Compressione dati e algoritmo di Huffman

17 maggio 2023

Questa foto di Autore sconosciuto è concesso in licenza da CC BY-SA-NC

Un esempio di codifica: il codice Morse

Il **codice Morse** è un codice binario in cui le lettere sono codificate da una particolare sequenza di punti e di linee. Il codice Morse è conosciuto anche come alfabeto Morse e prende il nome dal suo inventore Samuel Morse. Il codice Morse viene inventato nel 1837 per trasmettere messaggi tramite il primo prototipo di telegrafo elettrico. Un messaggio scritto in linguaggio naturale viene tradotto, lettera dopo lettera, in codice Morse ed infine inviato sotto forma di segnali elettrici brevi o lunghi in un cavo elettrico.

Lettera	Codice	Lettera	Codice	Numeri	Codice	Varie	Codice
A	•—	N	—•	0	—————	•	•—•—•—
B	—•••	O	———	1	•————	,	——••——
C	—•—•	P	•——•	2	••———	:	———•••
D	—••	Q	——•—	3	•••——	?	••——••
E	•	R	•—•	4	••••—	=	—•••—
F	••—•	S	•••	5	•••••	-	—••••—
G	——•	T	—	6	—••••	(	—•——•
H	••••	U	••—	7	——•••	)	—•——•—
I	••	V	•••—	8	———••	"	•—••—•
J	•———	W	•——	9	————•	'	•—————
K	—•—	X	—••—			/	—••—•
L	•—••	Y	—•——				
M	——	Z	——••				

••• ——— •••

S O S

Decodifica univoca

Problema di ambiguità:

A è codificato con . _

E è codificato con .

T è codificato con _

. _ . _ è la codifica di ?

ETA, AA, ETET, oppure AET?

Importante: decodifica/decifrabilità univoca

Soluzione di Morse: introdurre «pausa», un terzo simbolo

Codifica binaria a lunghezza fissa

Poiché i computer operano (alla fine) su sequenze di bit 0/1, ci interessa la codifica binaria

I metodi di codifica dati standard, come il sistema ASCII o Unicode, usano stringhe binarie di lunghezza fissa (*blocchi*) per codificare caratteri (stringhe di lunghezza 8 nel sistema ASCII e di lunghezza 16 nel sistema Unicode). Ad esempio, nel codice ASCII il carattere A viene codificato con 01000001, B con 01000010, C con 01000011, e cosí via...

Vantaggio: I codici a lunghezza fissa assicurano l'univoca decifrabilità.

Nel codice ASCII, quale messaggio è codificato in:

01000010010000010100001 1 ?

Divido in blocchi di 8 bit:

01000010 01000001 01000011 = B A C

Codifica binaria a lunghezza fissa: svantaggi

C'è qualcosa in più che possiamo chiedere ad una codifica?

Per esempio: se volessimo codificare in binario un testo in lingua italiana su un alfabeto di 21 (?) caratteri con un codice binario a lunghezza fissa, dovremmo associare ad ogni carattere una stringa binaria con 5 bit almeno (4 non bastano).

Quindi ogni testo di 1.000 caratteri sarebbe codificato in una stringa binaria di 5.000 bit.

Però, tipicamente, in un testo in italiano ci saranno molte a, e, i, t, ma poche q, z, h.

E' necessario codificare tutti i caratteri con la stessa lunghezza?

Non sarebbe più conveniente codificare i caratteri più frequenti con stringhe corte e quelli meno frequenti con stringhe più lunghe?

Consideriamo allora la possibilità di una….

Codifica binaria a lunghezza variabile

- Esempio: codifico a, b, c, d, e, f con

  0, 100, 101, 111, 1101, 1100, rispettivamente.

La decodifica è univoca?

Quale messaggio è codificato da: 0 0 0 1 0 1 1 1 0 1 0 ?

0 0 0 101 1101 0 = a a a c e a

L'insieme {0, 100, 101, 111, 1101, 1100} è prefisso:

nessuna stringa è prefissa di un'altra. La decodifica è univoca (e istantanea).

Nota: Tutti i codici a lunghezza fissa sono prefissi

- Esempio (codice univ. dec. non prefisso): codifico a, b, c con 0, 01, 11
  L'insieme {0, 01, 11} non è prefisso: 0 è prefisso di 01, ma la decodifica è unica:

  01111101 = 01 11 11 01 = b c c b

  011111101 = 0 11 11 11 01 = a c c c b

- Esempio (codice non univ. dec.): 0, 10, 01; quale messaggio è codificato da 010?

Compressione Dati

Informalmente, il problema é il seguente:

- Abbiamo una stringa X su di un dato alfabeto (ad es., quello della lingua italiana), e vogliamo codificare X in una sequenza binaria Y che sia *la piú corta possibile* ("comprimere X").

Perché lo vogliamo fare?

- Per risparmiare spazio di memoria (ad es. per memorizzare documenti di grandi dimensioni in memorie di capacitá limitata.
- Per ridurre il tempo di trasmissione dati su canali di banda limitata (ad es., modem lenti o connessioni senza fili)

Per ottenere la codifica *più corta possibile* di un testo, studiamo prima le frequenze di ogni carattere all'interno del testo. L'idea sarà di sfruttare la possibilità di utilizzare stringhe binarie di varia lunghezza, per codificare i caratteri meno frequenti con stringhe più lunghe.

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

carattere	_	a	e	o	i	d	n	m	l	z	t	s	r	v	c
frequenza	6	3	3	2	3	2	3	3	2	2	2	1	1	1	1

	Blocchi	Huffman		Blocchi	Huffman
_	0000	00	a	0001	1101
c	0010	10001	d	0011	1010
e	0100	1100	i	0101	1111
l	0110	0101	m	0111	1110
n	1000	1011	o	1001	0100
r	1010	10000	s	1011	10011
t	1100	0111	v	1101	10010
z	1110	0110			

Abbiamo un testo composto da 15 distinti caratteri, quindi per una codifica a lunghezza fissa ci abbisognano almeno 4 bits per carattere. La tabella a fianco mostra una possibile codifica a blocchi ed una basata sul codice di Huffman.

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

Testo: `nel_ mezzo_ del_cammin_di_nostra_vita` di lunghezza 35

Codifica a lunghezza fissa:
10000100011000000111010011101110100100000011010000110...

Codifica di Huffman:
10111100010100111011000110011 0 0100001010110001010...

Per codificare tutto il testo con la codifica a blocchi si useranno 140 bits, se ne useranno 132 impiegando la codifica di Huffman.

Spesso, la codifica di Huffman puó portare a comprimere le dimensioni dei file fino al 50% della dimensione originale.

Confronto

carattere	a	b	c	d	e	f
Frequenza su 100	45	13	12	16	9	5
Lunghezza fissa	000	001	010	011	100	101
Lunghezza variabile	0	100	101	111	1101	1100

Codice a lunghezza fissa

(Tutti i caratteri sono codificati con lo stesso numero di bit):

per 100.000 caratteri servono 300.000 bit

Codice a lunghezza variabile

(I caratteri sono codificati con un numero variabile di bit):

per 100.000 caratteri servono solo

45.000x1 + 13.000x3 + 12.000x3 + 16.000x3 + 9.000x4 + 5.000x4 = 224.000 bit

224.000 < 300.000

I codici a lunghezza variabile possono aiutare a minimizzare il numero di bit necessari

Lunghezza media di una codifica

carattere	a	b	c	d	e	f
Frequenza su 100	45	13	12	16	9	5
Lunghezza variabile	0	100	101	111	1101	1100

Esempio (continua):

C = {a, b, c, d, e, f} con codifica $\gamma : C \rightarrow \{0,1\}^*$

	Lunghezza
$a \rightarrow 0$	1
$b \rightarrow 100$	3
$c \rightarrow 101$	3
$d \rightarrow 111$	3
$e \rightarrow 1101$	4
$f \rightarrow 1100$	4

ABL(γ) = 2.24
Testo di lunghezza 100.000 caratteri è codificato mediamente con 224.000 bit (vedi slide precedente)

Per codificare n caratteri servono mediamente:

45/100 n X 1 + 13/100 n X 3+ 12/100 n X 3+ 16/100 n X 3 + 9/100 n X 4+ 5/100 n X 4 =

= $n \sum_c f(c) \, |\gamma(c)| = n$ X 2.24

Def. La lunghezza media della codifica γ è ABL(γ) = $\sum_c f(c) \, |\gamma(c)|$

ABL = Average Bit Length

Il problema

Input: $C = \{c_1, c_2, \ldots, c_n\}$

con relative frequenze $f(c_1), f(c_2), \ldots, f(c_n)$

Output: Codice prefisso binario

$\gamma : C \rightarrow \{0, 1\}^*$ tale che la quantità

$$ABL(\gamma) = \sum_{c \in C} f(c) \mid \gamma(c) \mid$$

sia minima

Vantaggi codici prefissi

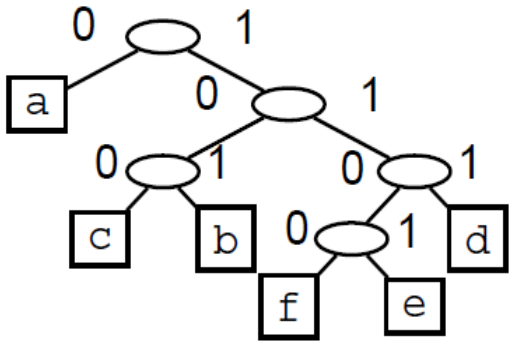
Abbiamo ristretto la ricerca ai codici prefissi perché:

- Teoria dell'informazione: «Se c'è un codice con una certa distribuzione di lunghezze, ve ne è uno prefisso con la stessa distribuzione di lunghezze»
- Decodifica univoca e 'istantanea'
- Facili da costruire…

Rappresentazione di codici prefisso

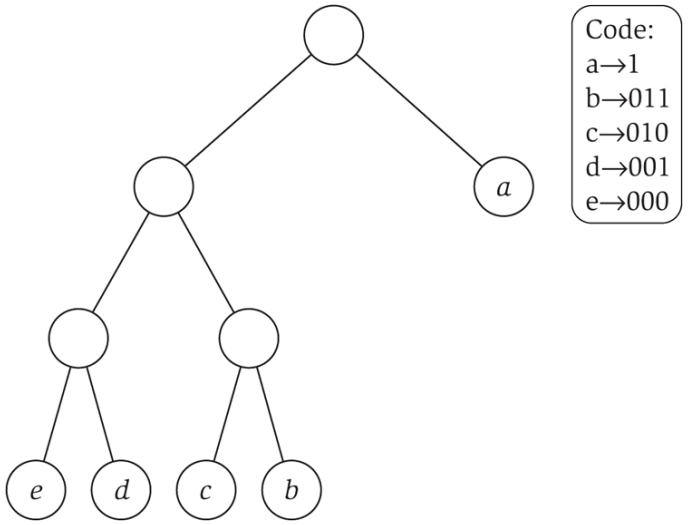
I codici prefisso possono essere rappresentati da alberi in cui le foglie corrispondono ai caratteri da codificare, ed i relativi percorsi radice-foglia corrispondono alle codifiche dei caratteri.

Esempio:

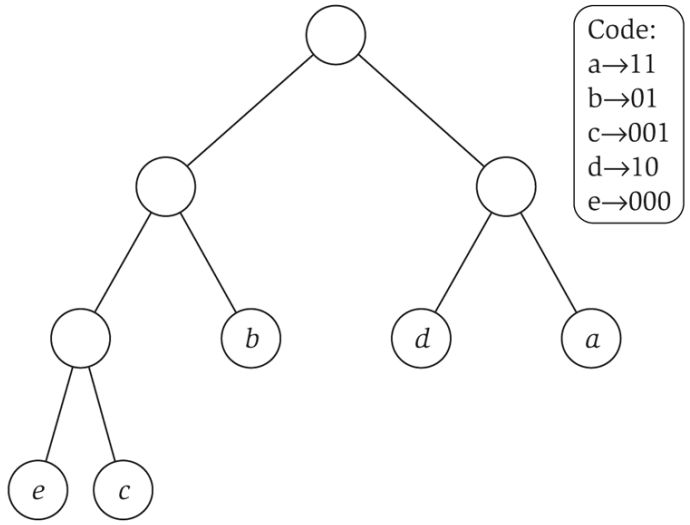


carattere	codifica
a	0
c	100
b	101
d	111
f	1100
e	1101

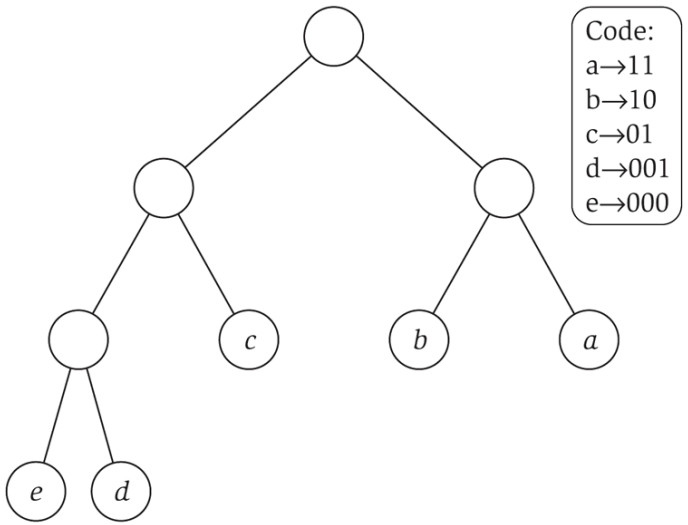
La profondità di una foglia nell'albero è uguale alla lunghezza della corrispondente codifica



(a)



(b)



(c)

Ad uno stesso insieme di caratteri possiamo associare vari alberi.
(a) e (b) differiscono nella forma
(b) e (c) differiscono solo nell'assegnazione dei caratteri alle foglie

Figure 4.16 Parts (a), (b), and (c) of the figure depict three different prefix codes for the alphabet $S = \{a, b, c, d, e\}$.

Il problema…

Input: $C = \{c_1, c_2, \ldots, c_n\}$

con relative frequenze $f(c_1), f(c_2), \ldots, f(c_n)$

Output: Codice prefisso binario

$\gamma : C \rightarrow \{0, 1\}^*$ tale che la quantità

$$ABL(\gamma) = \sum_{c \in C} f(c) \, |\gamma(c)|$$

sia minima

…. diventa

un problema su alberi

con *n* foglie

Dato un alfabeto di caratteri $C = \{c_1, c_2, \ldots, c_n\}$, con relative frequenze $f(c_1), f(c_2), \ldots, f(c_n)$, trovare un albero binario T ed una associazione di foglie a caratteri tale che la quantitá

$$ABL(T) = \sum_{c \in C} f(c) d_T(c)$$

sia la *minima* possibile. Un albero T siffatto verrà detto *ottimo*.

$d_T(c)$= profondità della foglia assegnata al carattere c nell'albero T

Alberi ottimali

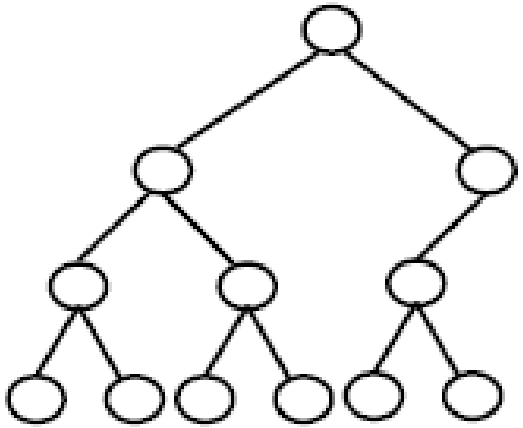
Ma vediamo come si presenta un albero ottimo rispetto:

- la forma e
- l'associazione dei caratteri alle foglie.

Intuizione di base (greedy):
conviene associare a caratteri meno frequenti, stringhe binarie più lunghe.

Domanda

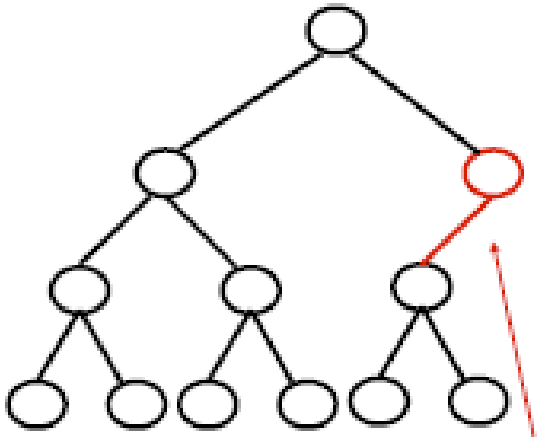
Potrebbe essere ottimo per un alfabeto S con 6 caratteri e frequenze $f(x)>0$, per $x \in S$?



A	000
B	001
C	010
D	011
E	100
F	101

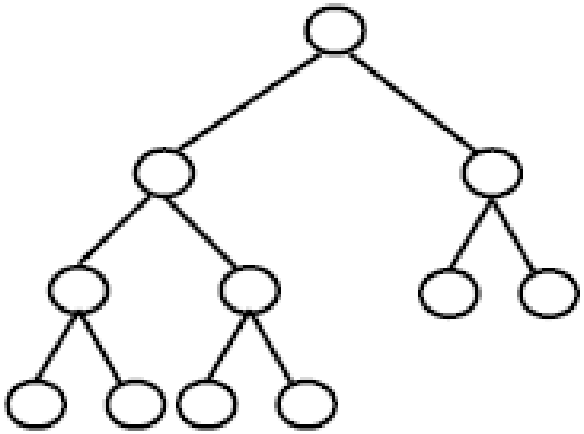
Domanda

Potrebbe essere ottimo per un alfabeto S con 6 caratteri e frequenze $f(x)>0$, per $x \in S$? No!



A	000
B	001
C	010
D	011
E	100
F	101

inutile

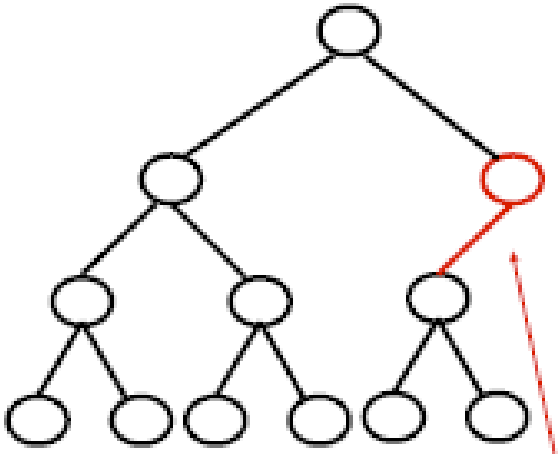


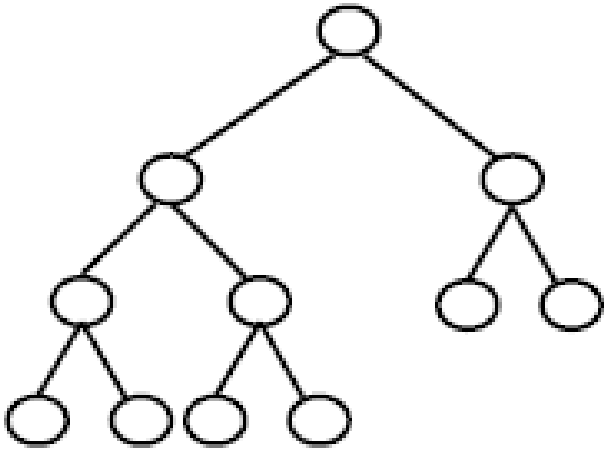
A	000
B	001
C	010
D	011
E	10
F	11

E' migliore!

Proprietà: gli alberi ottimi sono pieni

L'albero binario corrispondente ad un codice ottimo è pieno (full).
Ovvero, ogni nodo interno ha due figli.





A	000
B	001
C	010
D	011
E	100
F	101

inutile

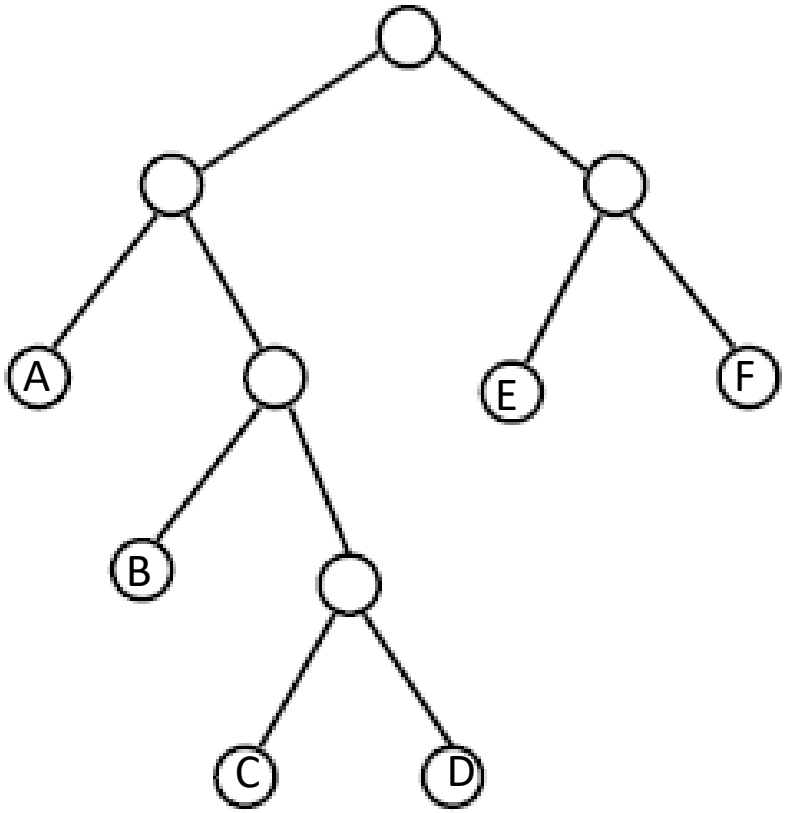
A	000
B	001
C	010
D	011
E	10
F	11

E' migliore!

Domanda

Potrebbe essere ottimo questo codice prefisso?

	frequenza	parola codice
A	25	00
B	25	010
C	6	0110
D	6	0111
E	13	10
F	25	11

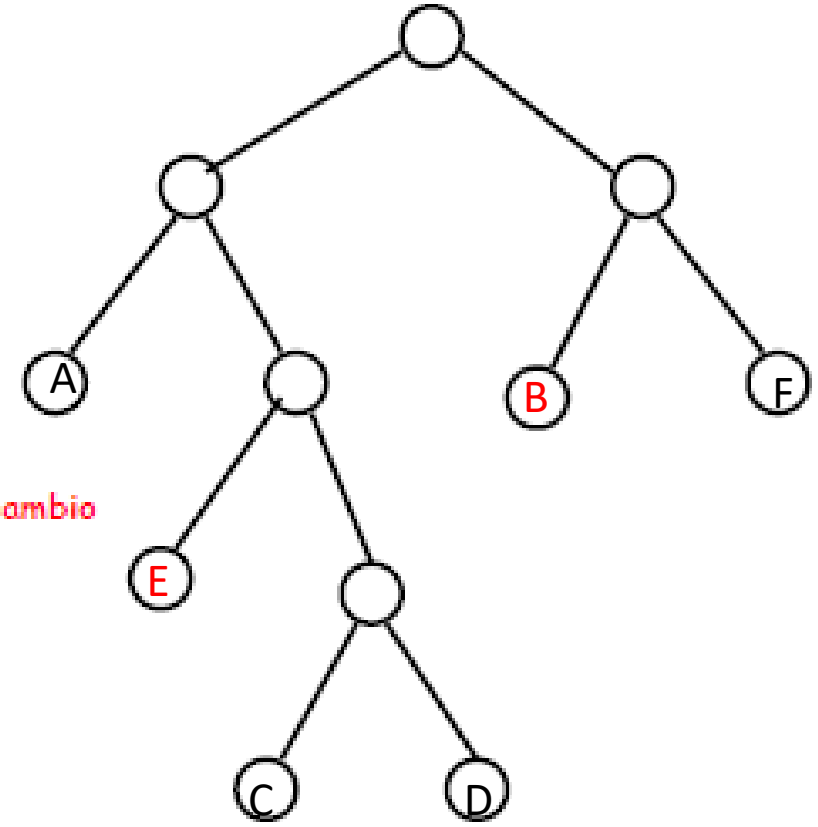


Domanda

Potrebbe essere ottimo questo codice prefisso? No!

	frequenza	parola codice
A	25	00
B	25	010
C	6	0110
D	6	0111
E	13	10
F	25	11

scambio



Guadagno in lunghezza:

(25x3 + 13x2) – (25x2 + 13x3) = 25x(3-2) -13x(3-2) = (25 – 13) (3-2) = 12

Nel calcolo dell'ABL:
il contributo di B ed E prima dello scambio > il contributo di B ed E dopo lo scambio.
L'ABL dopo lo scambio sarebbe strettamente minore!

Proprietà: associazione caratteri con parole codice

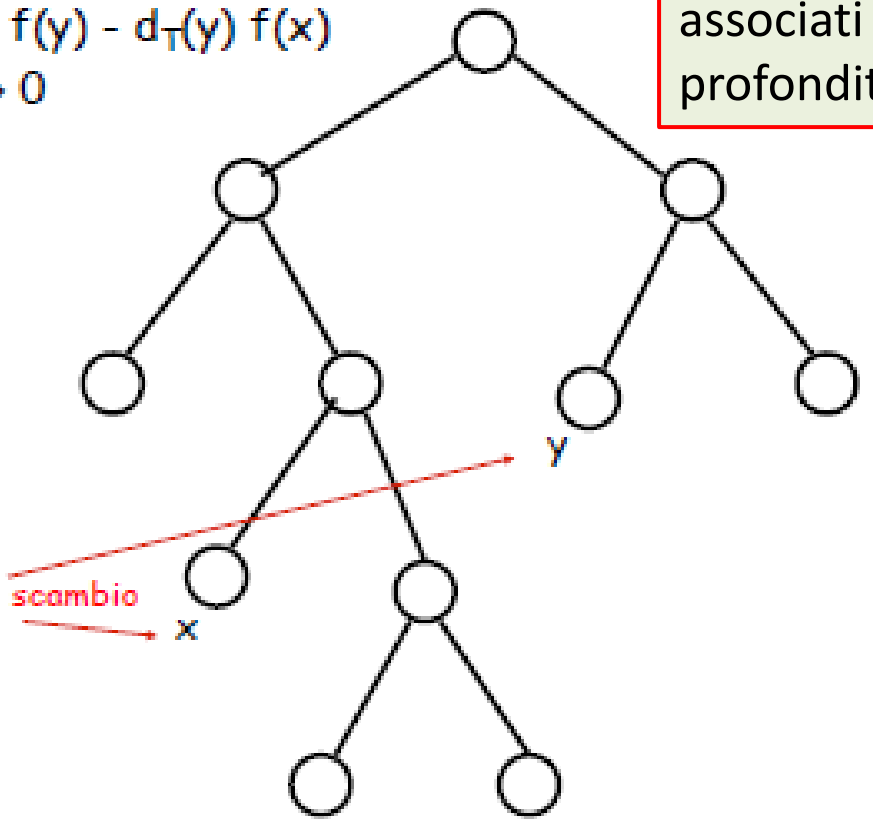
Teorema. Sia T un albero prefisso ottimale per l'alfabeto S con frequenze f(x), per $x \in S$. Se $d_T(x) > d_T(y)$ allora $f(x) \leq f(y)$.

Prova. (per assurdo) Supponiamo che $f(x) > f(y)$.
Scambiamo le parole codice. Il guadagno in lunghezza è:

$$d_T(x)\, f(x) + d_T(y)\, f(y) - d_T(x)\, f(y) - d_T(y)\, f(x)$$
$$= (d_T(x) - d_T(y))\, (f(x) - f(y)) > 0$$

In un albero ottimale, i caratteri con frequenze basse sono associati a foglie a profondità maggiore

	frequenza	parola codice
A	25	00
x	25	010
C	6	0110
D	6	0111
y	13	10
F	25	11



Costruzione albero ottimale

In un albero ottimale, i caratteri con frequenze basse sono associati a foglie a profondità maggiore

Da dove partire con la costruzione?

Dall'alto verso il basso: non sempre si ottiene un codice ottimale (codici di Shannon-Fano).

Soluzione di Huffman: dal basso verso l'alto!

Dal basso verso l'alto

Teorema. Esiste un albero ottimale in cui i due caratteri con frequenze minimali sono associati a due foglie «sorelle».

Prova. Sia T* un albero ottimale.

Nota: l'ordine con cui associamo dei caratteri a foglie che hanno la stessa profondità, è ininfluente (il loro contributo sarà lo stesso).

Consideriamo le foglie a profondità massima. Sia x una di queste foglie. Il padre w di x ha un altra figlia y (T* è pieno); e y è una foglia (perché la profondità di x e di y è massima). Le foglie x, y e tutte le (eventuali) altre foglie alla profondità massima conterranno i caratteri di frequenza minima (vedi Teorema precedente). Per la *Nota* sopra, esisterà un albero ottimale in cui x e y sono associate ai caratteri di frequenza minima.

Scelta *greedy* e sottostruttura ottima

Quindi, un algoritmo per la costruzione di un albero ottimale per un insieme di caratteri S, può partire associando i due caratteri di frequenza minimali, x e y, a due foglie «sorelle», cioè figlie di uno stesso nodo w.

Inoltre, vedremo che, il resto dell'albero, con w come foglia, sarà anch'esso un albero ottimale, per un nuovo sotto-problema, quello per $S\backslash\{x,y\} \cup \{z\}$, dove z è un carattere fittizio con frequenza $f_z = f_x + f_y$.

Sottoproblema

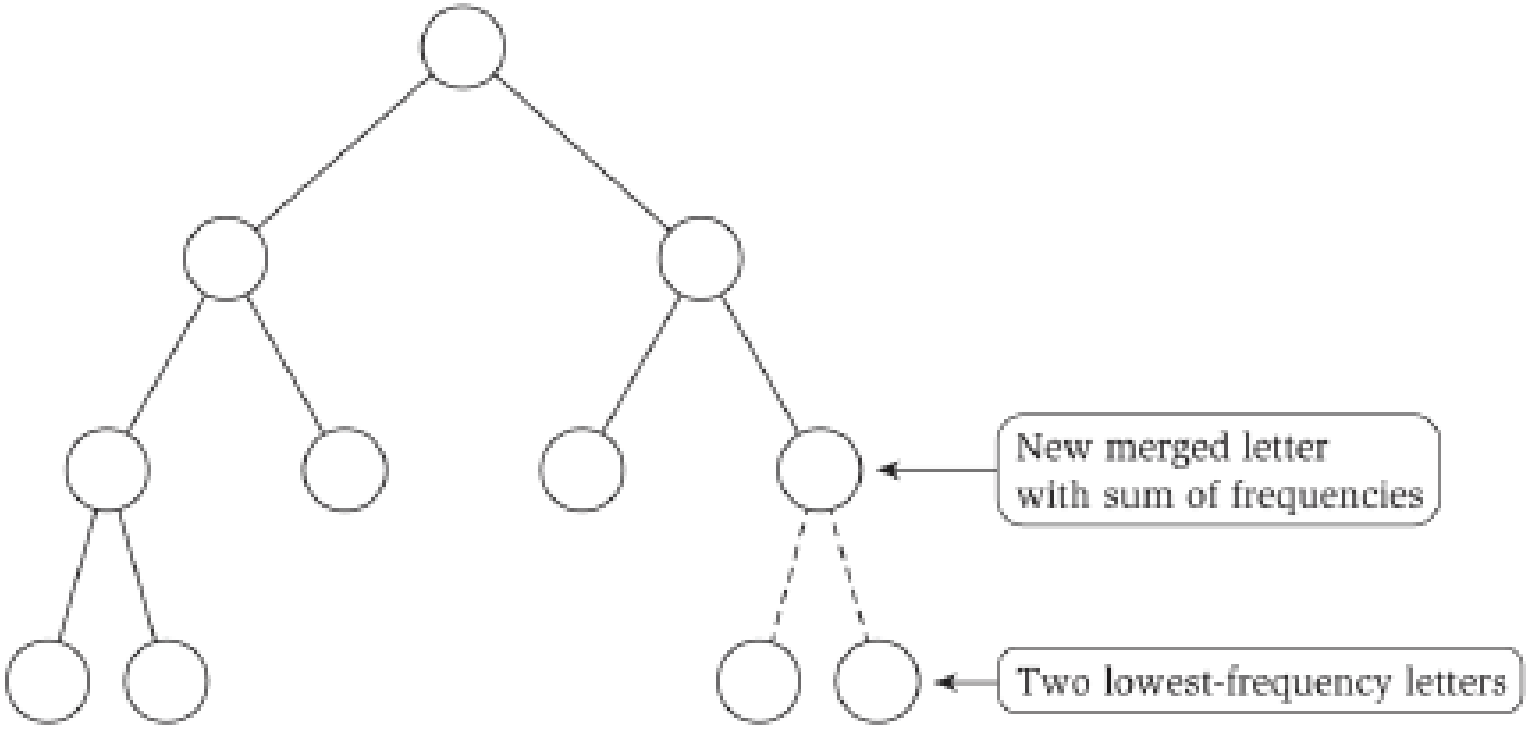
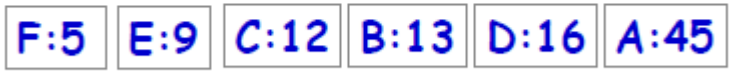
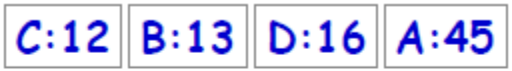
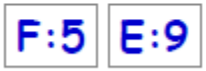


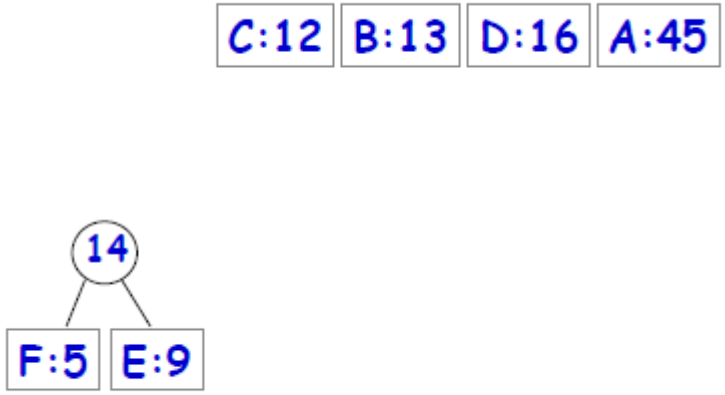
Figure 4.17 There is an optimal solution in which the two lowest-frequency letters label sibling leaves; deleting them and labeling their parent with a new letter having the combined frequency yields an instance with a smaller alphabet.

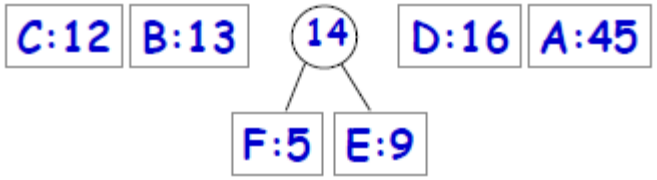
Esempio della costruzione

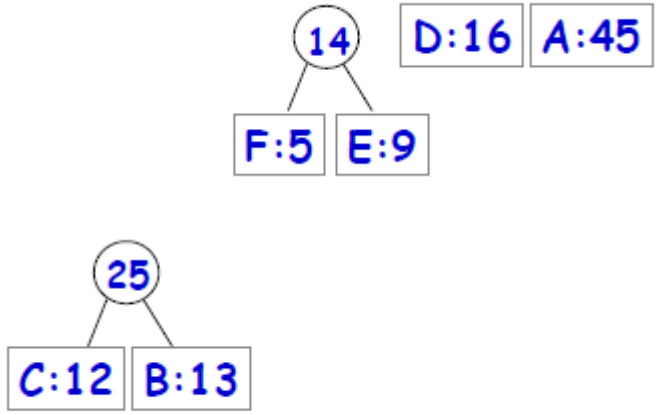


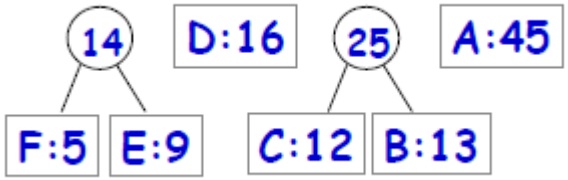

C:12
B:13
D:16
A:45

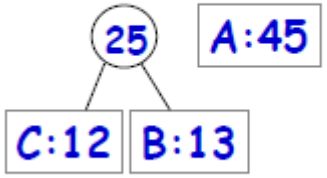

F:5
E:9

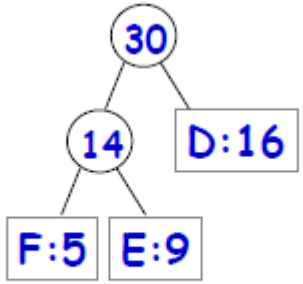

C:12
B:13
D:16
A:45
14
F:5
E:9

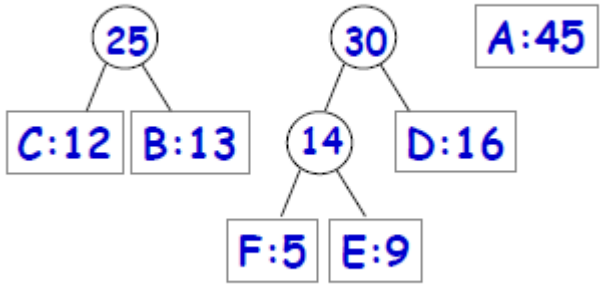

C:12
B:13
14
D:16
A:45
F:5
E:9

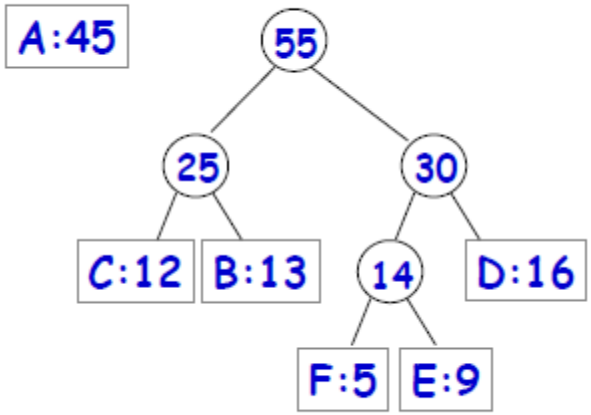

14
D:16
A:45
F:5
E:9
25
C:12
B:13

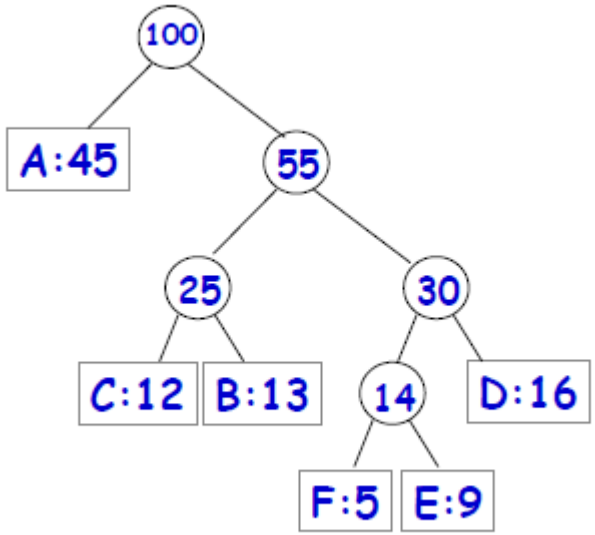

14
D:16
25
A:45
F:5
E:9
C:12
B:13


25
A:45
C:12
B:13

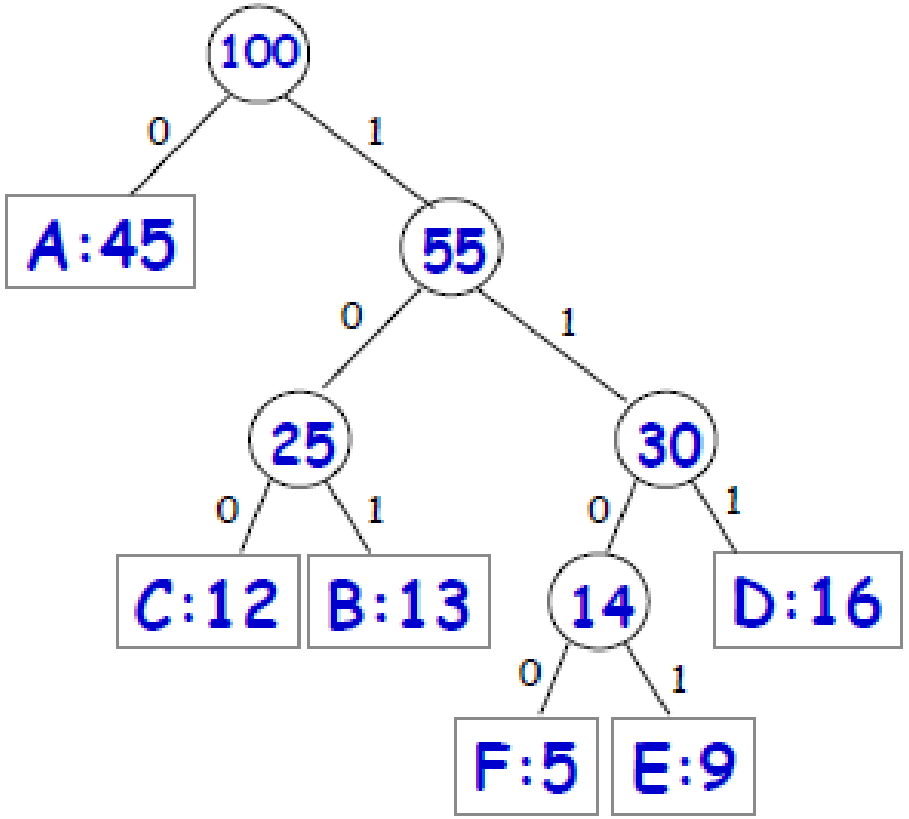

30
14
D:16
F:5
E:9


25
30
A:45
C:12
B:13
14
D:16
F:5
E:9


A:45
55
25
30
C:12
B:13
14
D:16
F:5
E:9


100
A:45
55
25
30
C:12
B:13
14
D:16
F:5
E:9

Esempio



	frequenza	parola codice
A	45	0
B	13	101
C	12	100
D	16	111
E	9	1101
F	5	1100

ABL(γ)= $\sum_c f(c) \mid \gamma(c)\mid$ =

45/100 X 1 + 13/100 X 3+ 12/100 X 3+ 16/100 X 3 + 9/100 X 4+ 5/100 X 4 = 2.24

Huffman: 2.24 è il minimo possibile!

Algorithm 4.6, page 172

To construct a prefix code for an alphabet S, with given frequencies:
  If S has two letters then
    Encode one letter using 0 and the other letter using 1
  Else
    Let y^* and z^* be the two lowest-frequency letters
    Form a new alphabet S' by deleting y^* and z^* and
      replacing them with a new letter ω of frequency $f_{y^*} + f_{z^*}$
    Recursively construct a prefix code γ' for S', with tree T'
    Define a prefix code for S as follows:
      Start with T'
      Take the leaf labeled ω and add two children below it
        labeled y^* and z^*
   Endif

Algoritmo di Huffman: implementazione

```
Huffman(A, n) {
    Inizializza coda a priorità vuota Q
    for i=1 to n {
        insert A[i] in Q
    }
    for i=1 to n-1 {
        x ← estrai elemento minimo da Q
        y ← estrai elemento minimo da Q
        "Crea nuovo albero con radice z e
         frequenza f(x)+f(y) e figli x e y"
        Insert z in Q
    }
}
```

Operazione PQ	Huffman	Array	Binary Heap
Insert	2n-1	1	log n
ExtractMin	2(n-1)	n	log n
ChangeKey	0	1	log n
IsEmpty	0	1	1
Total		$O(n^2)$	$O(n \log n)$

Nota sull'algoritmo di Huffman

Nell'algoritmo è detto:

estrai minimo di Q in x (primo minimo)

estrai minimo di Q in y (secondo minimo)

crea nodo z con figli x e y

L'implementazione più naturale assegnerà x=left[z] e y =right[z] (e non viceversa) ad ogni iterazione.

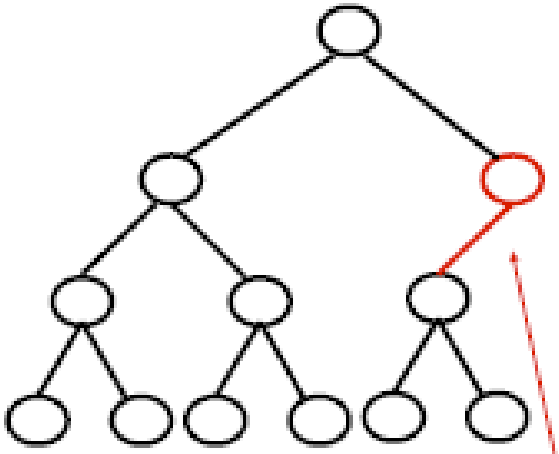
Noi adotteremo questa per convenzione negli esercizi (se non specificato).

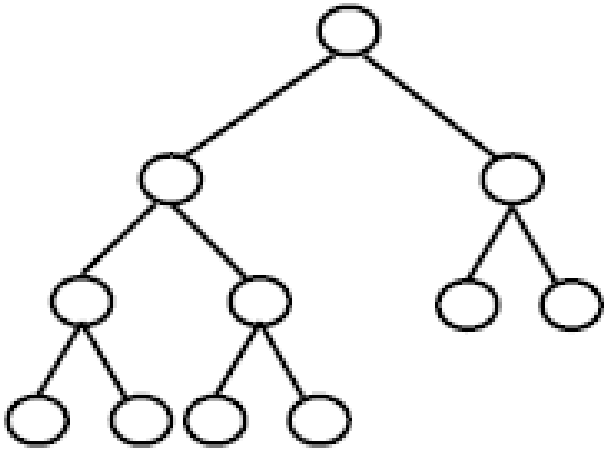
Cosa cambierebbe altrimenti?

Verso la correttezza

Proprietà: gli alberi ottimi sono pieni

L'albero binario corrispondente ad un codice ottimo è pieno (full).
Ovvero, ogni nodo interno ha due figli.





A	000
B	001
C	010
D	011
E	100
F	101

inutile

A	000
B	001
C	010
D	011
E	10
F	11

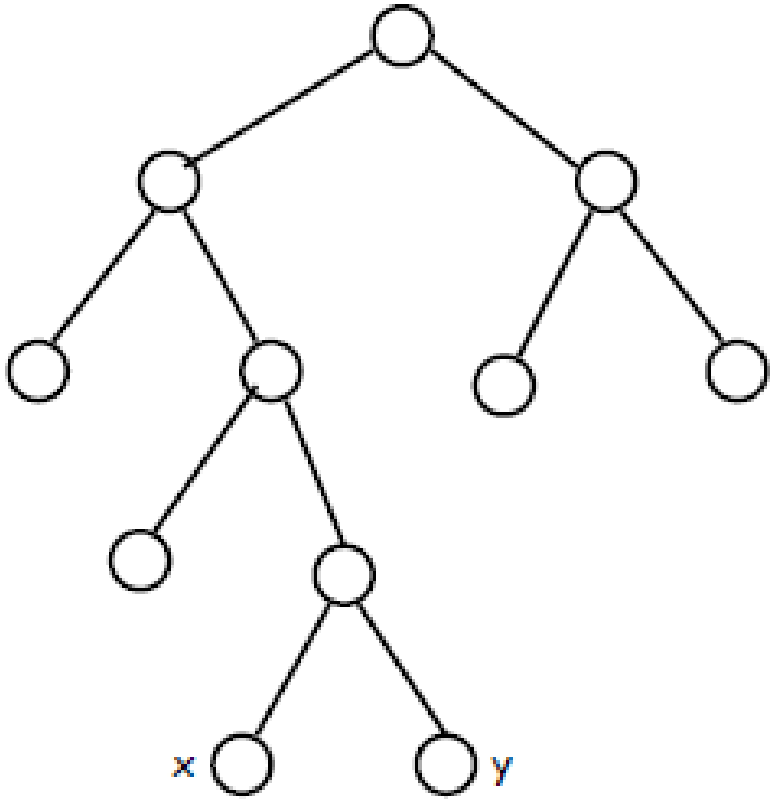
E' migliore!

Proprietà: associazione caratteri con parole codice

Teorema. Sia S un alfabeto con frequenze f(x), per $x \in S$. Siano $x,y \in S$ caratteri con le frequenze più basse, $f(x) \leq f(y)$. Allora esiste un codice ottimale con albero T* in cui x,y corrispondono a foglie che sono fratelli.

Esempio

	frequenza	parola codice
A	25	00
B	25	010
x	6	0110
y	6	0111
E	13	10
F	25	11



Algorithm 4.6, page 172

To construct a prefix code for an alphabet S, with given frequencies:
  If S has two letters then
    Encode one letter using 0 and the other letter using 1
  Else
    Let y^* and z^* be the two lowest-frequency letters
    Form a new alphabet S' by deleting y^* and z^* and
      replacing them with a new letter ω of frequency $f_{y^*} + f_{z^*}$
    Recursively construct a prefix code γ' for S', with tree T'
    Define a prefix code for S as follows:
      Start with T'
      Take the leaf labeled ω and add two children below it
        labeled y^* and z^*
   Endif

Codici di Huffman: correttezza (1)

Teorema: L' algoritmo di Huffman produce un codice prefisso ottimo

Prova.

Dato che l' algoritmo è ricorsivo, la prova è per induzione sulla cardinalità dell' alfabeto.

L' asserto è vero per k=2, cioè per un alfabeto con due simboli.

Supponiamo che sia ottimale per tutti gli alfabeti con k-1 simboli.

Mostreremo che è ottimale per gli alfabeti con k simboli.

Sia T l'albero restituito dall'algoritmo di Huffman per un alfabeto S con k simboli.

Quindi:

Siano y* e z* simboli di frequenza minimi e S' = S\ {y*, z* } U {w} con f(w)=f(y*)+f(z*)

Sia T' l'albero costruito ricorsivamente dall'algoritmo per S'. Per induzione T' è ottimale per S' poiché |S '| = k-1.

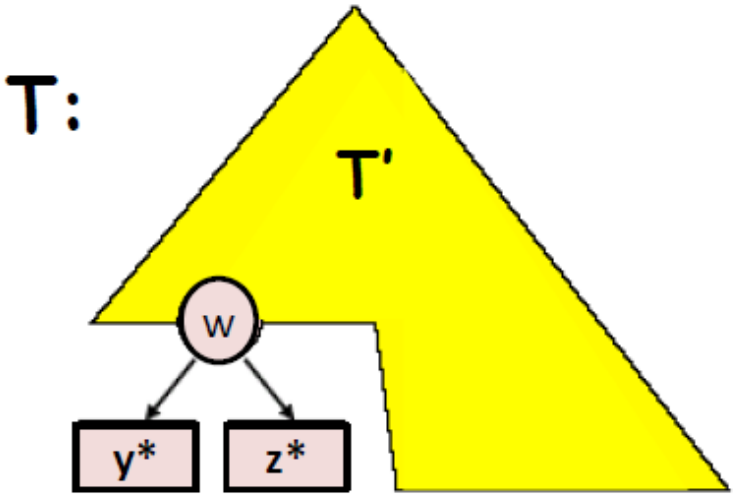
L'albero T è ottenuto da T' aggiungendo alla foglia corrispondente a w due figli etichettati y* e z*.

Si ha che: ABL(T) = ABL (T') + f(w), ovvero ABL(T') = ABL (T) - f(w) (segue dimostrazione)

Codici di Huffman Correttezza (2)

Prova (continua).

Adesso supponiamo per assurdo che T non sia ottimale. Esiste un albero Z per S con ABL(Z) < ABL(T).



Per quanto precedentemente dimostrato, possiamo supporre che Z sia uno di quelli in cui le foglie corrispondenti a y* e z* siano «sorelle».

Allora sia Z' l'albero ottenuto da Z eliminando y* e z* in cui facciamo corrispondere al nodo padre di y* e z* il simbolo w.

Abbiamo che: $ABL(Z') = ABL(Z) - f(w)$ e

$$ABL(T') = ABL(T) - f(w).$$

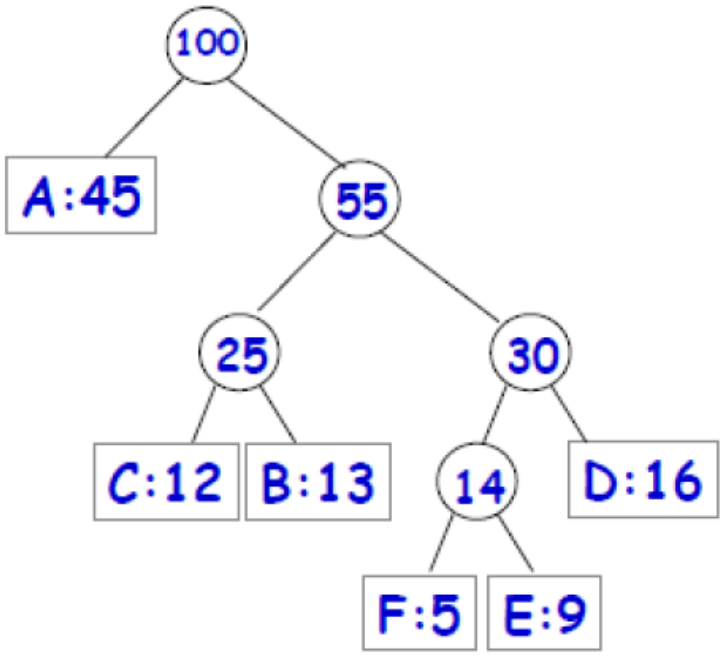
Quindi se $ABL(Z) < ABL(T)$ anche

$$ABL(Z') < ABL(T')$$

contro l'ottimalità di T ' !

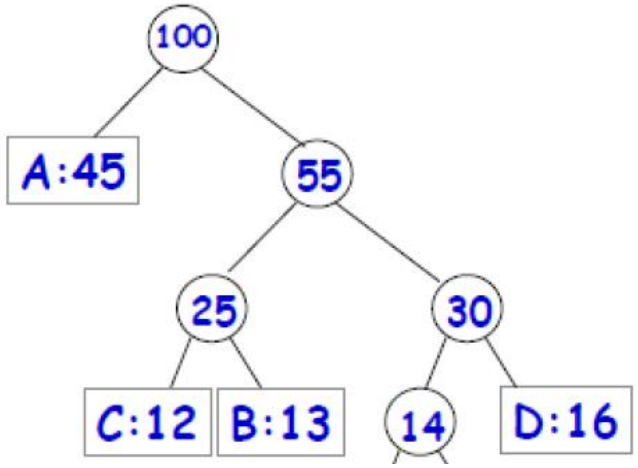
Esempio che ABL(T) = ABL(T')+f(w)

T



y*= F, z*= E

T'



w

con f(w) = 0,05 + 0,09 = 0,14

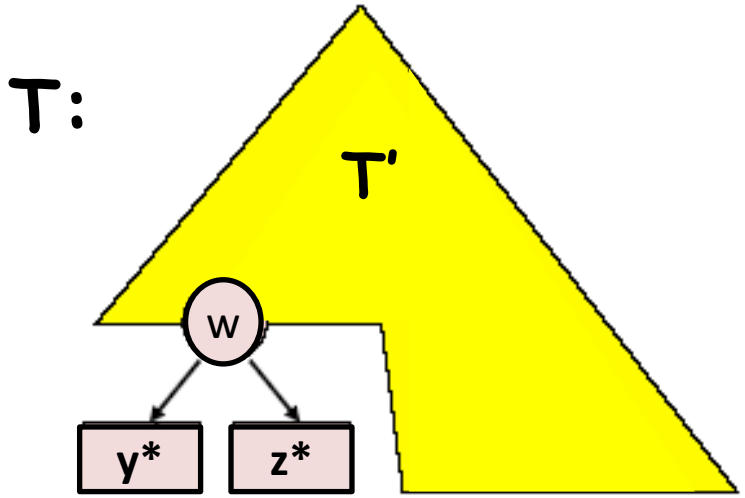
ABL(T') = (45X1 + 12X3 + 13X3 + 16X3)/100 + (14X3)/100 = 2,10

ABL(T) = (45X1 + 12X3 + 13X3 + 16X3)/100 + (5X4 + 9X4)/100 = 2,24

ABL(T) = 2,24 = 2,10 + 0,14 = ABL(T') + f(w)

Codici di Huffman: sottostruttura ottima

Lemma: Siano T e T ' come prima.
Allora:
ABL(T) = ABL (T ') + f(w)



Prova.

$$\begin{aligned}
\mathrm{ABL}(T) &= \sum_{x \in S} f_x \cdot \mathrm{depth}_T(x) \\
&= f_{y^*} \cdot \mathrm{depth}_T(y^*) + f_{z^*} \cdot \mathrm{depth}_T(z^*) + \sum_{x \neq y^*, z^*} f_x \cdot \mathrm{depth}_T(x) \\
&= (f_{y^*} + f_{z^*}) \cdot (1 + \mathrm{depth}_{T'}(\omega)) + \sum_{x \neq y^*, z^*} f_x \cdot \mathrm{depth}_{T'}(x) \\
&= f_\omega \cdot (1 + \mathrm{depth}_{T'}(\omega)) + \sum_{x \neq y^*, z^*} f_x \cdot \mathrm{depth}_{T'}(x) \\
&= f_\omega + f_\omega \cdot \mathrm{depth}_{T'}(\omega) + \sum_{x \neq y^*, z^*} f_x \cdot \mathrm{depth}_{T'}(x) \\
&= f_\omega + \sum_{x \in S'} f_x \cdot \mathrm{depth}_{T'}(x) \\
&= f_\omega + \mathrm{ABL}(T'). \quad \blacksquare
\end{aligned}$$

Esercizio

a) Sia C = $\{c_1, c_2, \ldots, c_n\}$ un insieme di caratteri ad ognuno dei quali é associata una frequenza $f(c_1), f(c_2), \ldots, f(c_n)$. Sia $\gamma : C \rightarrow \{0, 1\}^*$ una codifica binaria di C.

Definire cos'é la lunghezza media per bit di γ, ABL(γ), dall'inglese *Average Bit Length*.

b) Sia adesso C={ a,b,c,d,e, g} con le seguenti frequenze:

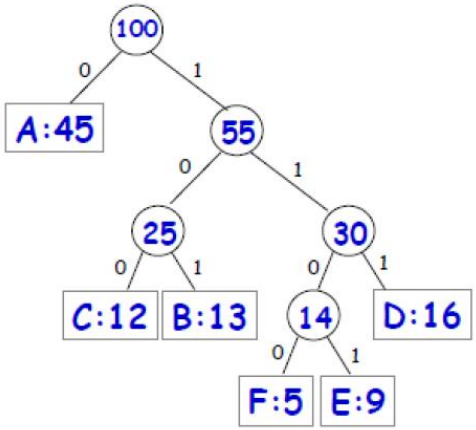
f[a]= 25, f[b]= 28, f[c]= 7, f[d]= 10, f[e]= 26, f[g]= 4.

b1) Descrivere una codifica binaria γ_1 di C a lunghezza fissa 3 e calcolare ABL(γ_1).

b2) Descrivere una codifica binaria γ_2 di C che minimizzi la lunghezza media per bit (utilizzando l'algoritmo studiato) e calcolare ABL(γ_2).

Seconda prova intercorso 31 maggio 2018

Quesito 2 (22 punti) (*Algoritmo di Huffman*)
Sia T il seguente albero risultante dall'esecuzione dell'algoritmo di Huffman sull'alfabeto {A, B, C, D, E, F} con frequenze: f(A)=45, f(B)=13, f(C)=12, f(D)=16, f(E)=9, f(F)=5.



a) Indicare la codifica binaria γ associata a T.
b) Calcolare la lunghezza media per bit di γ, ovvero ABL(T).
c) Si calcoli adesso la somma delle frequenze dei nodi **interni** di T. Quale proprietà osservate?
d) Dimostrare che la proprietà osservata al punto c) vale sempre, cioè per qualsiasi alfabeto finito di simboli con frequenze associate e per ogni albero ottimale costruito con l'algoritmo di Huffman.
Suggerimento: Usare la proprietà che esprime ricorsivamente il valore ABL(T) rispetto al valore ABL(T') relativo all'albero T' richiamato dall'algoritmo di Huffman nel passo ricorsivo.