Palermo

CODIFICA DI SORGENTE

Parte 1

Lezioni del corso di

TRASMISSIONE ED ELABORAZIONE NUMERICA DEI SEGNALI / COMUNICAZIONI ELETTRICHE (SUN)

AA 2016-17

Prof FRANCESCO A. N. PALMIERI

Capitolo 3

Codifica

In un sistema di comunicazione è spesso necessario trasformare la struttura di un alfabeto in un altro alfabeto ai fini di ottenere delle rappresentazioni più efficienti o più adatte al trasporto dei simboli. Tale operazione, detta *codifica*, consiste nella trasformazione di un alfabeto in un altro alfabeto, anche di dimensionalità diversa, detto *alfabeto di codice*.

Si parla di *codifica di sorgente*, quando a simboli di sorgente vengono associati altri simboli secondo criteri di efficienza dettati solo dalle caratteristiche della sorgente. Tipicamente, quando una sorgente di informazione mostra caratteristiche di *ridondanza*, la codifica di sorgente mira a fornire una rappresentazione più compatta dell'informazione (*compressione*). Si parla di *compressione senza perdite* quando la ricostruzione ottenibile dalla decodifica coincide esattamente con ciò che è stato generato dalla sorgente. Diversamente, si parla di *compressione con perdite* quando la ricostruzione fornita dal decodificatore non è esattamente coincidente con l'ingresso del codificatore (anche se tipicamente abbastanza fedele). Nelle applicazioni ci sono entrambi situazioni in cui è necessario utilizzare una codifica senza perdite e casi in cui è possibile accettare delle degradazioni.

Una codifica che tiene conto delle caratteristiche del canale è detta *codifica di canale*. Tipicamente, una codifica di canale mira a introdurre le opportune *protezioni* ai simboli trasmessi contro le distorsioni (errori) dovute al canale, generalmente inaffidabile. La codifica di canale consiste principalmente nella introduzione di *ridondanza* "ad hoc" ai fini di ottenere delle riproduzioni all'uscita del ricevitore repliche della sorgente che siano le più fedeli possibili .

Una codifica che tiene conto di entrambi sorgente e canale è detta *codifica congiunta sorgente-canale*.

Nel seguito di questo capitolo ci limiteremo a discutere la codifica di sorgente, rimandando la codifica di canale ai prossimi capitoli e la codifica congiunta sorgente-canale a successivi approfondimenti.

3.0.1 Definizione di codice

Una *codifica* è una operazione che associa due insiemi di simboli. Più formalmente, dato un alfabeto sorgente $\mathcal{A}$, un codice $\mathcal{C}$ è una applicazione su $\mathcal{A}$ che assume valori in un altro insieme $\mathcal{B}$ detto *alfabeto di codice*:

$$\mathcal{C} : \mathcal{A} = \{a_1, a_2, ..., a_n\} \longrightarrow \mathcal{B} = \{b_1, b_2, ..., b_m\} \tag{3.1}$$

Le cardinalità n e m dei due alfabeti possono essere diverse. Anche la associazione tra gli elementi dei due alfabeti può non essere biunivoca. Infatti è possibile definire codici non biunivoci, anche se la situazione più comune è quella per cui $n = m$ e la associazione è perfettamente reversibile. La figura 3.1 mostra alcuni codici in forma di grafo. I codici $\mathcal{C}_1$ e $\mathcal{C}_2$ sono biunivoci in quanto la associazione ingresso-uscita non è ambigua, sia in codifica che in decodifica. Il codice $\mathcal{C}_3$ è univoco in codifica ma non in decodifica: da un simbolo dell'alfabeto di codice non è sempre possibile associare il simbolo dell'alfabeto sorgente in maniera univoca. Il viceversa avviene nel codice $\mathcal{C}_4$ in cui la decodifica è univoca, a differenza della codifica.

Esempio 3.1 Un codice $\mathcal{C}$ consiste nella seguente associazione dei numeri interi da 1 a 6, a sei lettere dell'alfabeto

$$\mathcal{C} : \mathcal{A} = \{1, 2, 3, 4, 5, 6\} \longleftrightarrow \mathcal{B} = \{a, b, c, d, e, f\} \tag{3.2}$$

Il codice è biunivoco.

Esempio 3.2 Un codice $\mathcal{C}$ consiste nella trascrizione in codice binario decimale (BCD) dei numeri interi da 0 a 7

$$\mathcal{C} : \mathcal{A} = \{0, 1, 2, 3, 4, 5, 6, 7\} \longleftrightarrow \mathcal{B} = \{000, 001, 010, 011, 100, 101, 110, 111\} \tag{3.3}$$

Anche in questo caso la associazione è biunivoca.

Esempio 3.3 Un codice $\mathcal{C}$ consiste nella seguente associazione caratteri-stringhe binarie:

$$\mathcal{C} : \mathcal{A} = \{a, b, c, d, e\} \longleftrightarrow \mathcal{B} = \{0, 01, 0100, 011, 111\} \tag{3.4}$$

Il codice è biunivoco.

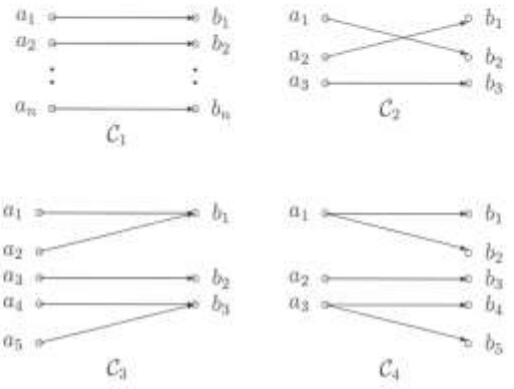


Figura 3.1: Esempi di codici in forma grafica

Esempio 3.4 In codice ASCII (America Standard Code for International Interchange) è uno dei primi esempi di codici per la comunicazione tra computer. Esso fu adottato nel 1963 e poi aggiornato nel 1967. Esso consiste nella associazione dei 128 caratteri mostrati in tabella 3.1 a delle stringhe binarie di 7 bit. L'insieme di caratteri include le lettere dell'alfabeto maiuscole e minuscole, e alcuni caratteri speciali. La tabella si legge associando ad ognuno dei caratteri la stringa di bit identificata dalla riga e dalla colonna corrispondente (ad esempio al carattere "G" va associata la stringa binaria 1000111). Si tratta di un codice con $n = m = 128$, ovviamente biunivoco. Nei sistemi numerici viene qualche volta aggiunto ai sette bit della parola codice un ottavo bit (non riportato in tabella), calcolato come bit di parità. Se la parità è *pari*, il bit 8 sarà "1" se il numero di bit nella parola è pari, e "0" altrimenti. Se la parità è dispari si adotta la associazione opposta (il roulo del bit di parità sarà più evidente nei prossimi capitoli). Si noti come anche dopo la introduzione del bit di parità il codice è biunivoco con $n = m = 128$. In questo caso le parole dell'alfabeto di codice sono stringhe binarie di lunghezza 8 e solo 128 delle 256 stringhe binarie di 8 bit vengono utilizzate.

Esempio 3.5 Un esempio interessante di codice, progettato da "madre natura", è il *codice genetico*. Le cellule degli organismi viventi sintetizzano le loro catene proteiche tramite l'mRNA prelevando l'informazione dal DNA. La trascrizione

				7	0	0	0	0	1	1	1	1
				6	0	0	1	1	0	0	1	1
4	3	2	1	5	0	1	0	1	0	1	0	1
0	0	0	0		NUL	DLE	SP	0	@	P	'	p
0	0	0	1		SOH	DC1	!	1	A	Q	a	q
0	0	1	0		STX	DC2	"	2	B	R	b	r
0	0	1	1		ETX	DC3	#	3	C	S	c	s
0	1	0	0		EOT	DC4	$	4	D	T	d	t
0	1	0	1		ENQ	NAK	%	5	E	U	e	u
0	1	1	0		ACK	SYN	&	6	F	V	f	v
0	1	1	1		BEL	ETB	'	7	G	W	g	w
1	0	0	0		BS	CAN	(	8	H	X	h	x
1	0	0	1		HT	EM	)	9	I	Y	i	y
1	0	1	0		LF	SUB	*	:	J	Z	j	z
1	0	1	1		VT	ESC	+	;	K	[	k	{
1	1	0	0		FF	FS	,	<	L	\	l	:
1	1	0	1		CR	GS	-	=	M	]	m	}
1	1	1	0		SO	RS	.	>	N	^	n	~
1	1	1	1		SI	US	/	?	O	_	o	DEL

Tabella 3.1: Il codice ASCII

avviene mediante la concatenazione dei quatto acidi nucleici (codoni), U (Uracil), A (Adenine), G (Guanine), C (Cytosine). A sequenze di parole codice di tre codoni corrispondono i 20 amminoacidi più il simbolo di terminazione. Tale associazione è ridondante, nel senso che più triplette corrispondono allo stesso amminoacido. Il codice, con la indicazione mnemonica degli amminoacidi, è riportato in tabella 3.2. Nota come la ridondanza non sia uniforme[1]. E' evidente come la associazione amminoacido-parola codice non sia univoca, diversamente da quella inversa che garantisce la univocità della decodifica. In questo caso l'alfabeto sorgente, costituito dai 20 amminoacidi più il simbolo di terminazione, ha dimensione $n = 21$. L'alfabeto di codice è costituito dalle $m = 4^3 = 64$ possibili terne quaternarie, molte delle quali associate allo stesso simbolo di sorgente.

Esempio 3.6 *Il codice Morse* rappresenta uno dei primi esempi di codice numerico. Esso fu progettato insieme alla invenzione del telegrafo. Samuel Finley

[1]Anche se le parole codice sembrano essere raggruppate per similarità, non è interamente chiaro il perchè di tale associazione. Interessanti considerazioni sulla struttura del codice genetico sono riportate in: A. Findley, S.P. McGlynn, G.L. Findley, *The Geometry of Genetics*, Wiley, 1989.

Polinesi 5

	Amminoacido	Abbreviazione	Parola codice
1	Alanine	Ala	GCU,GCC, GCA, GCG
2	Arginine	Arg	AGA, AGG, CGU, CGC, CGA, CGG
3	Aspargine	Asn	AAU, AAC
4	Aspartic acid	Asp	GAU, GAC
5	Cysteine	Cys	UGU, UGC
6	Glutamine	Gln	CAA, CAG
7	Glutamic acid	Glu	GAA, GAG
8	Glycine	Gly	GGU, GGC, GGA, GGG
9	Histidine	His	CAU, CAC
10	Isoleucine	Ile	AUU, AUC, AUA
11	Leucine	Leu	UUA, UUG, CUU, CUC, CUA, CUG
12	Lysine	Lys	AAA, AAG
13	Methionine	Met	AUG
14	Phenylalanine	Phe	UUU, UUC
15	Proline	Pro	CCU, CCC, CCA, CCG
16	Serine	Ser	AGU, AGC, UCU, UCC, UCA, UGC
17	Threonine	Thr	ACU, ACC, ACA, AGC
18	Tryptophan	Trp	UGG
19	Tyrosine	Tyr	UAU, UAC
20	Valine	Val	GUU, GUC, GUA, GUG
-	Term. codon	TC	UAA, UAG, UGA

Tabella 3.2: Il codice genetico

Breese Morse (1791-1872), che fu l'inventore del telegrafo (1840), osservò che non tutte le lettere dell'alfabeto sono presenti in uguale misura in un testo scritto. Ad esempio, nella lingua inglese, la "e", la "i", la "t", sono più frequenti rispetto alle altre lettere. Morse pertanto propose uno schema di codifica binaria basata su parole codice costruite dai due simboli: "·" (punto) e "–" (linea). Il criterio guida per la costruzione del codice fu: associare stringhe più brevi ai caratteri più probabili. Ciò risulta evidentemente in una lunghezza media del messaggio codificato più breve rispetto ad una codifica a lunghezza fissa. La tabella 3.3 mostra la corrispondenza tra caratteri alfabetici, numerici e speciali con le parole codice. La temporizzazione è tale che la lunghezza della linea è circa tre volte quella del punto. Le spaziature all'interno di una parola codice sono pari alla durata di un punto. In una sequenza di parole codice, al fine di distinguere una parola codice dalla precedente e dalla successiva, viene usata una pausa temporale tra caratteri della durata di circa tre punti. La pausa di separazione tra parole è della durata

di cinque punti.

3.1 Codici binari a lunghezza fissa e variabile

Quando gli elementi dell'alfabeto di codice sono stringhe di simboli binari, si parla di *codice binario*. Più formalmente:

Un *codice binario a lunghezza fissa L*, detto anche *codice binario a blocchi di lunghezza L*, è la corrispondenza tra gli n simboli dell'alfabeto sorgente $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ e l'insieme di parole codice $\mathcal{B} = \{b_1, b_2, ..., b_m\}$ costituito da un sottoinsieme delle 2^L stringhe binarie di lunghezza L. Esempi notevoli sono: il codice ASCII, il codice BCD, eccetera. Se la cardinalità dell'alfabeto di codice è $m = 2^L$, il codice è detto *completo*.

Un *codice binario a lunghezza variabile* è una corrispondenza tra gli m simboli dell'alfabeto sorgente $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ e l'insieme di parole codice $\mathcal{B} = \{b_1, b_2, ..., b_m\}$, dove ogni b_i è una stringa di ℓ_i bit.

Si sarebbe tentati di definire la lunghezza media di un codice come

$$L_{\text{av}} = \frac{1}{m}\sum_{i=1}^{m} \ell_i \quad \text{bit}, \tag{3.5}$$

ma tale definizione non terrebbe in conto la probabilità con cui le varie parole codice vengono utilizzate. Pertanto la lunghezza media è più propriamente definita come la media statistica della lunghezza delle parole codice: detta B la variabile aleatoria che rappresenta la parola codice emessa dal codificatore, la *lunghezza media del codice* è:

$$\overline{L} = E[\ell] = \sum_{i=1}^{m} \ell_i Pr\{B = b_i\} \quad \text{bit}. \tag{3.6}$$

Ovviamente se le parole codice sono equiprobabili, $\overline{L} = L_{\text{av}}$. Nel seguito di questo capitolo, se non diversamente specificato, faremo riferimento a codici univoci con $m = n$. Pertanto la distribuzione delle probabilità dell'alfabeto sorgente coincide con quella dell'alfabeto di codice, ovvero $Pr\{B = b_i\} = Pr\{S = a_i\} = p_i$, $i = 1, ..., m$. Inoltre, la totale univocità della corrispondenza tra alfabeto di codice e alfabeto di sorgente ci consente di concentrarci esclusivamente sulla struttura dell'alfabeto di codice. Nel seguito faremo riferimento a "alfabeto di codice", o a "codice" in maniera intercambiabile sottintendendo la struttura biunivoca della corrispondenza tra l'alfabeto di sorgente e l'alfabeto di codice.

$\mathcal{A}$	$\mathcal{B}$	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{A}$	$\mathcal{B}$
A	·–	1	·– – – –	.	·–·–·–
B	–···	2	··– – –	,	– – ··– –
C	–·–·	3	···– –	?	··– –··
D	–··	4	····–	"	·–··–·
E	·	5	·····	:	– – – ···
F	··–·	6	–····	;	–·–·–·
G	– –·	7	– –···	)	–·– –·–
H	····	8	– – –··	(	–·– –·
I	··	9	– – – –·	'	·– – – –·
J	·– – –	0	– – – – –	Double dash (Break)	–····–
K	–·–			Error	·····
L	·–··			/	–··–·
M	– –			End of message (AR)	·–·–·
N	–·			End of Transmission (SK)	···–·–
O	– – –			SOS	···– – –···
P	·– –·			Wait (AS)	·–···
Q	– –·–				
R	·–·				
S	···				
T	–				
U	··–				
V	···–				
W	·– –				
X	–··–				
Y	–·– –				
Z	– –··				

Tabella 3.3: Il codice Morse

3.1.1 Codici univocamente decodificabili

Affinchè uno schema di codifica sia utilizzabile, deve verificarsi che una sequenza di simboli di sorgente codificati siano riottenibili univocamente al decodificatore. Assumendo che il meccanismo di trasporto sia perfetto, ovvero che non ci siano perdite o errori, il ricevitore dovrà operare su una sequenza binaria costituita dalla concatenazione di parole codice. E' evidente come sorga un primo importante problema: come identificare univocamente le parole codice se non si è introdotto un simbolo di separazione? Ad esempio nel codice Morse sono state introdotte delle pause ai fini di ottenere una discriminazione univoca tra le varie parole codice. Tale soluzione può risultare molto inefficiente a causa della ridondanza conseguente la introduzione dei simboli ausiliari. Certo, se le parole codice hanno una lunghezza fissa L, si può fare affidamento su un decodificatore sincronizzato che estrae esattamente L bit alla volta e li traduce secondo la sua tabella di decodifica. Viceversa se il codice è a lunghezza variabile, il problema resta: come è possibile suddividere la sequenza binaria in maniera univoca nelle sue parole codice costituenti?

Per vederci un pò più chiaro diamo prima la seguente definizione con riferimento ad una sequenza di lunghezza finita. Si dice che un codice è *univocamente decodificabile*, se è possibile risalire in maniera univoca alla sequenza delle parole codice costituenti, a partire da una qualunque sequenza di lunghezza finita costituita dalla concatenazione di parole codice. In particolare, se m è la cardinalità dell'alfabeto di codice e K la lunghezza della sequenza, diremo che il codice è univocamente decodificabile se e solo se per ognuna delle m^K configurazioni possibili, la identificazione delle K parole di codice costituenti è univoca. Quindi in generale, come è possibile intuire, anche disponendo di codici univoci, potrebbe essere necessario osservare tutta la sequenza prima di poter capire dove cadono i confini delle parole codice costituenti. E' piuttosto complesso verificare in generale se un codice è univocamente decodificabile se esso non ricade nelle categorie che saranno specificate qui di seguito. Infatti nelle applicazioni, anche per evitare che i decodificatori necessitino di complessi algoritmi di decodifica e per far si che essi possano operare anche su sequenze semi-infinite, si preferisce adottare la sottoclasse dei cosidetti *codici istantanei*. Un codice istantaneo è tale che il decodificatore nello scorrere la sequenza binaria da sinistra verso destra è in grado di riconoscere *immediatamente* l'inizio e la terminazione di ogni parola codice.

Una classe molto importante di codici istantanei è quella dei cosidetti

$\mathcal{A}$	Π	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$
a_1	0.1	0	0	01
a_2	0.25	1	10	011
a_3	0.5	00	110	0111
a_4	0.1	01	1110	01111
a_5	0.05	10	1111	011111
	$\overline{L}$	1.65	2.7	3.75

Tabella 3.4: Esempi di codici di sorgente. L'entropia della sorgente è $\mathcal{H}(\Pi) = 1.88$ bit.

codici a prefisso. In un codice a prefisso, ogni stringa binaria dell'alfabeto di codice è una sequenza di bit, che non costituisce l'inizio di nessuna altra parola codice. Più precisamente, un codice a prefisso è un codice costituito da parole codice tra cui nessuna di essa è prefisso di un'altra. Ad esempio: la parola 001 è prefisso di 00111, ma non è prefisso di 0101; la parola codice 0 è prefisso di 011, di 00, ma non di 100; la parola codice 10 è prefisso di 101 e di 100, ma non di 010. Costruendo con tale vincolo l'insieme delle parole codice, siamo sicuri che il decodificatore nello scorrere la sequenza binaria da sinistra verso destra potrà riconoscere in maniera univoca le parole codice senza bisogno di simboli di terminazione.

Tabella 3.4 mostra tre esempi di codici associati a cinque simboli di sorgente $\{a_1, a_2, a_3, a_4, a_5\}$.

Si nota immediatamente come codice $\mathcal{C}_1$ non sia un codice a prefisso, in quanto 0 è prefisso di 00 e di 01, e 1 è prefisso di 10. Si tratta inoltre di un codice non univocamente decodificabile visto che se consideriamo la concatenazione di 0, 1, 01, ovvero 0101, essa può essere decodificata come 01, 01, oppure come 0, 10, 1, oppure come 0, 1, 0, 1. Codice $\mathcal{C}_3$ è anch'esso un codice che non soddisfa la condizione del prefisso perché 01 è prefisso di tutte le altre parole codice. Codice $\mathcal{C}_3$ è però un codice univocamente decodificabile essendo anche un codice istantaneo, visto che il bit 0 può essere usato come identificatore dell'inizio di tutte le parole codice. Codice $\mathcal{C}_2$ invece, è un codice a prefisso in quanto soddisfa la condizione richiesta.

La figura 3.2 mostra un diagramma di Venn con la relazione di inclusione esistente tra codici univovocamente decodificabili, codici istantanei e codici a prefisso.

E' molto utile descrivere la struttura di un codice mediante il corrispondente albero binario. Si osservi figura 3.3 in cui è riportata la struttura ad

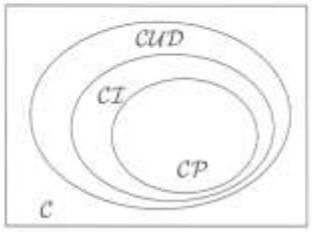


Figura 3.2: La relazione di inclusione tra i vari tipi di codice: $\mathcal{CUD}$-Codici Univocamente Decodificabili; $\mathcal{CI}$-Codici Istantanei; $\mathcal{CP}$-Codici a Prefisso.

albero dei tre codici di tabella 3.4. L'albero si legge partendo dalla radice e percorrendone i rami fino a incontrare una parola codice (pallino). L'albero consiste nella associazione di uno "0" e un "1" ad ogni ramo che si diparte da un nodo. Si noti come il codice a prefisso corrisponda ad un codice le cui parole codice sono associate solo ai nodi terminali dell'albero. Questo conferma come il decodificatore nell'interpretare una sequenza concatenata di parole codice, percorrendo l'albero dalla radice verso i rami, sarà certo di avere osservato una parola codice nel momento in cui avrà raggiunto un nodo terminale. Ad esempio, nel decodificare la sequenza del codice $\mathcal{C}_2$, 110101111, abbiamo che la decodifica è univocamente 110, 10, 1111, ovvero a_3, a_2, a_5.

3.1.2 La disuguaglianza di Kraft-McMillan

Dagli esempi forniti nella sezione precedente è evidente come la costruzione di un codice binario a prefisso richieda che le parole codice siano associate esclusivamente alle terminazioni dell'albero binario che lo rappresenta. Quindi se all'alfabeto di codice sono associate le lunghezza diverse $\{\ell_1, \ell_2, ..., \ell_m\}$, è abbastanza intuitivo che non troppe parole di codice possono avere un lunghezza piccola in quanto l'albero non avrebbe abbastanza terminazioni disponibili. Esiste infatti un risultato analitico che ci fornisce una condizione sulle possibili lunghezze di un codice a prefisso.

Teorema 3.1 *(Disuguaglianza di Kraft) Condizione necessaria e sufficiente per l'esistenza di un codice binario a prefisso con parole codice di lunghezze*

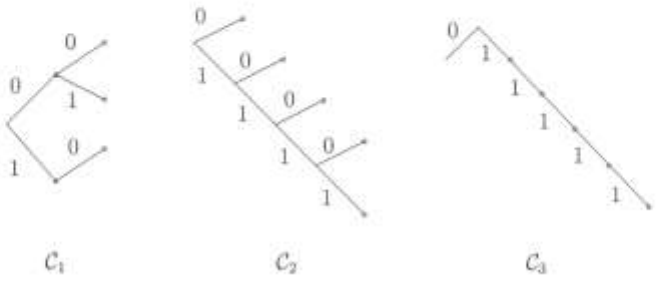


Figura 3.3: Gli alberi binari corrispondenti ai codici di tabella 3.4

$\{\ell_1, \ell_2, ..., \ell_m\}$ *è che sia soddisfatta la seguente disuguaglianza*

$$\sum_{i=1}^{m} 2^{-\ell_i} \leq 1. \tag{3.7}$$

Prova: Come già detto, le parole codice di un codice a prefisso sono associate ai nodi terminali di un albero. Cominciamo considerando il codice associato ad un albero completo di lunghezza L. Il numero totale di parole codice è in questo caso $mc = 2^L$. Quindi il risultato del teorema vale con il segno di uguaglianza poiché

$$\sum_{i=1}^{mc} 2^{-\ell_i} = \sum_{i=1}^{2^L} 2^{-L} = 2^L 2^{-L} = 1. \tag{3.8}$$

Figura 3.4(a) mostra l'albero associato ad un codice completo di lunghezza $L = 3$. Ora se consideriamo un codice a lunghezza variabile di cardinalità m, dove L è la lunghezza massima delle parole codice, abbiamo che ogni parola codice di lunghezza $\ell_i$nell'albero ostruisce $2^{L-\ell_i}$ nodi terminali dell'albero completo. Figura 3.4(b) mostra l'albero corrispondente a un codice di lunghezze $\{3, 3, 2, 1\}$. Infatti le prime due parole codice ostruiscono un solo nodo ognuna. La terza parola codice ostruisce due nodi terminali. Analogamente la quarta ne ostruisce quattro. Se tutti i nodi terminali di ordine $\{\ell_1, \ell_2, ..., \ell_m\}$ sono utilizzati, la somma dei nodi ostruiti è pari al numero totale dei nodi terminali disponibili, ovvero

$$\sum_{i=1}^{m} 2^{L-\ell_i} = 2^L. \tag{3.9}$$

Anche in questo caso $\sum_{i=1}^{m} 2^{-\ell_i} = 1$. Tale situazione è quella di un codice *compatto* i cui nodi terminali sono tutti utilizzati.

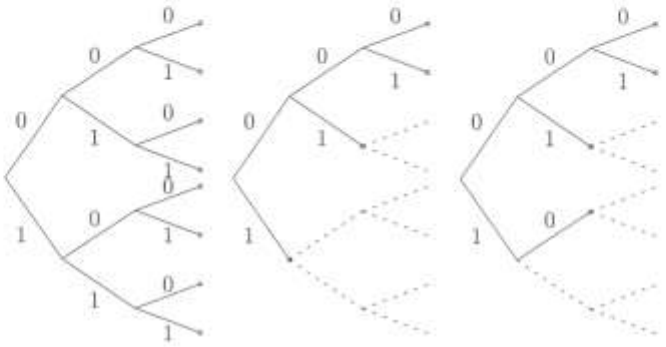


Figura 3.4: Gli alberi binari corrispondenti a codici a prefisso: (a) un codice completo; (b) un codice compatto; (c) un codice non compatto

Questa non è la situazione più generale per un codice a prefisso poiché qualche nodo terminale potrebbe non essere utilizzato, come in figura 3.4(c), le cui lunghezze sono $\{3, 3, 2, 2\}$. Pertanto in generale la somma dei nodi costruiti è solo *al più* 2^L. Quindi

$$\sum_{i=1}^{m} 2^{L-\ell_i} \leq 2^L, \tag{3.10}$$

che immediantamente implica la disuguaglianza di Kraft. Questo prova che dato un codice a prefisso vale la disuguaglianza (condizione necessaria). Per provare la condizione sufficiente bisogna supporre che la disuguaglianza sia verificata e provare che è possibile costruire un codice a prefisso. Se n_j è il numero di parole codice di lunghezza j e L è la lunghezza massima, abbiamo che la disuguaglianza deve essere soddisfatta per tutti i seguenti casi

$$n_1 2^{-1} \leq 1,$$
$$n_1 2^{-1} + n_2 2^{-2} \leq 1,$$
$$\ldots\ldots \tag{3.11}$$
$$n_1 2^{-1} + n_2 2^{-2} + \ldots + n_L 2^{-L} \leq 1. \tag{3.12}$$
$$\tag{3.13}$$

Le disuguaglianze possono essere riscritte come

$$n_1 \leq 2,$$

$$n_2 \leq 2^2 - n_1 2,$$
$$\ldots\ldots \tag{3.14}$$
$$n_L \leq 2^L - n_1 2^{L-1} - n_2 2^{L-2} - \ldots - n_{L-1} 2. \tag{3.15}$$
$$\tag{3.16}$$

Le disuguaglianze possono essere intrepretate come le condizioni per la costruzione del codice sull'albero: al primo livello (lunghezza uno) ci sono 2 nodi disponibili, al livello 2 (lughezza 2) ci sono 2^2 nodi disponibili meno quelli già ostruiti dalle parole codice nel livello 1. Al generico livello, la disuguaglianza esprime il numero di nodi disponibili avendo ecluso quelli già ostruiti ai livelli precedenti. Quindi è possibile costruire un codice a prefisso se le lunghezze soddisfano la disuguaglianza di Kraft. $\triangle$

Si noti come la disuguaglianza di Kraft non ci dica come costruire un codice efficiente, ma ci indichi soltanto quali sono le conseguenze della condizione del prefisso sulla lunghezza delle parole. Infatti, se riscriviamo la disuguaglianza come

$$\sum_{i=1}^{n} \frac{1}{2^{\ell_i}} \leq 1, \tag{3.17}$$

appare evidente la limitazione sulla lunghezza minima delle parole codice: in un codice a prefisso, non troppe parole codice possono essere corte. I casi limite si hanno quando l'albero di codice è un albero compatto o completo, in qual caso la condizione vale con il segno di uguaglianza. Si noti negli esempi di tabella 3.4 come codice $\mathcal{C}_1$ violi la condizione di Kraft, mentre il codice a prefisso $\mathcal{C}_2$ la soddisfi.

La domanda naturale che si pone a questo punto è: quanto limitativo è considerare solo i codici a prefisso? In altre parole, nel dominio più grande dei codici univocamente decodificabili esistono codici mediamente più corti? La risposta finale a questa domanda fu data da McMillan che dimostrò nel 1956 che la condizione del teorema di Kraft (1949) *vale anche per tutti i codici univocamente decodificabili.* Da qui il nome del teorema, che è noto come *disuguaglianza di Kraft-McMillan.* La estensione di McMillan del teorema è di fondamentale importanza, anche per i teoremi che seguiranno, perchè consente di trattare tutti i codici univocamente decodificabili in un'unica classe, senza doversi limitare ai codici istantanei, o più in particolare ai codici a prefisso. Inoltre i vincoli sulla lunghezza delle parole codice rende i codici non a prefisso poco interessanti per le applicazioni.

Teorema 3.2 *(Disuguaglianza di McMillan) Condizione necessaria e sufficiente per l'esistenza di un codice binario univocamente decodificabile è che*

le lunghezze delle parole codice $\{\ell_1, \ell_2, ..., \ell_m\}$ *soddisfino la disuguaglianza di Kraft*

$$\sum_{i=1}^{m} 2^{-\ell_i} \leq 1. \tag{3.18}$$

Prova: La condizione sufficiente è immediata dal teorema di Kraft in quanto, se la condizione è verificata, esiste sicuramente un codice a prefisso e quindi un codice univocamente decodificabile. Per la condizione necessaria dobbiamo supporre che ci sia un codice univocamente decodificabile con lunghezze $\{\ell_1, \ell_2, ..., \ell_m\}$ e derivare la disuguaglianza. Il passaggio cruciale per la dimostrazione è considerare l'espressione

$$(\sum_{i=1}^{m} 2^{-\ell_i})^K = \left(\sum_{i_1=1}^{m} 2^{-\ell_{i_1}}\right)\left(\sum_{i_2=1}^{m} 2^{-\ell_{i_2}}\right)\cdots\left(\sum_{i_m=1}^{m} 2^{-\ell_{i_m}}\right), \tag{3.19}$$

Espandendo il secondo membro dell'equazione abbiamo

$$(\sum_{i=1}^{m} 2^{-\ell_i})^K = \sum_{i_1=1}^{m}\sum_{i_2=1}^{m}\cdots\sum_{i_m=1}^{m} 2^{-\ell_{i_1}-\ell_{i_2}-\ldots-\ell_{i_m}}, \tag{3.20}$$

Sia l'indice J definito come

$$J = \ell_{i_1} + \ell_{i_2} + ... + \ell_{i_m}. \tag{3.21}$$

Si noti che J è anche la lunghezza totale della sequenza di K parole codice costituita da parole di lunghezza $\{\ell_{i1}, \ell_{i2}, ..., \ell_{im}\}$ espressa in bit. Se L è la lunghezza massima delle parole codice, J è almeno K e al più KL. Chiamando con N_J il numero di termini corrispondenti al termine 2^{-J} nello sviluppo, possiamo scrivere

$$(\sum_{i=1}^{m} 2^{-\ell_i})^K = \sum_{J=K}^{KL} N_J 2^{-J}. \tag{3.22}$$

Si noti ora che N_J è anche il numero di sequenze di K parole codice che hanno esattamente lunghezza complessiva pari a J. Se il codice è univocamente decodificabile, N_J non può essere superiore a 2^J. Pertanto

$$(\sum_{i=1}^{m} 2^{-\ell_i})^K \leq \sum_{J=K}^{KL} 2^J 2^{-J} = KL - K + 1 \leq KL, \tag{3.23}$$

poiché K è sicuramente maggiore di uno. Poichè la disuguaglianza deve valere per qualunque K, deve necessariamente accadere che

$$\sum_{i=1}^{m} 2^{-\ell_i} \leq 1. \tag{3.24}$$

La ragione di tale conclusioneè che se avessimo $\sum_{i=1}^{m} 2^{-\ell_i} > 1$, facendo crescere K, potrei trovare un valore K per cui $(\sum_{i=1}^{m} 2^{-\ell_i})^K > KL$, che violerebbe la condizione trovata. Questo prova l'asserto del teorema. $\triangle$

Tra gli esempi di tabella 3.4, è possibile verificare come il codice $\mathcal{C}_3$ che ha lunghezze $\{2, 3, 4, 5, 6\}$ soddifa la disuguaglianza di McMillan: $\frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \frac{1}{2^5} + \frac{1}{2^6} = \frac{31}{64} < 1$. E' opportuno comunque enfatizzare che avere un codice che soddisfa la disuguaglianza di McMillan, ovvero che ha le lunghezze opportune, non garantisce che il codice sia univocamente decodificabile. Il teorema di McMillan garantisce solo che esiste almeno un codice univocamente decodificabile con quelle lunghezze.

3.2 Codici M-ari

Anche se i pricipali codici di interesse sono quelli costituiti da parole di codice che sono delle stringhe binarie, va menzionato che tutte le definizioni e i risultati appena esposti possono essere applicati a codici formati da simboli M-ari (ternari, quaternari, eccetera). In particolare:

Un *codice M-ario a lunghezza fissa* L è la corrispondenza tra gli n simboli dell'alfabeto sorgente $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ e l'insieme di parole codice $\mathcal{B} = \{s_1, s_2, ..., s_m\}$ costituito da un sottoinsieme delle M^L stringhe M-arie di lunghezza L. Se tutte le stringhe $m = M^L$ sono utilizzate, il codice è detto *completo*.

Un *codice M-ario a lunghezza variabile* è una corrispondenza tra gli n simboli dell'alfabeto sorgente $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ e l'insieme di parole codice $\mathcal{B} = \{s_1, s_2, ..., s_m\}$, dove ogni s_i è una stringa di ℓ_i simboli M-ari.

La disuguaglianza di Kraft-McMillan si scrive per un codice M-ario come

$$\sum_{i=1}^{n} M^{-\ell_i} \leq 1, \tag{3.25}$$

Anche a un codice M-ario è possibile associare un albero. L'albero è tale che da ogni nodo si dipartono M rami. La lunghezza media di un codice M-ario è analogamente

$$\overline{L} = E[\ell] = \sum_{i=1}^{m} \ell_i Pr\{B = s_i\}. \tag{3.26}$$

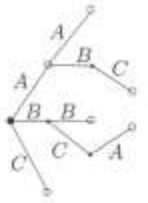
Essa si misura in unità diverse a seconda del valore di M, ovvero in trit (per $M = 3$), quadrit (per $M = 4$), eccetera. Se la si vuole esprimere in

bit, basta moltiplicare il valore ottenuto per il fattore di conversione $\log_2 M$. Alcuni esempi chiariranno in seguito le semplici formule.

Esempio 3.7 Si consideri il seguente codice ternario

$$\mathcal{A} = \{a_1, a_2, a_3, a_{4,5}, a_6\} \longmapsto \mathcal{B} = \{A, BB, ABC, AA, BCA, C\}. \tag{3.27}$$

L'alfabeto ternario con cui sono costruite le stringhe è $\{A, B, C\}$. L'albero relativo al codice è mostrato in figura, dove sono stati tracciati solo i rami relativi alle stringhe ternarie utilizzate.



Il codice non è compatto in quanto non tutti i nodi terminali sono stati utilizzati. Inoltre il codice non è a prefisso. Le lunghezze sono $\{1, 2, 3, 2, 3, 1\}$ trit. Anche se esse soddisfano la disuguaglianza di Kraft-McMillan,

$$\frac{1}{3} + \frac{1}{3^2} + \frac{1}{3^3} + \frac{1}{3^2} + \frac{1}{3^3} + \frac{1}{3} = \frac{26}{27} < 1, \tag{3.28}$$

il codice non è univocamente decodificabile. Infatti basta considerare la stringa a_1a_4, che è codificata come AAA. Essa può essere decodificata come a_1a_4 oppure come $a_1a_1a_1$. Se la distribuzione di probabilità dei simboli è $\{0.1, 0.1, 0.2, 0.4, 0.1, 0.1\}$, la lunghezza media è $\overline{L} = 2.1\text{trit} = 2.1 \log_2 3 = 3.33$ bit.

Capitolo 4

Compressione di sorgente

L'efficienza di un codice è collegata chiaramente alla sua lunghezza media. Abbiamo osservato come sarebbe desiderabile ottenere dei codici che abbiano una lunghezza media più piccola possibile e che siano univocamente decodificabili. Uno dei risultati fondamentali della teoria dell'informazione è il seguente teorema che lega l'entropia alla minima lunghezza media di un codice univocamente decodificabile.

Teorema 4.1 *Per ogni codice binario univocamente decodificabile, la lunghezza media soddisfa la seguente disuguaglianza*

$$\overline{L} \geq \mathcal{H}(\Pi). \tag{4.1}$$

Prima di discutere la dimostrazione del teorema, notiamo come tale risultato abbia delle conseguenze applicative formidabili: l'entropia di una sorgente, che possiamo calcolare dalle sue proprietà statistiche, ci fornisce il limite superiore teorico alla bontà di qualunque codice (utile) che andremo a progettare!!!

Si definisce pertanto *efficienza del codice* il rapporto

$$\eta = \frac{\mathcal{H}(\Pi)}{\overline{L}}, \tag{4.2}$$

con $0 < \eta \leq 1$. Guardando di nuovo agli esempi di tabella 3.4 notiamo che il limite inferiore teorico alla lunghezza media sia 1.88 bit. I due codici $\mathcal{C}_2$ e $\mathcal{C}_3$, che sono quelli univocamente decodificabili, hanno rispettivamente

lunghezze pari a 2.7 bit e 3.75 bit. Vedremo in seguito come sia possibile progettare dei codici più efficienti aventi lunghezza media molto più prossima all'entropia.

Prova del Teorema 4.1: Vogliamo dimostrare che per un codice univocamente decodificabile

$$\mathcal{H}(\Pi) - \overline{L} \leq 0 \quad . \tag{4.3}$$

Il termine sinistro dell'equazione può essere riscritto come

$$\begin{aligned} \mathcal{H}(\Pi) - \overline{L} &= -\sum_{i=1}^{m} p_i \log_2 p_i - \sum_{i=1}^{m} p_i \ell_i \\ &= -\sum_{i=1}^{m} p_i \log_2 p_i - \sum_{i=1}^{m} p_i \log_2 2^{\ell_i} \\ &= \sum_{i=1}^{m} p_i \log_2 \frac{2^{-\ell_i}}{p_i}. \end{aligned} \tag{4.4}$$

Poiché $\ln x \leq x - 1$, abbiamo che $\log_2 x \leq (x-1)\log_2 e$. Quindi limitando superiormente tutti i termini della sommatoria, abbiamo il risultato cercato

$$\begin{aligned} \mathcal{H}(\Pi) - \overline{L} &\leq \sum_{i=1}^{m} p_i \left(\frac{2^{-\ell_i}}{p_i} - 1 \right) \log_2 e = \left(\sum_{i=1}^{m} 2^{-\ell_i} - \sum_{i=1}^{m} p_i \right) \log_2 e \\ &= \left(\sum_{i=1}^{m} 2^{-\ell_i} - 1 \right) \log_2 e \leq 0, \end{aligned} \tag{4.5}$$

dove nell'ultimo passaggio si è fatto uso della disuguaglianza di Kraft-McMillan. △

Nel caso dei codici M-ari, è possibile generalizzare il teorema. Poiché in tal caso la lunghezza media del codice si misura in simboli M-ari abbiamo:

$$\mathcal{H}(\Pi) \leq \overline{L} \log_2 M, \tag{4.6}$$

con l'entropia espressa in bit. La prova è la stessa del caso binario e la si tralascia per brevità. Pertanto l'efficienza di un codice M-ario è definita come

$$\eta = \frac{\mathcal{H}(\Pi)}{\overline{L} \log_2 M}. \tag{4.7}$$

4.0.1 Il primo teorema di Shannon

Se le probabilità dei simboli di sorgente fossero esattamente $p_i = 2^{-\ell_i}$, $i = 1, ..., m$, ovvero avessimo un codice con lunghezze

$$\{\ell_1 = \log_2 \frac{1}{p_1}, ..., \ell_m = \log_2 \frac{1}{p_m}\}, \tag{4.8}$$

la efficienza del codice sarebbe massima poiché avremmo esattamente $\mathcal{H}(\Pi) = L$. Tale situazione è ovviamente ideale ed in pratica la distribuzione delle probabilità sui simboli sarà imposta dalla tipologia della sorgente. La domanda è ovviamente: quanto vicini al limite minimo teorico possiamo arrivare nel progettare un codice se le probabilità sono arbitrarie?

Vediamo se è possibile arrivare ad una risposta a questo quesito in maniera costruttiva. Abbiamo già detto che se $\log_2 \frac{1}{p_i}$ fosse un intero, dovremmo scegliere questo numero come lunghezza della parola codice i-esima. Diversamente sembra ragionevole prendere l'intero immediatamente superiore, salvo verificare che tale codice esista e più in particolare sia un codice univocamente decodificabile. Quindi potremmo scegliere le lunghezze delle parole codice come

$$\log_2 \frac{1}{p_i} \leq \ell_i < \log_2 \frac{1}{p_i} + 1, \tag{4.9}$$

per $i = 1, ..., m$. La prima cosa da verificare è che le lunghezze delle parole codice così prese soddisfino la disuguaglianza di Kraft-McMillan. Passando agli esponenziali nella disuguaglianza di sinistra abbiamo

$$\frac{1}{p_i} \leq 2^{\ell_i}. \tag{4.10}$$

Ovvero

$$p_i \geq 2^{-\ell_i}. \tag{4.11}$$

Sommando su i ambo i membri abbiamo

$$\sum_{i=1}^{m} p_i \geq \sum_{i=1}^{m} 2^{-\ell_i}. \tag{4.12}$$

Ovvero la disuguaglianza di Kraft

$$1 \geq \sum_{i=1}^{m} 2^{-\ell_i}. \tag{4.13}$$

polmuni.20

Pertanto le lunghezze così scelte si prestano alla progettazione di un codice univocamente decodificabile o a prefisso. Moltiplicando ambo i membri di equazione (4.9) per p_i e sommando, abbiamo

$$\mathcal{H}(\Pi) \leq \overline{L} < \mathcal{H}(\Pi) + 1. \tag{4.14}$$

E' importante puntualizzare comunque che nel trovare il risultato di equazione (4.14), si è fatto riferimento ad uno schema specifico di scelta delle lunghezze contrariamente al risultato del teorema 4.1 che vale per *tutti* i codici univocamente decodificabili. Quindi il limite superiore alla lunghezza media è garantito solo se usiamo lo schema di codifica di equazione (4.9).

Prima di andare oltre vediamo qualche esempio.

Esempio 4.1 Consideriamo una sorgente a 4 simboli: $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$ con probabilità $\Pi = \{\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\}$. La scelta delle lunghezze delle parole codice è ovviamente $\ell_i = \log_2 4 = 2$, $i = 1, 2, 3, 4$. Si può proporre pertanto il codice a prefisso corrispondente all'albero completo di lunghezza due: $\mathcal{B} = \{00, 01, 10, 11\}$. Il codice attinge il limite inferiore della lunghezza media: $\mathcal{H}(\Pi) = \overline{L} = 2$ bit. Si noti che la associazione tra i simboli e le stringhe può essere arbitraria in quanto essa non influisce sull'efficienza.

Esempio 4.2 Consideriamo ora una sorgente a 5 simboli $\mathcal{A} = \{a_1, a_2, a_3, a_4, a_5\}$ con probabilità $\Pi = \{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\}$. Poichè $\log_2 5 = 2.32$ non è un intero, la scelta delle lunghezze secondo il criterio di equazione (4.9) è: $\ell_i = 3$, $i = 1, 2, 3, 4, 5$. Un codice possibile è: $\mathcal{B}_1 = \{000, 001, 010, 100, 110\}$. Tale codice è a prefisso e ha lunghezza media $\overline{L}_1 = 3$ bit, mentre l'entropia è $\mathcal{H}(\Pi) = 2.32$ bit. E' abbastanza semplice intuire che ci sono codici a prefisso migliori. In particolare, poichè non tutte le 8 configurazioni di tre bit sono state usate, è possibile ottenere una lunghezza media più piccola potando opportunamente l'albero. Figure 4.1(a) e (b) mostrano i due alberi di codice corrispondenti. Il secondo codice ha un alfabeto di codice: $\mathcal{B}_2 = \{000, 001, 01, 10, 11\}$. La lunghezza media è ora $\overline{L}_2 = 2.4$ molto più vicina al minimo teorico. Si noti come tali lunghezze $\{3, 3, 2, 2, 2\}$ siano le minime ottenibili per la decodificabilità poichè la disuguaglianza di Kraft-McMillan è soddisfatta con l'uguaglianza: $\frac{1}{2^3} + \frac{1}{2^3} + \frac{1}{2^2} + \frac{1}{2^2} + \frac{1}{2^2} = 1$ (codice compatto).

Esempio 4.3 Consideriamo ora una sorgente a 5 simboli $\mathcal{A} = \{a_1, a_2, a_3, a_4, a_5\}$ con probabilità $\Pi = \{0.25, 0.15, 0.1, 0.1, 0.4\}$. Calcolando i valori:

$$\{\log_2 \frac{1}{0.25}, \log_2 \frac{1}{0.15}, \log_2 \frac{1}{0.1}, \log_2 \frac{1}{0.1}, \log_2 \frac{1}{0.4}\} = \{2, 2.74, 3.32, 3.32, 1.32\}, \tag{4.15}$$

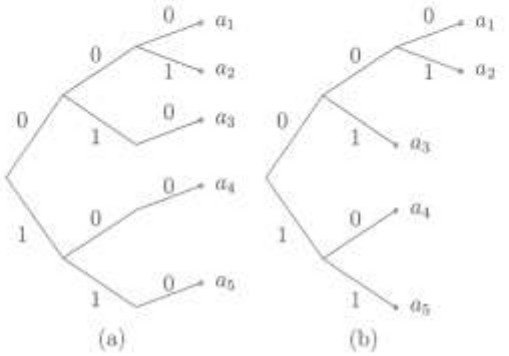


Figura 4.1: (a) L'albero di codice corrispondente all'esempio 4.2 con lunghezza media $\overline{L}_1 = 3$; (b) lo stesso albero potato con $\overline{L}_2 = 2.4$.

scegliamo le lunghezze come l'intero superiore più vicino a $\log_2 \frac{1}{p_i}$: $\{2, 3, 4, 4, 2\}$. Si noti come i simboli più probabili siano associati alle lunghezze minori. L'entropia della sorgente è $\mathcal{H}(\Pi) = 2.1$ bit. Figura 4.2(a) mostra un albero di codice costruito con le lunghezze trovate. L'alfabeto di codice corrispondente è $\mathcal{B}_1 = \{10, 010, 0000, 0001, 11\}$. La lunghezza media è $\overline{L}_1 = 2.55$ bit. Da notare come l'albero trovato non sia compatto. Pertanto accorciando e potando qualche ramo è possibile arrivare ad un codice migliore con $\mathcal{B}_2 = \{10, 01, 000, 001, 11\}$ e lunghezza media $\overline{L}_2 = 2.2$ bit.

Gli esempi riportati ci o hanno mostrato come sia possibile utilizzare il criterio di equazione (4.9), ma anche come questo possa portare a codici non necessariamente efficienti. I codici costruiti con il criterio di equazione (4.9) sono detti *codici di Shannon*. In effetti è bastato costruire l'albero con pò di attenzione per trovare dei codici più corti. La domanda che si pone ora è duplice: a) è possibile trovare degli algoritmi migliori per costruire dei codici che abbiano delle lunghezze medie più vicine al limite dell'entropia? b) è possibile dimostrare in generale che è sempre possibile avvicinarsi all'entropia quando l'insieme delle probabilità è arbitrario? La risposta è affermativa per entrambi i quesiti. Rimandiamo la risposta al quesito (a) alla prossima sezione, mentre risponderemo adesso al quesito (b).

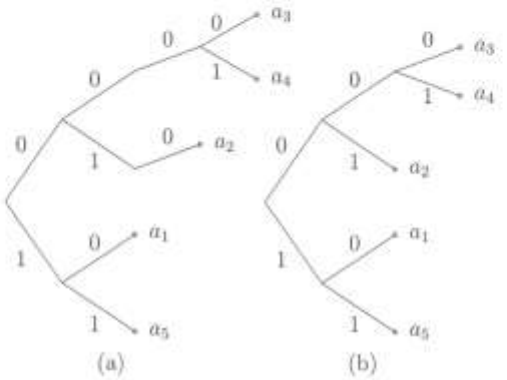


Figura 4.2: (a) L'albero di codice corrispondente all'esempio 4.3 con lunghezza media $\overline{L}_1 = 2.55$ bit; (b) lo stesso albero potato con $\overline{L}_2 = 2.2$ bit.

Assicurarsi che sia possibile sempre trovare un codice che si avvicina al limite teorico dell'entropia è una questione che ha una valenza sia teorica che pratica. Si tratta del problema che Shannon si pose quando studiò le sorgenti di informazione e che culminò nel teorema che andremo a presentare e che è appunto noto come *primo teorema di Shannon*. Nella letteratura il teorema è presentato a vari livelli di dettaglio matematico e per varie tipologie di sorgenti. In questo note ci limiteremo a presentare il risultato con riferimento a sorgenti discrete senza memoria con un livello di rigore matematico essenziale alla comprensione del risultato.

Teorema 4.2 *(Primo Teorema di Shannon) Data una sorgente discreta senza memoria con distribuzione dei simboli* $\Pi = \{p_1, ..., p_n\}$ *e entropia* $\mathcal{H}(\Pi)$. *Sia* $\Pi^N = \{p_1^N, p_2^N, ..., p_{n^N}^N\}$ *la distribuzione di probabilità della sorgente estesa di ordine* N. *Siano inoltre* $\mathcal{L}^N = \{\ell_1^N, \ell_2^N, ..., l_{n^N}^N\}$ *le lunghezze di un codice univocamente decodificabile per la sorgente estesa scelte con il criterio*

$$\log_2 \frac{1}{p_i^N} \leq \ell_i^N < \log_2 \frac{1}{p_i^N} + 1, \quad i = 1, ..., n^N. \tag{4.16}$$

Denotando con $\overline{L}_N$ *la lunghezza media del codice, il numero medio di bit*

Palmieri 23

Estensione del I Teorema di Shannon al caso di sorgenti dipendenti

~~[illegible]~~

Sia $\Pi^N = \{p_1^N, p_2^N, \dots, p_{n^N}^N\}$ la distr. di probabilità della sorgente composta da N simboli alla volta (non deve essere necessariamente il prodotto di Kronecker di quella di ordine inferiore)

Applicando la semplice regola del Teorema di Shannon-Fano-Elias

$$\log_2 \frac{1}{p_i^N} < l_i^N \leq \log_2 \frac{1}{p_i^N} + 1$$

METODO DI APPLICAZIONE

Quindi

$$\sum_{i=1}^{n^N} p_i^N \log_2 \frac{1}{p_i^N} \leq \sum_{i=1}^{n^N} p_i^N l_i^N \leq \sum_{i=1}^{n^N} p_i^N \log_2 \frac{1}{p_i^N} + 1$$

$$\frac{H(S[1], S[2], \dots, S[N])}{N} \leq \frac{\overline{L}_N}{N} \leq \frac{H(S[1], \dots S[N])}{N} + \frac{1}{N}$$

$\lim_{N \to \infty}$ (tanto cresce) $H(\mathcal{L})$ entropia estesa o $H_\infty(S)$

Quindi $\lim_{N \to \infty} \frac{\overline{L}_N}{N} = H_\infty(S)$

associato ad ogni simbolo di sorgente è $\frac{\overline{L}_N}{N}$ *ed è tale che*

$$\lim_{N\to\infty} \frac{\overline{L}_N}{N} = \mathcal{H}(\Pi). \tag{4.17}$$

Il primo teorema di Shannon, noto anche come *teorema della codifica senza rumore*, consiste nell'aver sancito che esiste una strategia di codifica asintoticamente efficiente ($\eta \to 1$). Il prezzo che si paga per avvicinarsi all'entropia sarà la complessità crescente necessaria alla costruzione di codici su sorgenti estese a cardinalità sempre crescente.

Prova: La dimostrazione del primo teorema di Shannon si ottiene facilmente mediante l'utilizzo delle proprietà delle sorgenti estese. Si ricordi come una sorgente estesa $\mathcal{S}^N$ di ordine N sia una sorgente formata raggruppando N simboli alla volta da una sorgente $\mathcal{S}$. Il nuovo alfabeto sorgente avrà pertanto n^N simboli e la lunghezza media del codice sarà

$$\overline{L}_N = \sum_{j=1}^{n^N} p_j^N \ell_j^N. \tag{4.18}$$

Se tali lunghezze sono scelte con lo stesso criterio di equazione (4.9) (anche se come abbiamo visto tale criterio potrebbe non essere il migliore) avremo

$$\mathcal{H}(\Pi^N) \leq \overline{L}_N < \mathcal{H}(\Pi^N) + 1. \tag{4.19}$$

Poiché la quantità $\frac{\overline{L}_N}{N}$ è il numero di bit mediamente associato ad un simbolo della sorgente originaria $\mathcal{S}$, e poichè sappiamo che l'entropia della sorgente estesa è $\mathcal{H}(\Pi^N) = N\mathcal{H}(\Pi)$, abbiamo che

$$\mathcal{H}(\Pi) \leq \frac{\overline{L}_N}{N} < \mathcal{H}(\Pi) + \frac{1}{N}. \tag{4.20}$$

Pertanto codificando la sorgente estesa abbiamo che asintoticamente

$$\lim_{N\to\infty} \frac{\overline{L}_N}{N} = \mathcal{H}(\Pi). \quad \triangle \tag{4.21}$$

Esempio 4.4 Consideriamo la sorgente binaria con alfabeto sorgente $\mathcal{A} = \{a_1, a_2\}$ e distribuzione di probabilità $\Pi = \{0.2, 0.8\}$. L'entropia della sorgente è $\mathcal{H}(\Pi) = 0.72$ bit. Applicando il criterio di Shannon per determinare le lunghezze della parole codice abbiamo:

$$\{\log_2 \frac{1}{0.2}, \log_2 \frac{1}{0.8}\} = \{2.32, 0.32\}, \tag{4.22}$$

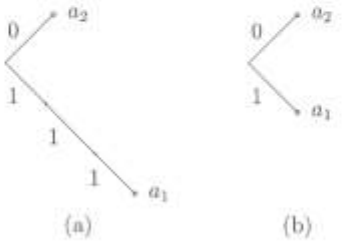


Figura 4.3: (a) L'albero di codice corrispondente alla sorgente dell'esempio 4.4 con $N = 1$ e lunghezza media $\overline{L}_1 = 1.4$ bit; (b) lo stesso albero accorciato con $\overline{L}_1 = 1$ bit.

ovvero parole codice di lunghezze: $\{3, 1\}$. La lunghezza media di un tale codice è $\overline{L}_1 = 1.4$ bit. Un tale codice, di cui un esempio è riportato in figura 4.3(a) con alfabeto di codice $\{111, 0\}$, è anche peggiore del codice binario più banale con alfabeto di codice $\{0, 1\}$ avente lunghezza media $\overline{L}_1 = 1$ riportato in Figura 4.3(b).

Incrementiamo ora l'ordine della sorgente considerando la sorgente estesa con $N = 2$ e assumendo che i simboli di sorgente siano emessi in maniera indipendente (sorgente senza memoria). Abbiamo pertanto: $\mathcal{A}^2 = \{a_1^2, a_2^2, a_3^2, a_4^2\} = \{a_1a_1, a_1a_2, a_2a_1, a_2a_2\}$. Le probabilità della nuova sorgente sono

$$\Pi^2 = \{0.04, 0.16, 0.16, 0.64\}. \tag{4.23}$$

Applicando il criterio di Shannon abbiamo

$$\{\log_2 \frac{1}{0.04}, \log_2 \frac{1}{0.16}, \log_2 \frac{1}{0.16}, \log_2 \frac{1}{0.64}\} = \{4.64, 2.64, 2.64, 0.64\}, \tag{4.24}$$

e pertanto lughezze $\{5, 3, 3, 1\}$. La lunghezza media di un tale codice sarà $\overline{L}_2 = 1.8$ bit. La lunghezza media impegnata per ogni simbolo della sorgente originaria è pertanto $\overline{L}_2/2 = 0.9$ bit, che è ovviamente tra l'entropia 0.72 e l'entropia più $\frac{1}{2}$, ovvero 1.22 bit: ci siamo ulteriormente avvicinati all'entropia. Un codice con queste lunghezze è mostrato in figura 4.4(a). Le parole codice corrispondenti sono: $\{11111, 100, 101, 0\}$. Una versione migliore del codice è mostrata in figura 4.4(b). Le lunghezze sono: $\{2, 3, 3, 1\}$ e la lunghezza media corrispondente $\overline{L}_2 = 1.68$ bit, ovvero $\overline{L}_2/2 = 0.84$ bit per simbolo di sorgente.

Facciamo ora un ulteriore passo avanti studiando la sorgente estesa di ordine

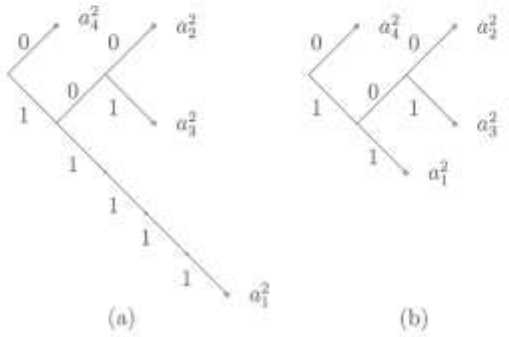


Figura 4.4: (a) L'albero di codice corrispondente alla sorgente estesa con $N = 2$ dell'esempio 4.4 con lunghezza media $\overline{L}_2 = 1.8$ bit; (b) lo stesso albero accorciato con $\overline{L}_2 = 1.68$ bit.

$N = 3$. L'alfabeto di sorgente è

$$\begin{aligned} \mathcal{A}^3 &= \{a_1^3, a_2^3, a_3^3, a_4^3, a_5^3, a_6^3, a_7^3, a_8^3\} \\ &= \{a_1a_1a_1, a_1a_1a_2, a_1a_2a_1, a_1a_2a_2, a_2a_1a_1, a_2a_1a_2, a_2a_2a_1, a_2a_2a_2\} \end{aligned} \tag{4.25}$$

Le probabilità relative sono

$$\Pi^3 = \{0.008, 0.032, 0.032, 0.128, 0.032, 0.128, 0.128, 0.512\}. \tag{4.26}$$

Le lunghezze secondo il criterio di Shannon saranno

$$\{7, 5, 5, 3, 5, 3, 3, 1\}. \tag{4.27}$$

Un codice con tali lunghezze ha lunghezza media: $\overline{L}_3 = 2.2$ bit. La lunghezza media per simbolo di sorgente è: $\frac{\overline{L}_3}{3} = 0.733$ bit. Figura 4.5(a) mostra un codice con queste lunghezze, mentre figura 4.5(b) ne mostra una versione accorciata con $\overline{L}_3 = 2.184$, ovvero $\frac{\overline{L}_3}{3} = 0.728$.

Si noti come si sia oramai molto vicini all'entropia. Il procedimento può essere iterato per avvicinarsi ulteriormente.

Il risultato del primo teorema di Shannon è di importanza fondamentale, ma purtroppo è basato su un algoritmo di costruzione del codice che

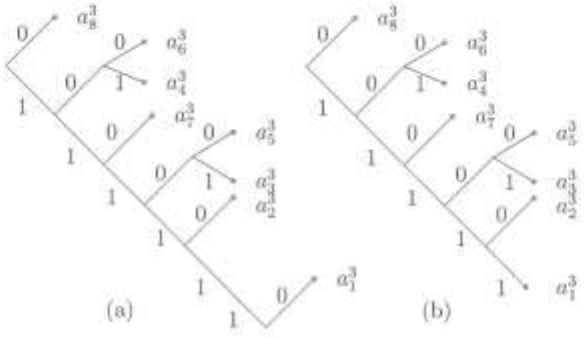


Figura 4.5: (a) L'albero di codice corrispondente alla sorgente estesa con $N = 3$ dell'esempio 4.4 con lunghezza media $\overline{L}_3 = 2.2$ bit; (b) lo stesso albero accorciato con $\overline{L}_3 = 2.184$ bit.

potrebbe non essere molto efficiente, come abbiamo già visto in alcuni esempi. In effetti il teorema è un risultato asintotico che non risponde direttamente alle tipiche situazioni di costruzione di codici di ordine limitato. Inoltre la inefficienza dell'algoritmo condiderato potrebbe essere ancora più marcata quanto più è grande l'ordine N della sorgente estesa. Il problema è pertanto: fermo restando che l'algoritmo del teorema garantisce l'efficienza asintotica, è possibile per un *fissato ordine* N trovare degli algoritmi che portino a codici più efficienti?

Nella letteratura della teoria dell'informazione sono state proposte varie strategie per la costruzione di un codice più efficiente di quello di Shannon[1]. Ci limiteremo in queste note a discutere l'algoritmo di Huffman, che costituisce la strategia ottima di costruzione di un codice a lunghezza variabile.

4.0.2 L'algoritmo di Huffman

Consideriamo una sorgente $\mathcal{S}(m)$ con alfabeto di sorgente

$$\mathcal{A}(m) = \{a_1(m), a_2(m), ..., a_m(m)\}, \tag{4.28}$$

[1]Vedi T. M. Cover, J. A. Thomas, *Elements of Information Theory*, J. Wiley, 1993 per una discussione aggiornata su vari algoritmi per la codifica di sorgente.

e distribuzione di probabilità $\Pi(m) = \{p_1(m), p_2(m), ..., p_m(m)\}$, dove abbiamo indicato esplicitamente l'ordine m della sorgente. Senza perdita di generalità, supponiamo che i simboli siano elencati in ordine di probabilità decrescente, ovvero $p_1(m) \geq p_2(m) \geq ... \geq p_m(m)$. Il primo passo dell'algoritmo consiste nell'accoppiare gli ultimi due simboli $a_{m-1}(m)$ e $a_m(m)$ in un unico simbolo, ottendo una nuova sorgente $\mathcal{S}(m-1)$ a $m-1$ simboli: $\mathcal{A}(m-1) = \{a_1(m-1), a_2(m-1), ..., a_m(m-1)\}$ e distribuzione di probabilità $\Pi(m-1) = \{p_1(m-1), p_2(m-1), ..., p_{m-1}(m-1)\}$, dove si è supposto di avere ordinato i simboli in ordine di probabilità decrescente. A questo punto il primo passo dell'algoritmo viene ripetuto e gli ultimi due simboli $a_{m-1}(m-1)$ e $a_{m-2}(m-1)$ in ordine di probabilità vengono accoppiati. La procedura si itera e si esaurisce in $m-1$ passi dopo aver costruito l'albero di codice binario andando dai rami verso la radice. Un esempio chiarirà la semplice procedura.

Esempio 4.5 Si consideri la sorgente a sei simboli $\mathcal{A} = \{a_1, a_2, a_3, a_4, a_5, a_6\}$, e probabilità $\Pi = \{0.4, 0.3, 0.11, 0.09, 0.06, 0.04\}$. L'entropia della sorgente è $\mathcal{H}(\Pi) = 2.14$ bit. Figura 4.6 mostra la sequenza delle sorgenti ridotte e il codice di Huffman risultante. La costruzione dell'albero binario si effettua dai rami verso la radice associando sempre le coppie di simboli meno probabili. Le stringhe di bit corrispondenti alle varie parole codice si ottengono leggendo l'albero dai rami verso la radice e riportando i bit in ordine inverso. La lunghezza delle parole codice che ne deriva è: $\{1, 2, 3, 4, 5, 5\}$. La lunghezza media è $\overline{L} = 2.19$ bit. A scopo di confronto valutiamo il risultato di un codice costruito secondo il criterio di Shannon. Poichè

$$\begin{aligned} &\{\log_2 \frac{1}{0.4}, \log_2 \frac{1}{0.3}, \log_2 \frac{1}{0.11}, \log_2 \frac{1}{0.09}, \log_2 \frac{1}{0.06}, \log_2 \frac{1}{0.04}\} \\ &= \{1.32, 1.74, 3.18, 3.47, 4.06, 4.64\}, \end{aligned} \tag{4.29}$$

le lunghezze scelte sono $\{2, 2, 4, 4, 5, 5\}$. La lughezza media del codice è $\overline{L} = 2.7$. Figura 4.7(a) mostra l'albero di un codice costruito con queste lunghezze. Figura 4.7(b) corrisponde ad una versione accorciata compatta con lunghezze: $\{2, 2, 3, 3, 4, 4\}$ e lunghezza media $\overline{L} = 2.4$. Si noti come il codice ottenuto dall'algoritmo di Huffman sia comunque il migliore.

Anche se tralasceremo la prova formale per brevità, va puntualizzato che l'algoritmo di Huffman garantisce che il codice è compatto e che la lunghezza media è minima rispetto a tutti i possibili codici univocamente decodificabili con quella distribuzione di probabilità. Si noti che nell'algoritmo di Huffman è possibile cambiare le etichettature dei rami dell'albero di codice ottenendo

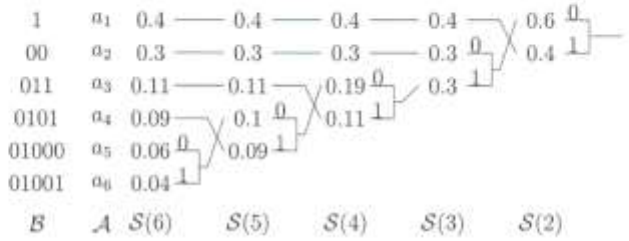


Figura 4.6: La costruzione del codice secondo l'algoritmo di Huffman

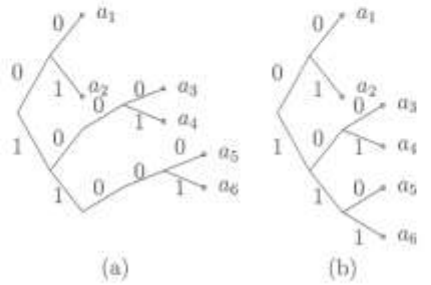


Figura 4.7: (a) Un codice progettato secondo le lunghezze dettate dal criterio di Shannon; (b) lo stesso codice ridotto ad una forma più compatta

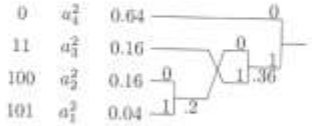


Figura 4.8: L'algoritmo di Huffman applicato alla sorgente estesa per $N = 2$ dell'esempio 4.4

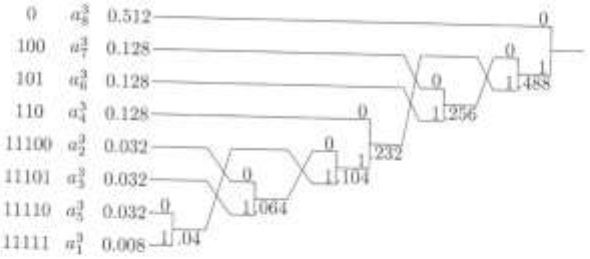


Figura 4.9: L'algoritmo di Huffman applicato alla sorgente estesa per $N = 3$ dell'esempio 4.4.

cosici diversi, ma equivalenti. Inoltre, in alcuni casi in cui le probabilità di tre o più simboli delle sorgenti ridotte hanno probabilità uguali, si può scegliere arbitrariamente come effettuare gli accoppiamenti ottenendo codici diversi, ma equivalenti dal punto di vista della lunghezza media. Qualche approfondimento su questo punto è suggerito negli esercizi.

Esempio 4.4 (cont.) Riprendiamo l'esempio 4.4 della sorgente binaria e delle sorgenti estese con $N = 2$ e $N = 3$ e applichiamo l'algoritmo di Huffman a ognuna di esse. Per $N = 1$, il codice è banalmente $\{0, 1\}$ con lunghezza media pari a 1. Per $N = 2$ Figura 4.8 mostra la costruzione del codice che fornisce: $\frac{\bar{L}_2}{2} = \frac{1.56}{2} = 0.78$ bit. Per $N = 3$ Figura 4.9 mostra la costruzione del codice con $\frac{\bar{L}_3}{3} = \frac{2.184}{3} = 0.728$ bit.

E' interessante andate a confrontare i vari codici ottenuti per questo esempio. Tabella 4.1 mostra in maniera compatta le lunghezze medie dei vari codici

ottenuti. Si noti come il limite superiore sia abbastanza lasco rispetto ai valori effettivi e come l'algoritmo di Huffman dia sempre il miglior risultato. Si noti inoltre la parità per $N = 3$ tra il risultato dell'algoritmo di Huffman e quella del codice derivato dall'accorciamento del codice di Shannon. La ragione di ciò è da attribuire alla piccola cardinalità del codice e al fatto che praticamente per $N = 3$ il limite dell'entropia è stato attinto.

	$N=1$	$N=2$	$N=3$
$\mathcal{H}(\Pi)+1/N$	1.72	1.22	1.05
Shannon	1.4	0.9	0.733
Shannon acc.	1	0.84	0.728
Huffman	1	0.78	0.728
$\mathcal{H}(\Pi)$	0.72	0.72	0.72

Tabella 4.1: Confronto della lunghezza media dei codici (in bit) e dei limiti del teorema di Shannon per l'esempio 4.4.

Per apprezzare ulteriormente l'efficienza di tali codici in un esempio reale, calcoliamo esplicitamente i risultati della codifica di un segmento di una realizzazione della sorgente. Ricordiamo che la sorgente emette i simboli in maniera indipendente e che a_1 è mediamente presente nel 20% dei casi e a_2 nell'80%. Un segmento di 30 simboli di sorgente è mostrato nella prima riga

$a_2a_1a_2a_2a_1a_2a_2a_2a_2a_2a_1a_2a_2a_2a_1a_2a_2a_2a_1a_2a_2a_2a_1a_2a_1a_2a_2a_2a_2a_2$

101101111101110111011101011111

110100001000100010010010000

101101010110001101011100

Le altre tre righe rappresentano il risultato della codifica secondo il codice di Huffman per $N = 1, 2, 3$ rispettivamente. Si noti che nel codice esteso con $N = 2$ il numero di bit totale è 28 contro i 30 iniziali. Per $N = 3$ il numero di bit totale è 24 riflettendo approssimativamente il risparmio del 28% predetto dall'entropia e dall'efficienza del terzo codice molto prossima a uno.

4.0.3 L'algoritmo di Huffman per codici M-ari

La stessa procedura discussa per la costruzione di codici binari può essere usata per la costruzione di codici M-ari. Nell'algoritmo basta raggruppare ad ogni fase dell'algoritmo le M parole di codice meno probabili e iterare. C'è però da considerare un problema che emerge nella costruzione

dell'albero: alla fine dell'algoritmo, ovvero nella ultima sorgente ridotta, ci si potrebbe trovare con un numero di simboli inferiore a M. Pertanto si sarebbe costretti a costruire un diramazione con meno di M rami con possibile perdita di efficienza del codice. Il rimedio al problema è semplice e consiste nell'aggiungere alla sorgente originaria un numero di simboli fittizi sufficiente a garantire che nello stadio finale avanzino proprio M nodi. A tali simboli si attribuisce probabilità nulla. Poiché un l'albero M-ario richiede un numero di parole codice pari a $M + k(M-1)$, dove $k-1$ è la lunghezza massima delle parole codice, è immediato capire quanti simboli ausiliari devono essere introdotti. Un esempio chiarirà il semplice algoritmo.

Esempio 4.6 Si consideri una sorgente a 10 simboli

$$\mathcal{A} = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}\}. \tag{4.30}$$

con probabilità

$$\Pi = \{0.22, 0.15, 0.12, 0.1, 0.1, 0.08, 0.07, 0.06, 0.05, 0.05\}. \tag{4.31}$$

Si progetti un codice ternario ($M = 3$)con l'algoritmo di Huffman.
Soluzione: Il primo valore della cardinalità che può essere ottenuto con l'equazione: $3 + k \cdot 2$, è 11. E' pertanto necessario introdurre un simbolo ausiliario (a_{11}) a probabilità nulla. La figura 4.10 mostra la costruzione del codice. La lunghezza media del codice risultante è $\overline{L} = 2.09$ trit, mentre l'entropia della sorgente è: $\mathcal{H}(\Pi) = 3.16$ bit. L'efficienza del codice diventa

$$\eta = \frac{\mathcal{H}(\Pi)}{\overline{L}\log_2 3} = 0.9547, \tag{4.32}$$

dove abbiamo convertito la lunghezza media da trit in bit: $\overline{L}\log_2 3 = 3.31$ bit.

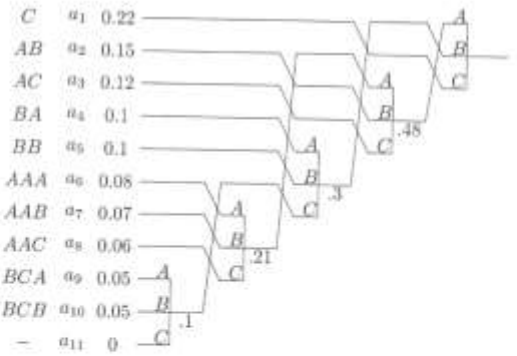


Figura 4.10: L'algoritmo di Huffman usato per la costruzione del codice ternario dell'esempio 4.6.