# TECNICA GREEDY: CODICI DI HUFFMAN

[Cormen] cap. 16

Sezione 16.3

# Codifica di caratteri

Una **codifica** (di caratteri/simboli) associa un **insieme di caratteri/simboli** (l'alfabeto) ad un insieme di altri elementi, denominati **parole in codice** (nel caso del computer, sono **sequenze di bit**).

Ad esempio, la codifica ASCII associa dei simboli alfanumerici a sequenze di 7 bit (e.g., il carattere 'a' è rappresentato in ASCII come 110 0001)

La codifica di caratteri può essere generalizzata alla codifica di un intero testo, tramite la **concatenazione** delle singole codifiche.

(e.g., 'ab' è rappresentato in ASCII come 110 0001 110 0010)

| Binary | Oct | Dec | Hex | Glyph |
|---|---|---|---|---|
| 010 0000 | 040 | 32 | 20 | |
| 010 0001 | 041 | 33 | 21 | ! |
| 010 0010 | 042 | 34 | 22 | " |
| 010 0011 | 043 | 35 | 23 | # |
| 010 0100 | 044 | 36 | 24 | $ |
| 010 0101 | 045 | 37 | 25 | % |
| 010 0110 | 046 | 38 | 26 | & |
| 010 0111 | 047 | 39 | 27 | ' |
| 010 1000 | 050 | 40 | 28 | ( |
| 010 1001 | 051 | 41 | 29 | ) |
| 010 1010 | 052 | 42 | 2A | * |
| 010 1011 | 053 | 43 | 2B | + |
| 010 1100 | 054 | 44 | 2C | , |
| 010 1101 | 055 | 45 | 2D | - |
| 010 1110 | 056 | 46 | 2E | . |
| 010 1111 | 057 | 47 | 2F | / |
| 011 0000 | 060 | 48 | 30 | 0 |
| 011 0001 | 061 | 49 | 31 | 1 |
| 011 0010 | 062 | 50 | 32 | 2 |
| 011 0011 | 063 | 51 | 33 | 3 |
| 011 0100 | 064 | 52 | 34 | 4 |
| 011 0101 | 065 | 53 | 35 | 5 |
| 011 0110 | 066 | 54 | 36 | 6 |
| 011 0111 | 067 | 55 | 37 | 7 |
| 011 1000 | 070 | 56 | 38 | 8 |
| 011 1001 | 071 | 57 | 39 | 9 |
| 011 1010 | 072 | 58 | 3A | : |
| 011 1011 | 073 | 59 | 3B | ; |
| 011 1100 | 074 | 60 | 3C | < |
| 011 1101 | 075 | 61 | 3D | = |
| 011 1110 | 076 | 62 | 3E | > |
| 011 1111 | 077 | 63 | 3F | ? |

| Binary | Oct | Dec | Hex | Glyph |
|---|---|---|---|---|
| 100 0000 | 100 | 64 | 40 | @ |
| 100 0001 | 101 | 65 | 41 | A |
| 100 0010 | 102 | 66 | 42 | B |
| 100 0011 | 103 | 67 | 43 | C |
| 100 0100 | 104 | 68 | 44 | D |
| 100 0101 | 105 | 69 | 45 | E |
| 100 0110 | 106 | 70 | 46 | F |
| 100 0111 | 107 | 71 | 47 | G |
| 100 1000 | 110 | 72 | 48 | H |
| 100 1001 | 111 | 73 | 49 | I |
| 100 1010 | 112 | 74 | 4A | J |
| 100 1011 | 113 | 75 | 4B | K |
| 100 1100 | 114 | 76 | 4C | L |
| 100 1101 | 115 | 77 | 4D | M |
| 100 1110 | 116 | 78 | 4E | N |
| 100 1111 | 117 | 79 | 4F | O |
| 101 0000 | 120 | 80 | 50 | P |
| 101 0001 | 121 | 81 | 51 | Q |
| 101 0010 | 122 | 82 | 52 | R |
| 101 0011 | 123 | 83 | 53 | S |
| 101 0100 | 124 | 84 | 54 | T |
| 101 0101 | 125 | 85 | 55 | U |
| 101 0110 | 126 | 86 | 56 | V |
| 101 0111 | 127 | 87 | 57 | W |
| 101 1000 | 130 | 88 | 58 | X |
| 101 1001 | 131 | 89 | 59 | Y |
| 101 1010 | 132 | 90 | 5A | Z |
| 101 1011 | 133 | 91 | 5B | [ |
| 101 1100 | 134 | 92 | 5C | \ |
| 101 1101 | 135 | 93 | 5D | ] |
| 101 1110 | 136 | 94 | 5E | ^ |
| 101 1111 | 137 | 95 | 5F | _ |

| Binary | Oct | Dec | Hex | Glyph |
|---|---|---|---|---|
| 110 0000 | 140 | 96 | 60 | ` |
| 110 0001 | 141 | 97 | 61 | a |
| 110 0010 | 142 | 98 | 62 | b |
| 110 0011 | 143 | 99 | 63 | c |
| 110 0100 | 144 | 100 | 64 | d |
| 110 0101 | 145 | 101 | 65 | e |
| 110 0110 | 146 | 102 | 66 | f |
| 110 0111 | 147 | 103 | 67 | g |
| 110 1000 | 150 | 104 | 68 | h |
| 110 1001 | 151 | 105 | 69 | i |
| 110 1010 | 152 | 106 | 6A | j |
| 110 1011 | 153 | 107 | 6B | k |
| 110 1100 | 154 | 108 | 6C | l |
| 110 1101 | 155 | 109 | 6D | m |
| 110 1110 | 156 | 110 | 6E | n |
| 110 1111 | 157 | 111 | 6F | o |
| 111 0000 | 160 | 112 | 70 | p |
| 111 0001 | 161 | 113 | 71 | q |
| 111 0010 | 162 | 114 | 72 | r |
| 111 0011 | 163 | 115 | 73 | s |
| 111 0100 | 164 | 116 | 74 | t |
| 111 0101 | 165 | 117 | 75 | u |
| 111 0110 | 166 | 118 | 76 | v |
| 111 0111 | 167 | 119 | 77 | w |
| 111 1000 | 170 | 120 | 78 | x |
| 111 1001 | 171 | 121 | 79 | y |
| 111 1010 | 172 | 122 | 7A | z |
| 111 1011 | 173 | 123 | 7B | { |
| 111 1100 | 174 | 124 | 7C | \| |
| 111 1101 | 175 | 125 | 7D | } |
| 111 1110 | 176 | 126 | 7E | ~ |

# Codici (senza) prefissi

Le codifiche possono avere **lunghezza fissa** (tutti i caratteri/simboli sono codificati da un numero costante di bit, come ASCII) o **variabile**.

Quando codifichiamo delle sequenze di caratteri, affinché la codifica risulti **non ambigua** (cioè nella sequenza di bit sia sempre determinabile dove termina la codifica di un carattere e inizia quello successivo) bisogna che **nessuna codifica di un carattere sia un prefisso di un'altra**, ad esempio (**01 e 010**)

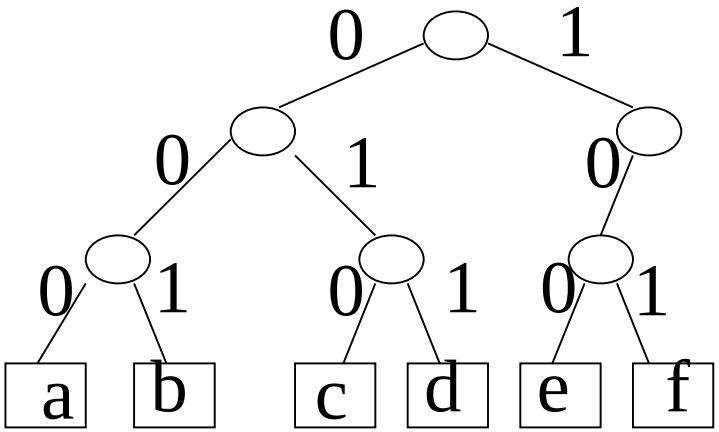
Un codice con questa proprietà si chiama **codice senza prefissi** o **codice prefisso**

Tutte le codifiche a lunghezza fissa sono prefisse, per quelle a lunghezza variabile non è detto.

# Rappresentazione di codici prefissi tramite alberi binari

Un codice binario prefisso può essere rappresentato in modo compatto tramite un **albero binario** in cui **le foglie rappresentano i caratteri** ed i **cammini dalla radice alle foglie rappresentano la codifica** dei caratteri.

L'albero prende il nome di **albero di codifica**.




a: 000, b: 001, c: 010, d: 011, e: 100, f: 101

# Costo di una codifica (di un testo)

Dato un **alfabeto C** e, un testo espresso usando C ed un **albero T di codifica** per C, si chiama **lunghezza media di codifica** o **costo di T**

$$L(T) = \sum_{c \in C} d_c \cdot f(c)$$

dove

**f(c)** è la **frequenza** con cui il carattere c compare nel testo

$d_c$ è il **livello**, quindi la lunghezza in bit della codifica, **del carattere c in T**.

Se **n** è il **numero di caratteri** che compongono il testo con frequenze date dalla funzione f, la lunghezza in bit della codifica del testo è ovviamente data da

$$\text{lunghezza testo codificato} = B(T) = \sum_{c \in C} d_c \cdot n \cdot f(c) = n \cdot L(T)$$

# Il problema: codifica ottima

Il problema affrontato dall'algoritmo di Huffman è il seguente:

Dato un testo scritto secondo un certo alfabeto C, trovare una codifica che sia **minimale**, cioè che renda minima la **lunghezza del testo codificato**.

… o meglio…

Dato l'alfabeto **C** e la funzione di **frequenza f(c)**, tra tutti gli alberi di codifica T per C, trovare quello (o uno di quelli) che **minimizza L(T)**.

La codifica di Huffman è quindi una tecnica di **compressione**, non è un altro modo di fare ciò che fa ASCII!

# Codifica a lunghezza fissa – costo

Una **codifica a lunghezza fissa** usa parole in codice tutte della stessa dimensione (ad es. ASCII).

Con questa codifica servono almeno $\lceil \log_2 n \rceil$ **bit** per rappresentare ogni parola in codice per un alfabeto di n elementi.

**ESEMPIO:** Si consideri un alfabeto di **6 caratteri:** a, b, c, d, e, f.

In un codice a lunghezza fissa servono **3 bit** **(L(T)=3)** per la loro rappresentazione: a: 000, b: 001, c: 010, d: 011, e: 100, f: 101.

Quindi un file di dati di **100.000 caratteri** richiede **300.000 bit** (indipendentemente dalla frequenza di ogni singolo carattere).

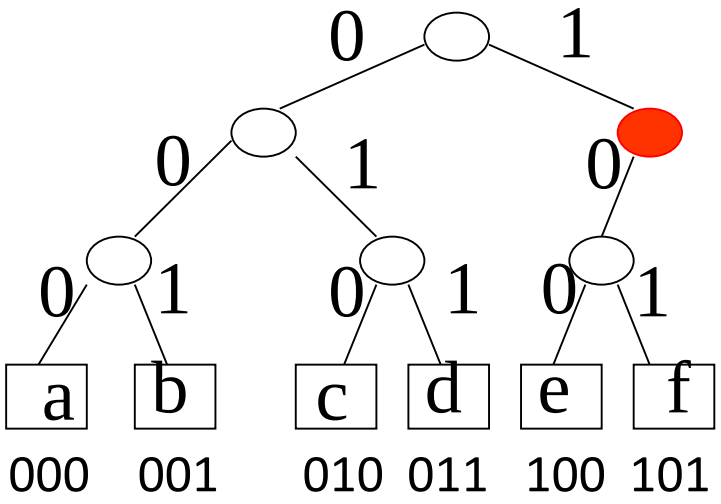
Vogliamo capire se è possibile usare **meno** caratteri.

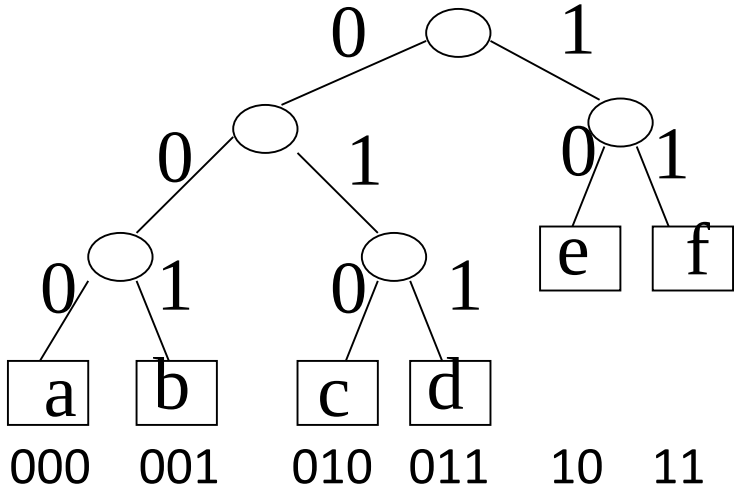
# Alberi Pieni

Un albero avente **nodi interni con un solo figlio non è ottimale**.

Un nodo con un solo figlio, infatti, può venire **eliminato** attaccando i suoi figli **direttamente al padre**.

Un **albero binario** in cui **ogni nodo interno ha esattamente due figli** si chiama albero (binario) **pieno**.



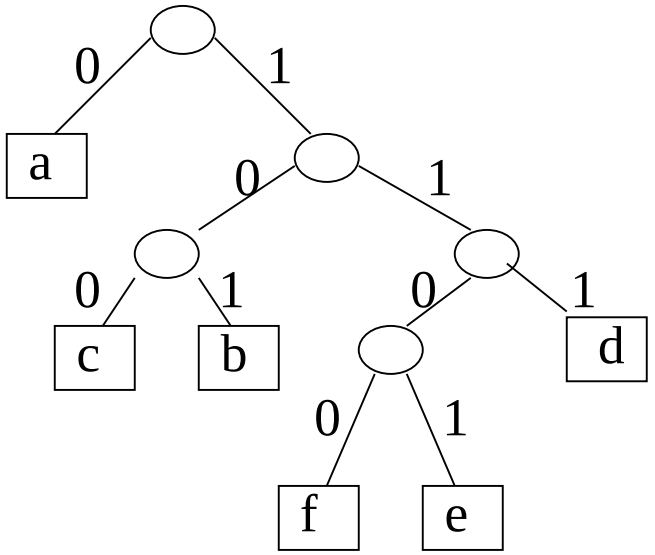
# A chi assegnare le codifiche più corte?

Gli alberi pieni, assegnano ad ogni carattere codifiche di lunghezza diversa. A chi assegno le codifiche più corte e a chi le più lunghe?

Per ottenere la **maggior compressione possibile**, i caratteri più **frequenti** devono avere **le codifiche più corte**, cioè comparire ai **livelli più alti dell'albero**.

**Ad es., per le frequenze:**

| | |
|---|---|
| **a** | **0.45** |
| **d** | **0.16** |
| **b** | **0.13** |
| **c** | **0.12** |
| **e** | **0.09** |
| **f** | **0.05** |



# Codifica a lunghezza variabile – costo

**COME:** Codificare con **meno bit** i simboli che compaiono **più di frequente**, con più bit quelli che compaiono più raramente.

Ciò può permettere un risparmio anche del **25-90%**.

**Esempio.** Supponiamo che nel testo considerato i sei caratteri compaiano con le frequenze sotto indicate, e usiamo la codifica descritta.

| **Caratteri:** | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| **Frequenze:** | 0,45 | 0,13 | 0,12 | 0,16 | 0,09 | 0,05 |
| **Codice l. fissa** | 000 | 001 | 010 | 011 | 100 | 101 |
| **Codice l. variabile** | 0 | 101 | 100 | 111 | 1101 | 1100 |

**Costo lunghezza fissa (L(T)=3):** 3*100.000 = **300.000**

**Costo lunghezza variabile:**

**L(T)=**1*0,45+3*0,13+3*0,12+3*0,16+4*0,09+4*0,05 = **2,24** 2,24*100.000 = **224.000**

# Quindi…

Una codifica ottimale molto probabilmente corrisponderà ad un **albero pieno**, in cui le foglie sono a livelli differenti.

Inoltre, le **foglie più alte** corrisponderanno ai **caratteri più frequenti**.

Ma abbiamo ancora un numero molto alto di soluzioni ammissibili, e non tutte saranno ottime!

Come otteniamo un codice che (dato un testo, o f(c)) abbia la **maggior** compressione possibile?

# Parametri dell'algoritmo di Huffman

**Input:**

- un **alfabeto**, cioè un insieme C di caratteri (distinti);
- una funzione f che dà la **frequenza di ciascun carattere** in un dato testo t o, equivalentemente, il numero di volte num(c, t) in cui ciascun carattere compare nel testo; ovviamente

$$f(c) = num(c, t) / lunghezza(t)$$

dove lunghezza(t) è il numero di caratteri del testo.

**Output:**

un **codice binario ottimo** per la compressione di quel testo t.

# L'algoritmo di Huffman

L’algoritmo di Huffman è un’applicazione della tecnica Greedy con appetibilità modificabili:

1. Per ciascun carattere crea un **albero formato solo da una foglia contenente il carattere e la frequenza del carattere**;

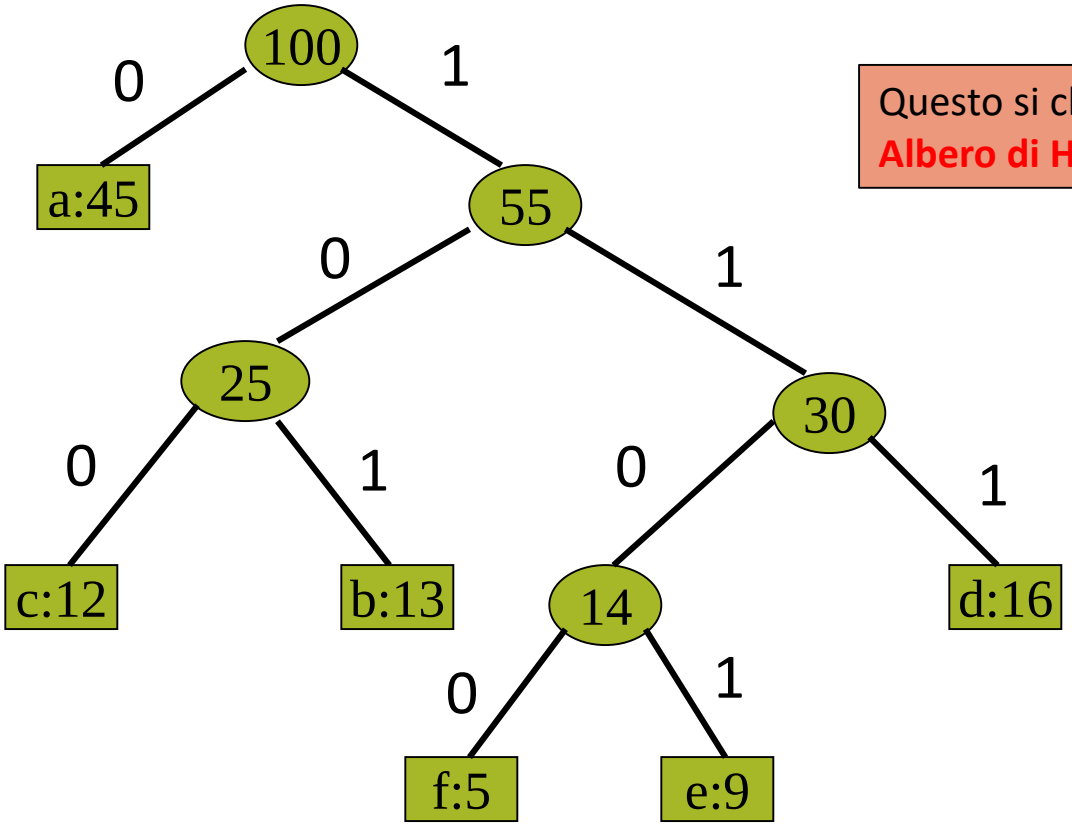
(Appetibilità: considera gli alberi in **ordine non decrescente di frequenza**)

1. Fondi i due alberi che hanno le due **frequenze minime** e costruisci un **nuovo albero** che ha come **frequenza la somma delle frequenze** degli alberi fusi;
2. **Ripeti** la fusione finché si ottiene un unico albero

# Esempio

**caratteri:** a, b, c, d, e, f

**frequenza:** 0.45, 0.13, 0.12, 0.16, 0.09, 0.05




| | |
|---|---|
| a | 0 |
| b | 101 |
| c | 100 |
| d | 111 |
| e | 1101 |
| f | 1100 |

Questo si chiama
**Albero di Huffman**

# Implementazione

Sia **C** un **insieme di |C| caratteri**. Per ogni carattere c, **f[c]** è la **frequenza**.

**Huffman(C, f)**

```
n <- |C|
Q <- empty_priority_queue()
foreach c in C
   enqueue(Q, createTreeNode(c,NULL,NULL), f[c]) // aggiungi c a Q con priorità f[c]
for i = 0; i < n-1; i++
   x <- dequeue_min(Q);
   y <- dequeue_min(Q);
   z <- createTreeNode(null, x, y); // interno: non contiene carattere
   f[z] <- f[x] + f[y];
   enqueue(Q, z, f[z]);
return dequeue_min(Q) // restituisce l'albero ottenuto
```

# Complessità

**Numero di iterazioni:** **O(n)** (esattamente, n-1).

**Costo dequeue_min:** **O(log n)**

**Costo enqueue:** **O(log n)**

**TOT: O(n log n)** se la coda con priorità è realizzata con uno **heap** binario.

(**$O(n^2)$** se la coda con priorità è realizzata con una **struttura con inserimento e/o estrazione** del minimo lineari)

# Correttezza

**DIMOSTRIAMO CHE: L'algoritmo restituisce un albero di Huffman, che rende minima la lunghezza media di codifica L(T).**

(ricorda, $L(T) = \sum_{c \in C} d_c \cdot f(c)$)

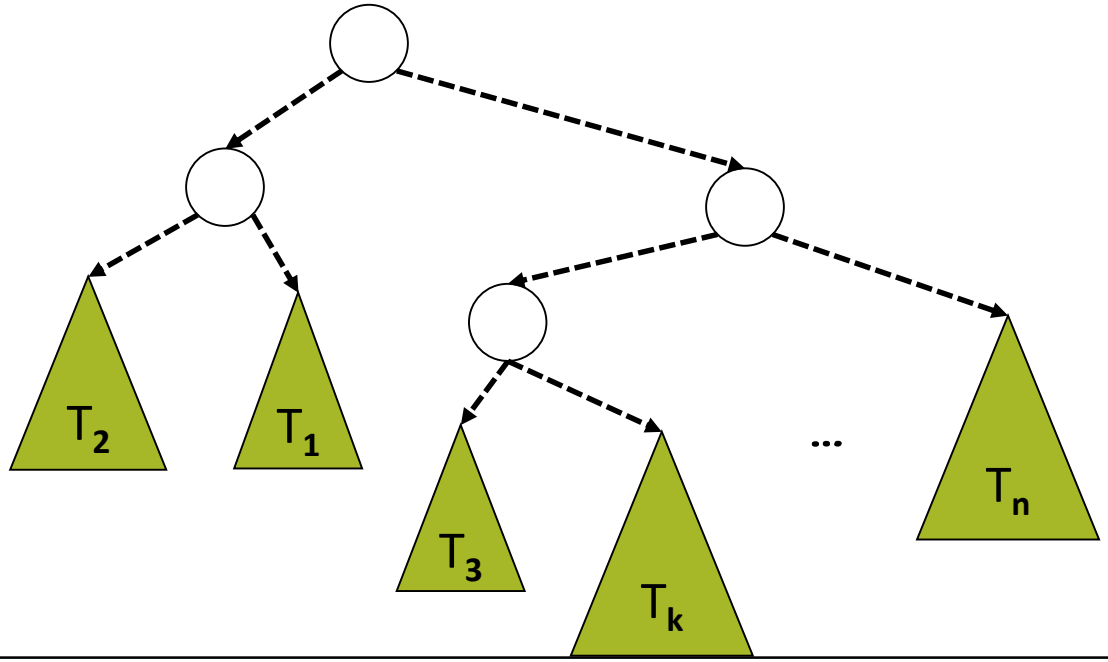
**COME:** per induzione. Partiamo dall'ipotesi (invariante) che ciò che è mantenuto dall'algoritmo può essere completato in un albero di Huffman T. Dimostriamo che se vale tale ipotesi, essa continua a valere dopo ogni passo (in particolare, esisterà un albero T''' in cui la foresta di Huffman risultante può essere completata).

# Definizione: Foresta di Huffman

**Foresta di Huffman** per un alfabeto C con funzione frequenza f:

è una foresta i cui elementi **$T_1$, $T_2$, … $T_n$** sono **sottoalberi di un albero di Huffman** T per l'alfabeto C con frequenze f;
cioè una foresta { $T_1$, $T_2$, … $T_n$ } tale che **esiste un albero di Huffman T per C di cui $T_1$, $T_2$, … $T_n$ sono sottoalberi.**

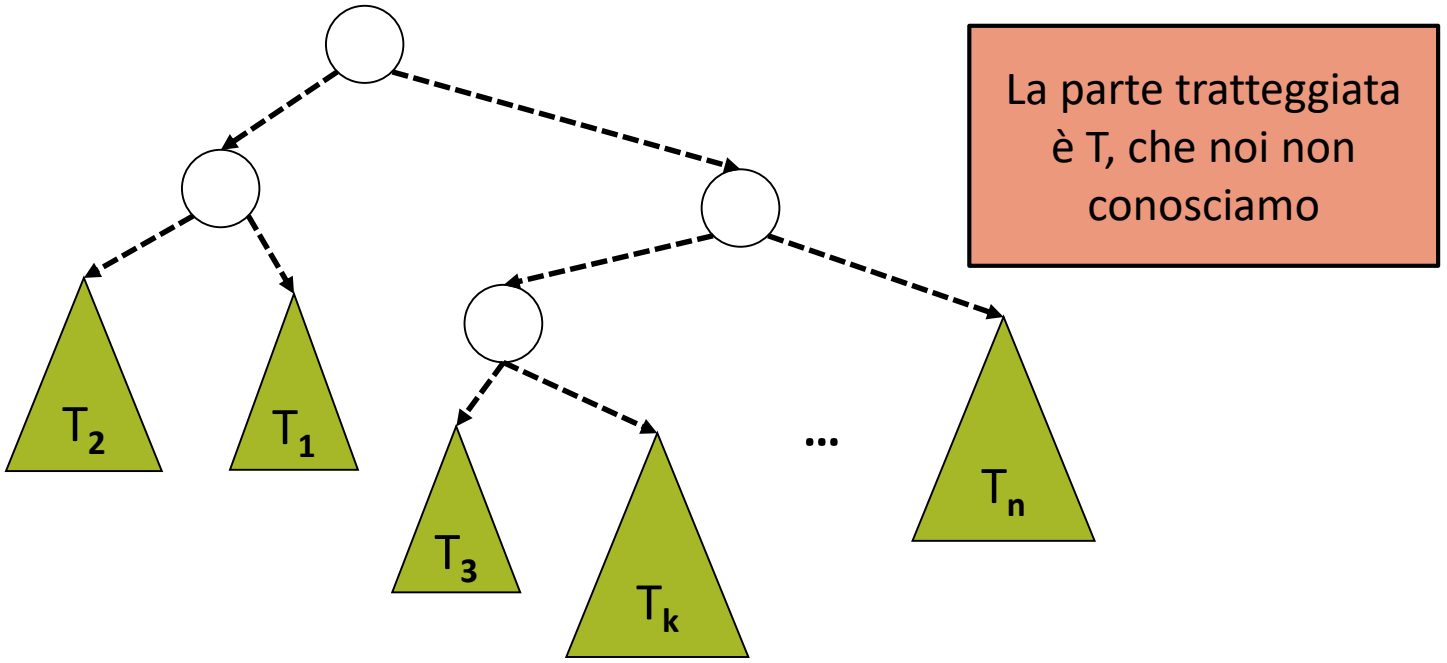
Inoltre le foglie degli alberi della foresta sono **tutti e soli i caratteri di C** (con associate le rispettive frequenze).



# Invariante di Ciclo

La foresta $\{T_1, T_2, \ldots T_n\}$ costruita dall'algoritmo al generico passo è una **foresta di Huffman** per C ed f;

cioè **esiste un albero di Huffman** T (che non conosciamo) di cui gli alberi $T_1, T_2, \ldots T_n$ **sono sottoalberi**, e le foglie di tali alberi sono tutti e soli i caratteri di C.



# Base dell'induzione

Nell'istante immediatamente **precedente l'esecuzione del ciclo l'invariante vale banalmente**.

Gli alberi sono infatti tutti costituiti da nodi singoli, cioè da **foglie corrispondenti ai caratteri dell'alfabeto C**.

Ovviamente essi sono sottoalberi banali di **qualunque albero di Huffman per C.**

# Passo induttivo

Assumiamo che **prima della k-esima iterazione** del ciclo l'invariante valga, cioè che **la foresta F = {$T_1$, $T_2$, … $T_n$} sia una foresta di Huffman** per il dato alfabeto C.

Mostriamo che **dopo il (k+1)-esimo passo dell'iterazione**, che **fonde due alberi** $T_a$ e $T_b$ aventi (le) due **frequenze minime** in nuovo albero $T_{ab}$, **la nuova foresta F – {$T_a$ , $T_b$ } U {$T_{ab}$} è ancora una foresta di Huffman**.

# Dimostrazione del passo induttivo

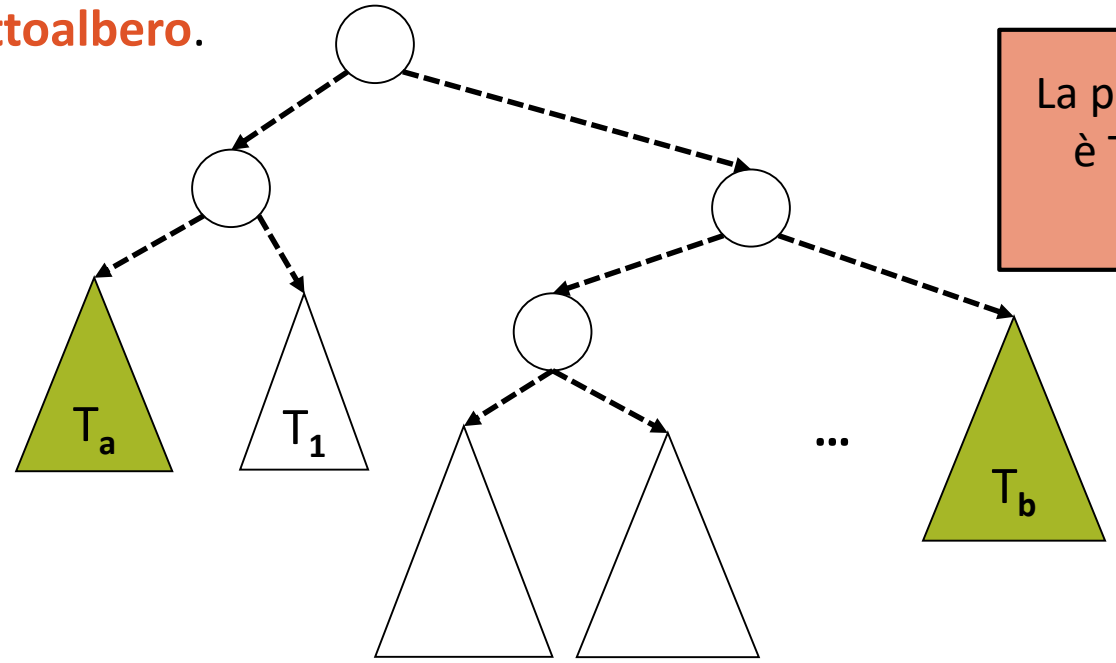
In altre parole, mostriamo che **fra tutti gli alberi di codifica** di cui $T_1$, … $T_n$ sono sottoalberi, vi è un albero (T''') la cui **lunghezza media di codifica L(T''')** è **minima** e di cui **$T_{ab}$ è un sottoalbero** (cioè $T_a$ e $T_b$ sono fratelli).

Quindi l'albero **$T_{ab}$ può essere inserito nella foresta al posto di $T_a$ e $T_b$**: la foresta risultante **$F - \{T_a , T_b \} \cup \{T_{ab}\}$ è** ancora **una foresta di Huffman**.

# Dimostrazione del passo induttivo

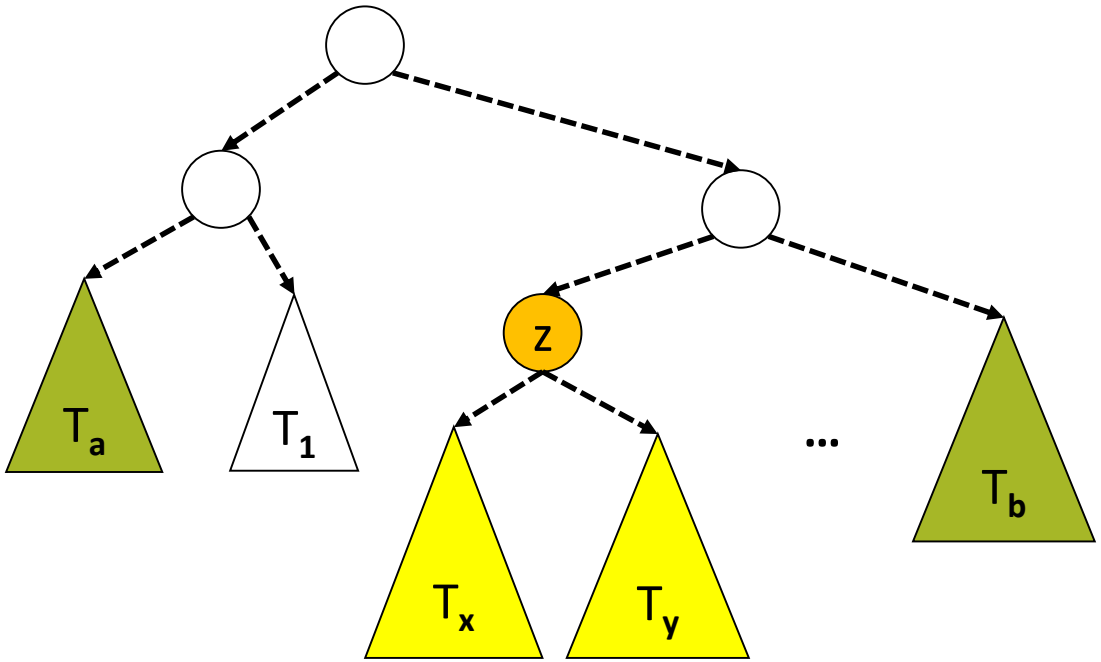
Per **ipotesi induttiva** esiste un albero di Huffman **T'** di cui gli alberi **$T_1$, …, $T_a$ , …, $T_b$ … $T_n$** sono **sottoalberi**, dove $T_a$ e $T_b$ sono, nella foresta al passo considerato, i due alberi di **frequenze minime** (o "pesi" minimi) $f(T_a)$ e $f(T_b)$.

Mostriamo che allora **esiste un albero di Huffman T''' avente $T_{ab}$ come sottoalbero**.




La parte tratteggiata è T', che noi non conosciamo

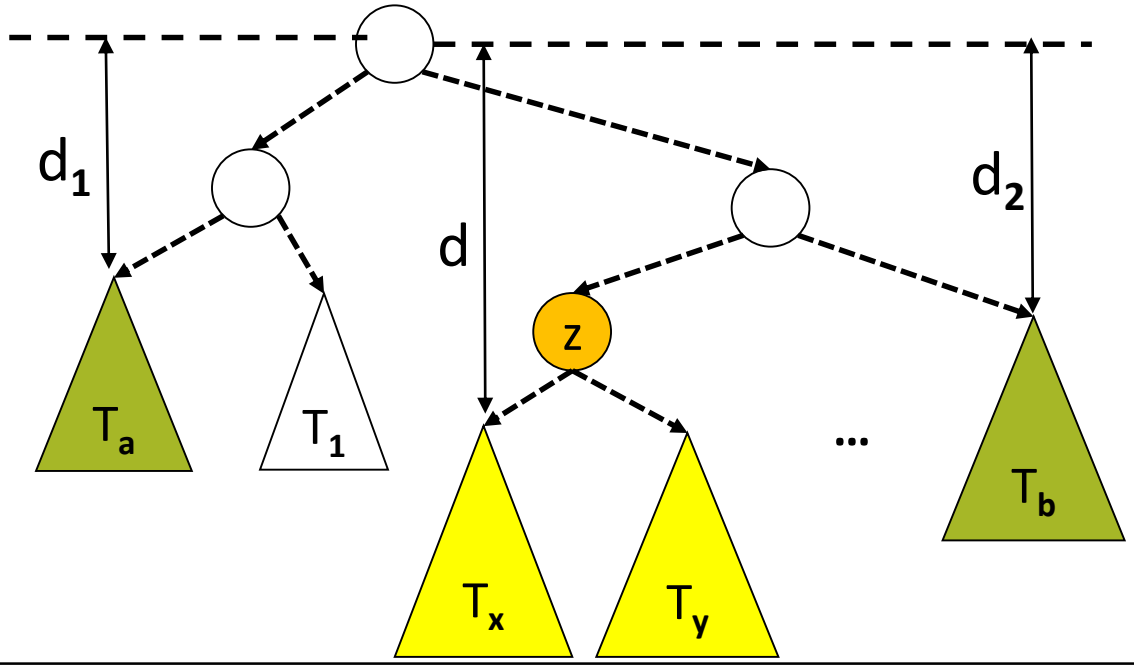
Consideriamo, fra i nodi (interni) di T' **non appartenenti alla foresta F**, cioè fra i nodi di T' che nel passo considerato non sono ancora stati creati, **quello** (o uno di quelli) di **profondità massima**. Sia esso z. Come ogni nodo interno, z **deve avere due sottoalberi-figli non nulli**, siano $T_x$ e $T_y$.



Poiché $T_a$ e $T_b$ sono, al passo considerato, i due alberi di pesi minimi, assumendo $f(T_a) <= f(T_b)$ e $f(T_x) <= f(T_y)$ (solo per ordinarli tra di loro) abbiamo:

$$f(T_a) <= f(T_x) \text{ e } f(T_b) <= f(T_y)$$

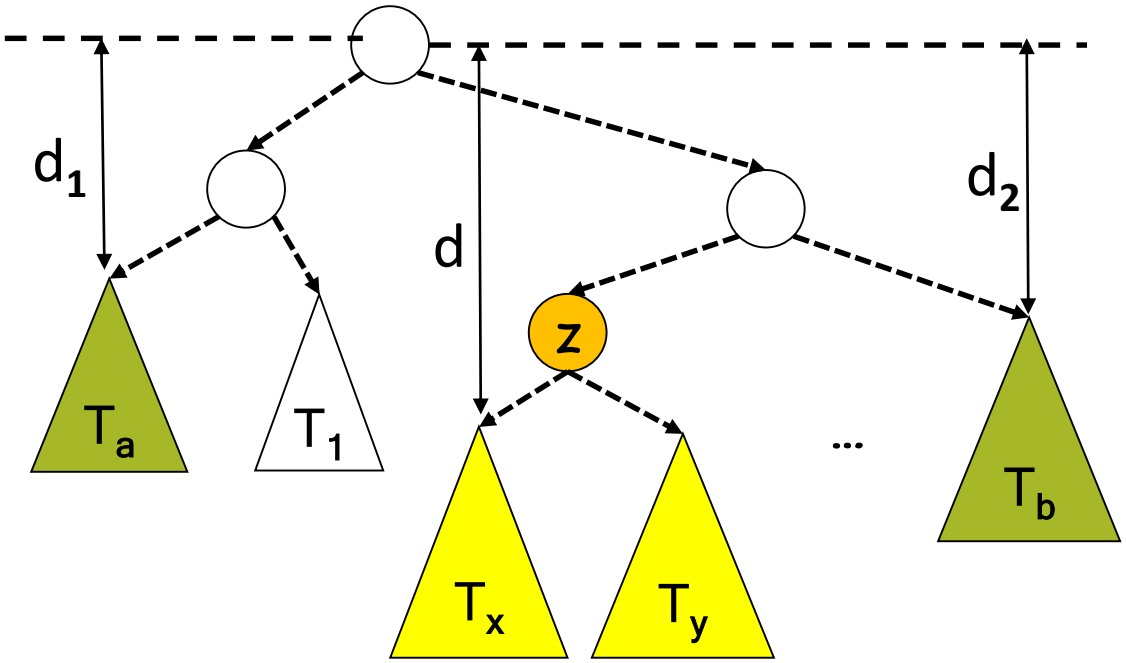
Poiché z è un nodo di profondità massima (fra quelli di T' non ancora creati), le radici degli alberi $T_x$ e $T_y$ si trovano in T' a profondità d non inferiore a quelle di $T_a$ e $T_b$ , siano $d_1$ e $d_2$.



Dunque:

$f(T_a) <= f(T_x)$ $f(T_b) <= f(T_y)$

$d_1 <= d$ $d_2 <= d$



**Scambiamo** di posizione $T_a$ e $T_x$ : otteniamo un albero **T''** di **lunghezza media non superiore**.

Ricordando $L(T) = \sum_{c \in C} d_c \cdot f(c)$ si ha infatti:

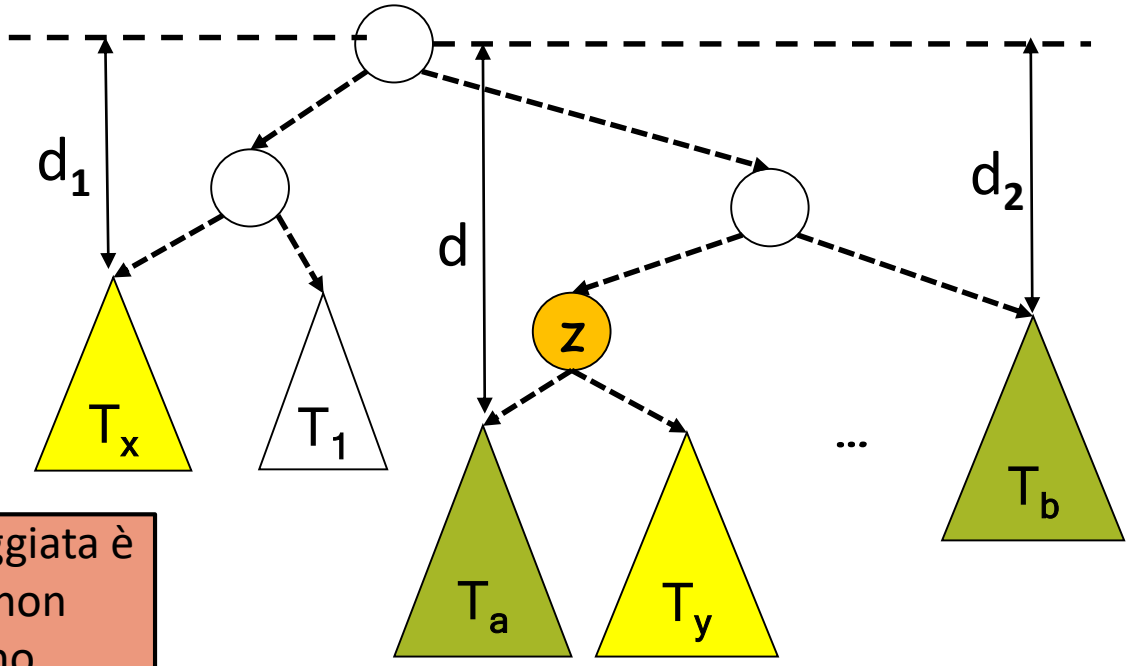
$$L(T'') = L(T') - d_1 f(T_a) - d\, f(T_x) + d_1 f(T_x) + d\, f(T_a)$$

$$= L(T') - d_1 (f(T_a) - f(T_x)) + d(f(T_a) - f(T_x))$$

$$= L(T') + (f(T_a) - f(T_x))\,(d - d_1)$$

ma $f(T_a) - f(T_x) <= 0$ e $d - d_1 >= 0$.

quindi $L(T') + (f(T_a) - f(T_x))\,(d - d_1) <= L(T')$



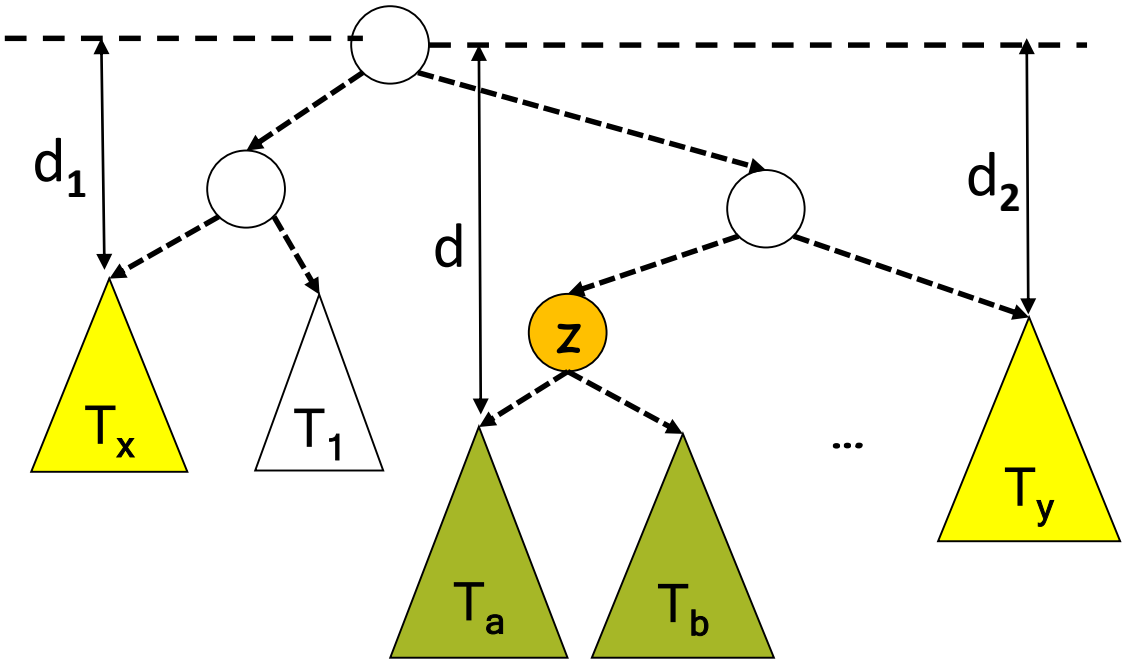

La parte tratteggiata è T'', che noi non conosciamo

Analogamente, scambiando di posizione $T_b$ e $T_y$ otteniamo un albero T''' di lunghezza media non superiore a quella di T''.

Si ha dunque:

$$L(T''') <= L(T'') <= L(T')$$

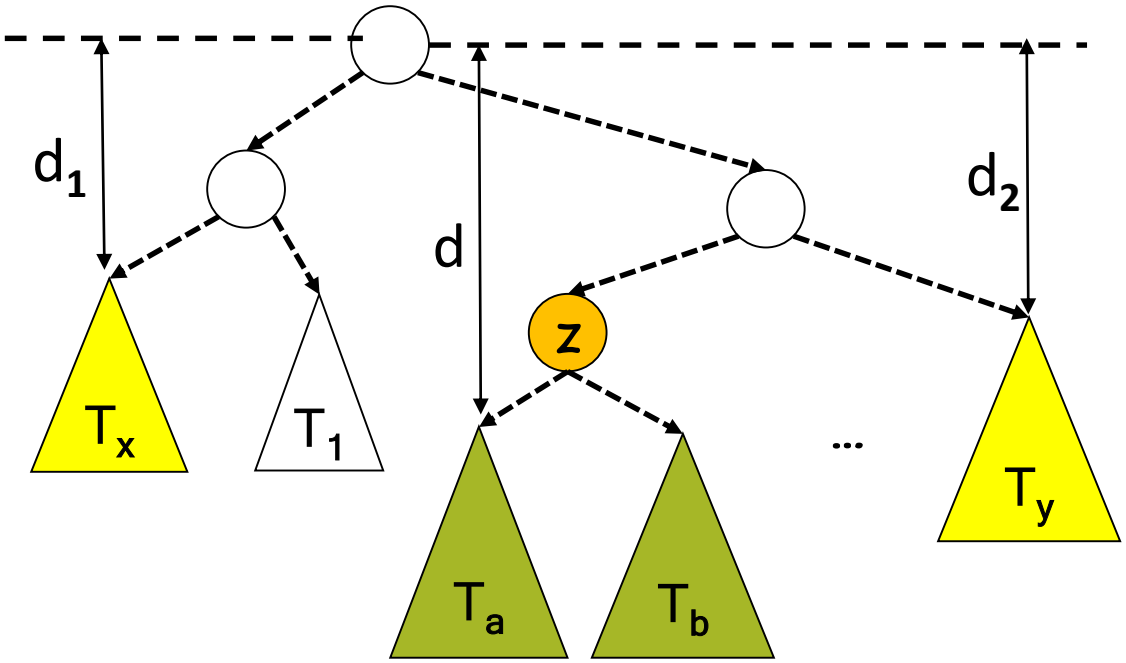
La parte tratteggiata è T''', che noi non conosciamo



Ma **T' è un albero di Huffman**, cioè avente **L(T') minimo**.

Quindi deve essere **L(T''') = L(T')**, e anche T''', che ha $T_{ab}$ come sottoalbero, **è un albero di Huffman**.

Dunque **la foresta F – {$T_a$ , $T_b$ } U {$T_{ab}$}** , ottenuta al k+1-esimo passo di iterazione, è "completabile" in un albero di Huffman, cioè **è ancora una foresta di Huffman**, CVD.



# Cosa devo aver capito fino ad ora

- Codifica di caratteri in sequenze di bit
- Codifica a lunghezza fissa e variabile
- Codifiche ottime
- Algoritmo di Huffman per trovare una codifica ottima
- Correttezza dell'algoritmo di Huffman

# …se non ho capito qualcosa

- Alzo la mano e chiedo
- Ripasso sul libro
- Chiedo aiuto sul forum
- Chiedo o mando una mail al docente