causa dell'*halting problem*. Per colpa di questo problema, della complessità di Kolmogorov avremo solo una limitazione superiore, salvo rarissimi casi. L'idea è comunque quella di usare la complessità di Kolmogorov per comprimere.

Lemma 1.43. Se A e B sono due l.d.p. allora $\forall x\colon K_A(x) \leq K_B(x) + c_{AB}$, dove c_{AB} è una costante che dipende solo dai due linguaggi.

Dimostrazione. Supponiamo $K_B(x) = k$, ovvero esiste un programma P nel linguaggio B tale che $[\![P]\!]^B() = x \wedge |P| = k$. So che esiste un programma I scritto in A che fa l'interprete del linguaggio B, ossia $\forall Q$ programma in $B, \vec{y}$ input$\colon [\![I]\!]^A(Q, \vec{y}) = [\![Q]\!]^B(\vec{y})$. Il programma "composto" (I, P) è un programma in A, lungo $|I| + |P|$ e che genera x. Allora $K_A(x) \leq |I| + K_B(x)$.

Corollario 1.44. Se A e B sono l.d.p. allora $\left|K_A(x) - K_B(x)\right| \leq c$, dove c è costante.

D'ora in poi astrarremo dal l.d.p., poiché è indifferente. Inoltre tutte le «c» denoteranno costanti sempre diverse e fissiamo «n» pari alla lunghezza della stringa x di volta in volta considerata.

Definizione 1.45. La complessità di Kolmogorov CONDIZIONATA di una stringa x è definita come $K(x \mid n) \coloneqq \min\{|P|\colon [\![P]\!](n) = x\}$.

Esempio 1.46. La stringa $x = \langle 01\ldots01\rangle$ può essere generata dal programma

```
for i := 1 to ◇ do
  write("01");
od
```

Al posto di «◇» il programma senza *input* avrà il valore $\frac{n}{2}$ in forma di stringa, mentre il programma con la lunghezza in *input* avrà semplicemente «`n / 2`». Quindi $K(x) \leq c + \log \frac{n}{2}$ mentre $K(x \mid n) \leq c$.

✎

✎

Lemma 1.47.

1. $\forall x\colon K(x \mid n) \leq n + c$.
2. $\forall x\colon K(x) \leq K(x \mid n) + 2\log n + c$.

Dimostrazione.

1. Sia x una stringa "senza struttura". Allora il programma che la genera è banalmente «`write(`x`);`», che ha lunghezza $c + 8n$, perché i *bits* salvati sono *bytes*.
   Allora si può creare un comando «`writeb(n, x)`» che stampa *bit* per *bit* una stringa codificata in forma di *bytes*. In questo modo il programma ha finalmente lunghezza $n + c$.
   Si noti che è necessario scrivere il numero di *bits* da leggere perché nella sequenza di *bits* codificati come *bytes* ci potrebbe essere un *byte* che si confonde col delimitatore della stringa.
2. Sfruttiamo il punto precedente. Il programma senza *input* è nella forma

   ```
   n := n;
   writeb(n, x);
   ```

   la cui lunghezza è tuttavia $8\log n + c$.
   Allora similmente a prima possiamo creare un comando «`bass(n,` n`)`» (*binary assignment*) in cui la stringa che rappresenta n è scritta di nuovo in *bits* codificati come *bytes*. Per capire dove finisce la stringa possiamo rappresentare n raddoppiando ogni singolo *bit* e aggiungere ‹01› come segnale.
   Ora il programma ha finalmente lunghezza $K(x) \leq K(x \mid n) + 2\log n + c$.

✎

Possiamo migliorare le prestazioni della codifica di n nel secondo punto del teorema precedente:

2	3	4	10	$n = 1023$		valore decimale
10	11	100	1010	0	1111111111	stringa
2	2	3	4		10	*bits* della rappresentazione

in cui lo ‹0› isolato è il segnale di fine. Questa stringa è lunga

$$\log n + \log\log n + \log\log\log n + \cdots + 3 \approx \log^* n \leq 2\log n\,.$$

Prendiamo $\mathcal{A} = \{0, 1\}$ alfabeto binario e consideriamo $\mathcal{A}^n$, che dividiamo in tipi. Allora $\gamma_n = n + 1$ esattamente. La cardinalità di un tipo identificato da (n_1, n_2) è $\binom{n}{n_1} = \binom{n}{n_1, n_2} = \binom{n}{n_2}$ e sappiamo anche $\frac{2^{nH(F)}}{\gamma_n} \leq |T| \leq 2^{nH(F)}$, con $F = \left(\frac{n_1}{n}, \frac{n_2}{n}\right) = \left(\frac{n_1}{n}, 1 - \frac{n_1}{n}\right)$ frequenza del tipo T.

L'entropia di una distribuzione binaria $H(x, 1-x)$ viene spesso denotata con «$H_0(x)$» o con «h_x». Abbreviamo con «u» il numero di ‹1› di un tipo T fissato, esprimibile anche come $u \coloneqq \sum_{i=1}^{n} x_i$. Vale che

$$\frac{2^{nH_0(u/n)}}{n+1} \leq \underbrace{\binom{n}{u}}_{=|T|} \leq 2^{nH_0(u/n)}\,.$$

Un programma che dice «Genera tutte le stringhe di n elementi con u ‹1›, ordinale lessicograficamente, stampa la i-esima» è sicuramente più compatto di un `write` se n è grande. Studiamo quindi $K(x \mid n)$ per il nuovo programma.

$$K(x \mid n) \leq \underbrace{c}_{\text{resto}} + \underbrace{\log(n+1)}_{u} + \underbrace{\log \binom{n}{u}}_{i} \leq c + \log(n+1) + nH_0\left(\frac{u}{n}\right) .$$

Confrontando questa complessità con quella del Teorema 1.47, abbiamo che per $u \approx \frac{n}{2} \Rightarrow H_0\left(\frac{u}{n}\right) \approx 1$ quindi conviene la `write`, altrimenti conviene la nuova codifica.

Proveremo ora a usare le idee impiegate per K come algoritmo di compressione. Associamo $\varphi(x)$ al più piccolo programma che stampa x. L'algoritmo ottenuto è B-LV e non realizzabile, sempre per l'impossibilità di determinare il programma più breve.

Proposizione 1.48. $H(P) \leq \frac{\mathrm{EL}}{n} \leq H(P) + \xi_n$, con ξ_n infinitesimo.

Dimostrazione. Programmi che generano stringhe diverse non possono essere uno prefisso dell'altro, quindi questo codice è a prefisso, è UD e dunque $\frac{\mathrm{EL}}{n} \geq H(P)$.

Innanzitutto vale $\dfrac{\mathrm{EL}}{n} = \dfrac{\sum\limits_{x \in \mathcal{A}^n} P^n(x) K(x \mid n)}{n}$. Usando una meta-notazione, scriviamo che

$$EK(x \mid n) \leq E\left(c + \log(n+1) + nH_0\left(\frac{\sum_{i=1}^n X_i}{n}\right)\right) = c + \log(n+1) + nEH_0\left(\frac{\sum_{i=1}^n X_i}{n}\right) ,$$

dove ogni X_i è una v.a. che è ‹1› con probabilità p. Allora, per la disuguaglianza di Jensen, si ha che

$$EH_0\left(\frac{\sum_{i=1}^n X_i}{n}\right) \leq H_0\left(\frac{E\sum_{i=1}^n X_i}{n}\right) = H_0\left(\frac{\cancel{n}p}{\cancel{n}}\right) = H(p, 1-p) = H(P) ,$$

da cui

$$EK(x \mid n) \leq c + \log(n+1) + nH(P) \Rightarrow \frac{\mathrm{EL}}{n} \leq \frac{c + \log(n+1)}{n} + \frac{\cancel{n}H(P)}{\cancel{n}} = H(P) + \xi_n .$$

2 Codici di canale

L'obiettivo è proteggere l'informazione da errori *casuali*, ossia non provocati da malintenzionati. L'idea generale consiste nell'introdurre delle *ridondanze sistematiche*.

Definizione 2.1. Il TASSO DI TRASMISSIONE è $R := \frac{\log|\mathcal{C}|}{n}$, dove $\mathcal{C}$ è l'alfabeto del codice e n è in numero di simboli del messaggio codificato.

In generale il metodo di decodifica è quello di MASSIMA VEROSIMIGLIANZA con un insieme di valori ammissibili. Il problema «Trovare un insieme di n-uple con e elementi che differiscono ognuno per almeno d *bits*» è $\mathbb{NP}$-completo.

Esempio 2.2.

- Nel *BIT* DI PARITÀ si aggiunge un *bit* corrispondente allo *xor* degli altri *bit*. Con un alfabeto di n-uple di *bits*, noi inviamo $n+1$ bit, con un tasso $R = \frac{7}{8}$. Se nella trasmissione avviene un numero dispari di errori si riconosce che c'è stato un errore, ma non lo si può correggere.

- Nel codice a RIPETIZIONE si ripete ogni *bit* r volte. Con un alfabeto di n-uple inviamo rn bit, con un tasso $R = \frac{1}{r}$. Anche se "sprechiamo" più *bits*, possiamo correggere fino a $\lfloor \frac{r}{2} \rfloor$ errori per ogni gruppo di r bit.

✎

✎

Definizione 2.3. Una matrice quadrata di lato n è di HADAMARD se $H_{ij} \in \{-1, 1\}$ e $HH^T = nI_n$.

✎

✎

Esempio 2.4. $\begin{pmatrix}1\end{pmatrix}$ e $\begin{pmatrix}-1\end{pmatrix}$ sono di Hadamard.

✎

✎

Lemma 2.5. Se H è di Hadamard allora $\begin{pmatrix} H & H \\ H & -H \end{pmatrix}$ è di Hadamard.

Dimostrazione. Se H è di Hadamard (di lato n) allora $H_{ij} \in \{-1, 1\}$ e ciò vale anche per la nuova matrice.

$$\begin{pmatrix} H & H \\ H & -H \end{pmatrix} \begin{pmatrix} H^T & H^T \\ H^T & -H^T \end{pmatrix} = \begin{pmatrix} HH^T + HH^T & HH^T - HH^T \\ HH^T - HH^T & HH^T + HH^T \end{pmatrix} = \begin{pmatrix} 2nI_n & 0 \\ 0 & 2nI_n \end{pmatrix} = 2nI_{2n} \,.$$

✎

Indichiamo con «H_{2^n}» la matrice di Hadamard ottenuta dalla matrice $\begin{pmatrix}1\end{pmatrix}$ con n applicazioni del lemma precedente.

✎

Lemma 2.6. Due qualsiasi righe distinte di H_{2^n} si differenziano in 2^{n-1} punti.

Dimostrazione. Prendiamo due righe a caso $(a_1, \ldots, a_{2^n})$ e $(b_1, \ldots, b_{2^n})$ distinte. Poiché $H_{2^n}(H_{2^n})^T = 2nI_{2^n}$, sappiamo che $\sum_{i=1}^{2^n} a_i b_i$ può valere 0 oppure $2n$, il secondo dei quali vale solo quando la riga è la stessa. Ma siccome le righe sono diverse per ipotesi, la sommatoria è nulla. Affinché questo accada a una somma di prodotti di 1 e -1, significa che metà sono uguali e metà diversi.

✎

Allora alla NASA hanno pensato di costruire la matrice $\begin{pmatrix} H_{32} \\ -H_{32} \end{pmatrix}$, che ha 64 righe e 32 colonne. Prendendo una riga di quella sopra e una di quella sotto, possono differire in 16 o 32 punti. In questo modo hanno scelto un codice $\mathcal{C}$ di 64 elementi per il quale vengono usati 32 bit. Allora $R = \frac{6}{32} \approx \frac{1}{5}$, potendo però correggere fino a 7 errori consecutivi.

Consideriamo

$$\xrightarrow{m} \boxed{\text{CC}} \xrightarrow{x} \boxed{\text{canale}} \xrightarrow{y} \boxed{\text{DC}} \xrightarrow{\hat{m}}$$

con $x \in \mathcal{X} \coloneqq \{a_1, \ldots, a_k\}$ e $y \in \mathcal{Y} \coloneqq \{b_1, \ldots, b_h\}$. Gli alfabeti sono teoricamente diversi perché in $\mathcal{Y}$ potrebbero esserci valori che indicano «indefinito», ma tipicamente $k = h = 2$. Un CANALE viene descritto con una matrice *stocastica* Γ tale che $\Gamma_{ij} \coloneqq \mathcal{P}(Y = b_j \mid X = a_i)$.

✎

Esempio 2.7. Il CANALE SIMMETRICO BINARIO (CSB) ha $\mathcal{X} = \mathcal{Y} = \{0, 1\}$ e $\Gamma = \begin{pmatrix} 1-\varepsilon & \varepsilon \\ \varepsilon & 1-\varepsilon \end{pmatrix}$, con ε probabilità di errore.

✎

Scriveremo «$\Gamma^n(x\,|\,y)$» per indicare $\mathcal{P}(Y^n = y\,|\,X^n = x)$. Nell'ipotesi di assenza di memoria di un canale, la probabilità della concatenazione è il prodotto delle probabilità. Scriveremo anche «$\Gamma\big(\{x_1, \dots, x_t\} \;\big|\; y\big)$» per intendere $\sum_{i=1}^{t} \Gamma^n(x_i \,|\, y)$.

Definizione 2.8. Dato un insieme di messaggi «$\mathcal{M}$», il CODIFICATORE DI CANALE (CC) è una $f\colon \mathcal{M} \to \mathcal{X}^n$ e il DECODIFICATORE DI CANALE (DC) è una $g\colon \mathcal{Y}^n \to \mathcal{M} \cup \{\perp\}$ (codifica MORBIDA) oppure $g\colon \mathcal{Y}^n \to \mathcal{M}$ (codifica RIGIDA). Il simbolo «$\perp$», opzionale, indica che non ho saputo decodificare.

Quindi $R \coloneqq \frac{\log|\mathcal{M}|}{n}$ e il $\mathcal{C}$ usato prima non è altro che $f(\mathcal{M}) \subseteq \mathcal{X}^n$. Ci piacerebbe che R fosse alto (il valore massimo è 1).

Definizione 2.9. La PROBABILITÀ DI ERRORE di un messaggio m è

$$p_{\text{err}}(f, g, m) \coloneqq \mathcal{P}\big(g(Y) \neq m \;\big|\; X = f(m)\big) = \mathcal{P}\big(Y \notin g^{-1}(m) \;\big|\; X = f(m)\big) = 1 - \Gamma^n\big(g^{-1}(m) \;\big|\; f(m)\big)\,.$$

Per la probabilità di errore di un codice possiamo scegliere il massimo, la media o la media pesata delle probabilità di errore dei messaggi. Quest'ultima è improponibile perché non possiamo conoscere a priori le probabilità dei messaggi. Per il CSB sono tutte uguali. Quindi in generale non viene definita.

Esempio 2.10. Come canale consideriamo il CSB.

- $f(x) = x \qquad g(x) = x$
  $R = 1 \qquad p_{\text{err}}(f, g, \text{‹0›}) = \Gamma(\text{‹1›} \mid \text{‹0›}) = \varepsilon \qquad p_{\text{err}}(f, g, \text{‹1›}) = \Gamma(\text{‹0›} \mid \text{‹1›}) = \varepsilon$
- $f(x) = xxx$

$$\begin{aligned} p_{\text{err}}(f, g, \text{‹0›}) &= \Gamma^3\big(\{\text{‹110›}, \text{‹011›}, \text{‹101›}, \text{‹111›}\} \;\big|\; \text{‹000›}\big) = \\ &= \Gamma^3(\text{‹110›} \mid \text{‹000›}) + \Gamma^3(\text{‹011›} \mid \text{‹000›}) + \Gamma^3(\text{‹101›} \mid \text{‹000›}) + \Gamma^3(\text{‹111›} \mid \text{‹000›}) = \\ &= 3\varepsilon^2(1 - \varepsilon) + \varepsilon^3 = \varepsilon^2(3 - 2\varepsilon) = p_{\text{err}}(f, g, \text{‹1›})\,. \end{aligned}$$

  Rispetto al caso precedente, la probabilità di errore è calata.
- Con 5 ripetizioni abbiamo $R = \frac{1}{5}$ e $p_{\text{err}} = \varepsilon^3(6\varepsilon^2 - 15\varepsilon + 10)$, che cala ulteriormente.

Possiamo migliorare la probabilità di errore, ferma restando Γ, senza perdere troppa velocità?

Definizione 2.11. La CAPACITÀ di un canale Γ si definisce come $C(\Gamma) \coloneqq \max_P I(X \wedge Y)$, dove P è la distribuzione di probabilità di X.

Teorema 2.12: Teorema di Shannon (per i codici di canale).

1. Esiste una famiglia di codici tale che la sua probabilità di errore tende a 0 per R che tende a $C(\Gamma)$.
2. Se un codice ha $R > C(\Gamma)$ allora $p_{\text{err}} > \eta$, con $\eta > 0$.

Esempio 2.13.

- Sia $\Gamma = \begin{pmatrix} 0.4 & 0.6 \\ 0.4 & 0.6 \end{pmatrix}$. Questa matrice ha le righe tutte uguali e a somma 1. L'uscita è indipendente dall'ingresso, infatti questo canale viene detto INUTILE (o "CAVO ROTTO"). Poiché

$$p_{ij} = \mathcal{P}(X = a_i \wedge Y = b_j) = \mathcal{P}(Y = b_j \mid X = a_i)\mathcal{P}(X = a_i) = \Gamma_{ij} p_i$$

e $\Gamma_{ij} = q_j$ (non dipende da i) allora

$$I(X \wedge Y) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{p_i q_j} = \sum_i \sum_j \log \underbrace{\frac{\cancel{\Gamma_{ij} p_i}}{\cancel{p_i q_j}}}_{=1} = 0$$

e in definitiva $C(\Gamma) = 0$.

- Sia $\Gamma = \begin{pmatrix} 0 & 0.35 & 0.65 \\ 1 & 0 & 0 \end{pmatrix}$. Questa matrice ha in ogni colonna al più un elemento non nullo. Un canale simile viene detto SENZA RUMORE, infatti dall'uscita posso determinare sempre l'ingresso. Indichiamo con «N_i» l'insieme degli indici di Y che sono "nomi" per a_i. Poiché $p_{ij} = p_{i|j} q_j$ allora

$$\begin{aligned} I(X \wedge Y) &= \sum_i \sum_j \log \frac{p_{ij}}{p_i q_j} = \sum_i \sum_{j \in N_i} p_{ij} \log \frac{p_{ij}}{p_i q_j} = \\ &= \sum_i \sum_{j \in N_i} q_j \log \frac{1}{p_i} = -\sum_i p_i \log p_i = H(P)\,. \end{aligned}$$

da cui $C(\Gamma) = \max_P H(P) = 1$.

- Sia $\Gamma = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$. Questa matrice ha in ogni riga esattamente un elemento non nullo. Un canale simile viene detto DETERMINISTICO. Pur non avendo rumore, introduce errore in decodifica. Poiché vale il ragionamento simmetrico rispetto al caso precedente, abbiamo $I(X \wedge Y) = H(Q)$, da cui $C(\Gamma) = \max_P H(Q) = \log_{|\mathcal{X}|} |\mathcal{Y}|$.

- Consideriamo il CSB e sfruttiamo $I(X \wedge Y) = H(Y) - H(Y \mid X)$.
  Abbiamo $H(Y \mid X) = \sum_{x \in \{0,1\}} P(x) H(Y \mid X = x)$ e

$$\begin{aligned} H(Y \mid X = \langle 0 \rangle) = &-\mathcal{P}(Y = \langle 0 \rangle \mid X = \langle 0 \rangle) \log \mathcal{P}(Y = \langle 0 \rangle \mid X = \langle 0 \rangle) - \\ &-\mathcal{P}(Y = \langle 1 \rangle \mid X = \langle 0 \rangle) \log \mathcal{P}(Y = \langle 1 \rangle \mid X = \langle 0 \rangle) = \\ &= -(1-\varepsilon)\log(1-\varepsilon) - \varepsilon \log \varepsilon = h_\varepsilon = H(Y \mid X = \langle 1 \rangle)\,, \end{aligned}$$

quindi $H(Y \mid X) = P(\langle 0 \rangle) h_\varepsilon + P(\langle 1 \rangle) h_\varepsilon = h_\varepsilon$. Allora $I(X \wedge Y) = H(Y) - h_\varepsilon$ e inoltre $C(\Gamma) = \max_P \big(H(Y) - h_\varepsilon\big)$, che è massima per Q uniforme e vale $C(\Gamma) = 1 - h_\varepsilon$.

. . . . . ✎

✎ . . . . . .

Esercizio 2.14. Dimostrare che nel CANALE SIMMETRICO q-ARIO (CSq) la capacità vale $C(\Gamma) = 1 - h_\varepsilon - \varepsilon \log(q-1)$ con

$$\Gamma = \underbrace{\begin{pmatrix} 1-\varepsilon & \eta & \cdots & \eta \\ \eta & 1-\varepsilon & \cdots & \eta \\ \vdots & \vdots & \ddots & \vdots \\ \eta & \eta & \cdots & 1-\varepsilon \end{pmatrix}}_{q \times q},$$

dove $\eta = \frac{\varepsilon}{q-1}$.

Abbiamo un messaggio m che diventa $x = f(m)$, che a sua volta diventa y passando attraverso il canale. La f è solo una riscrittura, quindi devo trovare x che massimizza $\mathcal{P}(Y = y \mid X = x)$ oppure $\mathcal{P}(Y = y \mid X = x)\mathcal{P}(X = x)$, ossia $\Gamma^n(y \mid x)$ oppure $\Gamma^n(y \mid x)P(x)$ in termini di Γ. Questi due criteri si chiamano rispettivamente MASSIMA VEROSIMIGLIANZA e MASSIMA PROBABILITÀ A POSTERIORI. La seconda è più precisa, ma siccome non abbiamo $P(x)$, l'unica decodifica possibile può essere quella a massima verosimiglianza (come già accennato).

Definizione 2.15. La DISTANZA DI HAMMING «$\mathrm{d_H}(x, y)$» è definita come il numero di simboli diversi tra x e y.

Allora nel CSB $\mathcal{P}(Y = y \mid X = x) = \varepsilon^{\mathrm{d_H}(x,y)}(1-\varepsilon)^{n-\mathrm{d_H}(x,y)} = (1-\varepsilon)^n\left(\frac{\varepsilon}{1-\varepsilon}\right)^{\mathrm{d_H}(x,y)}$, in cui il primo fattore è costante e il secondo è un esponenziale con base minore di 1, quindi devo minimizzare l'esponente $\mathrm{d_H}(x, y)$.

Definizione 2.16. Definiamo la DISTANZA MINIMA di un codice $\mathcal{C}$ come $d_{\min}(\mathcal{C}) \coloneqq \min\limits_{\substack{x,y\in\mathcal{C}\\ x\neq y}} \mathrm{d_H}(x, y)$.

Se il codice è fissato, scriveremo in breve «$d_{\min}$».

Lemma 2.17. Se $d_{\min} \geq 2t + 1$ (per $t \in \mathbb{N}$) allora so correggere tutte le configurazioni con t errori.

Soluzione. Sia y una stringa uscita dal canale, con $\mathrm{d_H}(x, y) \leq t$ per qualche $x \in \mathcal{C}$. Per ogni altra $x' \in \mathcal{C}$ vale $\mathrm{d_H}(x', y) > t$, altrimenti $\mathrm{d_H}(x, x') < 2t + 1$, contro l'ipotesi di minimalità di $d_{\min}$.

L'inverso di questo lemma afferma che se $d_{\min} \leq 2t$ allora esiste almeno una configurazione di t errori che non viene corretta.

Esercizio 2.18. Nel CSB con decodifica a minima $\mathrm{d_H}$, se $\frac{d_{\min}}{n} > 2\varepsilon$ allora so correggere tutte le n-uple δ-tipiche, ovvero p_{err} tende a 0.

2.1 Limitazioni asintotiche

Ci concentriamo su $\mathcal{X}$ e su $\mathcal{C} \subseteq \mathcal{X}^n$, indicando con «$q$» e «$M$» le rispettive cardinalità.

Definizione 2.19. Il TASSO DI CORREZIONE è definito come $\lambda \coloneqq \frac{d_{\min}}{n}$.

Naturalmente vogliamo sia R che λ alti.

Esempio 2.20.

	Parità	**Ripetizione**	**Hadamard**
R	$\frac{n-1}{n}$	$\frac{1}{n}$	$\frac{6}{32}$
λ	$\frac{2}{n}$	1	$\frac{1}{2}$

Singleton Prese tutte le n-uple di $\mathcal{C}$ e considerandone solo i primi $n-(d_{\min}-1)$ caratteri, sono tutte diverse. Infatti, se così non fosse, il codice avrebbe distanza minima pari a $d_{\min}-1$, che non ha senso. Allora $q^{n-(d_{\min}-1)} \geq M$ affinché valga la considerazione precedente. Asintoticamente vale

$$q^{n-(d_{\min}-1)} \geq M \Rightarrow \log q^{n-(d_{\min}-1)} \geq \log M \Rightarrow n-(d_{\min}-1) \geq \log M \Rightarrow 1-\frac{d_{\min}}{n}+\frac{1}{n} \geq \frac{\log M}{n} \Rightarrow R \leq 1-\lambda.$$

Plotkin Denotiamo con «D» il valore $\sum_{x_i,x_j \in \mathcal{C}} \mathrm{d_H}(x_i,x_j)$, contenente anche i doppioni. Sicuramente vale $D \geq M(M-1)$. Immaginiamo di incolonnare tutte le n-uple di $\mathcal{C}$ e denotiamo con «u_{ij}» il numero di simboli della colonna i uguali al j-esimo simbolo. Allora

$$D = \sum_{i=1}^{n}\sum_{j=1}^{q} u_{ij}(M-u_{ij}) = M\sum_{i=1}^{n}\underbrace{\sum_{j=1}^{q} u_{ij}}_{=M} - \sum_{i=1}^{n}\sum_{j=1}^{q} u_{ij}^2 = nM^2 - \sum_{i=1}^{n}\sum_{j=1}^{q} u_{ij}^2 \,.$$

✎ .....

Lemma 2.21: Disuguaglianza di Cauchy-Schwartz. $\displaystyle\sum_i a_i^2 \sum_i b_i^2 \geq \left(\sum_i a_i b_i\right)^2 .$

..... ✎

Esaminiamo il secondo addendo:

$$\sum_{i=1}^{n}\sum_{j=1}^{q} u_{ij}^2 = \sum_{i=1}^{n} \frac{\sum_{j=1}^{q} 1^2}{q} \sum_{j=1}^{q} u_{ij}^2 \geq \frac{1}{q}\sum_{i=1}^{n}\left(\underbrace{\sum_{j=1}^{q} u_{ij}}_{=M}\right)^2 = \frac{nM^2}{q} \,.$$

Allora $D \leq nM^2\frac{q-1}{q}$ e dunque asintoticamente

$$M(M-1)d_{\min} \leq nM^2\frac{q-1}{q} \Rightarrow \frac{d_{\min}}{n} \leq \frac{M}{M-1}\frac{q-1}{q} \Rightarrow \lambda \leq \frac{q^{nR}}{q^{nR}-1}\frac{q-1}{q} \Rightarrow \lambda \leq \frac{q-1}{q} \,.$$

Hamming

✎ .....

Definizione 2.22. La SFERA DI HAMMING di centro x e raggio r è definita come $S(x,r) \coloneqq \{y \in \mathcal{X}^n : \mathrm{d_H}(x,y) \leq r\}$. Il suo "volume" «$\big|S(x,y)\big|$» vale $\sum_{i=0}^{r}\binom{n}{i}(q-1)^i$.

..... ✎

Notiamo che le sfere di raggio $t = \left\lfloor \frac{d-1}{2} \right\rfloor$ con elementi di $\mathcal{C}$ come centro non si intersecano mai. Dunque

$$M\big|S(x,t)\big| \leq q^n = M\sum_{i=0}^{t}\binom{n}{i}(q-1)^i \leq q^n \Rightarrow$$

$$\Rightarrow \log M + \log\sum_{i=0}^{t}\binom{n}{i}(q-1)^i \leq n \Rightarrow R \leq 1 - \frac{1}{n}\log\sum_{i=0}^{t}\binom{n}{i}(q-1)^i \,.$$

Gilbert Con un algoritmo *greedy* scelgo una n-upla x_1 e costruisco $S(x_1,d-1)$, per qualche d. Poi scelgo un'altra n-upla x_2 tra quelle non appartenenti alla sfera e reitero fermandomi quando $i\big|S(x,d-1)\big| > q^n$, per qualche x. Questa limitazione afferma che qualunque codice con $d_{\min}=d$ è migliore o uguale a questo, ossia $M\sum_{i=0}^{d-1}\binom{n}{i}(q-1)^i \geq q^n \Rightarrow \cdots \Rightarrow R \geq 1 - \frac{1}{n}\log\sum_{i=0}^{d-1}\binom{n}{i}(q-1)^i$.

Calcoli finali Abbiamo lasciato in sospeso le ultime due limitazioni in cui il membro di destra della disuguaglianza è nella forma $\sum_{i=0}^{z} \binom{n}{i}(q-1)^i$, per un certo z.

✎ .....

Definizione 2.23. L'ENTROPIA q-ARIA si definisce come $H_q(p) \coloneqq H\Big(1-p, \underbrace{\frac{p}{q-1}, \dots, \frac{p}{q-1}}_{q-1 \text{ elementi}}\Big)$.

..... ✎

Dalla definizione si ricava che l'entropia q-aria vale

$$\begin{aligned} H_q(p) &= -(1-p)\log(1-p) - \cancel{(q-1)}\frac{p}{\cancel{q-1}}\log\frac{p}{q-1} = \\ &= \underbrace{-(1-p)\log(1-p) - p\log p}_{h_p} + p\log(q-1) = h_p + p\log(q-1)\,. \end{aligned}$$

Sappiamo che $\frac{q^{nH(F)}}{n+1} \le \binom{n}{i} \le q^{nH(F)}$, con $F = \left(\frac{i}{n}, \frac{n-i}{n}\right)$, e possiamo riscrivere $(q-1)^i$ come $\exp_q\big(n\frac{i}{n}\log(q-1)\big)$. Allora

$$\begin{aligned} &\frac{\exp_q\Big(n\big(H_0(\frac{i}{n}) + \frac{i}{n}\log(q-1)\big)\Big)}{n+1} \le \binom{n}{i}(q-1)^i \le \exp_q\Big(n\big(H_0(\tfrac{i}{n}) + \tfrac{i}{n}\log(q-1)\big)\Big) \Rightarrow \\ &\Rightarrow \frac{q^{nH_q(\frac{i}{n})}}{n+1} \le \binom{n}{i}(q-1)^i \le q^{nH_q(\frac{i}{n})}\,. \end{aligned}$$

Considerando la sommatoria $\sum_{i=0}^{z} \binom{n}{i}(q-1)^i$, l'idea è che l'argomento cresce fino a $\frac{n}{2}$, ma non ci si arriva. Nell'approssimazione in cui d non può essere troppo grande rispetto a n, per $i = z$ avremo il massimo, quindi

$$\begin{aligned} &\frac{q^{nH_q(\frac{z}{n})}}{n+1} \le \sum_{i=0}^{z}\binom{n}{i}(q-1)^i \le (z+1)q^{nH_q(\frac{z}{n})} \Rightarrow \\ &\Rightarrow \frac{\cancel{n}H_q\left(\frac{z}{n}\right)}{\cancel{n}} - \underbrace{\frac{\log(n+1)}{n}}_{\text{infinitesimo}} \le \frac{\log\sum_{i=0}^{z}\binom{n}{i}(q-1)^i}{n} \le \underbrace{\frac{\log(z+1)}{n}}_{\text{infinitesimo}} + \frac{\cancel{n}H_q\left(\frac{z}{n}\right)}{\cancel{n}} \Rightarrow \\ &\Rightarrow \frac{1}{n}\log\sum_{i=0}^{z}\binom{n}{i}(q-1)^i \approx H_q\left(\frac{z}{n}\right)\,. \end{aligned}$$

$H_q(y)$ è massima quando $1-y = \frac{y}{q-1} \Rightarrow y = \frac{q-1}{q}$, per cui le due limitazioni che avevamo lasciato in sospeso diventano $R \lesssim 1 - H_q\left(\frac{d-1}{2n}\right) \lesssim 1 - H_q\left(\frac{\lambda}{2}\right)$ e $R \gtrsim 1 - H_q\left(\frac{d-1}{n}\right) \gtrsim 1 - H_q(\lambda)$, rispettivamente.

Uno schema riassuntivo è presentato in Figura 2.1. Non ci si lasci ingannare dal fatto che nel grafico esemplificativo la limitazione ① non intervenga attivamente.

2.2 Codici algebrici

Un CODICE ALGEBRICO è un codice che trasforma k-uple di $\mathcal{M}$ in n-uple di $\mathcal{X}^n$ utilizzando una matrice «G» (di dimensione $k \times n$), ossia $\vec{x} = \vec{m}G$.

Sorgono subito alcune domande:

- Sarà UD?
- Corregge? Quanto corregge?
- Se $\mathcal{X} = \{\text{‹a›}, \text{‹b›}, \dots, \text{‹z›}\}$, cosa significa ‹a› · ‹b›? Posso invertirlo?

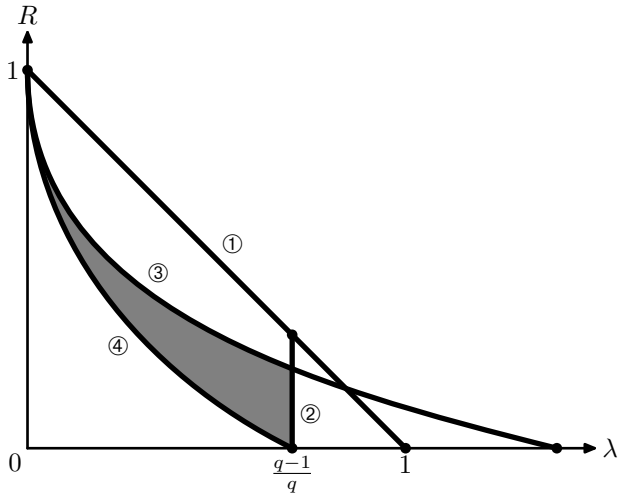

Figura 2.1: Limitazioni asintotiche nei codici di canale:
① Singleton, ② Plotkin, ③ Hamming, ④ Gilbert

Abbiamo bisogno di una struttura algebrica in cui valgano le "solite" regole e dotata di inverso/reciproco per la decodifica. La struttura che ci serve è il *campo finito*.

Teorema 2.24. Se F è un campo finito, allora ha p^r elementi, con p primo e $r \geq 1$.

Perciò possiamo considerare solo $\mathcal{X}$ tale che $|\mathcal{X}| = p^r$, che è comunque unico a meno di isomorfismi.

Definizione 2.25. Un CAMPO DI GALOIS «GF(p^g)» è un campo finito basato sull'algebra dei polinomi di grado $g-1$ a coefficienti in $\mathbb{Z}_p$.

Se $r = 1$ allora il campo finito è $\mathbb{Z}_p$. Siccome lavoreremo con potenze di 2, dovremo generare dei GF(2^g), in cui i polinomi possono essere visti come g-uple.

Esempio 2.26. Queste sono le tabelle dell'addizione e della moltiplicazione per GF(4):

$+$	$\mathbf{0}$	$\mathbf{1}$	$\boldsymbol{x}$	$\boldsymbol{x+1}$
$\mathbf{0}$	0			
$\mathbf{1}$	1	0		
$\boldsymbol{x}$	x	$x+1$	0	
$\boldsymbol{x+1}$	$x+1$	x	1	0

$\cdot$	$\mathbf{0}$	$\mathbf{1}$	$\boldsymbol{x}$	$\boldsymbol{x+1}$
$\mathbf{0}$	0			
$\mathbf{1}$	0	1		
$\boldsymbol{x}$	0	x	$x+1$	
$\boldsymbol{x+1}$	0	$x+1$	1	x

In generale per ottenere la seconda tabella si moltiplica modulo POLINOMIO IRRIDUCIBILE di grado g. In questo caso (per $g = 2$) esso è unico e vale $x^2 + x + 1$.

Per dar luogo a una codifica iniettiva, G deve avere le k righe linearmente indipendenti. Se ciò avviene, G identifica un sottospazio di $\mathcal{X}^n$ di dimensione k. Per comodità suddividiamo G in una parte sinistra quadrata «A_k» e una destra «P», con $\det(A_k) \neq 0$. Se $A_k = I_k$ diremo che G è in FORMA CANONICA. Valgono $R = \frac{k}{n}$, $|\mathcal{M}| = q^k$ e $|\mathcal{X}| = q^n$.

Se non c'è errore, per decodificare bisogna invertire G. Trovo k colonne linearmente indipendenti, le inverto e seleziono le componenti di $\vec{x}$ corrispondenti.

Esempio 2.27.

$$\underbrace{\begin{pmatrix}0 & 1 & 1\end{pmatrix}}_{\vec{m}}\underbrace{\left(\begin{array}{ccc|cc}1 & 1 & 1 & 0 & 0\\0 & 1 & 1 & 0 & 1\\0 & 0 & 1 & 1 & 0\end{array}\right)}_{G=(A|P)} = \underbrace{\left(\begin{array}{ccc|cc}0 & 1 & 0 & 1 & 1\end{array}\right)}_{\vec{x}}$$

$$\underbrace{\begin{pmatrix}0 & 1 & 0\end{pmatrix}}_{\text{parte di } \vec{x}}\underbrace{\begin{pmatrix}1 & 1 & 0\\0 & 1 & 1\\0 & 0 & 1\end{pmatrix}}_{A^{-1}} = \underbrace{\begin{pmatrix}0 & 1 & 1\end{pmatrix}}_{\vec{m}}$$

Osserviamo che se $\vec{x}_1, \vec{x}_2 \in \mathcal{C}$ allora sia $\vec{x}_1 + \vec{x}_2 \in \mathcal{C}$ sia $\vec{x}_1 - \vec{x}_2 \in \mathcal{C}$, infatti $\mathcal{M}$ è fatto in modo tale da contenere tutte le k-uple. Inoltre $\vec{0} \in \mathcal{C}$ sempre.

Definizione 2.28. Il PESO di un codice è $w(\mathcal{C}) \coloneqq \min\limits_{\vec{x}\in\mathcal{C}\setminus\{\vec{0}\}} \mathrm{d_H}(\vec{x}, \vec{0})$.

Osserviamo che $\vec{0} \in \mathcal{C} \Rightarrow w(\mathcal{C}) \geq d_{\min}(\mathcal{C})$ (la premessa serve affinché funzioni anche nei codici non algebrici) e in particolare nei codici algebrici $w(\mathcal{C}) = d_{\min}(\mathcal{C})$. Supponendo infatti che la distanza minima del codice si ottenga come $d_{\min} = \mathrm{d_H}(\vec{x}_1, \vec{x}_2)$, poiché $\vec{x}_1 - \vec{x}_2 \in \mathcal{C}$ allora $\mathrm{d_H}(\vec{x}_1 - \vec{x}_2, \vec{0}) = d_{\min}$.

Esempio 2.29.

- $G = \left(\begin{array}{ccc|c}1 & \cdots & 0 & 1\\ \vdots & \ddots & \vdots & \vdots\\0 & \cdots & 1 & 1\end{array}\right)$ è il *bit* di parità, che ha $w = d_{\min} = 2$.
- $G = \underbrace{\left(\begin{array}{c|ccc}1 & 1 & \cdots & 1\end{array}\right)}_{n \text{ colonne}}$ è il codice a ripetizione, che ha $w = d_{\min} = n$.

Come esempio di riferimento della prossima parte consideriamo

$$\underbrace{\begin{pmatrix}m_1 & m_2 & m_3\end{pmatrix}}_{\vec{m}}\underbrace{\left(\begin{array}{ccc|cc}1 & 0 & 0 & 1 & 1\\0 & 1 & 0 & 0 & 1\\0 & 0 & 1 & 1 & 0\end{array}\right)}_{G=(I_3|P)} = \underbrace{\begin{pmatrix}x_1 & x_2 & x_3 & x_4 & x_5\end{pmatrix}}_{\vec{x}},$$

notando che G è in forma canonica. Allora $x_i = m_i$ per $i \in \{1, 2, 3\}$, $x_4 = m_1 + m_3 = x_1 + x_3 \Rightarrow$ $\Rightarrow x_1 + x_3 - x_4 = 0$ e $x_5 = m_1 + m_2 = x_1 + x_2 \Rightarrow x_1 + x_2 - x_5 = 0$. Le ultime due sono dette EQUAZIONI DI CONTROLLO e danno luogo alla MATRICE DI CONTROLLO, solitamente denotata con «H», con $n-k$ righe ed n colonne. In questo esempio abbiamo $H = \left(\begin{array}{ccc|cc}1 & 0 & 1 & -1 & 0\\1 & 1 & 0 & 0 & -1\end{array}\right)$. Poiché G era in forma canonica, vale infatti che $H = (P^T \mid -I_{n-k})$.

Siccome $H\vec{x}^T = \begin{pmatrix}x_1 + x_3 - x_4\\x_1 + x_2 - x_5\end{pmatrix}$, allora $H\vec{x}^T = \vec{0} \Leftrightarrow \vec{x} \in \mathcal{C}$, anche se $H\vec{x}^T = \vec{0}$ non implica assenza di errore.

Definizione 2.30. La SINDROME di $\vec{y}$ si definisce come $s(\vec{y}) \coloneqq H\vec{y}^T$.

Quando una $\vec{x}$ entra nel canale, ne esce una $\vec{y} = \vec{x} + \vec{e}$, con $\vec{e}$ errore. Siccome

$$H\vec{y}^T = H(\vec{x} + \vec{e})^T = H(\vec{x}^T + \vec{e}^T) = \underbrace{H\vec{x}^T}_{=\vec{0}} + H\vec{e}^T = H\vec{e}^T$$

si può costruire un algoritmo di decodifica basato sulle sindromi (METODO DI SLEPIAN), provando a risalire a $\vec{x}$ da $s(\vec{y})$.

2.2.1 Codice di Hamming

Consideriamo $G = \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right)$ e la sua $H = \left(\begin{array}{cccc|ccc} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right)$, con $k = 4$, $n = 7$, $R = \frac{4}{7}$. Indichiamo «$\vec{h}_i$» l'i-esima colonna di H.

In questa H le colonne sono tutte diverse, quindi se c'è stato 1 errore non solo me ne accorgo ma so anche dove è avvenuto e lo correggo calcolando $\vec{x} = \vec{y} - \vec{e}$. Se invece ci sono stati 2 o più errori sbaglierei, perché $\vec{e} = \vec{h}_i$ per qualche i, pur essendo la somma di più colonne. In sintesi, questo codice corregge 1 errore.

Permutando le colonne di H in modo da ottenere $H' = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$ si ottiene un codice equivalente, col vantaggio che $H'\vec{y}^T$ mi dice in binario il *bit* con l'errore, rendendo più facile l'implementazione. Quest'ultimo è il CODICE DI HAMMING, che ha $d_{\min} = 3$ e $\lambda = \frac{3}{7}$.

In generale un codice di Hamming ha H matrice $r \times q^{r-1}$, da cui $R = \frac{q^r - 1 - r}{q^r - 1}$ e $\lambda = \frac{r}{q^r - 1}$.

Sia $\hat{H} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$, dove la parte inferiore sinistra è la H' di poco fa. Mi aspetto un $\vec{x}$ in cui l'ultimo *bit* è di parità. $\hat{H}\vec{y}^T$ può essere nella forma $\begin{pmatrix} 1 \\ \vec{0} \end{pmatrix}$, $\begin{pmatrix} 0 \\ \vec{0} \end{pmatrix}$, $\begin{pmatrix} 0 \\ \vec{h}_i \end{pmatrix}$ oppure $\begin{pmatrix} 1 \\ \vec{h}_i \end{pmatrix}$. Con 1 errore ottengo $\begin{pmatrix} 1 \\ \vec{h}_i \end{pmatrix}$ oppure $\begin{pmatrix} 1 \\ \vec{0} \end{pmatrix}$ e posso correggere. Con 2 errori ottengo $\begin{pmatrix} 0 \\ \vec{h}_i \end{pmatrix}$ e me ne accorgo.

2.2.2 Codice BCH

Sia $H_2 = \begin{pmatrix} h_1 & \cdots & h_n \\ k_1 & \cdots & k_n \end{pmatrix}$, dove le colonne k_i provengono da una matrice «K» in stile Hamming.

Con 1 errore nel *bit* i, $H_2\vec{y}^T = \begin{pmatrix} \vec{h}_i \\ \vec{k}_i \end{pmatrix}$, riconosciamo la colonna di H_2 e correggiamo. Con 2 errori nei *bits* i, j, $H_2\vec{y}^T = \begin{pmatrix} \vec{h}_i + \vec{h}_j \\ \vec{k}_i + \vec{k}_j \end{pmatrix}$. Siamo in grado di risalire a i, j? Se $H = K$ no. In alternativa K potrebbe essere una permutazione di H, quindi con 2 errori $H_2\vec{y}^T$ non corrisponderebbe ad alcuna colonna di H_2.

Per trovare la giusta K esprimiamo $\vec{k}_i$ in funzione di $\vec{h}_i$. Se proviamo con $\vec{k}_i = \vec{h}_i^2$ si ha $\begin{pmatrix} \vec{h}_i + \vec{h}_j \\ \vec{k}_i + \vec{k}_j \end{pmatrix} = \begin{pmatrix} \vec{h}_i + \vec{h}_j \\ \vec{h}_i^2 + \vec{h}_j^2 \end{pmatrix}$. Indicando con «$s$» la somma e con «$q$» la somma dei quadrati abbiamo $s^2 = q$, che non fornisce alcuna informazione aggiuntiva.

Con $\vec{k}_i = \vec{h}_i^3$ invece funziona, perché $(\vec{h}_i + \vec{h}_j)^3 = \vec{h}_i^3 + \vec{h}_j^3 + \vec{h}_i^2\vec{h}_j + \vec{h}_i\vec{h}_j^2 \Rightarrow s^3 = q + ps \Rightarrow$ $\Rightarrow p = (s^3 - q)s^{-1}$, dove stavolta «$q$» denota la somma dei cubi e «p» il prodotto. Resta ora da capire cosa siano il cubo e l'inverso di un vettore-colonna, ma possiamo considerare le colonne di H come elementi di $\mathrm{GF}(n+1) \setminus \{0\}$. In questo esempio, prendendo $x^3 + x + 1$ come polinomio irriducibile per

l'algebra in GF(8), dopo qualche calcolo otteniamo $K = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$. Per quanto riguarda gli inversi valgono $(x+1)(x^2+x) = 1$, $x(x^2+1) = 1$ e $x^2(x^2+x+1) = 1$.

Ora dovremmo risolvere $z^2 - sz + p = 0$, ma poiché in un campo finito di polinomi a coefficienti in $\mathbb{Z}_2$ la formula risolutiva delle equazioni di secondo grado non funziona, è necessario riscriverla nella forma $(\alpha x^2 + \beta x + \gamma)^2 - s(\alpha x^2 + \beta x + \gamma) + p = 0$ e "semplificarla".

✎ .....

Esempio 2.31. Supponiamo di avere errori nel *bit* 2 e 4. Allora $s = q = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = x^2 + x$ (casualmente uguali).

$$p = \big((x^2+x)^3 - (x^2+x)\big)(x^2+x)^{-1} = (x^2+x+1-x^2-x)(x+1) = x+1\,.$$

$$(\alpha x^2 + \beta x + \gamma) - (x^2+x)(\alpha x^2 - \beta x + \gamma) + (x+1) = 0 \Rightarrow \begin{cases} \gamma = 0 \\ \alpha + \beta = 1\,, \end{cases}$$

da cui le soluzioni attese $\vec{h}_2, \vec{h}_4$.

..... ✎

Nel caso particolare utilizzato in questa sezione, H_2 era 6×7, quindi la G corrispondente è 1×7, che non è altro che il codice a ripetizione.

In generale il codice BCH corregge 1 errore di più della matrice H da cui viene generato.

2.2.3 Codici di Reed-Müller

Ogni funzione booleana $B\colon \{0,1\}^m \to \{0,1\}$ può essere identificata univocamente con un vettore $(a_1, \ldots, a_{2^m})$ di elementi in $\{0,1\}$. La possiamo allora scrivere nella cosiddetta FORMA NORMALE ALGEBRICA $B(X_1, \ldots, X_m) = a_1 + a_2X_1 + a_3X_2 + \cdots + a_{2^m}X_1 \cdots X_m$, dove le operazioni sono in $\mathbb{Z}_2$.

✎ .....

Esempio 2.32. Per $m = 3$ costruiamo la seguente tabella (omettendo gli ‹0›):

	0	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1	1
X_1					1	1	1	1
X_2			1	1			1	1
X_3		1		1		1		1
X_1X_2							1	1
X_1X_3						1		1
X_2X_3				1				1
$X_1X_2X_3$								1

in cui la prima riga è semplicemente riempita di ‹1›, la fascia delle variabili singole è la rappresentazione binaria dei numeri soprastanti e le fasce successive sono compilate con la congiunzione delle variabili di ciascuna riga.

..... ✎

Parte della matrice dell'esempio precedente può essere usata per codificare, considerando come G la parte di tabella dalla cima fino alle r-uple, con $r \in \{1, \ldots, m-1\}$. La G avrà allora $n = 2^m$ colonne e $k = \sum_{i=0}^{r} \binom{m}{i}$ righe.

Per ogni grado i abbiamo 2^{m-1} equazioni di controllo, con il minimo per $i = r$. Allora la capacità di correzione di questo codice è 2^{m-r-2}.

2.2.4 Codici ciclici

Definizione 2.33. Un codice $\mathcal{C} \subseteq \mathcal{X}^n$ è CICLICO se è un codice algebrico e se per ogni $\vec{x} \in \mathcal{C}$ anche tutte le permutazioni cicliche di $\vec{x}$ sono in $\mathcal{C}$.

Esempio 2.34. $\mathcal{C} = \{\langle 000\rangle, \langle 101\rangle, \langle 110\rangle, \langle 011\rangle\}$ è un codice ciclico con $G = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$.

Sia $(c_0, c_1, \cdots, c_{n-2}, c_{n-1}) \in \mathcal{C}$ e applichiamo le seguenti trasformazioni:

- interpreto la n-upla come un polinomio (attenzione che già le c_i sono elementi del campo finito $\mathcal{X}$): $c_0 + c_1 x + \cdots + c_{n-2}x^{n-2} + c_{n-1}x^{n-1}$;
- lo moltiplico per x: $c_0 x + c_1 x^2 + \cdots + c_{n-2}x^{n-1} + c_{n-1}x^n$;
- applico il modulo $x^n - 1$: $c_{n-1} + c_0 x + c_1 x^2 + \cdots + c_{n-2}x^{n-1}$;
- lo reinterpreto come n-upla: $(c_{n-1}, c_0, c_1, \ldots, c_{n-2})$.

Così facendo abbiamo ottenuto la prima permutazione ciclica della n-upla iniziale. Riassumendo, se $\vec{c} \in \mathcal{C}$ considero il polinomio associato $c(x)$, di grado $n-1$. Allora $xc(x), \ldots, x^{n-2}c(x) \mod x^n - 1$ sono le sue permutazioni cicliche.

Nei codici algebrici vale anche la proprietà che $a \in \mathcal{X} \wedge \vec{x} \in \mathcal{C} \Rightarrow a\vec{x} \in \mathcal{C}$. Per questo motivo, nel codici ciclici vale che se $c(x) \in \mathcal{C}$ e $p(x)$ è un qualunque polinomio a coefficienti in $\mathcal{X}$ allora $\big(c(x)p(x) \mod x^n - 1\big) \in \mathcal{C}$.

Definizione 2.35. Un polinomio è MONICO se il coefficiente del termine di grado massimo è 1.

Lemmi 2.36.

- In un codice ciclico c'è un unico polinomio «$g(x)$» monico di grado $r \neq 0$ minimo.

  Dimostrazione. Se fossero due distinti, sottraendoli otterrei un polinomio di grado minore non nullo.

- $\mathcal{C} = \big\langle g(x) \big\rangle$, ovvero se $c(x) \in \mathcal{C}$ allora esiste f polinomio di grado minore di n tale che $c(x) = f(x)g(x) \mod x^n - 1$.

- $g(x)$ è fattore di $x^n - 1$, ovvero $\exists h\colon g(x)h(x) = x^n - 1$.

  Dimostrazione. $x^n - 1 = g(x)h(x) + s(x)$, con s di grado minore di r. Allora $g(x)h(x) \equiv -s(x) \mod x^n - 1$, ma $g(x) \in \mathcal{C} \Rightarrow g(x)h(x) \in \mathcal{C} \Rightarrow s(x) = 0$, perché lo zero è l'unico polinomio del codice di grado minore di r.

- $|\mathcal{C}| = q^{n-r}$.

- Ogni $c(x) \in \mathcal{C}$ può essere scritto univocamente come $c(x) = f(x)g(x)$, con f polinomio di grado minore di $n - r$.

- Il codice che si ottiene ha

$$G = \begin{pmatrix} g_0 & \cdots & g_r & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & g_0 & \cdots & g_r \end{pmatrix}, \qquad H = \begin{pmatrix} 0 & \cdots & h_{n-r} & \cdots & h_0 \\ \vdots & \iddots & \iddots & \iddots & \vdots \\ h_{n-r} & \cdots & h_0 & \cdots & 0 \end{pmatrix},$$

  rispettivamente con $n - r$ ed r righe, dove $\vec{g} = (g_0, \ldots, g_r, 0, \ldots, 0)$ e $\vec{h} = (h_0, \ldots, h_{n-r}, 0, \ldots, 0)$ sono n-uple.

- $c(x) \in \mathcal{C} \Rightarrow c(x)h(x) \equiv 0 \mod x^n - 1$.

  Dimostrazione.

$$c(x) \in \mathcal{C} \Rightarrow c(x) = f(x)g(x) \Rightarrow c(x)h(x) = f(x)\underbrace{g(x)h(x)}_{\equiv 0} \Rightarrow c(x)h(x) \equiv 0 \mod x^n - 1\,.$$

Esempio 2.37. Sia $n = 1$ e $\mathcal{X} = \mathbb{Z}_2$. Siccome $x^7 - 1 = (x+1)(x^3+x^2+1)(x^3+x+1)$, abbiamo vari modi per scegliere g e h.

- $g(x) = x^3 + x^2 + 1$, $h(x) = x^4 + x^3 + x^2 + 1$

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}, \qquad H = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

  Questa H la conosciamo già: è il codice di Hamming.

- $g(x) = x + 1$, $h(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$

$$G = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}, \qquad H = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

  Questo è ancora una volta il codice di parità.

Come si fattorizza $x^n - 1$? Che relazioni ci sono tra g, h e la capacità di correzione?

Definizione 2.38. Un elemento a di un campo finito di cardinalità n si dice RADICE PRIMITIVA DELL'UNITÀ se $a^{n-1} \equiv 1$.

Il Teorema 3.8 ci dice che se $n = p^r - 1$, ogni elemento non nullo di $\mathrm{GF}(p^r)$ è radice primitiva dell'unità. Inoltre, preso un qualsiasi elemento a, tutti gli elementi del campo di ottengono come potenza di a.

Scomporre $x^n - 1$ significa scrivere $(x - \alpha_1)\cdots(x - \alpha_n)$ avendo le soluzioni $\alpha_1, \ldots, \alpha_n$ di $x^n = 1$. Ciò sembra contrapporsi alla presenza di polinomi irriducibili, secondo quanto visto sopra. L'"imbroglio" sta nel fatto che gli α_i sono coefficienti in $\mathrm{GF}(p^r)$, non in $\mathbb{Z}_p$.

Tuttavia, se abbiamo un polinomio ℓ e α è radice di ℓ, allora anche $\alpha^{p^1}, \alpha^{p^2}, \ldots$ sono radici di ℓ. Nel nostro campo di riferimento $\mathrm{GF}(8)$, se a è radice di $x^7 - 1$, i due polinomi irriducibili sono proprio $(x-a)(x-a^2)(x-a^4) = x^3 + x + 1$ e $(x-a^3)(x-a^6)(x-a^5) = x^3 + x^2 + 1$ (infatti $5 \equiv_7 12$).

Prendiamo $g(x) = x^3 + x + 1$, le cui radici sono $a, a^2, a^2 + a$. Allora

$$c(x) \in \mathcal{C} \wedge a \text{ radice di } g \Rightarrow c(a) = 0 \Rightarrow$$

$$\Rightarrow c(a) = \begin{pmatrix} 1 & a & \cdots & a^{n-1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}}_{H \text{ di Hamming}} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix}.$$

Naturalmente vale anche $c(a^2) = \begin{pmatrix}1 & a^2 & \cdots & a^{2(n-1)}\end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix}$. Se però uniamo le due matrici di controllo otteniamo $\begin{pmatrix}1 & a & \cdots & a^{n-1} \\ 1 & a^2 & \cdots & a^{2(n-1)}\end{pmatrix}$, che il primo tentativo di BCH. Come sappiamo, esso corregge esattamente come quello singolo, proprio perché le radici "coniugate" non aumentano la capacità di correzione. Dunque possiamo arricchire $g(x) = (x^3 + x + 1)(x^3 + x^2 + 1)$, ottenendo così la matrice di controllo $\begin{pmatrix}1 & a & \cdots & a^{n-1} \\ 1 & a^3 & \cdots & a^{3(n-1)}\end{pmatrix}$, in grado di correggere 1 errore in più.

Teorema 2.39. Se $g(x)$ ammette radici $\underbrace{\alpha^b, \alpha^{b+1}, \ldots, \alpha^{b+\delta-2}}_{\delta-1 \text{ radici consecutive}}$ (dove α è un elemento del campo) allora con

$$H = \begin{pmatrix} 1 & \alpha^b & (\alpha^b)^2 & \cdots & (\alpha^b)^{n-1} \\ 1 & \alpha^{b+1} & (\alpha^{b+1})^2 & \cdots & (\alpha^{b+1})^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{b+\delta-2} & (\alpha^{b+\delta-2})^2 & \cdots & (\alpha^{b+\delta-2})^{n-1} \end{pmatrix}$$

il codice ha $d_{\min} \geq \delta$.

Esempio 2.40. Rivediamo alcuni codici alla luce del teorema appena enunciato:

- $g(x) = x^3 + x + 1$ ha radici a, a^2, a^4, di cui solo le prime 2 consecutive, per cui il codice avrà $d_{\min} \geq 3$. Noi sappiamo che questo codice è Hamming e possiamo notare a posteriori che 1 delle 2 righe generate è ridondante.
- $g(x) = (x^3 + x + 1)(x^3 + x^2 + 1)$ ha radici $a, a^2, a^3, a^4, a^5, a^6$, per cui il codice avrà $d_{\min} \geq 7$. Noi sappiamo che questo codice è BCH e possiamo notare a posteriori che 4 delle 6 righe generate sono ridondanti.

Per quanto riguarda i tassi, $\lambda \geq \frac{\delta}{n}$ e $R \geq \frac{n - r(\delta-1)}{n}$. La prima ha un «$\geq$» perché $d_{\min} \geq \delta$, la seconda perché nella H potrebbero esserci righe superflue.

2.2.5 Codice di Reed-Solomon

Supponiamo di avere una matrice di controllo $H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \cdots \\ 1 & \alpha^2 & \alpha^4 & \cdots \\ 1 & \alpha^3 & \alpha^6 & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ 1 & \alpha^\delta & \alpha^{2\delta} & \cdots \end{pmatrix}$ con $n = 2^r - 1$ colonne.

Si correggono $\frac{\delta}{2}$ errori e i tassi valgono $R = \frac{(2^r-1) - r\delta}{2^r - 1}$ e $\lambda = \frac{\delta+1}{2^r-1}$. L'idea di Reed e Solomon fu quella di prendere i coefficienti da $\mathrm{GF}(2^t)$, invece che in $\mathbb{Z}_2$. Si ottiene così una matrice formata da t-ple di *bits*, quindi il numero di colonne effettivo diventa $t(2^r - 1)$. I tassi diventano $R = ?$ e $\lambda = ?$ DA CHIEDERE, per cui anche se si va più veloce sembra che la capacità di correzione sia diminuita. In realtà questo codice corregge non solo $\frac{\delta}{2}$ errori lontani, ma anche $t\frac{\delta}{2}$ errori vicini, perché agisce all'interno del *byte*.

Nel Reed-Solomon effettivo abbiamo $n = 2^{16} - 1$ ($\sim 64\,\mathrm{k}$), $\delta = 512$, $t = 16$, quindi corregge 256 errori sparsi e 4096 errori contigui in 1 M.

2.3 Codici convolutivi

Immaginiamo di avere un circuito sequenziale come quello in Figura 2.2. I «+» sono in $\mathbb{Z}_2$ e le lettere sono registri che memorizzano i risultati precedenti. Ogni circuito introduce 3 parametri: numero di uscite «u»

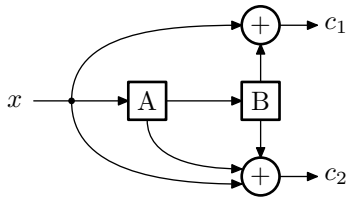

Figura 2.2: Esempio di circuito che descrive un codice convolutivo

($R = \frac{1}{u}$), numero di memorie «k» (influisce sulla difficoltà della decodifica) e collegamenti/forma (studiati insieme per costruire un buon codice).

✎ .....

Esempio 2.41: Viterbi.

- Codifica – Sia ‹1011101› la stringa in ingresso.
  La stringa che esce sarà ‹11 01 00 10 01 10 00 | 01 11›.
  I *bits* dopo la barra verticale sono ottenuti in una fase finale di "svuotamento" dei registri, simulando l'ingresso di ulteriori k ‹0›. Ciò peggiora lievemente il tasso di trasmissione.

- Decodifica in assenza di errore – Si usa un automa a stati finiti con *output*, dotato di 2^k stati (Figura 2.3). Notiamo che questa particolare rete sequenziale ha la proprietà che le uscite da ciascuno stato differiscono sempre di 2 bit. Per la decodifica si usa questo stesso automa scambiando l'*input* con l'*output*.

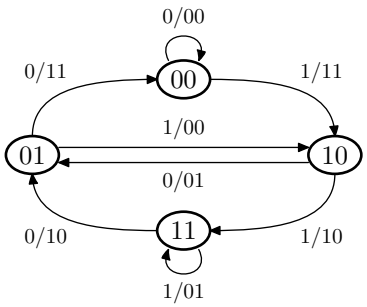

Figura 2.3: L'automa a stati finiti che corrisponde al circuito

  Facendo un *unfolding* parziale dell'automa si ottiene un *TRELLIS* («TRALICCIO») di altezza 2^k.

- Decodifica in presenza di errore – Costruiamo il *trellis* completo, di lunghezza proporzionale alla lunghezza della stringa. Percorriamo ipoteticamente tutte le strade, segnando gli errori su ciascuna strada. Se un nodo è raggiungibile da più di una strada, il percorso a partire da esso terrà conto solo della strada col minimo numero di errori; se la strada con numero minimo di errori non è unica, se ne sceglie una a caso.

- Capacità di correzione – Si considera un qualsiasi percorso chiuso del *trellis*. Se esso ha lunghezza ℓ e le due stringhe che individua hanno distanza d, diremo che la capacità di correzione è di $\left\lfloor \frac{d-1}{2} \right\rfloor$ su ℓ bit. La capacità di correzione è dunque variabile in funzione della lunghezza del percorso considerato.

..... ✎

3 Codici segreti[1]

3.1 Cifrario di Vigenère: decrittazione (Cifrari a sostituzione polialfabetica)

L'idea chiave per decrittare il cifrario di Vigenère è quella di trovare μ tale che tra il testo in cifra e se stesso traslato di μ ci sia il massimo numero di simboli coincidenti.

Sia «k» la cardinalità dell'alfabeto. Associato ad esso ci sarà una distribuzione di probabilità $P^0 := (p_0, \dots, p_{k-1})$. Siano X, Y v.a. che rappresentano rispettivamente il simbolo in chiaro e quello in cifra. Definiamo inoltre «P^i» come P^0 spostato a destra ciclicamente di i passi. Allora, riassumendo, $\mathcal{P}(X = x_i) = p_i = P^0_i$ e $\mathcal{P}(Y = y_i) = P^\delta_i$, dove δ è la differenza tra y_i e il simbolo in chiaro che l'ha generato.

Se consideriamo Y' v.a. che indica il simbolo corrispondente a Y nel testo in cifra traslato di μ, abbiamo $\mathcal{P}(Y = y_i) = P^{\delta_1}_i$ e $\mathcal{P}(Y' = y_i) = P^{\delta_2}_i$ per ogni i. Allora $\mathcal{P}(Y = Y') = \sum_{i=0}^{k-1} P^{\delta_1}_i P^{\delta_2}_i = P^{\delta_1} \cdot P^{\delta_2}$. Poiché $P^{\delta_1} \cdot P^{\delta_2} = p_0 p_{|\delta_1 - \delta_2|} + \dots + p_{k-1} p_{|\delta_1 - \delta_2| + k - 1}$, questo prodotto scalare dipende solo da $|\delta_1 - \delta_2|$, il cui valore massimo è per $\delta_1 = \delta_2$. Ciò succede quando allineo i due testi in cifra con un μ multiplo della lunghezza della chiave.

3.2 Introduzione all'RSA

✎ .....

Pseudocodice 3.1: Euclide. MCD(a, b), con $a \geq b$:

```
if (b = 0) then
  return a;
else
  return MCD(b, a mod b);
fi
```

..... ✎

L'algoritmo funziona perché $c \mid a \wedge c \mid b \Leftrightarrow c \mid b \wedge c \mid a \bmod b$.

✎ .....

Lemma 3.2. Se MCD(a, b) esegue più di $k \geq 1$ chiamate ricorsive, allora $a \geq F_{k+1}$ e $b \geq F_k$, con «F_i» i-esimo numero di Fibonacci.

Dimostrazione.

(B) $k = 1 \Rightarrow b > 0 \wedge a \geq b \Rightarrow b \geq 1 = F_1 \wedge a \geq 1 = F_2$.

(P) Per ipotesi induttiva vale $b \geq F_k \wedge a \bmod b \geq F_{k-1}$, da cui otteniamo banalmente $b \geq F_k$. Poiché $a = qb + (a \bmod b)$, se valesse $q \geq 1$ saremmo a posto perché $a \geq b + (a \bmod b) \geq F_k + F_{k-1} = F_{k+1}$; ma in effetti $a \geq b \Rightarrow q \geq 1$.

..... ✎

[1] Ecco i *link* ai PDF preparati dal docente:

1. Storia e terminologia. Cifrari a SOSTITUZIONE MONOALFABETICA. Cifrari OMOFONICI e NOMENCLATORI.
2. Cifrari POLIALFABETICI: tecniche per scoprire lunghezza della chiave e chiave. Esempio pratico di cifrazione VIGENÈRE e sua decrittazione.
3. Cifrari polialfabetici "algebrici". *ONE-TIME-PAD*: perfezione. Automazione della crittografia. Il rotore di Jefferson. ENIGMA: storia, funzionamento, debolezze.
4. Codici a TRASPOSIZIONE. *DATA ENCRYPTION STANDARD* (DES): funzionamento e decrittazione.
5. *ADVANCED ENCRYPTION STANDARD* (AES).
6. Crittografia a CHIAVE PUBBLICA. Diffie e Hellman e il logaritmo finito. Diffie e Merkle e il cifrario del fusto. Il cifrario RSA (RIVEST SHAMIR ADLEMANN).

Quanto esposto in questa sezione comprende solo i contenuti non presenti nei PDF.

✎ .....

Teorema 3.3: Teorema di Lamé. Se $a \geq b \geq 0$ e $b \leq F_k$ allora $\mathrm{MCD}(a, b)$ impiega meno di k chiamate ricorsive.

..... ✎

Poiché i numeri di Fibonacci crescono esponenzialmente, il numero di passi è dell'ordine $\log b$, che è anche il numero di cifre b.

✎ .....

Pseudocodice 3.4: Euclide esteso. $\mathrm{EE}(a, b)$, con $a \geq b$:

```
if (b = 0) then
  return (a, 1, 0);
else
  (d, x, y) := EE(b, a mod b);
  return (d, y, x - (a div b) * y);
fi
```

..... ✎

L'algoritmo di Euclide esteso (EE) permette di trovare i coefficienti d, x, y tali che $ax + by = d = \mathrm{MCD}(a, b)$. La sua complessità è la stessa di Euclide "semplice".

Se siamo in $\mathbb{Z}_b$ (b primo) e scegliamo $a \in \mathbb{Z}_b \setminus \{0\}$, a e b sono sempre primi tra loro ed $\mathrm{EE}(a, b) = (1, x, y)$, con x inverso di a in $\mathbb{Z}_b$.

✎ .....

Lemma 3.5. $ab \equiv_n ac \wedge \mathrm{MCD}(a, n) = 1 \Rightarrow b \equiv_n c$.

Dimostrazione. Per EE esistono x, y tali che $ax + ny = 1 \Rightarrow (ab - ac)x + n(b - c)y = b - c$ quindi $b - c$ è multiplo di n perché $ab - ac$ è multiplo di n.

..... ✎

✎ .....

Corollario 3.6. $ab \equiv_n 0 \wedge \mathrm{MCD}(a, n) = 1 \Rightarrow b \equiv_n 0$.

..... ✎

✎ .....

Teorema 3.7: Piccolo teorema di Fermat. Se p è primo e $p \nmid a$ allora $a^{p-1} \equiv_p 1$.

Dimostrazione. Sia $S = \mathbb{Z}_p \setminus \{0\}$. Definiamo $\psi(x) = ax \bmod p$ con $x \in S$ e mostriamo che è una biiezione $\psi \colon S \to S$. Sicuramente $\psi(x) \in \mathbb{Z}_p$, ma se fosse $\psi(x) = 0 \Rightarrow ax \equiv_p 0 \overset{p \nmid a}{\Rightarrow} x \equiv_p 0 \lightning$ con l'ipotesi che $x \in S$.

..... ✎

✎ .....

Corollario 3.8. In $\mathbb{Z}_p$ (p primo) ogni elemento a non nullo è tale che $a^{p-1} \equiv_p 1$, ossia è radice primitiva dell'unità.

..... ✎

✎ .....

Definizione 3.9. La FUNZIONE DI EULERO è $\Phi(n) \coloneqq \left|\left\{x \in \{1, \dots, n-1\} \colon \mathrm{MCD}(n, x) = 1\right\}\right|$, ossia restituisce il numero di numeri minori di n coprimi con n.

..... ✎

Calcolando $\Phi(n)$, se p è primo e $p \mid n$ allora $\mathrm{MCD}(kp, n) \neq 1$ per ogni k tale che $kp < n$. Dunque

$$\Phi(n) = n \prod_{\substack{p \text{ primo} \\ p \mid n}} \left(1 - \frac{1}{p}\right).$$

In particolare se p è primo $\Phi(p) = p-1$. Inoltre se $n = pq$, con p, q primi, allora $\Phi(n) = pq\left(1-\frac{1}{p}\right)\left(1-\frac{1}{q}\right) = $
$= (p-1)(q-1) = \Phi(p)\Phi(q)$.

✎ .....

Teorema 3.10: Teorema di Eulero. $\mathrm{MCD}(a,n) = 1 \Rightarrow a^{\Phi(n)} \equiv_n 1$.

Dimostrazione. Sia $S = \{x \in \{1,\dots,n-1\} \colon \mathrm{MCD}(n,x) = 1\}$, per cui vale $\Phi(n) = |S|$. Definiamo $\psi(x) = ax \bmod n$ e mostriamo che è una biiezione $\psi \colon S \to S$. Sicuramente $\psi(x) \in \{0,\dots,n-1\}$, ma se fosse $\psi(x) = 0 \Rightarrow ax \equiv_p 0 \overset{p \nmid a}{\Rightarrow} x \equiv_p 0 \not\downarrow$ con l'ipotesi che $x \in S$. Poi $\mathrm{MCD}(a,x) = 1$ (per ipotesi) e $\mathrm{MCD}(x,n) = 1$ (perché $x \in S$) implicano $\mathrm{MCD}(ax,n) = 1 \Rightarrow ax \bmod n \in S$. Inoltre $ax \equiv_n ay \Rightarrow x \equiv_n y$ quindi è iniettiva e dunque biiettiva.

Ciò premesso, scrivendo $S = \{x_1,\dots,x_{\Phi(n)}\}$, possiamo scrivere

$$x_1 \cdots x_{\Phi(n)} = \psi(x_1)\cdots\psi(x_{\Phi(n)}) \equiv_n ax_1 \cdots ax_{\Phi(n)} \equiv_n a^{\Phi(n)} x_1 \cdots x_{\Phi(n)} \Rightarrow a^{\Phi(n)} \equiv_n 1\,,$$

poiché nell'ultimo passo dividiamo di volta in volta per $x_1,\dots,x_{\Phi(n)}$.

..... ✎

✎ .....

Lemma 3.11. Se p e q sono primi diversi e $a \equiv_p b \wedge a \equiv_q b$ allora $a \equiv_{pq} b$.

Dimostrazione.

$$\left.\begin{array}{l} a \equiv_p b \Rightarrow a - b = ph \\ a \equiv_q b \Rightarrow a - b = qk \end{array}\right\} \Rightarrow ph = qk \Rightarrow h = q\alpha \wedge k = p\alpha \Rightarrow a - b = pq\alpha \Rightarrow a \equiv_{pq} b\,.$$

..... ✎

L'ESPONENZIALE FINITO consente di calcolare in maniera efficiente $b^a \bmod n$. I passi sono i seguenti:

- Calcolo $b^{2^0},\dots,b^{2^k}$ con $k = \lfloor \log_2 a \rfloor$. In questa fase i numeri crescono temporaneamente fino a $(n-1)^2$, ma ne devo calcolare solo un numero logaritmico.
- Scrivo a in base 2, che posso denotare come $(c_k,\dots,c_0)$.
- Calcolo $b^a = \prod_{i=0}^{k} c_i b^{2^i}$, facendo ogni moltiplicazione in modulo n.

Il numero di operazioni è dunque nell'ordine di $\log a$. Al contrario, il calcolo del LOGARITMO FINITO $\log_b a \bmod n$ è difficile (secondo le conoscenze attuali).

3.3 Complessità nell'ambito della crittografia[2]

Una MdT può servire per decidere o per calcolare una funzione. Ad esempio, a SAT è strettamente associato un problema funzionale FSAT, ossia trovare (se esiste) un assegnamento di verità. Se ho un algoritmo per FSAT, allora lo posso usare anche per SAT, il che è banale.

Tuttavia se ho M che decide SAT, posso costruire M' per FSAT. Infatti sia φ una formula con variabili $x_1,\dots,x_n$. Se $M(\varphi) = \text{no}$ allora subito $M'(\varphi) = \text{no}$. Altrimenti provo $M\big(\varphi[x_1/0]\big)$ e a seconda del risultato scopro l'assegnamento per x_1. Procedo quindi similmente per stabilire le altre variabili.

Secondo le conoscenze attuali, non è però sempre vero che il problema esistenziale risolve facilmente quello funzionale. Infatti nella coppia

- ¬PRIMES (esistenziale) – dato n, stabilire se esiste p primo tale che $p \mid n$;
- ONE-FACTOR (funzionale) – dato n, calcolare p primo (ma non necessariamente) tale che che $p \mid n$ (se esiste);

[2] Questa sezione dà per noti i concetti di complessità ivi utilizzati.

la negazione del primo è in $\mathbb{P}$, mentre non sappiamo dire nulla del secondo, che è alla base della crittografia a chiave pubblica.

Definizione 3.12. La classe di complessità «$\mathbb{FP}$» è definita come l'insieme delle funzioni f calcolate da una MdT che opera in tempo polinomiale.

Definizione 3.13. Una relazione $R \subseteq \Sigma^* \times \Sigma^*$ si dice:

- POLINOMIALMENTE DECIDIBILE se esiste una MdT deterministica che decide $(x, y) \in R$ in tempo polinomiale;
- POLINOMIALMENTE BILANCIATA se $\exists h \colon \forall (x, y) \in R \colon |y| \leq |x|^h$.

Teorema 3.14. Sia $L \in \Sigma^*$. $L \in \mathbb{NP}$ se e solo se esiste una relazione R_L polinomialmente decidibile e polinomialmente bilanciata tale che $L = \left\{x \colon \exists y \colon (x, y) \in R_L\right\}$.

È proprio questo teorema che caratterizza $\mathbb{NP}$ come l'insieme dei problemi "*guess-and-verify*", ossia che possono essere verificati in tempo polinomiale conoscendo un assegnamento.

Definizione 3.15. La classe di complessità «$\mathbb{FNP}$» viene definita come l'insieme dei problemi formulati nel seguente modo:
Sia $L \in \mathbb{NP}$. Dato $x \in \Sigma^*$ trovare $y \in \Sigma^*$ tale che $(x, y) \in R_L$ se $x \in L$, altrimenti rispondere negativamente.

Notiamo che $\mathbb{FP}$ non è altro che la restrizione di $\mathbb{FNP}$ per problemi in P. Inoltre si ottiene facilmente che $\mathbb{FP} = \mathbb{FNP} \Leftrightarrow \mathbb{P} = \mathbb{NP}$.

Definizione 3.16. Una funzione f si dice *ONE-WAY* (ossia MONODIREZIONALE) se:

0. f è iniettiva;
1. $\exists k \colon \forall x \in \Sigma^* \colon \sqrt[k]{|x|} \leq \left|f(x)\right| \leq |x|^k$;
2. $f \in \mathbb{FP}$;
3. $f^{-1} \notin \mathbb{FP}$ (con f^{-1} definita laddove possibile).

Osserviamo che una conseguenza della definizione è che se f è *one-way* allora $f^{-1} \in \mathbb{FNP}$. Inoltre non è detto che esistano funzioni *one-way*, infatti possiamo solo *presumere* che il logaritmo finito e ONE-FACTOR non siano in $\mathbb{FP}$. Più precisamente vale che esistono funzioni *one-way* solo se $P \neq \mathbb{NP}$.

Definizione 3.17. La classe di complessità $\mathbb{UP}$ (la «U» sta per «*UNAMBIGUOUS*») è definita come l'insieme dei linguaggi L decisi da una ND-MdT che opera in tempo polinomiale e tale che per ogni $x \in L$ esiste un'unica computazione accettante.

Chiaramente $\mathbb{UP} \subseteq \mathbb{NP}$, ma si sospetta che l'inclusione sia stretta. Inoltre $P \subseteq \mathbb{UP}$, infatti in una MdT deterministica c'è al più una computazione.

Teorema 3.18. $\mathbb{UP} = \mathbb{P}$ se e solo se non ci sono funzioni *one-way*.

Dimostrazione. Per la dimostrazione consideriamo la formulazione equivalente: esiste una funzione *one-way* se e solo se $\mathbb{UP} \neq \mathbb{P}$.

($\Rightarrow$) Sia f una funzione *one-way* e sia k il grado del polinomio che collega la dimensione dell'*input* con quella dell'*output*. Consideriamo una relazione d'ordine tra stringhe $x \prec y \Leftrightarrow |x| < |y| \vee \big(|x| = |y| \wedge x <_{\text{lex}} y\big)$. Sia $L_f = \big\{(x, y) \colon \exists z \preceq x \colon f(z) = y\big\}$.

Se voglio risolvere $(x, y) \in L_f$ genero nondeterministicamente tutte le $z \preceq x$, escludendo eventualmente quelle che non rispettano i vincoli di lunghezza. Tra le z rimaste lancio f (che è polinomiale), ottenendo al più un valore uguale a y, perché f è iniettiva. Allora $L_f \in \mathbb{UP}$.

Supponiamo per assurdo che $L_f \in \mathbb{P}$. Allora esiste una MdT deterministica M che decide L_f in tempo polinomiale, che possiamo usare per calcolare f^{-1}. Se abbiamo $f^{-1}(y)$ lancio $M(\underbrace{\langle 1 \cdots 1 \rangle}_{|y|^k}, y)$ cercando l'eventuale soluzione per bisezione. Ma allora $\lightning$ col fatto che f^{-1} non sia polinomiale.

($\Leftarrow$) Sia $L \in \mathbb{UP} \setminus \mathbb{P}$ e sia M la ND-MdT (che per semplicità supporremo a un nastro) che lo decide. Chiamiamo stringhe di tipo ① quelle che descrivono una computazione accettante di M per una qualche stringa y e di tipo ② tutte le altre.

Allora definiamo $f_M(x) = \begin{cases} (1, y) & \text{se } x \text{ è di tipo ①} \\ (0, x) & \text{se } x \text{ è di tipo ②} \end{cases}$ e procediamo con il controllo dei punti della Definizione 3.16:

0. Date due stringhe x_1, x_2 distinte: se sono di tipo diverso allora le rispettive f_M hanno prefisso diverso; se sono entrambe di tipo ② differiscono nel suffisso; se infine sono entrambe di tipo ①, la y a cui si riferiscono è necessariamente diversa per l'unicità della computazione accettante.
1. $|f_M(x)| < |x|$ per le stringhe di tipo ① e $|f_M(x)| = c + |x|$ per quelle di tipo ②. Dopo qualche calcolo di ottiene anche che $|f_M(x)| \geq \sqrt[k]{|x|}$.
2. Posso determinare il tipo di x e costruire l'*output* in tempo polinomiale.
3. Supponiamo di voler calcolare $f^{-1}(z)$. Se z è nella forma «$(0, z)$» basta restituire z. Altrimenti se è «$(1, y)$» devo restituire la descrizione dell'unica computazione accettante per y con M. Tuttavia in tal caso M opererebbe in tempo polinomiale, $\lightning$ con l'ipotesi iniziale.

Abbiamo dunque trovato una funzione *one-way*.