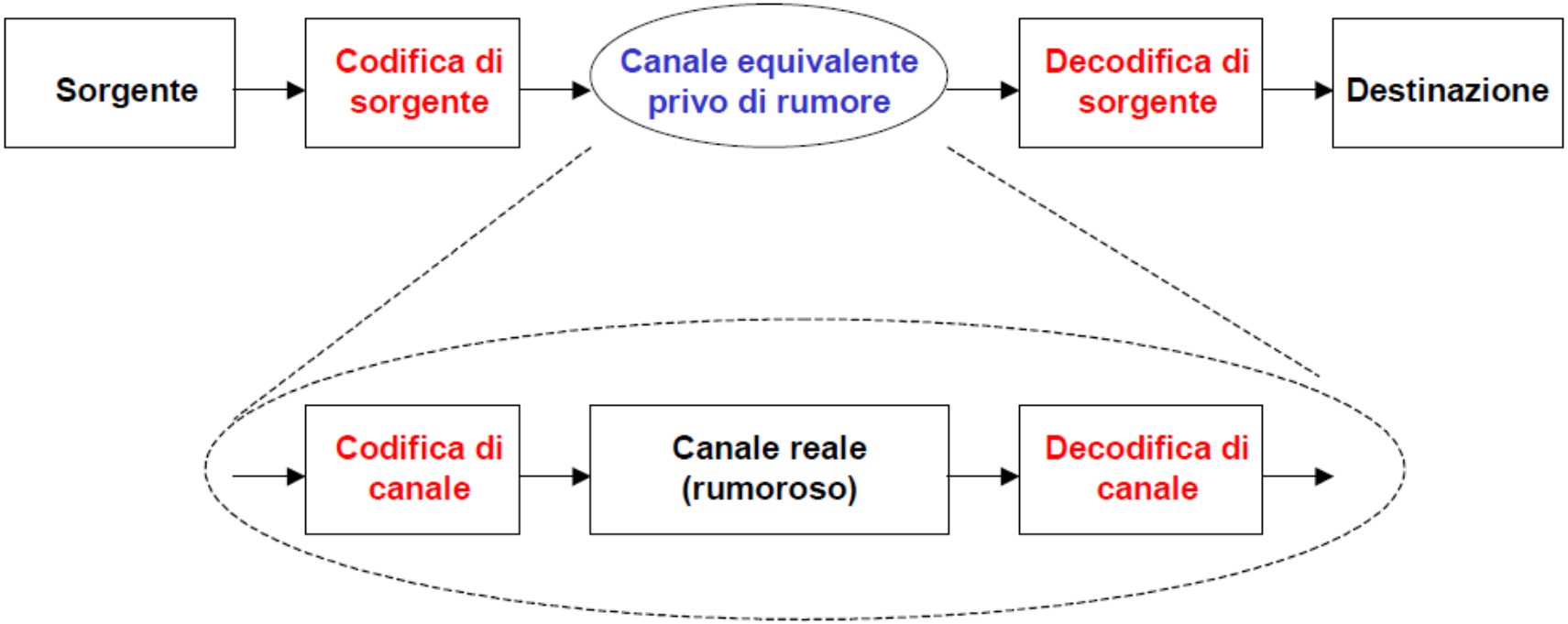
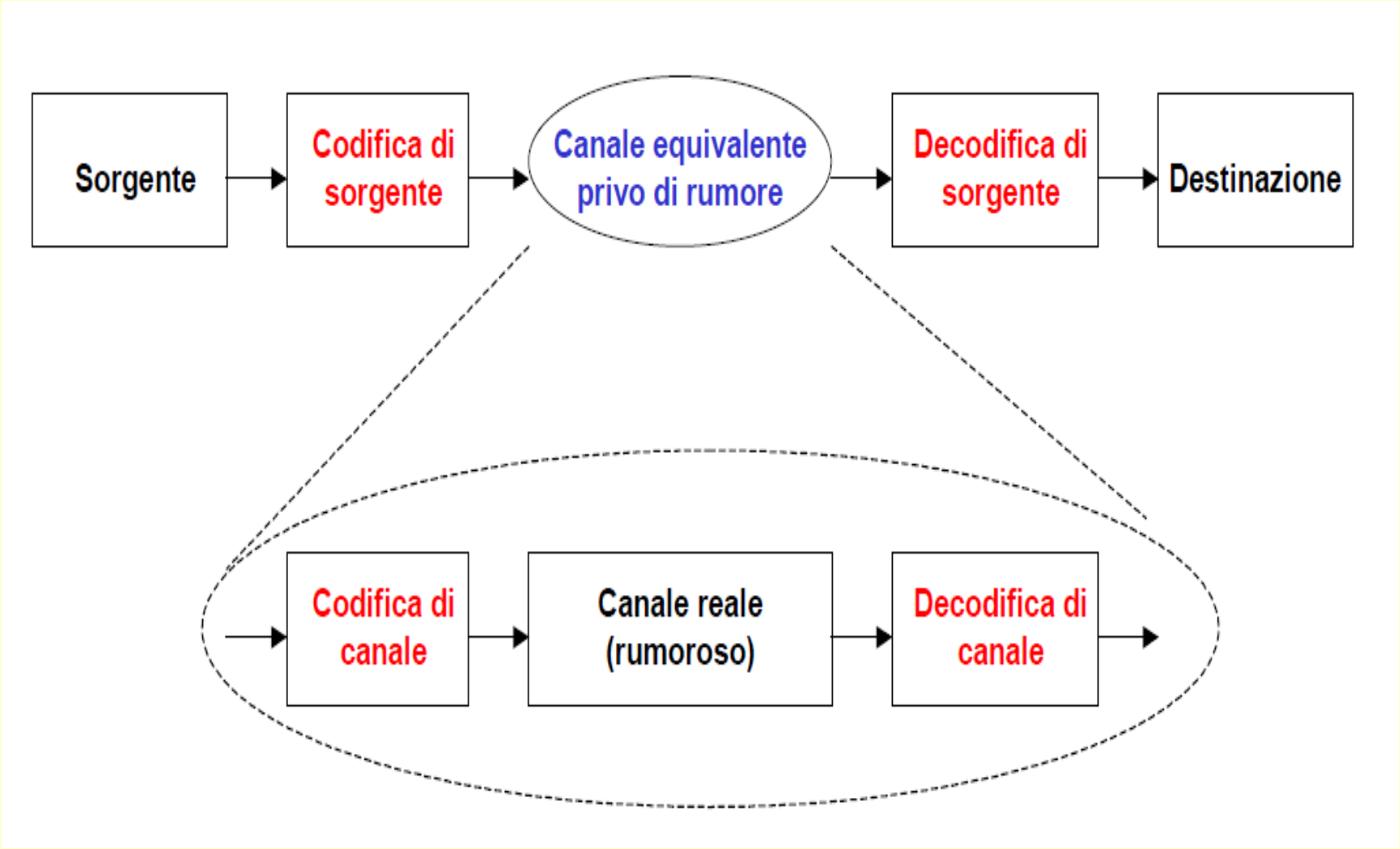
La codifica di sorgente

La **codifica di sorgente** è la rappresentazione efficiente dei dati generati da una sorgente discreta al fine poi di trasmetterli su di un opportuno canale privo di rumore.



La **codifica di canale** permette di trasmettere l'informazione emessa dalla sorgente (opportunamente trattata mediante la codifica di sorgente) **in maniera affidabile** su un canale reale caratterizzato da limitazioni fisiche (es. rumore).


Sorgente
Codifica di sorgente
Canale equivalente privo di rumore
Decodifica di sorgente
Destinazione
Codifica di canale
Canale reale (rumoroso)
Decodifica di canale

Il codice adottato da una sorgente deve essere tale da

garantire

che il ricevitore distingua senza ambiguità

i messaggi inviati dal trasmettitore

(a meno di eventuali errori introdotti dal canale di comunicazione)

Quindi la codifica di sorgente deve soddisfare le seguenti condizioni:

- **Ogni simbolo** deve essere **riconoscibile** (codici diversi)

  Es. A=0 B=1 C=10 D=1
  non riconoscibile

- **Ogni simbolo** deve essere **identificabile** qualsiasi posizione occupi nella sequenza trasmessa

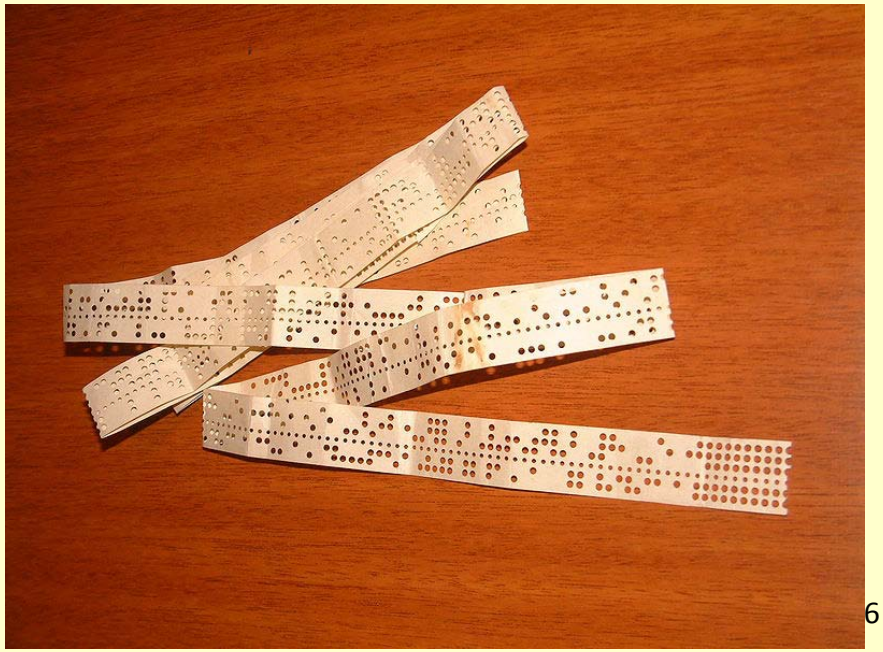
  Es. A=0 B=1 C=10 D=11
  non identificabile
  1 0 1 0 1 1 ? C C D
  ? B A C B B
  ? …….

Se il codice è a **lunghezza fissa** le due condizioni sono facilmente realizzabili, perché è sufficiente che ogni simbolo codificato si differenzi dall'altro almeno per un bit.

Per le sorgenti discrete, nel caso in cui si debbano trasmettere messaggi di tipo testo (alfanumerici) si usano spesso codici in cui ad ogni **carattere** o simbolo viene associata una **sequenza di cifre binarie di lunghezza fissa**.

Esempi di codici di sorgenti:

- **Codice ASCII** a 7 o 8 bit
- **Codice BCD o EBCDIC** a 4 o 8 bit (sistemi IBM)
- **Codice Baudot** a 5 bit (codifica 32 caratteri tra simboli e di controllo ed era usato nelle telescriventi)



Sia dato un file di 120 caratteri con frequenze:

carattere	a	b	c	d	e	f
frequenza	57	13	12	24	9	5

Usando un **codice a lunghezza fissa** occorrono 3 bit per rappresentare 6 caratteri.
Ad esempio:

carattere	a	b	c	d	e	f
cod. fisso	000	001	010	011	100	101

Per codificare il file occorrono 120 × 3 = **360 bit.**

Possiamo fare meglio con un **codice a lunghezza variabile** che assegni codici più corti ai caratteri più frequenti.

Ad esempio con il codice:

carattere	a	b	c	d	e	f
frequenza	57	13	12	24	9	5
cod. var.	0	101	100	111	1101	1100

Bastano 57 × 1 + 49 × 3 + 14 × 4 = **260 bit.**

Codici a lunghezza variabile

Tengono conto della **frequenza** con cui determinati simboli vengono emessi dalla sorgente.

Questi codici sono molto utilizzati nella codifica dei dati digitali, soprattutto quelli multimediali, quali immagini, audio e video (**tecniche di compressione**), che cerca appunto di **ridurre il numero di bit necessari** per immagazzinare un'informazione.

Le **tecniche di compressione** vengono applicate da appositi programmi immediatamente **prima della memorizzazione** o **trasmissione dei dati**.

Ovviamente in fase di lettura o ricezione analoghi programmi devono effettuare la **decompressione.**

Se i simboli da trasmettere non sono equiprobabili

utilizzando i **codici a lunghezza variabile** si ottengono

dei codici molto più efficienti

rispetto a quelli a lunghezza fissa

perché

consentono di ottenere

un minor numero medio di bit

per trasmettere un certo messaggio.

Tecniche di compressione

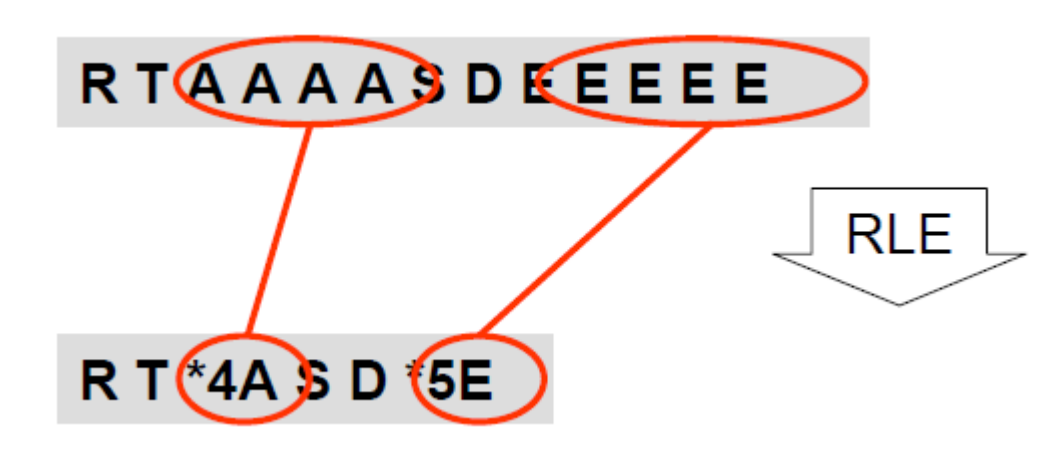
- **Algoritmi di compressione lossless o senza perdita (non distruttivi)**
  Comprimono i dati attraverso un processo *senza perdita d'informazione* e si basano sulla codifica in forma compatta di sequenze di dati uguali, oppure sulla codifica con un numero ridotto dei bit di dati statisticamente più frequenti.
  Dopo la decodifica il messaggio originale viene ricostruito interamente.
  Un esempio sono i formati ZIP per i file e GIF, PNG e TIFF per le immagini.

  Gli algoritmi di compressione senza perdita più importanti sono la **codifica RLE** (Run Length Encoding), la **codifica LZW** (Lempel-Ziv-Welch) e la **codifica di Huffman**.

- **Algoritmi di compressione lossy o con perdita (distruttivi)**
  Comprimono i dati attraverso un processo *con perdita d'informazione* che sfrutta le ridondanze nell'utilizzo dei dati e non possono assicurare una reversibilità assoluta. La parte meno significativa del messaggio viene soppressa in modo irreversibile. Vengono usati per comprimere file multimediali (immagini, audio, video).
  Sfruttano il fatto che l'informazione multimediale può essere sottoposta a trasformazioni pur rimanendo intelegibile per noi.
  Un esempio sono i formati JPEG per le immagini, MP3 per il suono e MPEG per le immagini in movimento. In molti casi lo spazio che si riesce a risparmiare effettuando la compressione con algoritmi lossy è molto maggiore rispetto a quello risparmiato usando algoritmi lossless.

Run Length Encoding (RLE)

La sequenza di *N campioni uguali (run) viene sostituita (encoding) dal numero di* occorrenze del valore (length) ed il valore.



L'**inizio di un blocco** codificato deve essere **identificabile**: in questo esempio si usa il carattere * per indicare che seguono due valori in codifica run-length

La codifica di Huffman

Il **codice di *Huffman*** *costruisce una tabella* di codifica-decodifica utilizzando un numero di bit differente a seconda della probabilita' che si ha di trovare uno specifico valore.

Utilizzando: **meno bit per i codici più probabili**,
e **più bit per quelli meno probabili**

si hanno dei vantaggi:

- **risparmio di spazio** nella memorizzazione
- **risparmio di tempo** nella trasmissione
- **minore occupazione di banda** del canale

Esempio di codifica di Huffman

Supponiamo che una sorgente possa emettere solo i seguenti caratteri dell'alfabeto e di conoscere anche le loro probabilità di essere inviati.

1. Si ordinano i simboli per probabilità decrescente

Simbolo	Prob.
L	0,30
A	0.25
E	0.20
F	0.10
S	0.10
M	0.05

2. finché c'è più di un elemento nella tabella ripeti:

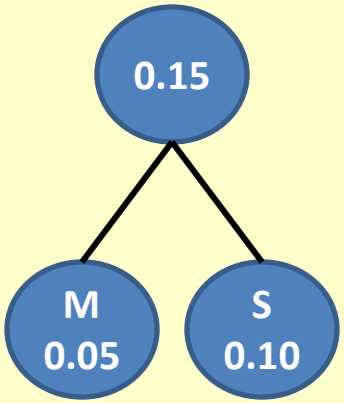
a) associa a ciascuno dei due simboli con probabilità minore un nodo foglia dell'albero e considera rimossi i due simboli dalla tabella;

b) crea un nuovo nodo interno all'albero con questi due nodi come figli, e con probabilità pari alla somma delle loro probabilità;

c) aggiungi il nuovo elemento alla tabella con la probabilità somma calcolata, rispettando l'ordinamento;

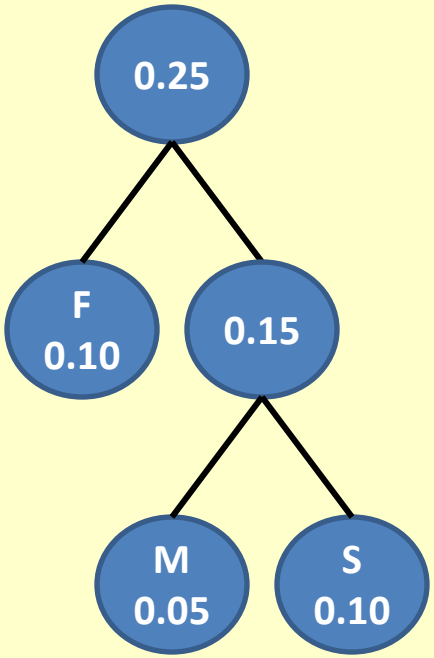
Esempio di codifica di Huffman

Simbolo	Prob.	
L	0,30	
A	0.25	
E	0.20	
		SM 0.15
F	0.10	
S	0.10	
M	0.05	



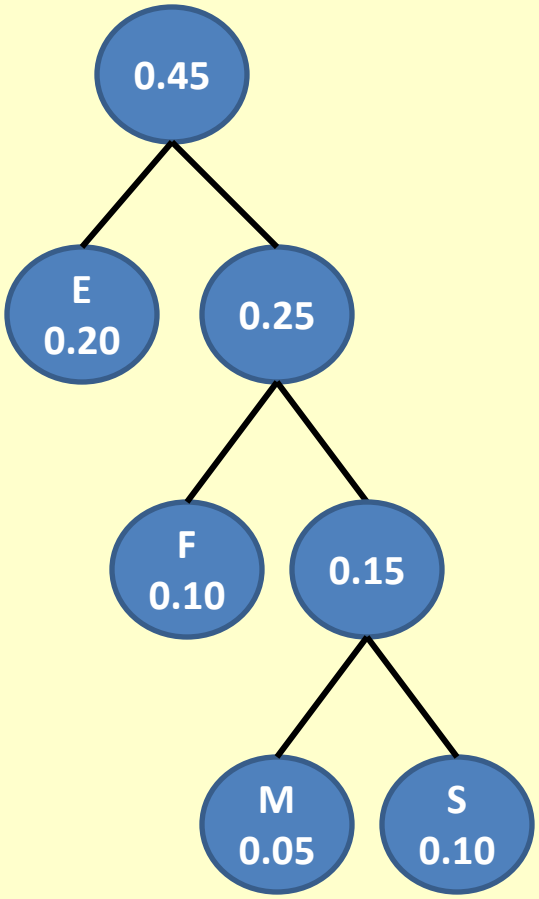
Esempio di codifica di Huffman

Simb.	Prob.		
L	0,30		
A	0.25		
			FSM 0.25
E	0.20		
		SM 0.15	
F	0.10		
S	0.10		
M	0.05		



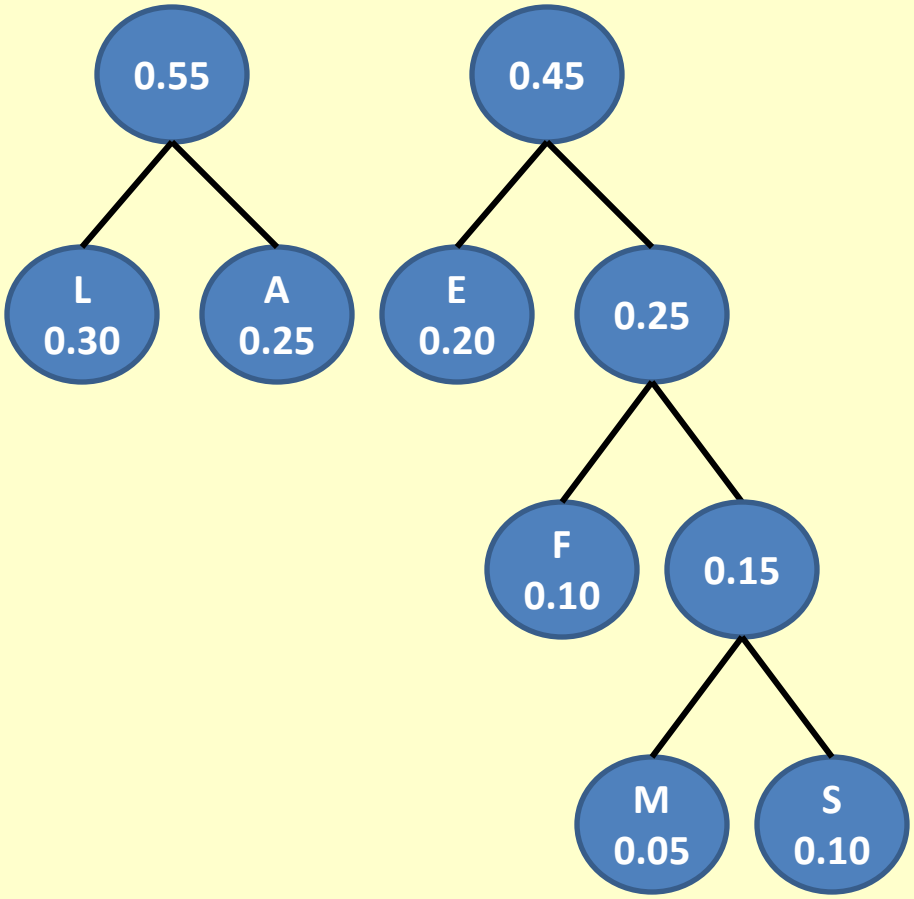
Esempio di codifica di Huffman

Simb.	Prob.			
				EFSM 0.45
L	0,30			
A	0.25			
			FSM 0.25	
E	0.20			
		SM 0.15		
F	0.10			
S	0.10			
M	0.05			



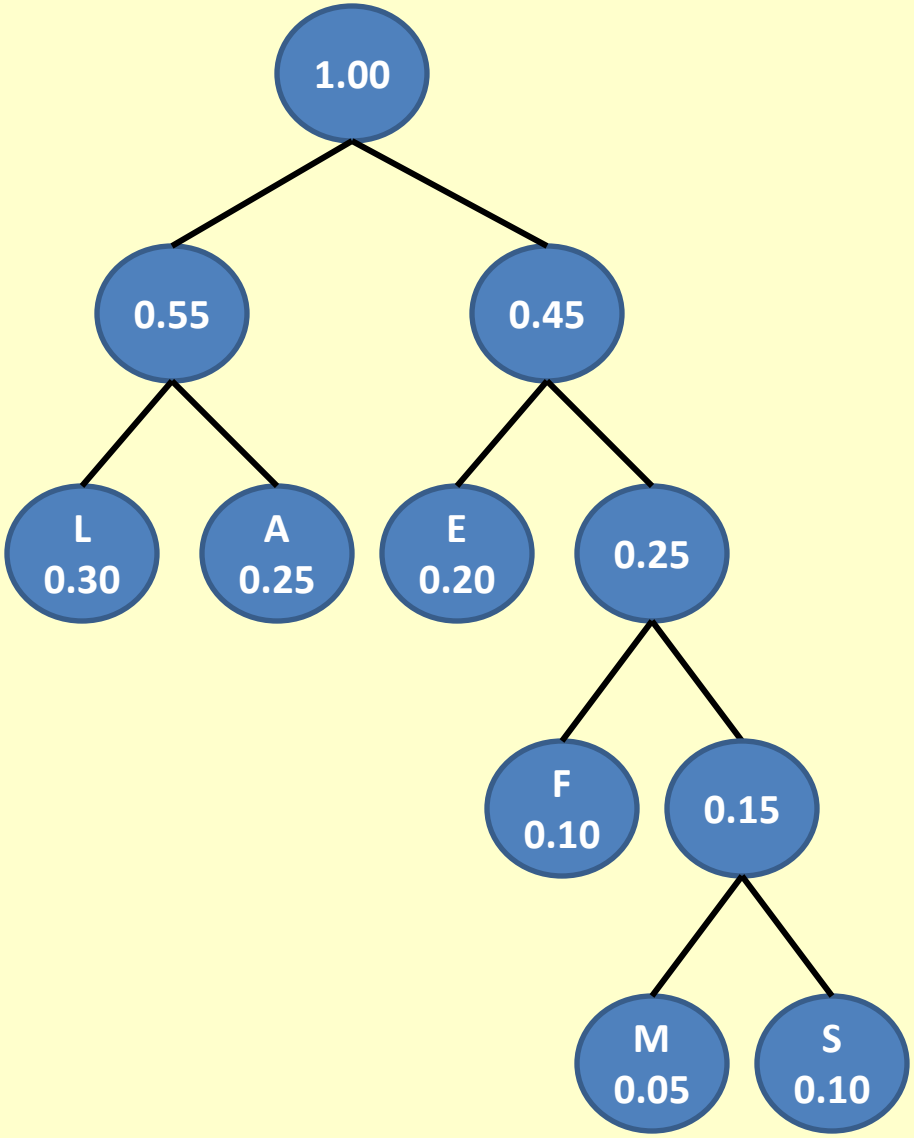
Esempio di codifica di Huffman

Simb.	Prob.			
				LA 0.55
				EFSM 0.45
L	0,30			
A	0.25			
			FSM 0.25	
E	0.20			
		SM 0.15		
F	0.10			
S	0.10			
M	0.05			



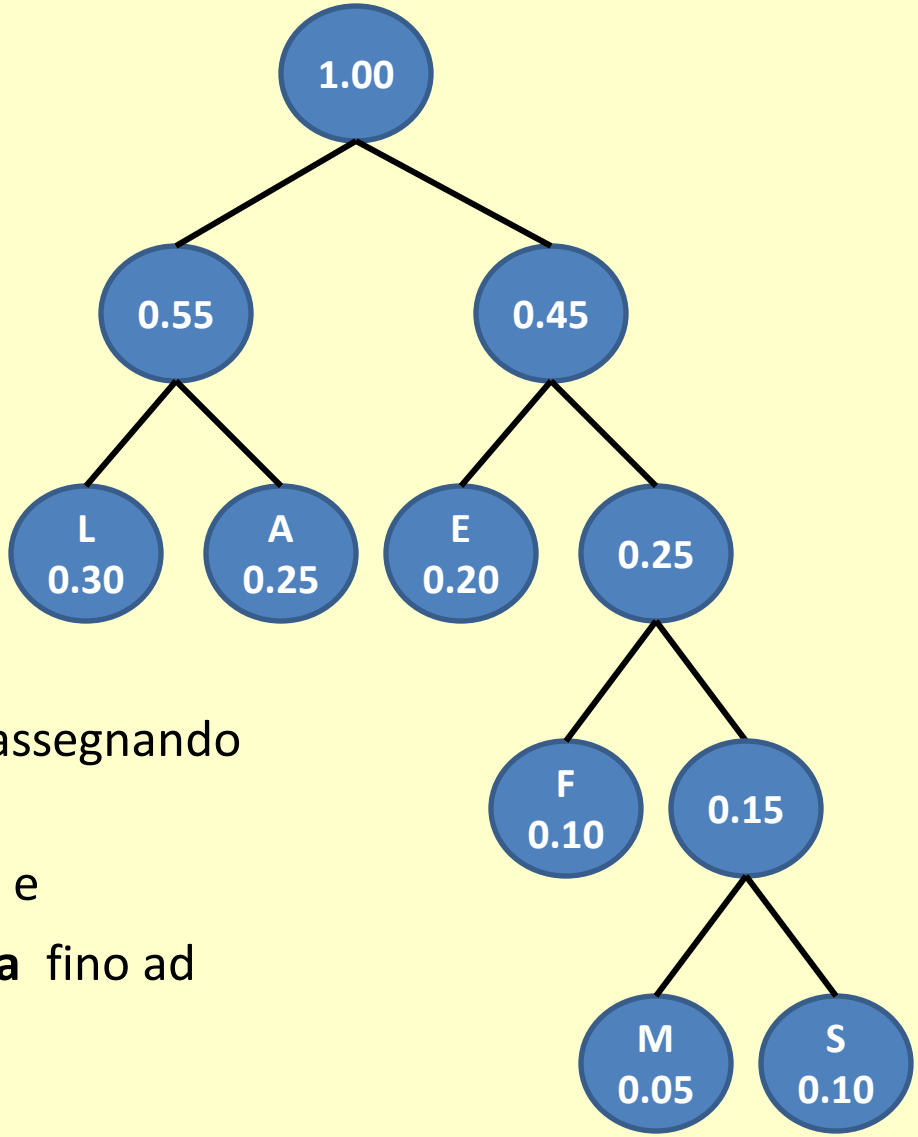
Esempio di codifica di Huffman

Simb.	Prob.			
				LA 0.55
				EFSM 0.45
L	0,30			
A	0.25			
			FSM 0.25	
E	0.20			
F	0.10	SM 0.15		
S	0.10			
M	0.05			



3. Il nodo rimanente è la radice con probabilità 1, e l'albero è completo.

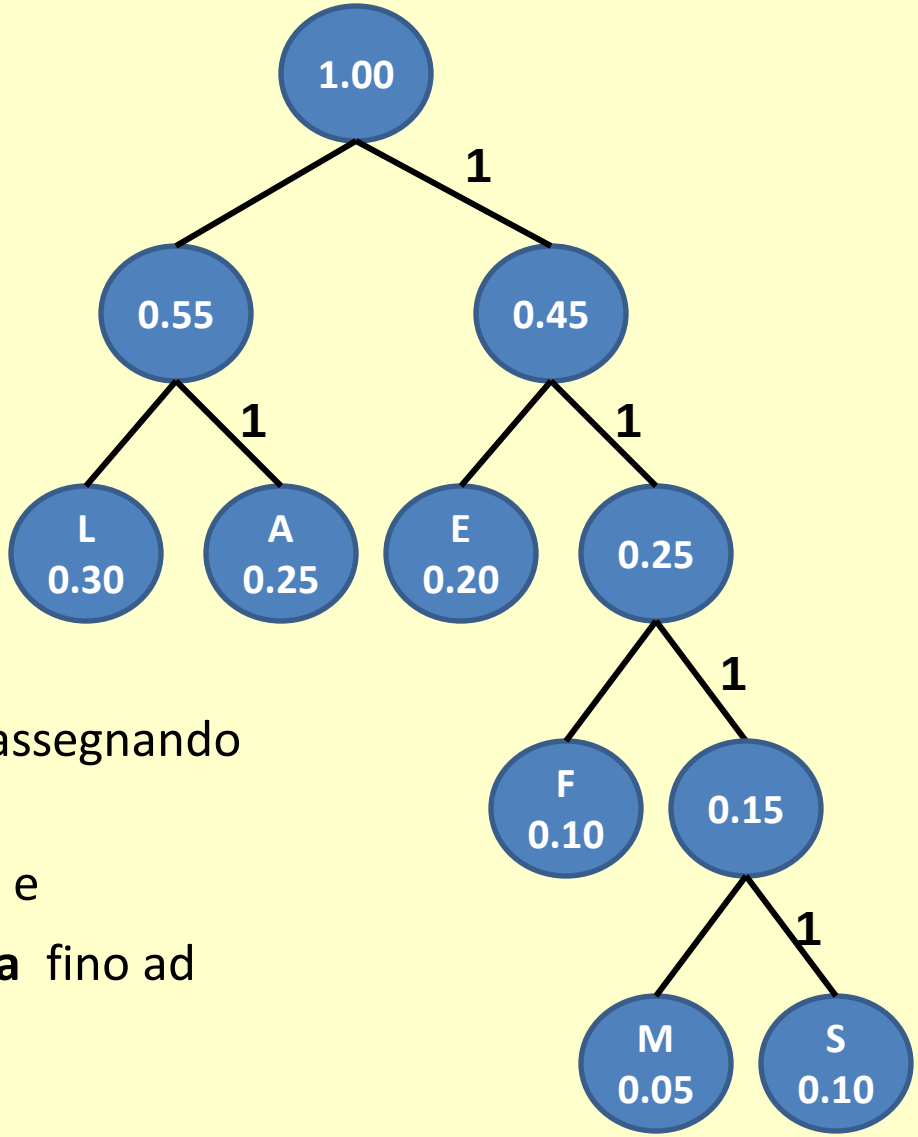
Esempio di codifica di Huffman



4. A partire dalla radice si visita l'albero assegnando ogni volta:

- **1** al ramo del **sottoalbero di destra** e
- **0** al ramo del **sottoalbero di sinistra** fino ad arrivare a tutte le foglie.

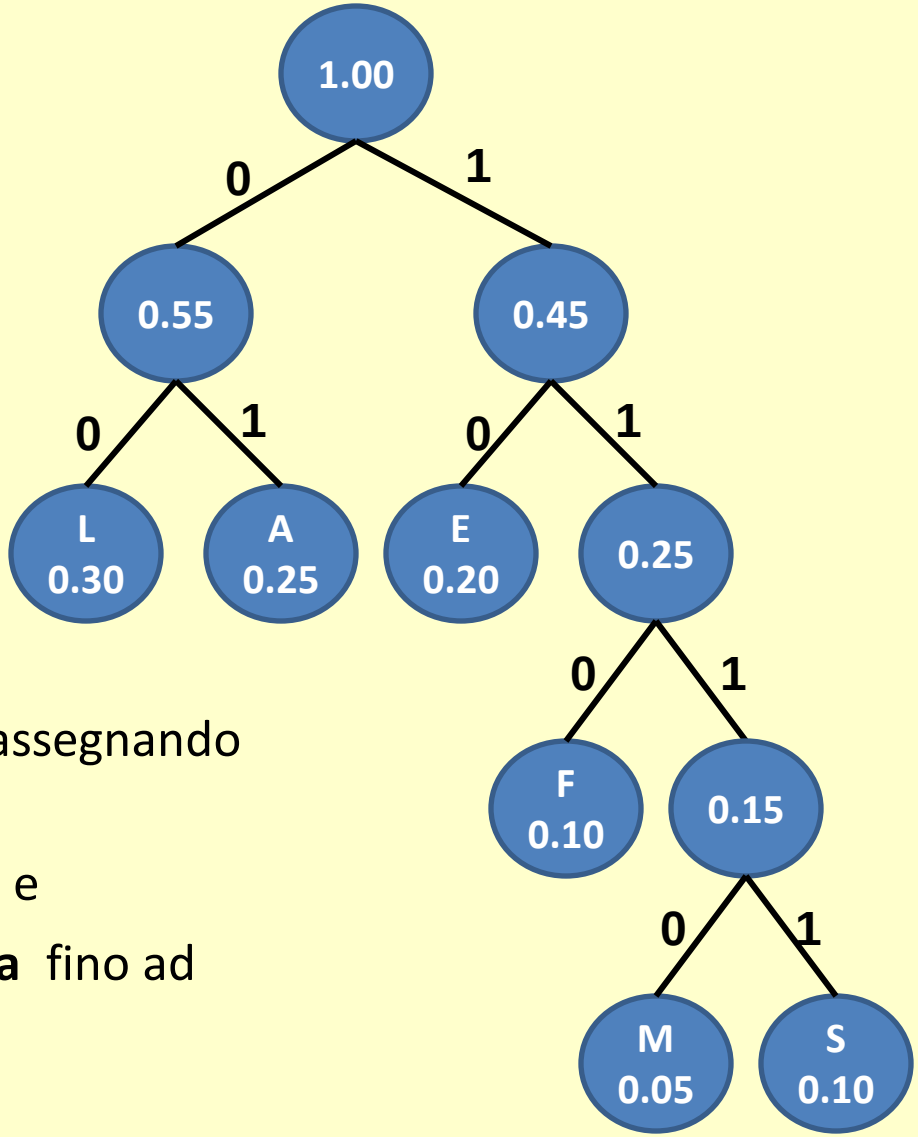
Esempio di codifica di Huffman



4. A partire dalla radice si visita l'albero assegnando ogni volta:

- **1** al ramo del **sottoalbero di destra** e
- **0** al ramo del **sottoalbero di sinistra** fino ad arrivare a tutte le foglie.

Esempio di codifica di Huffman



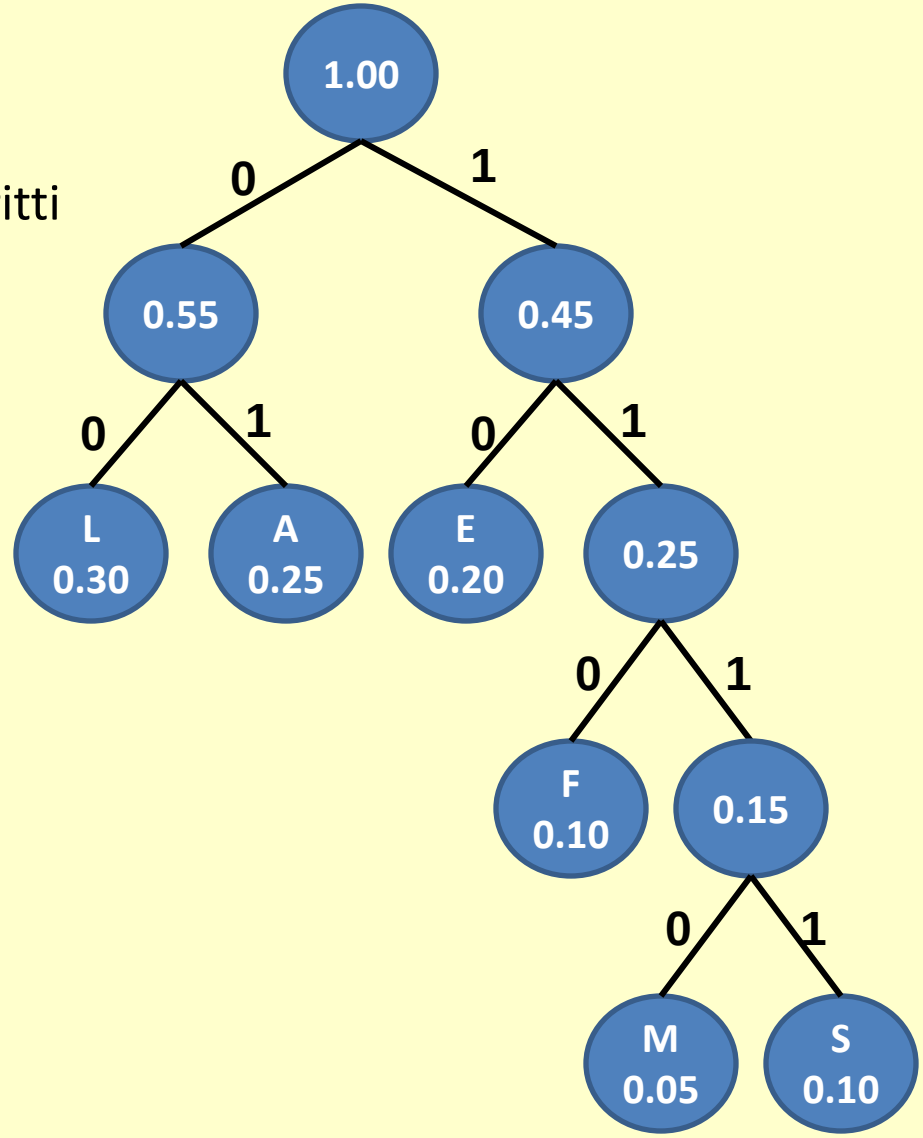
4. A partire dalla radice si visita l'albero assegnando ogni volta:

- **1** al ramo del **sottoalbero di destra** e
- **0** al ramo del **sottoalbero di sinistra** fino ad arrivare a tutte le foglie.

Esempio di codifica di Huffman

5. Si codifica ogni simbolo leggendo la sequenza di 0 e 1 che si incontrano scritti sui rami, attraversando l'albero dalla radice al simbolo stesso.

Simbolo	Prob.	Codice	Lunghezza
L	0,30	**0 0**	**2**
A	0.25	**0 1**	**2**
E	0.20	**1 0**	**2**
F	0.10	**1 1 0**	**3**
S	0.10	**1 1 1 1**	**4**
M	0.05	**1 1 1 0**	**4**



Esempio di codifica di Huffman

Simbolo	Prob.	Codice	Lunghezza
L	0,30	**0 0**	**2**
A	0.25	**0 1**	**2**
E	0.20	**1 0**	**2**
F	0.10	**1 1 0**	**3**
S	0.10	**1 1 1 1**	**4**
M	0.05	**1 1 1 0**	**4**

Si può ora calcolare la **lunghezza media del codice** così costruito utilizzando la formula seguente:

La **lunghezza media** del codice

$$L = \sum_{i=1}^{N} p(s_i) \cdot l(s_i) \quad bit / simbolo$$

L = **0,30 · 2 + 0,25 · 2 + 0,20 · 2 + 0,10 · 3 + 0,10 · 4 + 0,05 · 4 = 2,4** bit/simbolo

Il **codice** di Huffman generato secondo l'algoritmo appena esposto è il **migliore possibile** nel caso in cui la statistica dei simboli di sorgente sia nota a priori, nel senso che produce una codifica con il **minor numero possibile di bit/simbolo medi** per rappresentare un messaggio.

Inoltre l'etropia della sorgente diviso la lunghezza media del codice generato consente di misurare l'**efficienza** del codice stesso.

La **lunghezza media** del codice

$$L = \sum_{i=1}^{N} p(s_i) \cdot l(s_i) \quad bit / simbolo$$

L' **entropia** della sorgente

$$H(S) = \sum_{i=1}^{N} p(s_i) \cdot \log_2 \frac{1}{p(s_i)} \quad bit / simbolo$$

L' **efficienza** del codice costruito

$$\eta = \frac{H(S)}{L}$$

1° Teorema di Shannon afferma che: $\mathbf{L \geq H(S)}$

Un **codice decodificabile** istantaneamente non può avere lunghezza media **L** inferiore all'entropia **H(S)** della sorgente.

Questo significa che in un **codice decodificabile**
la **lunghezza media di un simbolo**, in bit,
non può essere inferiore
alla **quantità media di informazione che può trasportare**
e
l'**efficienza H(S)/L** sarà sempre un valore $\leq \mathbf{1}$.

Inoltre un codice si dice:

efficiente	se	**L = H**
ridondante	se	**L > H**
ambiguo	se	**L < H**

In questo caso non è garantita la corretta decodifica.