Teoria dell'informazione e della trasmissione

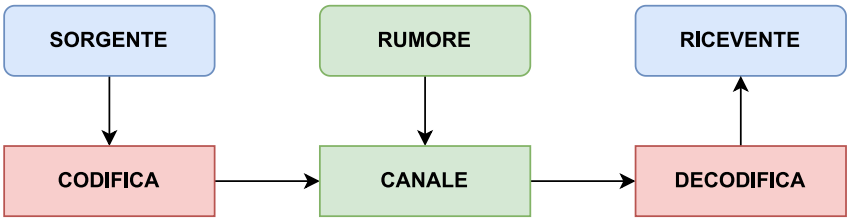
Indice

1. Lezione 01 [24/09]

2. Lezione 02 [01/10]

2.1. Introduzione storica

Lo schema di riferimento che andremo ad usare durante tutto il corso è il seguente:



La prima persona che lavorò alla teoria dell'informazione fu **Claude Shannon**, un impiegato della TNT (Telecom americana) al quale è stato commissionato un lavoro: massimizzare la quantità di dati che potevano essere trasmessi sul canale minimizzando il numero di errori che poteva accadere durante la trasmissione.

Nel 1948 pubblica un lavoro intitolato "A mathematical theory of comunication", un risultato molto teorico nel quale modella in maniera astratta il canale e capisce come l'informazione può essere spedita "meglio" se rispetta certe caratteristiche. Ci troviamo quindi di fronte ad un risultato che non ci dice cosa fare nel caso specifico o che codice è meglio, ma è probabilistico, ti dice nel caso medio cosa succede.

Questo approccio è sicuramente ottimale, ma rappresenta un problema: va bene il caso medio, ma a me piacerebbe sapere cosa succede nel caso reale. Questo approccio più reale è quello invece seguito dal russo **Kolmogorov**, un accademico che vuole capire cosa succede nei singoli casi senza usare la probabilità. A metà degli anni '60 propone la sua idea di teoria dell'informazione, focalizzandosi quindi sui casi reali e non sui casi medi.

Questi due mostri sacri della teoria dell'informazione, nel nostro schema di lavoro, si posizionano nei rettangoli di sorgente e codifica, mentre la teoria della trasmissione si concentra sui rettangoli sottostanti.

Altri due personaggi che hanno lavorato allo stesso problema di Kolmogorov sono due russi, che lavoravano uno in America e uno nell'est del mondo, ma non sono ricordati perché Kolmogorov era il più famoso dei tre.

Un altro personaggio importante è **Richard Hamming**, un ricercatore di Bell Lab che doveva risolvere un problema: i job mandati in esecuzione dalle code batch delle macchine aziendali se si piantavano durante il weekend potevano far perdere un sacco di tempo. La domanda che si poneva Hamming era "che maroni, perché se le schede forate hanno errori, e le macchine lo sanno, io non posso essere in grado di correggerli?"

Lui sarà il primo che, per risolvere un problema pratico, costruisce il primo codice di rilevazione e correzione degli errori, il famoso e usatissimo **codice di Hamming**.

Vediamo alcune date che rivelano quanto è stata importante la teoria dell'informazione:

- 1965: prima foto di Marte in bianco e nero grande $\approx 240K$ bit, inviata con velocità 6 bit al secondo, ci metteva ore per arrivare a destinazione;
- 1969: stessa foto ma compressa, inviata con velocità $16K$ bit al secondo, ci metteva pochi secondi;
- 1976: prima foto di Marte a colori da parte di Viking;
- 1979: prima foto di Giove e delle sue lune a colori da parte di Voyager;

- anni ‘80: prima foto di Saturno e delle sue lune da parte di Voyager.

2.2. Cosa faremo

Nel corso vedremo due operazioni fondamentali per spedire al meglio un messaggio sul canale:

- **compressione**: dobbiamo ottimizzare l’accesso al canale, ovvero se possiamo mandare meno bit ma questi mi danno le stesse informazioni dei bit totali ben venga;
- **aggiunta di ridondanza**: dobbiamo aggiungere dei bit per permettere il controllo, da parte del ricevente, dell’integrità del messaggio.

La compressione riguarda la **source coding**, e viene rappresentata nel **primo teorema di Shannon**, mentre la ridondanza riguarda la **channel coding**, e viene rappresentata nel **secondo teorema di Shannon**.

Voglio massimizzare l’informazione da spedire sul canale ma alla quale devo aggiungere ridondanza.

Quello che fa Shannon è modellare il canale secondo una matrice stocastica, che quindi permette di sapere in media cosa succede evitando tutti i casi precisi.

IN/OUT	a	b	c	d	e
a	0.7	0.0	0.1	0.1	0.1
b	0.2	0.8	0.0	0.0	0.0
c	0.1	0.0	0.6	0.2	0.1
d	0.0	0.0	0.2	0.5	0.3
e	0.0	0.0	0.0	0.0	1.0

Questa matrice indica, per ogni carattere del nostro alfabeto, quale è la probabilità di spedire tale carattere e ottenere i vari caratteri presenti nell’alfabeto.

2.3. Esempio: informazione di un messaggio

Quando un messaggio contiene più informazioni di un altro?

Lancio n volte due monete, una truccata e una non truccata.

Quale delle due monete mi dà più informazioni? Sicuramente quella non truccata: il lancio della moneta “classica” è un evento randomico, non so mai cosa aspettarmi, mentre il lancio della moneta truccata è prevedibile, so già cosa succederà.

Possiamo quindi dire che un evento prevedibile porta pochissima informazione, mentre un evento imprevedibile porta tantissima informazione.

2.4. Esempio: codice ZIP

Uno dei codici di compressione più famosi è il codice **ZIP**.

Come funziona:

1. creo un dizionario, inizialmente vuoto, che contiene le coppie (stringa-sorgente,codifica), dove “codifica” è un numero;
2. usando un indice che scorre la stringa carattere per carattere, e partendo con numero di codifica uguale a 1, fino all’ultimo carattere eseguo iterativamente:
    1. parti da una stringa vuota che useremo come accumulatore;

2. aggiungi il carattere corrente all'accumulatore;
3. se l'accumulatore non è presente come chiave nel dizionario la aggiungo a quest'ultimo con codifica il numero corrente di codifica; riparto poi dal punto 2.1 con numero di codifica aumentato di 1;
4. se l'accumulatore è presente come chiave nel dizionario riparto dal punto 2.2.

Ad esempio, la stringa $AABACDABDCAABBA$ viene codificata con:

- $A \longrightarrow 1$;
- $A \longrightarrow AB \longrightarrow 2$;
- $A \longrightarrow AC \longrightarrow 3$;
- $D \longrightarrow 4$;
- $A \longrightarrow AB \longrightarrow ABD \longrightarrow 5$;
- $C \longrightarrow 6$;
- $A \longrightarrow AA \longrightarrow 7$;
- $B \longrightarrow 8$;
- $B \longrightarrow BA \longrightarrow 9$.

Prima avevo 15 caratteri, ora ne uso 9, letsgosky.

2.5. Richiami matematici

In questa parte abbiamo rivisto la definizione di monoide, gruppo, anello e campo, oltre alla definizione di generatore.

Abbiamo anche visto i **campi di Galois**, utilissimi per rendere il nostro calcolatore un campo algebrico, ovvero il campo $\mathbb{GF}(2^{64})$ dei polinomi di grado massimo 63 con coefficienti sul campo $\mathbb{Z}_2$.

3. Lezione 03 [04/10]

Dalla lezione scorsa abbiamo capito che dobbiamo fare due cose:

1. massimizzare l'informazione trasmessa ad ogni utilizzo del canale; il "ogni utilizzo" è importante perché ogni volta che utilizzo il canale devo essere top, adesso lo devo essere. Se devo mandare n informazioni lo posso fare in n accessi oppure in 1 accesso, in entrambi i casi massimizzo l'informazione trasmessa ma non uso sempre tutta la banda possibile;
2. minimizzare il numero di errori di trasmissione.

Shannon userà la tecnica del Divide Et Impera, ovvero risolverà le due task separatamente e poi unirà i due risultati parziali. Per puro culo, questa soluzione sarà ottima (*non sempre succede*).

Il punto 1 riguarda la sorgente, il punto 2 riguarda la codifica e il canale.

3.1. Codici

Prima di vedere un po' di teoria dei codici diamo alcune basi matematiche.

Sia X un insieme di **simboli** finito. Un/a **messaggio/parola** $\overline{x} = x_1 \cdot ... \cdot x_n \in X^n$ è una concatenazione di n caratteri x_i di X.

Dato $d > 1$, definiamo $D = \{0, ..., d-1\}$ insieme delle **parole di codice**.

Definiamo infine $c : X \longrightarrow D^+$ **funzione di codifica** che associa ad ogni carattere di X una parola di codice. Questa è una codifica in astratto: a prescindere da come è formato l'insieme X io uso le parole di codice che più mi piacciono (????). L'insieme D^+ è tale che

$$D^+ = \bigcup_{n \geq 1}^{\infty} \{0, ..., d-1\}^n.$$

Voglio massimizzare la compressione: sia $l_c(x)$ la lunghezza della parola $x \in X$ nel codice c.

Quello che ho ora mi basta? NO: mi serve anche la **probabilità** $p(x)$ di estrarre un simbolo. Questo perché alle parole estratte molto spesso andremo a dare una lunghezza breve, mentre alle parole estratte di rado andremo a dare una lunghezza maggiore. Un codice che segue queste convenzioni è il codice morse.

Definiamo quindi il **modello sorgente** la coppia $\langle X, p \rangle$.

Ora va bene? NON ANCORA: noi non vogliamo la probabilità di estrarre un simbolo perché noi lavoriamo con le parole, non con i simboli.

Definiamo quindi $P_n(\overline{x} = x_1 \cdot ... \cdot x_n) = \prod_{i=1}^{n} p(x_i)$

Qua vediamo una prima enorme semplificazione di Shannon: stiamo assumendo l'indipendenza tra più estrazioni consecutive, cosa che nelle lingue invece non è vera.

Il codice ZIP invece non assume l'indipendenza e lavora con la lingua che sta cercando di comprimere.

Dati il modello $\langle X, p \rangle$ e la base $d > 1$, dato il codice $c : X \longrightarrow D^+$ il modello deve essere tale che

$$\mathbb{E}[l_c] = \sum_{x \in X} l_c(x) p(x)$$

sia minimo.

Il primo problema che incontriamo sta nella fase di decodifica: se codifico due simboli con la stessa parola di codice come faccio in fase di decodifica a capire quale sia il simbolo di partenza?

Imponiamo che il codice sia un **codice non singolare**, ovvero che la funzione c sia iniettiva. Stiamo imponendo quindi che il codominio sia abbastanza capiente da tenere almeno tutti i simboli di X.

Definiamo $C : X^+ \longrightarrow D^+$ l'**estensione del codice** c che permette di codificare una parola intera.

Se c è non singolare, come è C?

Purtroppo, la non singolarità non si trasmette nell'estensione del codice.

Andiamo a restringere l'insieme dei codici "buoni": chiamiamo codici **univocamente decodificabili** (*UD*) i codici che hanno C non singolare.

Per dimostrare che un codice è UD esista l'algoritmo di Sardinas-Patterson, che lavora in tempo $O(mL)$, con $m = |X|$ e $L = \sum_{x \in X} l_c(x)$.

Ora però abbiamo un altro problema: gli UD non sono **stream**. In poche parole, un codice UD non ci garantisce di decodificare istantaneamente una parola di codice in un carattere di X quando mi arrivano un po' di bit, ma dovrei prima ricevere tutta la stringa e poi decodificare.

Sono ottimi codici eh, però ogni tanto aspetteremo tutta la codifica prima di poterla decodificare. Eh ma non va bene: in una stream non posso permettermi tutto ciò, e inoltre, se la codifica è veramente grande potrei non riuscire a tenerla tutta in memoria.

Potremmo utilizzare i **codici a virgola**, ovvero codici che hanno un carattere di terminazione per dire quando una parola è finita, e quindi risolvere il problema stream negli UD, ma noi faremo altro.

Restringiamo per l'ultima volta, definendo i **codici istantanei** (*CI*). Questi codici hanno la proprietà che stiamo cercando, ovvero permettono una decodifica stream, quindi non dobbiamo aspettare tutta la codifica prima di passare alla decodifica, ma possiamo farla appena riconosciamo una parola di codice.

Per capire se un codice è CI devo guardare i prefissi: nessuna parola di codice deve essere prefissa di un'altra. Questo controllo è molto più veloce rispetto al controllo che dovevo fare nei codici UD.

Ho quindi la seguente gerarchia:

$$\text{CI} \subset \text{UD} \subset \text{NS} \subset \text{codici} \ .$$

Ritorniamo all'obiettivo di prima: minimizzare la quantità

$$\mathbb{E}[l_c] = \sum_{x \in X} l_c(x) p(x).$$

Vediamo come, mettendo ogni volta dei nuovi vincoli sul codice, questo valore atteso aumenta, visto che vogliamo dei codici con buone proprietà ma questo va sprecato in termini di bit utilizzati. Inoltre, il codice che abbiamo sotto mano è il migliore?

A noi gli UD andavano bene (*stream esclusi*), quanto pago per andare nei codici istantanei?

Teorema: Se un codice è istantaneo allora è anche univocamente decodificabile.

Dimostrazione: Dimostro il contrario, ovvero che se un codice non è UD allora non è CI.

Sia $c : X \longrightarrow D^+$ e sia C la sua estensione. Assumiamo che c sia non singolare. Se c non è UD allora esistono due messaggi $x_1 e x_2$ diversi che hanno la stessa codifica $C(x_1) = C(x_2)$.

Per mantenere i due messaggi diversi possono succede due cose:

- un messaggio è prefisso dell'altro: se x_1 è formato da x_2 e altri m caratteri, vuol dire che i restanti m caratteri di x_1 devono essere codificati con la parola vuota, che per definizione di codice non è possibile, e soprattutto la parola vuota sarebbe prefissa di ogni altra parola di codice, quindi il codice c non è istantaneo;
- esiste almeno una posizione in cui i due messaggi differiscono: sia i la prima posizione dove i due messaggi differiscono, ovvero $x_1[i] \neq x_2[i]$ e $x_1[j] = x_2[j]$ per $1 \leq j \leq i-1$, ma allora $c(x_1[i]) \neq c(x_2[i])$ e $c(x_1[j]) = c(x_2[j])$ perché c è non singolare, quindi per avere la stessa codifica devo tornare al punto 1 e avere x_1 come prefisso di x_2 (o viceversa), ma, quindi otteniamo che c non è istantaneo.

Ma allora il codice non è istantaneo. ■

4. Lezione 04 [08/10]

4.1. Disuguaglianza di Kraft

I codici UD ci vanno bene (stream escluso), mentre i CI sono perfetti ma perché ci danno di più ma ci fanno spendere più bit: quanti?

Noi stavamo minimizzando il valore atteso delle lunghezze delle parole di codice, ovvero

$$\min_{c \in \mathcal{C}} \quad \mathbb{E}[l_c] = \sum_{x \in X} l_c(x) p(x).$$

Cosa possiamo dire sui CI? Sono molto graditi perché, oltre a identificare un CI in maniera molto semplice guardando i prefissi, possiamo capire **se esiste** un codice istantaneo senza vedere il codice, ovvero guardando solo le lunghezze delle parole di codice.

Questa è una differenza abissale: prima guardavamo le parole di codice e controllavo i prefissi, ora non ho le parole, posso guardare solo le lunghezze delle parole di codice e, in base a queste, posso dire se esiste un CI con quelle lunghezze. Il problema principale è che non so che codice è quello istantaneo con quelle lunghezze, sto solo dicendo che sono sicuro della sua esistenza

Date le lunghezze, l'unica cosa che posso dire è se un CI non esiste (quando sono sbagliate le lunghezze date), ma non posso dire che il codice che ho dato con quelle lunghezze è un CI, perché questo ci dice solo della sua esistenza, non se il codice che ho dato con quelle lunghezze è CI, per quello dovrei guardare i prefissi.

Come mai seguire questa via e non quelle dei prefissi? Se il codice è molto grande, guardare questa proprietà è meno oneroso rispetto al guardare i prefissi.

Questa proprietà è detta **disuguaglianza di Kraft**.

Teorema (Disuguaglianza di Kraft): Data una sorgente $X = \{x_1, ..., x_m\}$, data $d > 1$ base del codice e dati m interi $l_1, ..., l_m > 0$ che mi rappresentano le lunghezze dei simboli del codice, esiste un CI $c : X \longrightarrow D^+$ tale che

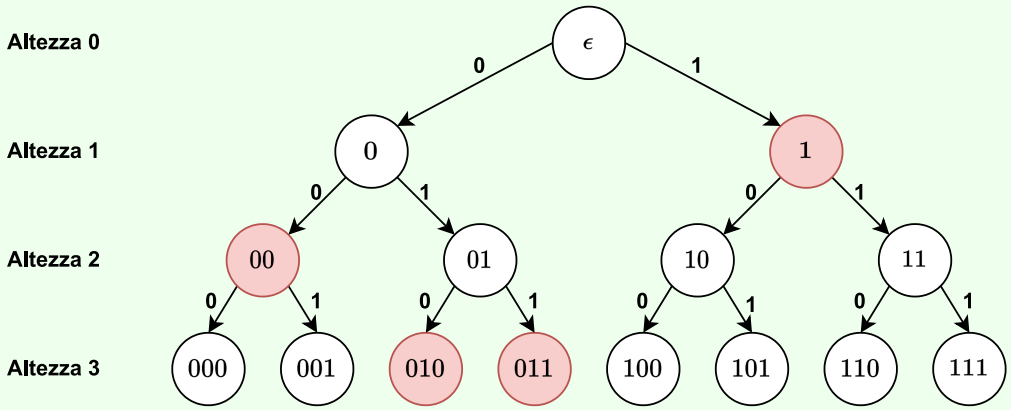
$$l_c(x_i) = l_i \quad \forall i = 1, ..., m$$

se e solo se

$$\sum_{i=1}^{m} d^{-l_i} \leq 1.$$

Dimostrazione:
($\Longrightarrow$) Esiste un CI con quelle proprietà, dimostro che vale la disuguaglianza di Kraft.

Sia c il nostro CI e d la base di c, dobbiamo definire la profondità del nostro codice. Possiamo usare un albero di codifica, ovvero un albero d-ario che rappresenta come sono codificate le varie parole di codice. Vediamo un breve esempio.



Vogliamo sapere

$$l_{\max} = \max_{i=1,\dots,m} l_c(x_i).$$

Costruiamo l'albero di codifica per il nostro CI e mettiamo dentro questo albero le parole del nostro codice. Come lo costruiamo? Creiamo l'albero d-ario alto $l_{\max}$ completo e scegliamo, in questo albero, delle parole ad altezza $l_i \quad \forall i = 1, \dots, m$.

Dividiamo il nostro albero in sotto-alberi grazie alle parole che abbiamo inserito: ogni sotto-albero ha come radice una delle parole che abbiamo scelto. Contiamo quante foglie che ogni nodo copre. Sono tutti alberi disgiunti, visto che non posso avere prefissi essendo c un CI. Il numero massimo di foglie è $d^{l_{\max}}$, noi non le potremmo coprire tutte quindi

$$\sum_{i=1}^{m} |A_i| \leq d^{l_{\max}},$$

con A_i sotto-albero con radice la parola x_i. Notiamo che

$$\sum_{i=1}^{m} d^{l_{\max} - l_i} = \sum_{i=1}^{m} |A_i| \leq d^{l_{\max}}.$$

Ora possiamo dividere tutto per $d^{l_{\max}}$ e ottenere

$$\sum_{i=1}^{m} \frac{d^{l_{\max} - l_i}}{d^{l_{\max}}} = \sum_{i=1}^{m} d^{-l_i} \leq 1.$$

($\Longleftarrow$) Vale la disuguaglianza di Kraft, dimostro che esiste un CI con quelle proprietà.

Abbiamo le lunghezze che rendono vera la disuguaglianza di Kraft, quindi costruiamo l'albero di codifica: scegliamo un nodo ad altezza l_i e poi proibiamo:

- a tutti gli altri nodi di inserirsi nel suo sotto-albero;
- di inserire nodi che conterrebbero dei nodi già inseriti.

Una volta costruito l'albero vedo che rappresenta un CI, perché tutti i nodi non hanno nodi nel loro sotto-albero, quindi non ho prefissi per costruzione, quindi questo è un CI. ■

Il nostro obiettivo rimane sempre e comunque quello di cercare il codice migliore, quello che minimizza il valore atteso delle lunghezze di tale codice, ovvero vogliamo trovare delle lunghezze $l_1, ..., l_m$ tali che

$$\min_{l_1,...,l_m} \sum_{i=1}^{m} l_i p_i.$$

Non voglio solo minimizzare, voglio che valga anche Kraft, così da avere un codice che sia istantaneo, quindi devono valere contemporaneamente

$$\begin{cases} \min\limits_{l_1,...,l_m} \sum\limits_{i=1}^{m} l_i p_i \\ \sum\limits_{i=1}^{m} d^{-l_i} \leq 1 \end{cases}$$

con $p_i = p(x_i) \quad \forall i = 1, ..., m$. Cosa possiamo dire?

Notiamo che

$$\sum_{i=1}^{m} d^{-l_i} \leq \sum_{i=1}^{m} p_i = 1.$$

Per comodità, associamo p_i al simbolo x_i con lunghezza l_i. Allora guardiamo tutti i singoli valori della sommatoria, perché *"se rispetto i singoli rispetto anche le somme"*, quindi

$$d^{-l_i} \leq p_i \quad \forall i = 1, ..., m$$
$$l_i \geq \log_d\left(\frac{1}{p_i}\right).$$

Con questa relazione ho appena detto come sono le lunghezze l_i del mio codice: sono esattamente

$$l_i \geq \left\lceil \log_d\left(\frac{1}{p_i}\right) \right\rceil.$$

Posso quindi costruire un codice mettendo in relazione le lunghezze del codice con la probabilità di estrarle dalla sorgente.

Esempio: Dati

$$X = \{x_1, x_2, x_3, x_4\} \quad | \quad P = \left\{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}\right\} \quad | \quad d = 2$$

costruire un codice con la tecnica appena vista.

- $l_1 \geq \lceil \log_2(2) \rceil = 1 \longrightarrow 0$;
- $l_2 \geq \lceil \log_2(4) \rceil = 2 \longrightarrow 10$;
- $l_3 \geq \lceil \log_2(8) \rceil = 3 \longrightarrow 110$;
- $l_4 \geq \lceil \log_2(8) \rceil = 3 \longrightarrow 111$.

Questo codice ha valore atteso delle lunghezze

$$\mathbb{E}[l_c] = \frac{2}{4} + \frac{2}{4} + \frac{3}{4} = \frac{7}{4}.$$

Il valore atteso ora diventa

$$\mathbb{E}[l_c] = \sum_{i=1}^{m} p_i \log_d \left(\frac{1}{p_i} \right).$$

grazie alla nuova definizione delle lunghezze delle parole di codice.

La tecnica che associa ad ogni lunghezza un valore in base alla probabilità ci consente di generare un codice chiamato **codice di Shannon** (*o Shannon-Fano*. Il bro Fano era il dottorando di Shannon).

Inoltre, notiamo come il valore atteso delle lunghezze dipenda solo dalla distribuzione di probabilità dei simboli sorgente.

La quantità

$$\sum_{i=1}^{m} p_i \log_d \left(\frac{1}{p_i} \right)$$

viene chiamata **entropia**. Come vedremo, sarà una misura che definisce quanto i nostri codici possono essere compressi, una sorta di misura di compattezza: oltre quella soglia non posso andare sennò perdo informazione.

Questa costruzione dei codici di Shannon-Fano è molto bella:

- se ho una probabilità bassa di estrarre il simbolo x_i allora $\frac{1}{p_i}$ è grande e $\log_d \left(\frac{1}{p_i} \right)$ è grande;
- se ho una probabilità alta di estrarre il simbolo x_i allora $\frac{1}{p_i}$ è piccolo e $\log_d \left(\frac{1}{p_i} \right)$ è piccolo.

Purtroppo, i codici di Shannon-Fano non sono ottimali: una sorgente con due simboli a probabilità $p_1 = 0.1$ e $p_2 = 0.9$ darà un codice che non è ottimale.

Esempio: Siano

$$m = 4 \quad | \quad c : 1, 011, 01, 111.$$

Fai alcune considerazioni su questo codice.

Sicuramente non è CI: abbiamo dei prefissi che non vanno bene. Non è nemmeno UD: abbiamo una codifica ambigua per 111.

Esempio: Siano

$$m = 5 \quad | \quad c : 1, 001, 0000, 01, 0001.$$

Fai alcune considerazioni su questo codice.

Sicuramente è CI: non abbiamo prefissi, quindi è anche UD.

Posso vederlo come codice a virgola con il simbolo 1 che fa da separatore tra tutte le sequenze di 0.

Esempio: Siano

$$m = 5 \quad | \quad c : 000, 001, 01, 111, 110.$$

Fai alcune considerazioni su questo codice.

Sicuramente è CI: non abbiamo prefissi, quindi è anche UD.

Non è ottimale perché due foglie dell'albero non vengono coperte. Si potrebbe ridurre $\mathbb{E}[l_c]$ se scambiassimo 110 con 10 e 111 con 11.

Esempio: Siano

$$X = \{x_1, ...x_6\} \quad | \quad d = 2 \quad | \quad P = \left\{\frac{1}{15}, \frac{1}{3}, \frac{1}{6}, \frac{1}{9}, \frac{1}{5}, \frac{1}{29}\right\}.$$

Costruisci un codice di Shannon per queste lunghezze.

Le probabilità non sommano a 1.

Esempio: Siano

$$X = \{x_1, ...x_6\} \quad | \quad d = 2 \quad | \quad P = \left\{\frac{1}{12}, \frac{1}{3}, \frac{1}{5}, \frac{1}{3}, \frac{1}{72}, x\right\}.$$

Costruisci un codice di Shannon per queste lunghezze.

Le probabilità devono sommare a 1 quindi

$$p_6 = 1 - \left(\frac{1}{12} + \frac{1}{3} + \frac{1}{5} + \frac{1}{3} + \frac{1}{72}\right) = 1 - \frac{347}{360} = \frac{13}{360}.$$

- $l_1 \geq \lceil \log_2(12) \rceil = 4 \longrightarrow 1100$;
- $l_2 \geq \lceil \log_2(3) \rceil = 2 \longrightarrow 01$;
- $l_3 \geq \lceil \log_2(5) \rceil = 3 \longrightarrow 100$;
- $l_4 \geq \lceil \log_2(3) \rceil = 2 \longrightarrow 00$;
- $l_5 \geq \lceil \log_2(72) \rceil = 7 \longrightarrow 1111111$;
- $l_6 \geq \lceil \log_2(\approx 28) \rceil = 5 \longrightarrow 11010$.