Lezione 20

Sommario della lezione

Applicazione della tecnica Greedy a:

- Compressione Dati

Compressione Dati

Informalmente, il problema è il seguente:
Abbiamo una stringa X su di un dato alfabeto

Compressione Dati

Informalmente, il problema è il seguente:
Abbiamo una stringa X su di un dato alfabeto (ad es., quello della lingua italiana),

Compressione Dati

Informalmente, il problema è 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 più corta possibile* ("comprimere X").

Compressione Dati

Informalmente, il problema è 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 più corta possibile* ("comprimere X").

Perchè lo vogliamo fare?

Compressione Dati

Informalmente, il problema è 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 più corta possibile* ("comprimere X").

Perchè lo vogliamo fare?

1. Per risparmiare spazio di memoria (ad es. per memorizzare documenti di grandi dimensioni in memorie di capacità limitata).

Compressione Dati

Informalmente, il problema è 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 più corta possibile* ("comprimere X").

Perchè lo vogliamo fare?

1. Per risparmiare spazio di memoria (ad es. per memorizzare documenti di grandi dimensioni in memorie di capacità limitata).
2. Per ridurre il tempo di trasmissione dati su canali di banda limitata (ad es., modem lenti o connessioni wireless)

Compressione Dati

Informalmente, il problema è 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 più corta possibile* ("comprimere X").

Perchè lo vogliamo fare?

1. Per risparmiare spazio di memoria (ad es. per memorizzare documenti di grandi dimensioni in memorie di capacità limitata).
2. Per ridurre il tempo di trasmissione dati su canali di banda limitata (ad es., modem lenti o connessioni wireless)
3. ...

I metodi di codifica dati standard, come il sistema ASCII o Unicode, usano stringhe binarie di lunghezza fissa (*blocchi*) per codificare caratteri

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).

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 così via...

Nel sistema di codifica che vedremo (detta di Huffman) ogni simbolo viene codificato con una sequenza binaria di lunghezza *variabile*.

Nel sistema di codifica che vedremo (detta di Huffman) ogni simbolo viene codificato con una sequenza binaria di lunghezza *variabile*.

La codifica dipende dalle *frequenze* di apparizione dei caratteri nella sequenza da codificare X.

Nel sistema di codifica che vedremo (detta di Huffman) ogni simbolo viene codificato con una sequenza binaria di lunghezza *variabile*.

La codifica dipende dalle *frequenze* di apparizione dei caratteri nella sequenza da codificare X.

Quindi, occorre prima calcolare (o già conoscere per altra via) il valore $f(c) =$ numero di volte che il generico carattere c appare in X, per ogni carattere c in X.

Nel sistema di codifica che vedremo (detta di Huffman) ogni simbolo viene codificato con una sequenza binaria di lunghezza *variabile*.

La codifica dipende dalle *frequenze* di apparizione dei caratteri nella sequenza da codificare X.

Quindi, occorre prima calcolare (o già conoscere per altra via) il valore $f(c) =$ numero di volte che il generico carattere c appare in X, per ogni carattere c in X.

L'idea alla base della codifica di Huffman è le seguente: Per usare meno spazio delle codifiche a lunghezza fissa, nella codifica di Huffman si useranno stringhe "corte" per codificare caratteri con frequenze grandi,

Nel sistema di codifica che vedremo (detta di Huffman) ogni simbolo viene codificato con una sequenza binaria di lunghezza *variabile*.

La codifica dipende dalle *frequenze* di apparizione dei caratteri nella sequenza da codificare X.

Quindi, occorre prima calcolare (o già conoscere per altra via) il valore $f(c) =$ numero di volte che il generico carattere c appare in X, per ogni carattere c in X.

L'idea alla base della codifica di Huffman è le seguente: Per usare meno spazio delle codifiche a lunghezza fissa, nella codifica di Huffman si useranno stringhe "corte" per codificare caratteri con frequenze grandi, e stringhe "lunghe" per codificare caratteri con frequenza basse.

Esempio: codifica di: `nel_mezzo_del_cammin_di_nostra_vita`

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

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

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

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

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

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

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

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

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

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			

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 di sopra mostra una possibile codifica a blocchi ed una basata sul codice di Huffman.

Dalla codifica di caratteri alla codifica di testi

Dalla codifica di caratteri alla codifica di testi

- Una volta aver definito come codificare i *singoli* caratteri del testo, si potrà ottenere la codifica dell'intero testo semplicemente *concatenando* (ovvero scrivendo le une appresso alle altre) le codifiche individuali dei caratteri del testo.

Dalla codifica di caratteri alla codifica di testi

- Una volta aver definito come codificare i *singoli* caratteri del testo, si potrà ottenere la codifica dell'intero testo semplicemente *concatenando* (ovvero scrivendo le une appresso alle altre) le codifiche individuali dei caratteri del testo.
- Le stringhe che codificano i caratteri singoli del testo verranno chiamate *parole codice*, e l'insieme di tutte le parole codice verrà chiamato il *codice*.

	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			

	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			

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

	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			

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

Testo:

Codifica a lunghezza fissa:

Codifica di Huffman:

	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			

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

Testo: `n`

Codifica a lunghezza fissa:
1000

Codifica di Huffman:
1011

	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			

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

Testo: `ne`

Codifica a lunghezza fissa:
10000100

Codifica di Huffman:
10111100

	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			

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

Testo: `nel`

Codifica a lunghezza fissa:
100001000110

Codifica di Huffman:
101111000101

	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			

Esempio: codifica di `nel␣mezzo␣del␣cammin␣di␣nostra␣vita`

Testo: `nel␣`

Codifica a lunghezza fissa:
1000010001100000

Codifica di Huffman:
10111100010100

	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			

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

Testo: `nel_ m`

Codifica a lunghezza fissa:
10000100011000000111

Codifica di Huffman:
1011110001010011110

	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			

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

Testo: `nel_ me`

Codifica a lunghezza fissa:
1000010001100000011101000

Codifica di Huffman:
1011110001010011101100

	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			

Esempio: codifica di `nel␣mezzo␣del␣cammin␣di␣nostra␣vita`

Testo: `nel␣ mez`

Codifica a lunghezza fissa:
1000010001100000011101001110

Codifica di Huffman:
10111100010100111011000110

	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			

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

Testo: `nel_ mezz`

Codifica a lunghezza fissa:
1000010001100000011101001110 1110

Codifica di Huffman:
101111000101001110110001100110

	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			

Esempio: codifica di `nel␣mezzo␣del␣cammin␣di␣nostra␣vita`

Testo: `nel␣ mezzo`

Codifica a lunghezza fissa:
1000010001100000011101001110111101001

Codifica di Huffman:
10111100010100111011000110011000100

	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			

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

Testo: `nel_ mezzo_`

Codifica a lunghezza fissa:
10000100011000000111010011101110100100000

Codifica di Huffman:
10111100010100111011000110011001000

	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			

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

Testo: `nel_ mezzo_d`

Codifica a lunghezza fissa:
1000010001100000011101001110111010010000 0011

Codifica di Huffman:
10111100010100111011000110011001000 01010

	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			

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

Testo: `nel_ mezzo_de`

Codifica a lunghezza fissa:
100001000110000001110100111011101001000000110100

Codifica di Huffman:
10111100010100111011000110011001000010101100

	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			

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

Testo: `nel_ mezzo_del`

Codifica a lunghezza fissa:
1000010001100000011101001110111010010000001101000110

Codifica di Huffman:
10111100010100111011000110011001000001010110000101

	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			

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

Testo: `nel_ mezzo_del`. . .

Codifica a lunghezza fissa:
100001000110000001110100111011101001000000110100 0110. . .

Codifica di Huffman:
1011110001010011101100011001100100000101011000101. . .

Per codificare tutto il testo con la codifica a blocchi si useranno 140 bits,

	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			

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

Testo: `nel_ mezzo_del`. . .

Codifica a lunghezza fissa:
1000010001100000011101001110111010010000001101000110. . .

Codifica di Huffman:
10111100010100111011000110011001000001010110000101. . .

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

	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			

Esempio: codifica di `nel_mezzo_del_cammin_di_nostra_vita`

Testo: `nel_ mezzo_del`...

Codifica a lunghezza fissa:
1000010001100000011101001110111010010000001101000110...

Codifica di Huffman:
1011110001010011101100011001100100001010110001 01...

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 può portare a comprimere le dimensioni dei file fino al 50% della dimensione originale.

Se osserviamo la tabella che descrive il codice di Huffman per l'esempio considerato, noteremo che nessuna stringa binaria in esso è inizio (prefisso) di alcun altra.

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Se osserviamo la tabella che descrive il codice di Huffman per l'esempio considerato, noteremo che nessuna stringa binaria in esso è inizio (prefisso) di alcun altra.

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Sistemi di codifica con tale proprietà vengono detti codifiche prefisso (o *codici prefisso*).

Se osserviamo la tabella che descrive il codice di Huffman per l'esempio considerato, noteremo che nessuna stringa binaria in esso è inizio (prefisso) di alcun altra.

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Sistemi di codifica con tale proprietà vengono detti codifiche prefisso (o *codici prefisso*).

In pratica, si usano quasi esclusivamente codici prefisso, per due motivi:

Se osserviamo la tabella che descrive il codice di Huffman per l'esempio considerato, noteremo che nessuna stringa binaria in esso è inizio (prefisso) di alcun altra.

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Sistemi di codifica con tale proprietà vengono detti codifiche prefisso (o *codici prefisso*).

In pratica, si usano quasi esclusivamente codici prefisso, per due motivi:

1. I codici prefisso ammettono semplici algoritmi di codifica/decodifica, ed ammettono semplici ed efficienti rappresentazioni.

Se osserviamo la tabella che descrive il codice di Huffman per l'esempio considerato, noteremo che nessuna stringa binaria in esso è inizio (prefisso) di alcun altra.

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Sistemi di codifica con tale proprietà vengono detti codifiche prefisso (o *codici prefisso*).

In pratica, si usano quasi esclusivamente codici prefisso, per due motivi:

1. I codici prefisso ammettono semplici algoritmi di codifica/decodifica, ed ammettono semplici ed efficienti rappresentazioni.

2. Per ottenere la migliore compressione possibile, la restrizione a codici prefisso *non è* una limitazione.

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
1011110001010011101100011001100100001010110001 01

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
1011110001010011101100011001100100001010110001 01

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
101111000101001110110001100110010000101011000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la **codifica** di qualche carattere:

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
101111000101001110110001100110010000101011000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
101111000101001110110001100110010000101011000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
101111000101001110110001100110010000101011000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
101111000101001110110001100110010000101011000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:
`n`

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
1011**1**1000101001110110001100110010000101011000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:
n

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
␣	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
1011**11**000101001110110001100110010000101011000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:
n

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
1011**110**00101001110110001100110010000101011000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:
n

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
1011**1100**0101001110110001100110010000101011000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:
n

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
1011**1100**010100111011000110011001000010101 1000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:
`ne`

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
10111100**0**101001110110001100110010000101011000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:
`ne`

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
10111100**01**010011101100011001100100001010110000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:
`ne`

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
10111100**010**10011101100011001100100001010111000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:
ne

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
10111100**0101**0011101100011001100100001010110001 01

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:
ne

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
10111100**0101**0011101100011001100100001010110000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:
`nel`

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
101111000101**0**01110110001100110010000101011000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:
`nel`

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
101111000101**00**111011000110011001000010101100010

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:
`nel`

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
_	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
101111000101**00**1110110001100110010000101011000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:
`nel_`

Esempio: decodifica

car.	codifica	car.	codifica	car	codifica
␣	00	a	1101	c	10001
d	1010	e	1100	i	1111
l	0101	m	1110	n	1011
o	0100	r	10000	s	10011
t	0111	v	10010	z	0110

Vogliamo decodificare la stringa binaria:
101111000101001110110001100110010000101011000101

Leggeremo la stringa sequenzialmente da sinistra a destra, fin quando non individuiamo la codifica di qualche carattere:
`nel␣`...

Ma la proprietà di codice prefisso serve proprio?

Ma la proprietà di codice prefisso serve proprio? Si.

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola:

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011,

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01,

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa “molto lunga” 01111111111111111111111111111111....

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa "molto lunga" 0111111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come 01

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa “molto lunga”
011111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa “molto lunga”
0111111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa “molto lunga” 0111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come 01 11 11 11

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa "molto lunga"
0111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa “molto lunga” 0111111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come 01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa “molto lunga”
01111111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero `b`

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa “molto lunga”
01111111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero `bc`

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa "molto lunga"
0111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero `bcc`

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa “molto lunga”
01111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero `bccc`

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa "molto lunga"
0111111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero `bccccccccccccccc`...)

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri a,b, e c venisse codificato secondo questa regola: il carattere a viene codificato con 011, il carattere b con 01, ed il carattere c con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa "molto lunga"
01111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero bccccccccccccccc...)

oppure la si potrebbe decodificare come
011

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa “molto lunga”
01111111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero `bccccccccccccccc`...)

oppure la si potrebbe decodificare come
011 11

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri a,b, e c venisse codificato secondo questa regola: il carattere a viene codificato con 011, il carattere b con 01, ed il carattere c con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa “molto lunga”
0111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero bccccccccccccccc...)

oppure la si potrebbe decodificare come
011 11 11

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa “molto lunga”
011111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero `bccccccccccccccc`...)

oppure la si potrebbe decodificare come
011 11 11 11

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa "molto lunga"
01111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero `bcccccccccccccc`...)

oppure la si potrebbe decodificare come
011 11 11 11 11

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa "molto lunga"
0111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero bccccccccccccccc...)

oppure la si potrebbe decodificare come
011 11 11 11 11 11

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa “molto lunga”
0111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero `bccccccccccccccc`...)

oppure la si potrebbe decodificare come
011 11 11 11 11 11 11 11 11 11 11 11 11 11 11 1....

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa “molto lunga”
0111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero `bccccccccccccccc`...)

oppure la si potrebbe decodificare come
011 11 11 11 11 11 11 11 11 11 11 11 11 11 11 1....
(ovvero `a`

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa "molto lunga"
0111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero `bccccccccccccccc`...)

oppure la si potrebbe decodificare come
011 11 11 11 11 11 11 11 11 11 11 11 11 11 11 1....
(ovvero `ac`

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`, `b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa “molto lunga”
01111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero `bccccccccccccccc`...)

oppure la si potrebbe decodificare come
011 11 11 11 11 11 11 11 11 11 11 11 11 11 11 1....
(ovvero `acc`

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa "molto lunga"
01111111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero `bccccccccccccccc`...)

oppure la si potrebbe decodificare come
011 11 11 11 11 11 11 11 11 11 11 11 11 11 11 1....
(ovvero `accc`

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri a,b, e c venisse codificato secondo questa regola: il carattere a viene codificato con 011, il carattere b con 01, ed il carattere c con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa “molto lunga”
0111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero bccccccccccccccc...)

oppure la si potrebbe decodificare come
011 11 11 11 11 11 11 11 11 11 11 11 11 11 11 1....
(ovvero acccc

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa "molto lunga"
01111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero `bccccccccccccccc`...)

oppure la si potrebbe decodificare come
011 11 11 11 11 11 11 11 11 11 11 11 11 11 11 1....
(ovvero `acccccccccccccc`...)

Ma la proprietà di codice prefisso serve proprio? Si.

Immaginiamo la situazione in cui un testo composto solo di caratteri `a`,`b`, e `c` venisse codificato secondo questa regola: il carattere `a` viene codificato con 011, il carattere `b` con 01, ed il carattere `c` con la stringa 11.

Il codice *non* è prefisso, in quanto 01 è parte inziale di 011.

Potremmo dover decodificare la stringa "molto lunga"
01111111111111111111111111111111....

Leggendo da sinitra a destra, la stringa si potrebbe decodificare come
01 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11....
(ovvero `bccccccccccccccc`...)

oppure la si potrebbe decodificare come
011 11 11 11 11 11 11 11 11 11 11 11 11 11 11 1....
(ovvero `acccccccccccccc`...)

Questa ambiguità di decodifica della stringa binaria potrà essere risolta solo quando si conoscerà la *fine* della stringa binaria.

In altri termini....

- Con la codifica precedente, non saremmo in grado di stabilire nemmeno chi è il primo carattere del testo, almeno fin quando non si è letto *interamente* (cioè fino alla fine) la sua codifica binaria.

In altri termini....

- Con la codifica precedente, non saremmo in grado di stabilire nemmeno chi è il primo carattere del testo, almeno fin quando non si è letto *interamente* (cioè fino alla fine) la sua codifica binaria.
- Questo è chiaramente inaccettabile dal punto di vista pratico.

In altri termini....

- Con la codifica precedente, non saremmo in grado di stabilire nemmeno chi è il primo carattere del testo, almeno fin quando non si è letto *interamente* (cioè fino alla fine) la sua codifica binaria.
- Questo è chiaramente inaccettabile dal punto di vista pratico.
- I codici prefisso non soffrono di questo problema, in quanto è possibile iniziare la decodifica non appena compare la stringa binaria che codifica un qualche carattere, e poi iterare.

In altri termini....

- Con la codifica precedente, non saremmo in grado di stabilire nemmeno chi è il primo carattere del testo, almeno fin quando non si è letto *interamente* (cioè fino alla fine) la sua codifica binaria.
- Questo è chiaramente inaccettabile dal punto di vista pratico.
- I codici prefisso non soffrono di questo problema, in quanto è possibile iniziare la decodifica non appena compare la stringa binaria che codifica un qualche carattere, e poi iterare.
- La correttezza dell'algoritmo di decodifica segue dal fatto che nessuna codifica di carattere è parte iniziale (prefisso) di alcun altra codifica di altri caratteri.

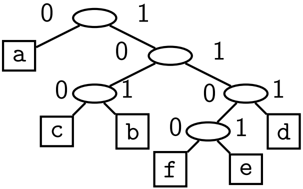
Rappresentazione di codici prefisso

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.

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.

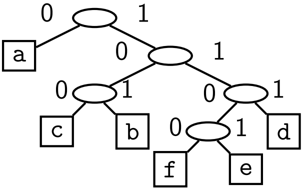
Esempio:



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:

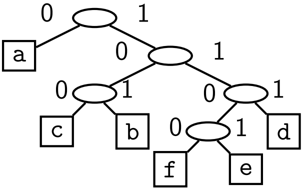


carattere	codifica

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:

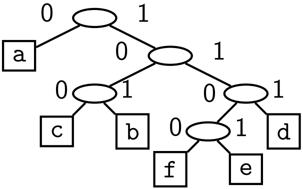


carattere	codifica
a	0

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:

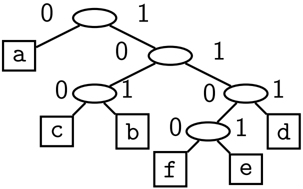


carattere	codifica
a	0
c	100

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:

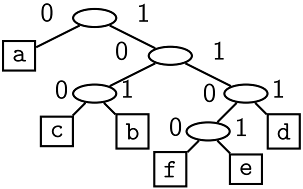


carattere	codifica
a	0
c	100
b	101

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:

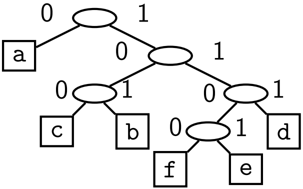


carattere	codifica
a	0
c	100
b	101
d	111

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:

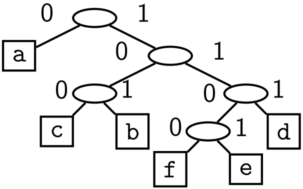


carattere	codifica
a	0
c	100
b	101
d	111
f	1100

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

Riassumendo:
Una codifica (binaria) è rappresentabile da un albero binario in cui:

1. Ogni foglia rappresenta un carattere da codificare.

Riassumendo:
Una codifica (binaria) è rappresentabile da un albero binario in cui:
1. Ogni foglia rappresenta un carattere da codificare.
2. L'arco da un nodo al suo figlio sinistro è etichettato con 0, l'arco da un nodo al suo figlio destro è etichettato con 1.

Riassumendo:
Una codifica (binaria) è rappresentabile da un albero binario in cui:
1. Ogni foglia rappresenta un carattere da codificare.
2. L'arco da un nodo al suo figlio sinistro è etichettato con 0, l'arco da un nodo al suo figlio destro è etichettato con 1.
3. La sequenza di etichette che si leggono su di un percorso radice-foglia, corriponde alla codifica binaria del relativo carattere associato alla foglia.

Riassumendo:
Una codifica (binaria) è rappresentabile da un albero binario in cui:
1. Ogni foglia rappresenta un carattere da codificare.
2. L'arco da un nodo al suo figlio sinistro è etichettato con 0, l'arco da un nodo al suo figlio destro è etichettato con 1.
3. La sequenza di etichette che si leggono su di un percorso radice-foglia, corriponde alla codifica binaria del relativo carattere associato alla foglia.
4. La lunghezza della parola codice associata ad un generico carattere $\mathtt{x}$ è uguale alla lunghezza del cammino dalla radice dell'albero alla foglia associata ad $\mathtt{x}$ (=profondità $d_T(\mathtt{x})$ della foglia)

Riassumendo:
Una codifica (binaria) è rappresentabile da un albero binario in cui:

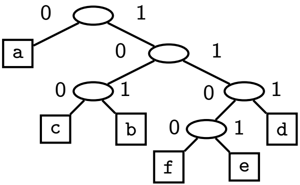
1. Ogni foglia rappresenta un carattere da codificare.

2. L'arco da un nodo al suo figlio sinistro è etichettato con 0, l'arco da un nodo al suo figlio destro è etichettato con 1.

3. La sequenza di etichette che si leggono su di un percorso radice-foglia, corriponde alla codifica binaria del relativo carattere associato alla foglia.

4. La lunghezza della parola codice associata ad un generico carattere `x` è uguale alla lunghezza del cammino dalla radice dell'albero alla foglia associata ad `x` (=profondità $d_T(\mathtt{x})$ della foglia)

Esempio:



Riassumendo:
Una codifica (binaria) è rappresentabile da un albero binario in cui:

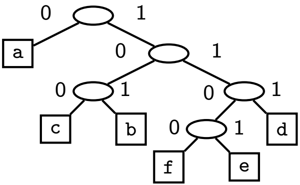
1. Ogni foglia rappresenta un carattere da codificare.

2. L'arco da un nodo al suo figlio sinistro è etichettato con 0, l'arco da un nodo al suo figlio destro è etichettato con 1.

3. La sequenza di etichette che si leggono su di un percorso radice-foglia, corriponde alla codifica binaria del relativo carattere associato alla foglia.

4. La lunghezza della parola codice associata ad un generico carattere `x` è uguale alla lunghezza del cammino dalla radice dell'albero alla foglia associata ad `x` (=profondità $d_T(\mathtt{x})$ della foglia)

Esempio:



La rappresentazione ad albero della codifica è anche molto utile nelle fasi di codifica e decodifica. Ad es., per decodificare la stringa 11100101 basta seguire i relativi percorsi radice-foglia per ottenere il testo `d`

Riassumendo:
Una codifica (binaria) è rappresentabile da un albero binario in cui:

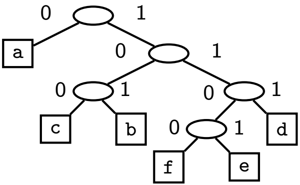
1. Ogni foglia rappresenta un carattere da codificare.

2. L'arco da un nodo al suo figlio sinistro è etichettato con 0, l'arco da un nodo al suo figlio destro è etichettato con 1.

3. La sequenza di etichette che si leggono su di un percorso radice-foglia, corriponde alla codifica binaria del relativo carattere associato alla foglia.

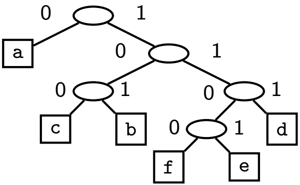
4. La lunghezza della parola codice associata ad un generico carattere `x` è uguale alla lunghezza del cammino dalla radice dell'albero alla foglia associata ad `x` (=profondità $d_T(\mathtt{x})$ della foglia)

Esempio:



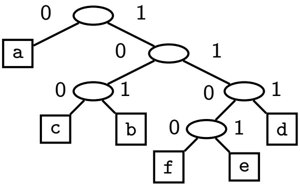
La rappresentazione ad albero della codifica è anche molto utile nelle fasi di codifica e decodifica. Ad es., per decodificare la stringa 11100101 basta seguire i relativi percorsi radice-foglia per ottenere il testo da

Riassumendo:
Una codifica (binaria) è rappresentabile da un albero binario in cui:
1. Ogni foglia rappresenta un carattere da codificare.
2. L'arco da un nodo al suo figlio sinistro è etichettato con 0, l'arco da un nodo al suo figlio destro è etichettato con 1.
3. La sequenza di etichette che si leggono su di un percorso radice-foglia, corriponde alla codifica binaria del relativo carattere associato alla foglia.
4. La lunghezza della parola codice associata ad un generico carattere `x` è uguale alla lunghezza del cammino dalla radice dell'albero alla foglia associata ad `x` (=profondità $d_T(\mathtt{x})$ della foglia)
Esempio:



La rappresentazione ad albero della codifica è anche molto utile nelle fasi di codifica e decodifica. Ad es., per decodificare la stringa 11100101 basta seguire i relativi percorsi radice-foglia per ottenere il testo `daa`

Riassumendo:
Una codifica (binaria) è rappresentabile da un albero binario in cui:
1. Ogni foglia rappresenta un carattere da codificare.
2. L'arco da un nodo al suo figlio sinistro è etichettato con 0, l'arco da un nodo al suo figlio destro è etichettato con 1.
3. La sequenza di etichette che si leggono su di un percorso radice-foglia, corriponde alla codifica binaria del relativo carattere associato alla foglia.
4. La lunghezza della parola codice associata ad un generico carattere `x` è uguale alla lunghezza del cammino dalla radice dell'albero alla foglia associata ad `x` (=profondità $d_T(\mathtt{x})$ della foglia)
Esempio:



La rappresentazione ad albero della codifica è anche molto utile nelle fasi di codifica e decodifica. Ad es., per decodificare la stringa 11100101 basta seguire i relativi percorsi radice-foglia per ottenere il testo daab.

Ritorniamo al nostro problema di partenza, che è :

Data una stringa X su di un alfabeto C di n caratteri (ad es., quello della lingua italiana), vogliamo codificare X in una sequenza binaria Y che sia la più corta possibile.

Ritorniamo al nostro problema di partenza, che è :

Data una stringa X su di un alfabeto C di n caratteri (ad es., quello della lingua italiana), vogliamo codificare X in una sequenza binaria Y che sia la più corta possibile.

Per codificare X supponiamo di usare una data codifica rappresentata da un albero T.

Ritorniamo al nostro problema di partenza, che è :

Data una stringa X su di un alfabeto C di n caratteri (ad es., quello della lingua italiana), vogliamo codificare X in una sequenza binaria Y che sia la più corta possibile.

Per codificare X supponiamo di usare una data codifica rappresentata da un albero T. Quanto sarà la lunghezza $|Y|$ della codifica binaria Y di X che otterremo?

Ritorniamo al nostro problema di partenza, che è :

Data una stringa X su di un alfabeto C di n caratteri (ad es., quello della lingua italiana), vogliamo codificare X in una sequenza binaria Y che sia la più corta possibile.

Per codificare X supponiamo di usare una data codifica rappresentata da un albero T. Quanto sarà la lunghezza $|Y|$ della codifica binaria Y di X che otterremo?

$\forall$ carattere $c \in C$, sia $f(c)$ la frequenza di c in X

Ritorniamo al nostro problema di partenza, che è :

$\boxed{\text{Data una stringa } X \text{ su di un alfabeto } C \text{ di } n \text{ caratteri (ad es., quello della lingua italiana), vogliamo codificare } X \text{ in una sequenza binaria } Y \text{ che sia la più corta possibile.}}$

Per codificare X supponiamo di usare una data codifica rappresentata da un albero T. Quanto sarà la lunghezza $|Y|$ della codifica binaria Y di X che otterremo?

$\forall$ carattere c $\in C$, sia $f(\text{c})$ la frequenza di c in X
Sia $d_T(\text{c})$ la profondità della foglia associata al carattere c nell'albero T (corrispondente alla *lunghezza* della codifica di c).

Ritorniamo al nostro problema di partenza, che è :

Data una stringa X su di un alfabeto C di n caratteri (ad es., quello della lingua italiana), vogliamo codificare X in una sequenza binaria Y che sia la più corta possibile.

Per codificare X supponiamo di usare una data codifica rappresentata da un albero T. Quanto sarà la lunghezza $|Y|$ della codifica binaria Y di X che otterremo?

$\forall$ carattere $c \in C$, sia $f(c)$ la frequenza di c in X
Sia $d_T(c)$ la profondità della foglia associata al carattere c nell'albero T (corrispondente alla *lunghezza* della codifica di c).
Varrà

$$|Y| = \sum_{c \in C} f(c) d_T(c)$$

Ritorniamo al nostro problema di partenza, che è :

Data una stringa X su di un alfabeto C di n caratteri (ad es., quello della lingua italiana), vogliamo codificare X in una sequenza binaria Y che sia la più corta possibile.

Per codificare X supponiamo di usare una data codifica rappresentata da un albero T. Quanto sarà la lunghezza $|Y|$ della codifica binaria Y di X che otterremo?

$\forall$ carattere $c \in C$, sia $f(c)$ la frequenza di c in X
Sia $d_T(c)$ la profondità della foglia associata al carattere c nell'albero T (corrispondente alla *lunghezza* della codifica di c).
Varrà

$$|Y| = \sum_{c \in C} f(c) d_T(c) = B(T)$$

dove $B(T)$ rappresenterà il costo del'albero T (ovvero della codifica rappresentata da T).

Da cui il problema di ottimizzazione seguente:

Dato un alfabeto di caratteri $C = \{c_1, c_2, \ldots, c_n\}$, con relative frequenze $f(c_1), f(c_2), \ldots, f(c_n)$,

Da cui il problema di ottimizzazione seguente:

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

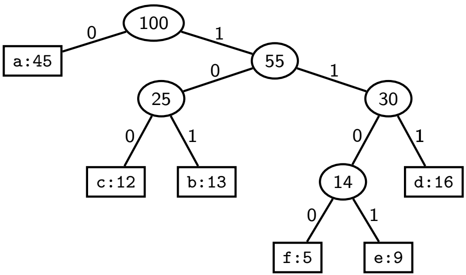
$$B(T) = \sum_{c \in C} f(c) d_T(c)$$

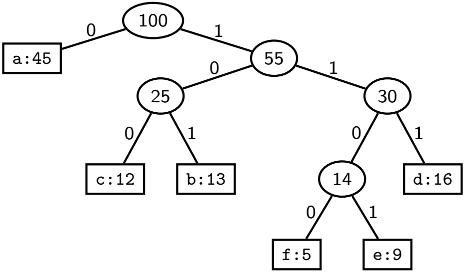
Da cui il problema di ottimizzazione seguente:

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

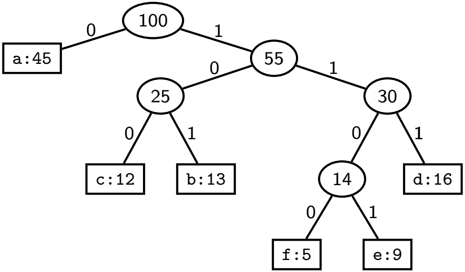
$$B(T) = \sum_{c \in C} f(c) d_T(c)$$

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

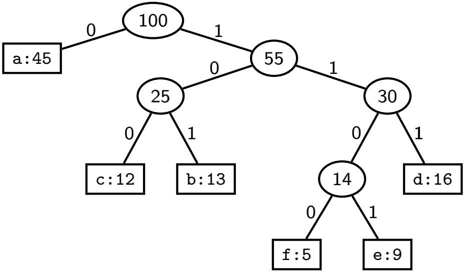

100
0
1
a:45
55
0
1
25
30
0
1
0
1
c:12
b:13
14
d:16
0
1
f:5
e:9



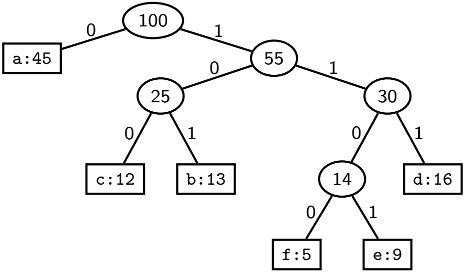
La lunghezza del file codificato (in bit) è :
45×1



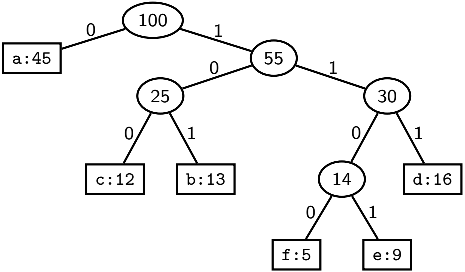
La lunghezza del file codificato (in bit) è :
$45 \times 1 + 12 \times 3$



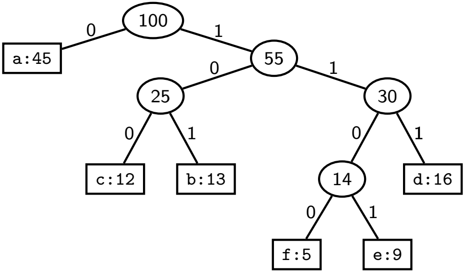
La lunghezza del file codificato (in bit) è :
$45 \times 1 + 12 \times 3 + 13 \times 3$



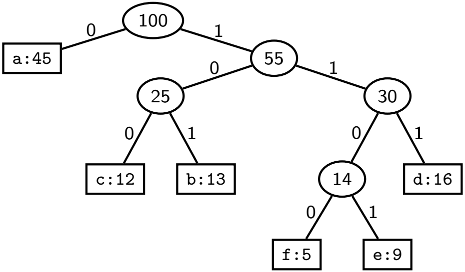
La lunghezza del file codificato (in bit) è :
$45 \times 1 + 12 \times 3 + 13 \times 3 + 5 \times 4$



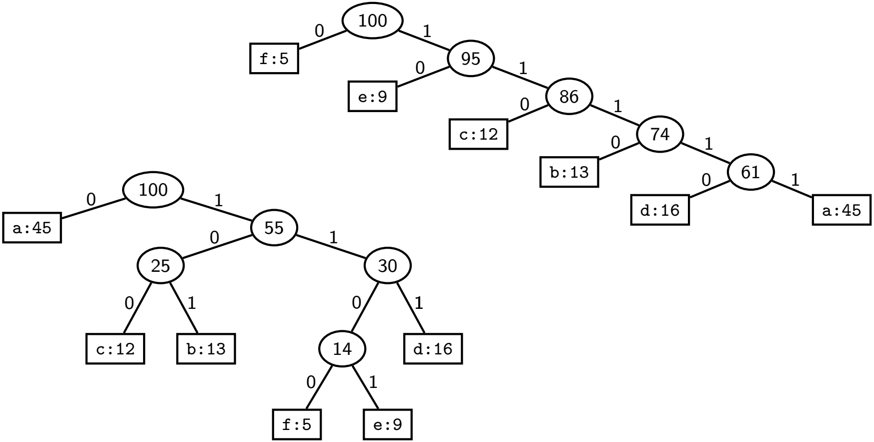
La lunghezza del file codificato (in bit) è :
$45 \times 1 + 12 \times 3 + 13 \times 3 + 5 \times 4 + 9 \times 4$



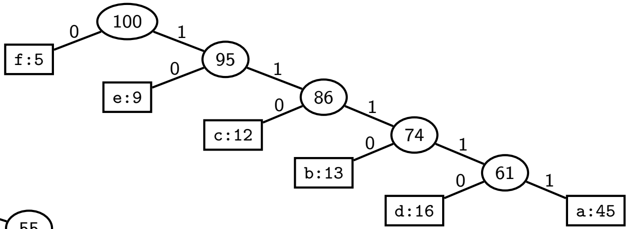
La lunghezza del file codificato (in bit) è :
$45 \times 1 + 12 \times 3 + 13 \times 3 + 5 \times 4 + 9 \times 4 + 16 \times 3$

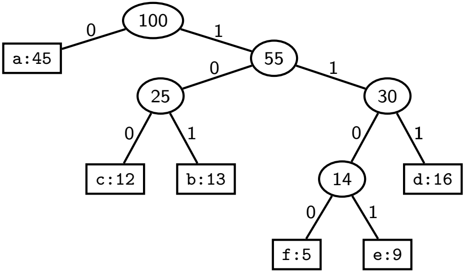


La lunghezza del file codificato (in bit) è :
$45 \times 1 + 12 \times 3 + 13 \times 3 + 5 \times 4 + 9 \times 4 + 16 \times 3 = 224$



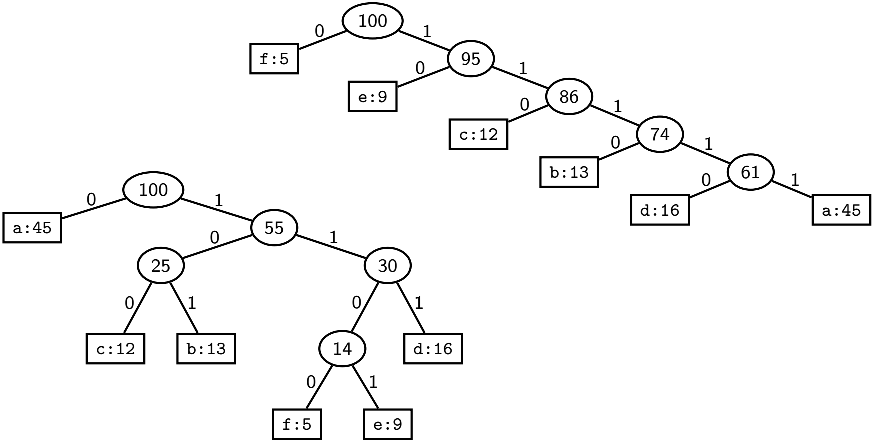
La lunghezza del file codificato (in bit) è :
$45 \times 1 + 12 \times 3 + 13 \times 3 + 5 \times 4 + 9 \times 4 + 16 \times 3 = 224$





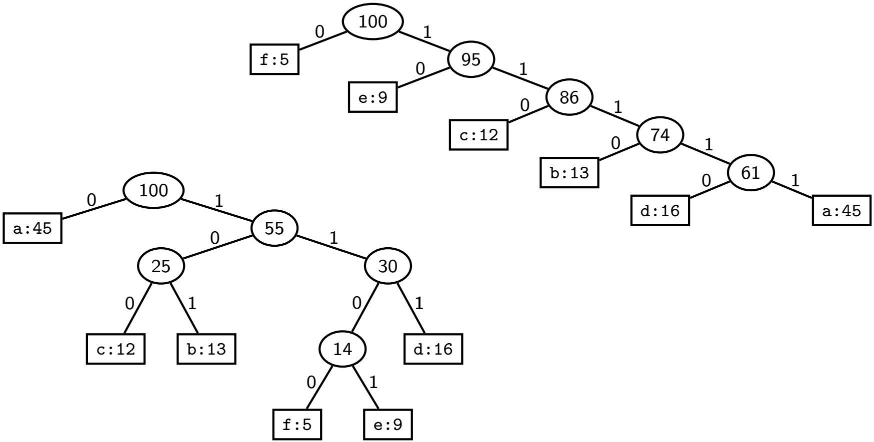
La lunghezza del file codificato (in bit) è :
$45 \times 1 + 12 \times 3 + 13 \times 3 + 5 \times 4 + 9 \times 4 + 16 \times 3 = 224$

La lunghezza del file codificato con il secondo albero sarà
45×5



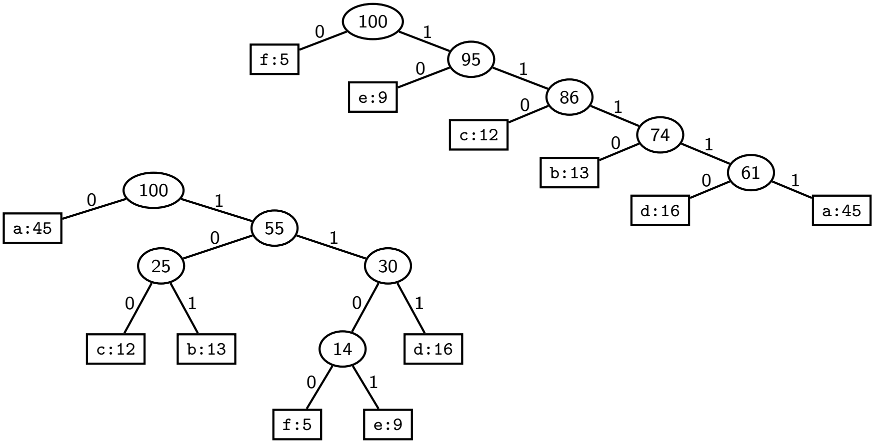
La lunghezza del file codificato (in bit) è :
$45 \times 1 + 12 \times 3 + 13 \times 3 + 5 \times 4 + 9 \times 4 + 16 \times 3 = 224$

La lunghezza del file codificato con il secondo albero sarà
$45 \times 5 + 16 \times 5$



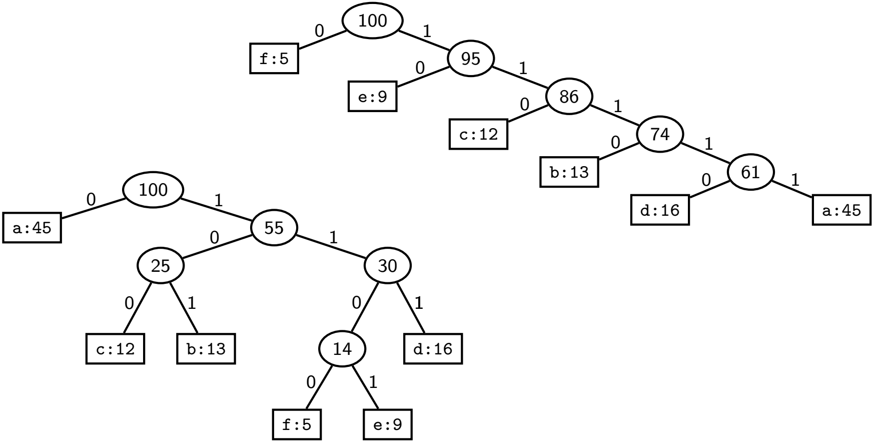
La lunghezza del file codificato (in bit) è :
$45 \times 1 + 12 \times 3 + 13 \times 3 + 5 \times 4 + 9 \times 4 + 16 \times 3 = 224$

La lunghezza del file codificato con il secondo albero sarà
$45 \times 5 + 16 \times 5 + 13 \times 4$



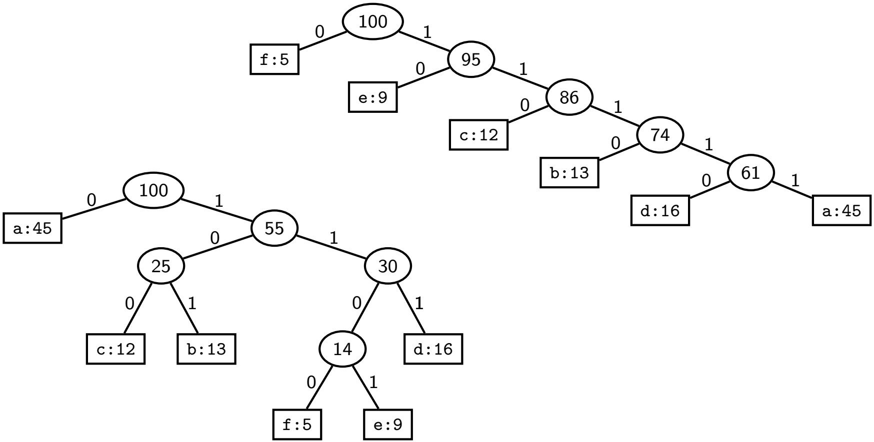
La lunghezza del file codificato (in bit) è :
$45 \times 1 + 12 \times 3 + 13 \times 3 + 5 \times 4 + 9 \times 4 + 16 \times 3 = 224$

La lunghezza del file codificato con il secondo albero sarà
$45 \times 5 + 16 \times 5 + 13 \times 4 + 12 \times 3$



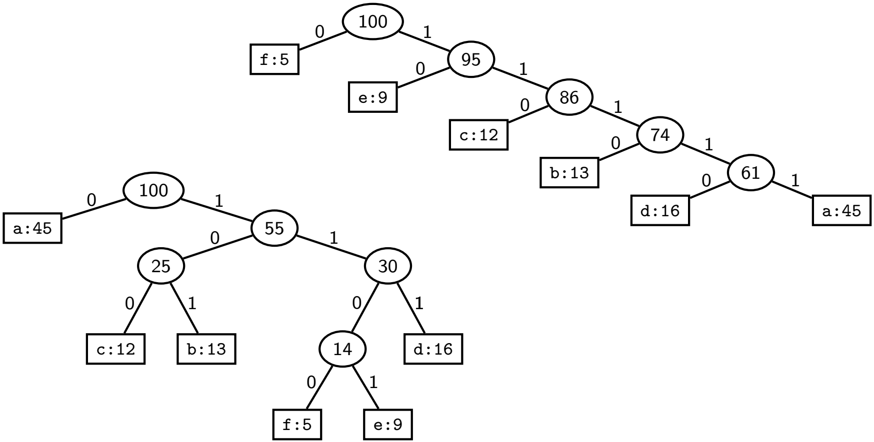
La lunghezza del file codificato (in bit) è :
$45 \times 1 + 12 \times 3 + 13 \times 3 + 5 \times 4 + 9 \times 4 + 16 \times 3 = 224$

La lunghezza del file codificato con il secondo albero sarà
$45 \times 5 + 16 \times 5 + 13 \times 4 + 12 \times 3 + 2 \times 9$



La lunghezza del file codificato (in bit) è :
$45 \times 1 + 12 \times 3 + 13 \times 3 + 5 \times 4 + 9 \times 4 + 16 \times 3 = 224$

La lunghezza del file codificato con il secondo albero sarà
$45 \times 5 + 16 \times 5 + 13 \times 4 + 12 \times 3 + 2 \times 9 + 5 \times 1$



La lunghezza del file codificato (in bit) è :
$45 \times 1 + 12 \times 3 + 13 \times 3 + 5 \times 4 + 9 \times 4 + 16 \times 3 = 224$

La lunghezza del file codificato con il secondo albero sarà
$45 \times 5 + 16 \times 5 + 13 \times 4 + 12 \times 3 + 2 \times 9 + 5 \times 1 = 416$

Ritorniamo al nostro problema di ottimizzazione:

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

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

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

Ritorniamo al nostro problema di ottimizzazione:

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

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

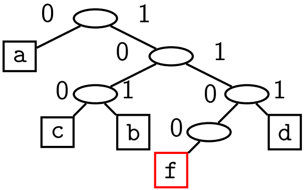
sia la *minima* possibile. Un albero T siffatto verrà detto *ottimo*.

Per facilitare la nostra ricerca di alberi ottimi, cercheremo innanzitutto di meglio comprendere come questi sono "fatti".

E come è fatto un albero ottimo?

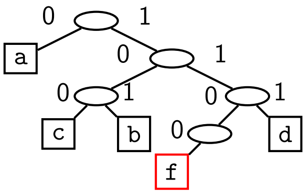
E come è fatto un albero ottimo?
Osservazione: per trovare alberi ottimi possiamo limitarci a cercare la soluzione solo tra alberi binari *completi*, ovvero in cui ogni nodo interno ha esattamente due figli.

E come è fatto un albero ottimo?
Osservazione: per trovare alberi ottimi possiamo limitarci a cercare la soluzione solo tra alberi binari *completi*, ovvero in cui ogni nodo interno ha esattamente due figli. Infatti

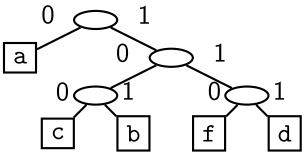


E come è fatto un albero ottimo?

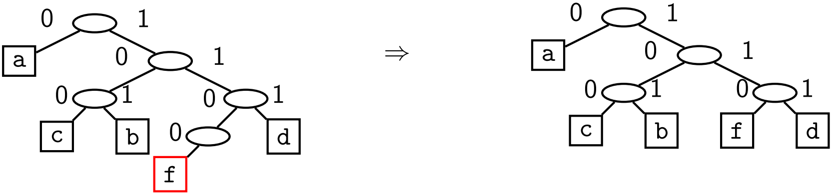
Osservazione: per trovare alberi ottimi possiamo limitarci a cercare la soluzione solo tra alberi binari *completi*, ovvero in cui ogni nodo interno ha esattamente due figli. Infatti



$\Rightarrow$



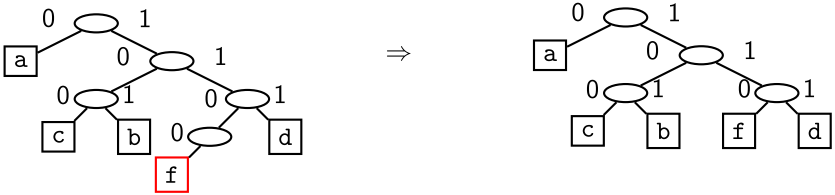
E come è fatto un albero ottimo?
Osservazione: per trovare alberi ottimi possiamo limitarci a cercare la soluzione solo tra alberi binari *completi*, ovvero in cui ogni nodo interno ha esattamente due figli. Infatti



É chiaro che l'albero di destra ha una quantità $B(\cdot)$ inferiore a quella dell'albero di sinistra, qualunque siano le frequenze $f(\cdot)$ dei caratteri,

E come è fatto un albero ottimo?

Osservazione: per trovare alberi ottimi possiamo limitarci a cercare la soluzione solo tra alberi binari *completi*, ovvero in cui ogni nodo interno ha esattamente due figli. Infatti



É chiaro che l'albero di destra ha una quantità $B(\cdot)$ inferiore a quella dell'albero di sinistra, qualunque siano le frequenze $f(\cdot)$ dei caratteri, quindi quello di sinistra non potrà *mai* essere ottimo.

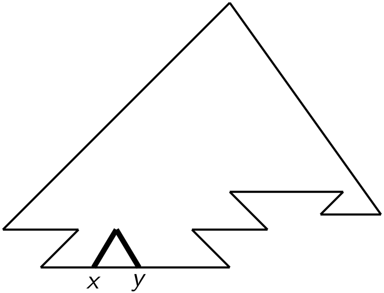
Ancora...

Fatto 1: Esiste *sicuramente* un albero ottimo T in cui i due caratteri $x, y \in C$ di frequenze minime appaiono nell'albero T alla profondità massima, e sono fratelli.

Ancora...

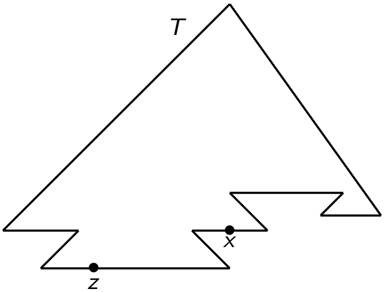
Fatto 1: Esiste *sicuramente* un albero ottimo T in cui i due caratteri $x, y \in C$ di frequenze minime appaiono nell'albero T alla profondità massima, e sono fratelli.

In altri termini, vogliamo provare che esiste *almeno* un albero ottimo fatto così

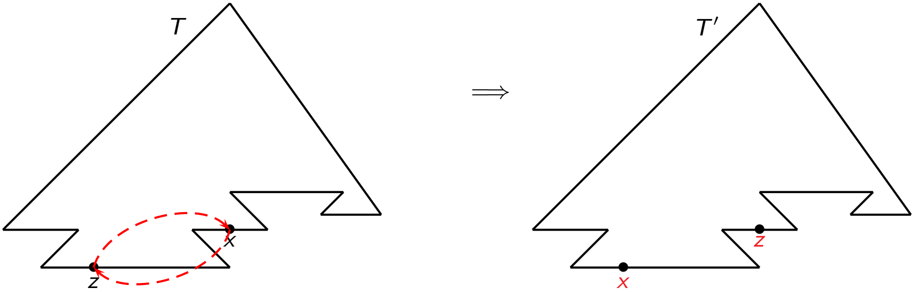


Proviamo innanzitutto che esiste un albero ottimo in cui il carattere x con la frequenza minore appre nel livello *più basso* dell'albero.

Supponiamo invece che qualcuno ci dia un albero ottimo T fatto così

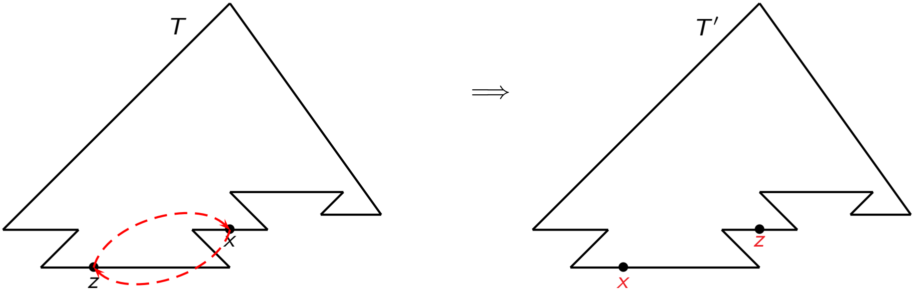


Supponiamo invece che qualcuno ci dia un albero ottimo T fatto così



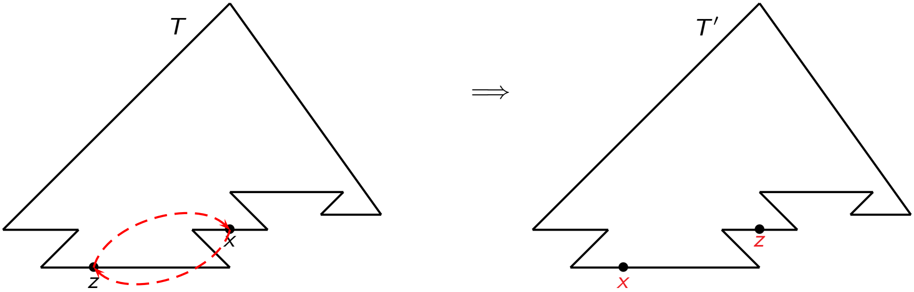
Nell'albero T scambiamo di posizione i caratteri x e z ed otteniamo T'.

Supponiamo invece che qualcuno ci dia un albero <u>ottimo</u> T fatto così



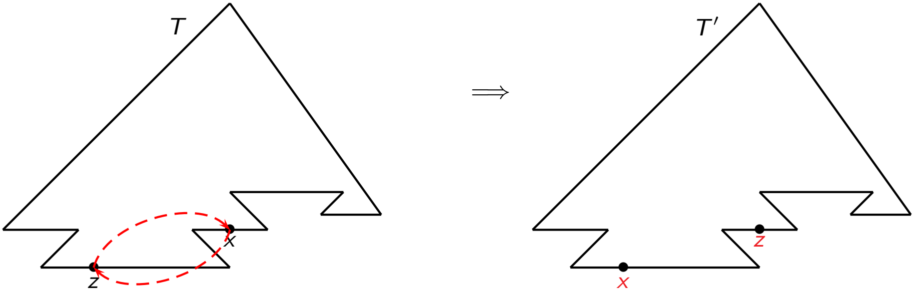
Nell'albero T scambiamo di posizione i caratteri x e z ed otteniamo T'.
Calcoliamo $B(T) - B(T')$

Supponiamo invece che qualcuno ci dia un albero ottimo T fatto così



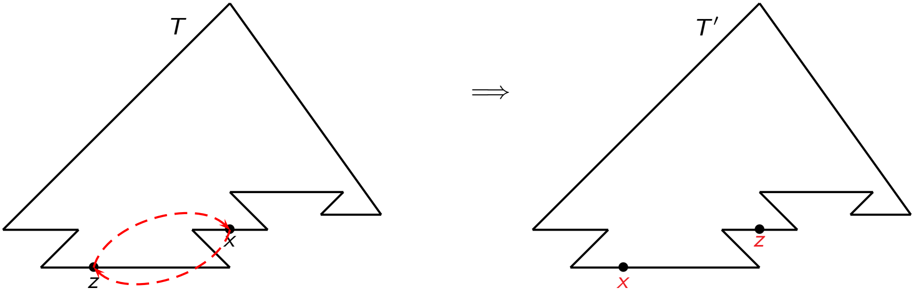
Nell'albero T scambiamo di posizione i caratteri x e z ed otteniamo T'.
Calcoliamo $B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)$

Supponiamo invece che qualcuno ci dia un albero <u>ottimo</u> T fatto così



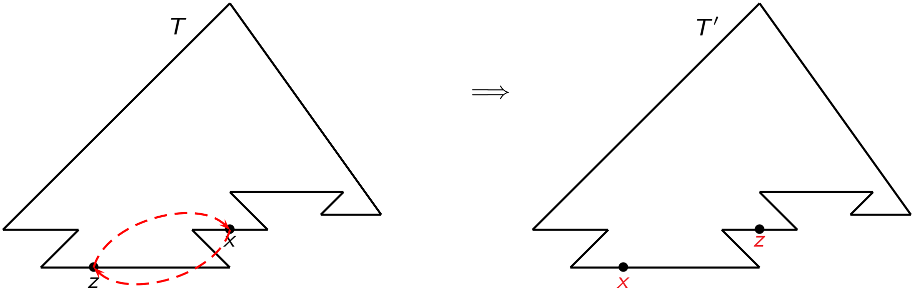
Nell'albero T scambiamo di posizione i caratteri x e z ed otteniamo T'.
Calcoliamo $B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)$
$= f(z) d_T(z) + f(x) d_T(x) - f(z) d_{T'}(z) - f(x) d_{T'}(x)$

Supponiamo invece che qualcuno ci dia un albero ottimo T fatto così



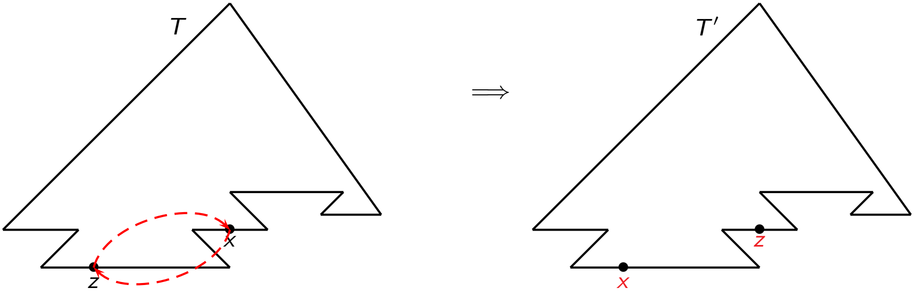
Nell'albero T scambiamo di posizione i caratteri x e z ed otteniamo T'.
Calcoliamo $B(T) - B(T') = \sum_{c \in C} f(c)d_T(c) - \sum_{c \in C} f(c)d_{T'}(c)$
$= f(z)d_T(z) + f(x)d_T(x) - f(z)d_{T'}(z) - f(x)d_{T'}(x)$
$= f(z)d_T(z) + f(x)d_T(x) - f(z)d_T(x) - f(x)d_T(z)$

Supponiamo invece che qualcuno ci dia un albero <u>ottimo</u> T fatto così



Nell'albero T scambiamo di posizione i caratteri x e z ed otteniamo T'.
Calcoliamo $B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)$
$= f(z)d_T(z) + f(x)d_T(x) - f(z)d_{T'}(z) - f(x)d_{T'}(x)$
$= f(z)d_T(z) + f(x)d_T(x) - f(z)d_T(x) - f(x)d_T(z)$
$= f(z)(d_T(z) - d_T(x))$

Supponiamo invece che qualcuno ci dia un albero ottimo T fatto così



Nell'albero T scambiamo di posizione i caratteri x e z ed otteniamo T'.

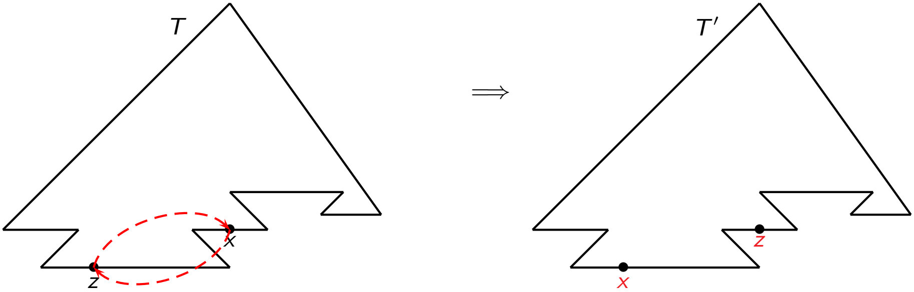
Calcoliamo $B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)$

$= f(z)d_T(z) + f(x)d_T(x) - f(z)d_{T'}(z) - f(x)d_{T'}(x)$

$= f(z)d_T(z) + f(x)d_T(x) - f(z)d_T(x) - f(x)d_T(z)$

$= f(z)(d_T(z) - d_T(x)) - f(x)(d_T(z) - d_T(x))$

Supponiamo invece che qualcuno ci dia un albero <u>ottimo</u> T fatto così



Nell'albero T scambiamo di posizione i caratteri x e z ed otteniamo T'.

Calcoliamo $B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)$

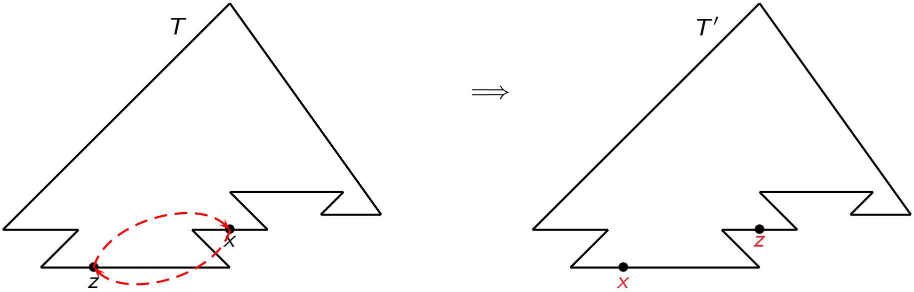
$= f(z)d_T(z) + f(x)d_T(x) - f(z)d_{T'}(z) - f(x)d_{T'}(x)$

$= f(z)d_T(z) + f(x)d_T(x) - f(z)d_T(x) - f(x)d_T(z)$

$= f(z)(d_T(z) - d_T(x)) - f(x)(d_T(z) - d_T(x))$

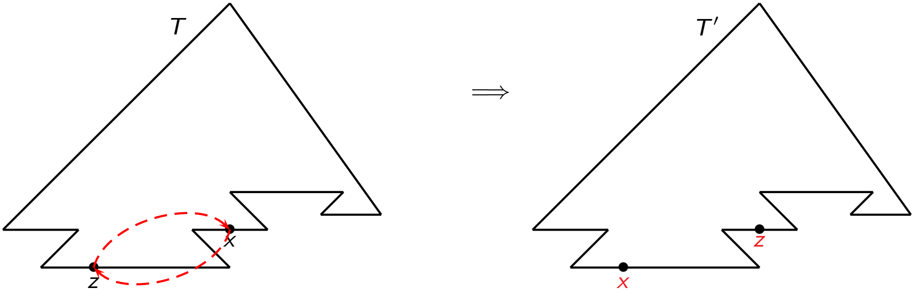
$= (f(z) - f(x)(d_T(z) - d_T(x))$

Supponiamo invece che qualcuno ci dia un albero <u>ottimo</u> T fatto così



Nell'albero T scambiamo di posizione i caratteri x e z ed otteniamo T'.
Calcoliamo $B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)$
$= f(z)d_T(z) + f(x)d_T(x) - f(z)d_{T'}(z) - f(x)d_{T'}(x)$
$= f(z)d_T(z) + f(x)d_T(x) - f(z)d_T(x) - f(x)d_T(z)$
$= f(z)(d_T(z) - d_T(x)) - f(x)(d_T(z) - d_T(x))$
$= (f(z) - f(x)(d_T(z) - d_T(x)) \geq 0$

Supponiamo invece che qualcuno ci dia un albero ottimo T fatto così



Nell'albero T scambiamo di posizione i caratteri x e z ed otteniamo T'.

Calcoliamo $B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)$

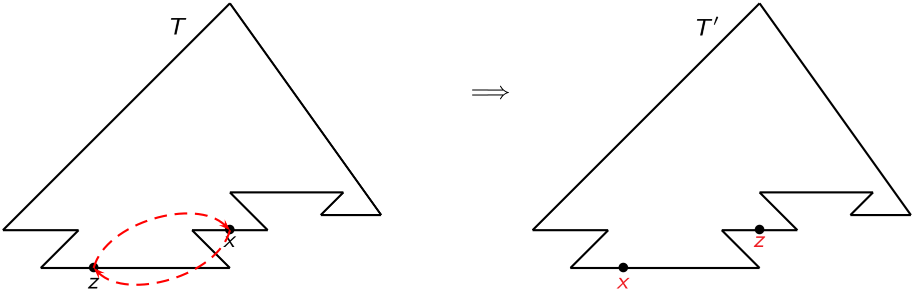
$= f(z)d_T(z) + f(x)d_T(x) - f(z)d_{T'}(z) - f(x)d_{T'}(x)$

$= f(z)d_T(z) + f(x)d_T(x) - f(z)d_T(x) - f(x)d_T(z)$

$= f(z)(d_T(z) - d_T(x)) - f(x)(d_T(z) - d_T(x))$

$= (f(z) - f(x)(d_T(z) - d_T(x)) \geq 0 \Rightarrow B(T) - B(T') \geq 0$

Supponiamo invece che qualcuno ci dia un albero <u>ottimo</u> T fatto così



Nell'albero T scambiamo di posizione i caratteri x e z ed otteniamo T'.

Calcoliamo $B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)$

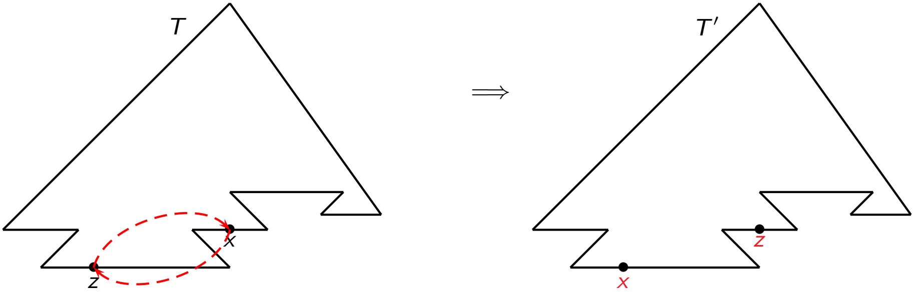
$= f(z)d_T(z) + f(x)d_T(x) - f(z)d_{T'}(z) - f(x)d_{T'}(x)$

$= f(z)d_T(z) + f(x)d_T(x) - f(z)d_T(x) - f(x)d_T(z)$

$= f(z)(d_T(z) - d_T(x)) - f(x)(d_T(z) - d_T(x))$

$= (f(z) - f(x))(d_T(z) - d_T(x)) \geq 0 \Rightarrow B(T) - B(T') \geq 0 \Rightarrow B(T) \geq B(T')$

Supponiamo invece che qualcuno ci dia un albero <u>ottimo</u> T fatto così



Nell'albero T scambiamo di posizione i caratteri x e z ed otteniamo T'.
Calcoliamo $B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)$
$= f(z)d_T(z) + f(x)d_T(x) - f(z)d_{T'}(z) - f(x)d_{T'}(x)$
$= f(z)d_T(z) + f(x)d_T(x) - f(z)d_T(x) - f(x)d_T(z)$
$= f(z)(d_T(z) - d_T(x)) - f(x)(d_T(z) - d_T(x))$
$= (f(z) - f(x)(d_T(z) - d_T(x)) \geq 0 \Rightarrow B(T) - B(T') \geq 0 \Rightarrow B(T) \geq B(T')$
$\Rightarrow$ allora anche $B(T')$ è minimo (ovvero anche T' è ottimo, e adesso x appare finalmente all'ultimo livello)

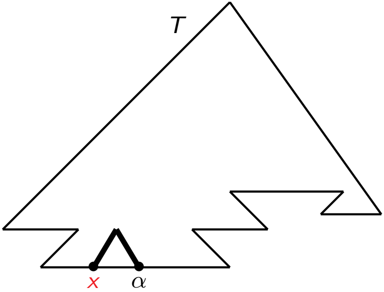
Noi dovevamo provare che

Fatto 1: Esiste *sicuramente* un albero ottimo T in cui i due caratteri $x, y \in C$ di frequenze minime appaiono nell'albero T alla profondità massima, e sono fratelli.

Noi dovevamo provare che

Fatto 1: Esiste *sicuramente* un albero ottimo T in cui i due caratteri $x, y \in C$ di frequenze minime appaiono nell'albero T alla profondità massima, e sono fratelli.

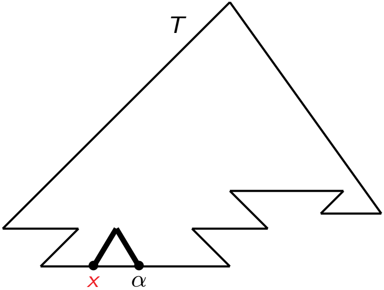
Abbiamo provato che esiste un albero ottimo in cui x appare all'ultimo livello



Noi dovevamo provare che

Fatto 1: Esiste *sicuramente* un albero ottimo T in cui i due caratteri $x, y \in C$ di frequenze minime appaiono nell'albero T alla profondità massima, e sono fratelli.

Abbiamo provato che esiste un albero ottimo in cui x appare all'ultimo livello

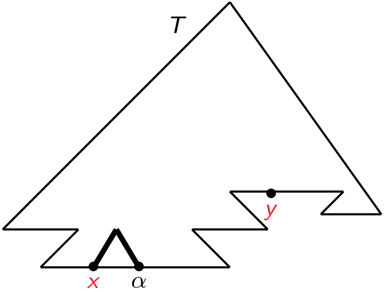


Ora, o $\alpha = y$ (ed abbiamo terminato)

Noi dovevamo provare che

Fatto 1: Esiste *sicuramente* un albero ottimo T in cui i due caratteri $x, y \in C$ di frequenze minime appaiono nell'albero T alla profondità massima, e sono fratelli.

Abbiamo provato che esiste un albero ottimo in cui x appare all'ultimo livello

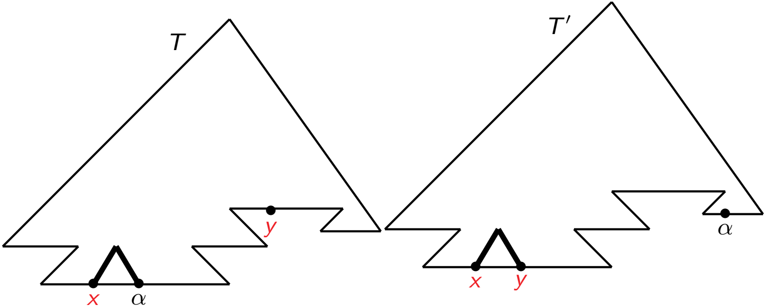


Ora, o $\alpha = y$ (ed abbiamo terminato) oppure y sta da qualche altra parte.

Noi dovevamo provare che

Fatto 1: Esiste *sicuramente* un albero ottimo T in cui i due caratteri $x, y \in C$ di frequenze minime appaiono nell'albero T alla profondità massima, e sono fratelli.

Abbiamo provato che esiste un albero ottimo in cui x appare all'ultimo livello

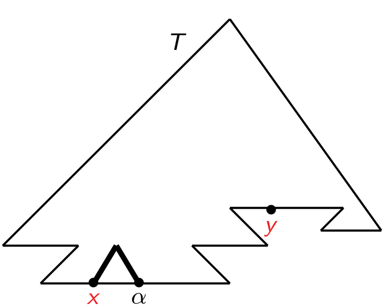


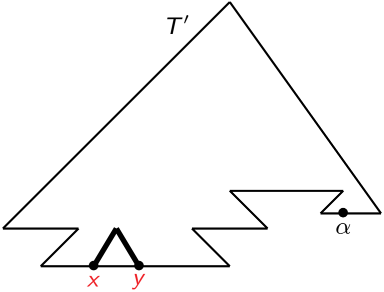
Ora, o $\alpha = y$ (ed abbiamo terminato) oppure y sta da qualche altra parte. Allora, come prima, scambiamo di posto α e y

Noi dovevamo provare che

<u>Fatto 1</u>: Esiste *sicuramente* un albero ottimo T in cui i due caratteri $x, y \in C$ di frequenze minime appaiono nell'albero T alla profondità massima, e sono fratelli.

Abbiamo provato che esiste un albero ottimo in cui x appare all'ultimo livello





Ora, o $\alpha = y$ (ed abbiamo terminato) oppure y sta da qualche altra parte. Allora, come prima, scambiamo di posto α e y ed otteniamo T', sempre ottimo, ma questa volta x e y sono fratelli e compaiono all'ultimo livello.

L'idea e l'intuizione per l'algoritmo Greedy

Visto che esiste un albero ottimo in cui i caratteri di frequenza più piccola sono fratelli e compaiono al livello più basso dell'albero, nel costruire il nostro albero ci conviene assicurarci che questo accada.

L'idea e l'intuizione per l'algoritmo Greedy

Visto che esiste un albero ottimo in cui i caratteri di frequenza più piccola sono fratelli e compaiono al livello più basso dell'albero, nel costruire il nostro albero ci conviene assicurarci che questo accada.

1. Sia $C = \{c_1, c_2, \ldots, c_n\}$, un alfabeto di caratteri, con frequenze $f(c_1), f(c_2), \ldots, f(c_n)$.

L'idea e l'intuizione per l'algoritmo Greedy

Visto che esiste un albero ottimo in cui i caratteri di frequenza più piccola sono fratelli e compaiono al livello più basso dell'albero, nel costruire il nostro albero ci conviene assicurarci che questo accada.

1. Sia $C = \{c_1, c_2, \ldots, c_n\}$, un alfabeto di caratteri, con frequenze $f(c_1), f(c_2), \ldots, f(c_n)$.

2. Costruiremo l'albero di codifica un passo alla volta, in maniera "bottom-up" partendo da n foglie, ciascuna contenente un carattere c e la sua frequenza $f(c)$.

L'idea e l'intuizione per l'algoritmo Greedy

Visto che esiste un albero ottimo in cui i caratteri di frequenza più piccola sono fratelli e compaiono al livello più basso dell'albero, nel costruire il nostro albero ci conviene assicurarci che questo accada.

1. Sia $C = \{c_1, c_2, \ldots, c_n\}$, un alfabeto di caratteri, con frequenze $f(c_1), f(c_2), \ldots, f(c_n)$.

2. Costruiremo l'albero di codifica un passo alla volta, in maniera "bottom-up" partendo da n foglie, ciascuna contenente un carattere c e la sua frequenza $f(c)$.

3. Ad ogni passo, prendiamo i due nodi p e q che hanno le frequenze *minime*, e creiamo un nuovo nodo nell'albero che sarà il loro padre (p e q diventeranno quindi fratelli). La frequenza del nuovo nodo sarà pari alla somma delle frequenze di p e q.

L'idea e l'intuizione per l'algoritmo Greedy

Visto che esiste un albero ottimo in cui i caratteri di frequenza più piccola sono fratelli e compaiono al livello più basso dell'albero, nel costruire il nostro albero ci conviene assicurarci che questo accada.

1. Sia $C = \{c_1, c_2, \ldots, c_n\}$, un alfabeto di caratteri, con frequenze $f(c_1), f(c_2), \ldots, f(c_n)$.

2. Costruiremo l'albero di codifica un passo alla volta, in maniera "bottom-up" partendo da n foglie, ciascuna contenente un carattere c e la sua frequenza $f(c)$.

3. Ad ogni passo, prendiamo i due nodi p e q che hanno le frequenze *minime*, e creiamo un nuovo nodo nell'albero che sarà il loro padre (p e q diventeranno quindi fratelli). La frequenza del nuovo nodo sarà pari alla somma delle frequenze di p e q.

4. Itereremo **3.** fin quando l'albero non e' completamente costruito.

L'idea e l'intuizione per l'algoritmo Greedy

Visto che esiste un albero ottimo in cui i caratteri di frequenza più piccola sono fratelli e compaiono al livello più basso dell'albero, nel costruire il nostro albero ci conviene assicurarci che questo accada.

1. Sia $C = \{c_1, c_2, \ldots, c_n\}$, un alfabeto di caratteri, con frequenze $f(c_1), f(c_2), \ldots, f(c_n)$.

2. Costruiremo l'albero di codifica un passo alla volta, in maniera "bottom-up" partendo da n foglie, ciascuna contenente un carattere c e la sua frequenza $f(c)$.

3. Ad ogni passo, prendiamo i due nodi p e q che hanno le frequenze *minime*, e creiamo un nuovo nodo nell'albero che sarà il loro padre (p e q diventeranno quindi fratelli). La frequenza del nuovo nodo sarà pari alla somma delle frequenze di p e q.

4. Itereremo **3.** fin quando l'albero non e' completamente costruito.

Ulteriore intuizione: Così facendo, le foglie con caratteri associati di "alta frequenza" verranno presi nel passo **3.** il più tardi possibile,

L'idea e l'intuizione per l'algoritmo Greedy

Visto che esiste un albero ottimo in cui i caratteri di frequenza più piccola sono fratelli e compaiono al livello più basso dell'albero, nel costruire il nostro albero ci conviene assicurarci che questo accada.

1. Sia $C = \{c_1, c_2, \ldots, c_n\}$, un alfabeto di caratteri, con frequenze $f(c_1), f(c_2), \ldots, f(c_n)$.

2. Costruiremo l'albero di codifica un passo alla volta, in maniera "bottom-up" partendo da n foglie, ciascuna contenente un carattere c e la sua frequenza $f(c)$.

3. Ad ogni passo, prendiamo i due nodi p e q che hanno le frequenze *minime*, e creiamo un nuovo nodo nell'albero che sarà il loro padre (p e q diventeranno quindi fratelli). La frequenza del nuovo nodo sarà pari alla somma delle frequenze di p e q.

4. Itereremo **3.** fin quando l'albero non e' completamente costruito.

Ulteriore intuizione: Così facendo, le foglie con caratteri associati di "alta frequenza" verranno presi nel passo **3.** il più tardi possibile, e quindi avranno una "piccola" profondità nell'albero (=codifica "corta"),

L'idea e l'intuizione per l'algoritmo Greedy

Visto che esiste un albero ottimo in cui i caratteri di frequenza più piccola sono fratelli e compaiono al livello più basso dell'albero, nel costruire il nostro albero ci conviene assicurarci che questo accada.

1. Sia $C = \{c_1, c_2, \ldots, c_n\}$, un alfabeto di caratteri, con frequenze $f(c_1), f(c_2), \ldots, f(c_n)$.

2. Costruiremo l'albero di codifica un passo alla volta, in maniera "bottom-up" partendo da n foglie, ciascuna contenente un carattere c e la sua frequenza $f(c)$.

3. Ad ogni passo, prendiamo i due nodi p e q che hanno le frequenze *minime*, e creiamo un nuovo nodo nell'albero che sarà il loro padre (p e q diventeranno quindi fratelli). La frequenza del nuovo nodo sarà pari alla somma delle frequenze di p e q.

4. Itereremo **3.** fin quando l'albero non e' completamente costruito.

Ulteriore intuizione: Così facendo, le foglie con caratteri associati di "alta frequenza" verranno presi nel passo **3.** il più tardi possibile, e quindi avranno una "piccola" profondità nell'albero (=codifica "corta"), mentre quelli di "bassa frequenza" verranno presi nel passo **3.** il più presto possibile,

L'idea e l'intuizione per l'algoritmo Greedy

Visto che esiste un albero ottimo in cui i caratteri di frequenza più piccola sono fratelli e compaiono al livello più basso dell'albero, nel costruire il nostro albero ci conviene assicurarci che questo accada.

1. Sia $C = \{c_1, c_2, \ldots, c_n\}$, un alfabeto di caratteri, con frequenze $f(c_1), f(c_2), \ldots, f(c_n)$.

2. Costruiremo l'albero di codifica un passo alla volta, in maniera "bottom-up" partendo da n foglie, ciascuna contenente un carattere c e la sua frequenza $f(c)$.

3. Ad ogni passo, prendiamo i due nodi p e q che hanno le frequenze *minime*, e creiamo un nuovo nodo nell'albero che sarà il loro padre (p e q diventeranno quindi fratelli). La frequenza del nuovo nodo sarà pari alla somma delle frequenze di p e q.

4. Itereremo **3.** fin quando l'albero non e' completamente costruito.

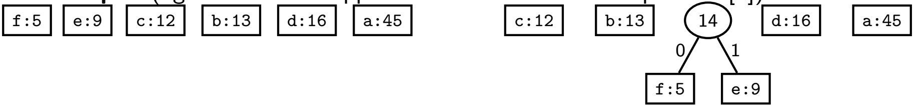
Ulteriore intuizione: Così facendo, le foglie con caratteri associati di "alta frequenza" verranno presi nel passo **3.** il più tardi possibile, e quindi avranno una "piccola" profondità nell'albero (=codifica "corta"), mentre quelli di "bassa frequenza" verranno presi nel passo **3.** il più presto possibile, e quindi avranno una "alta" profondità nell'albero (=codifica "lunga").

Esempio: (ogni carattere c appare con a fianco la sua frequenza $f[c]$)

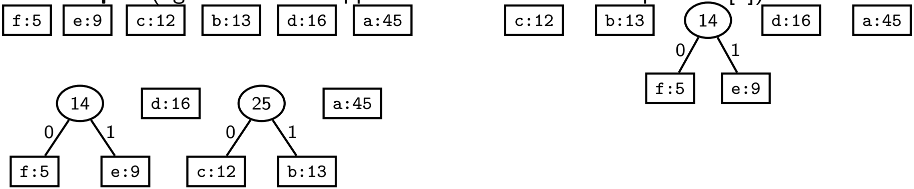
Esempio: (ogni carattere c appare con a fianco la sua frequenza $f[\mathrm{c}]$)

$\boxed{\texttt{f:5}}$ $\boxed{\texttt{e:9}}$ $\boxed{\texttt{c:12}}$ $\boxed{\texttt{b:13}}$ $\boxed{\texttt{d:16}}$ $\boxed{\texttt{a:45}}$

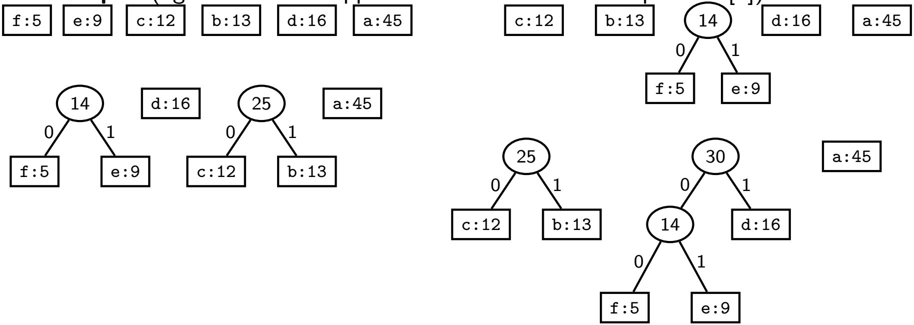
Esempio: (ogni carattere c appare con a fianco la sua frequenza $f[c]$)



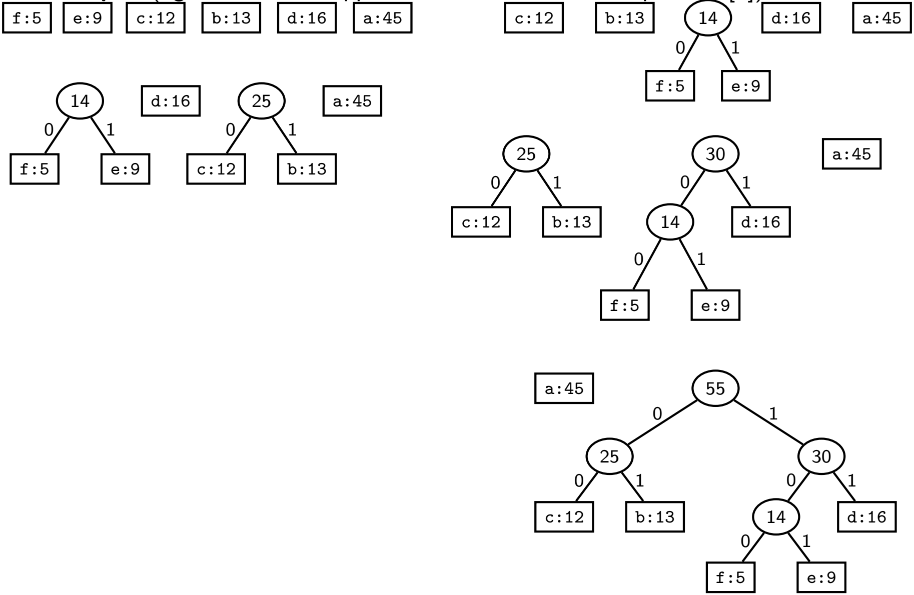
Esempio: (ogni carattere c appare con a fianco la sua frequenza $f[c]$)



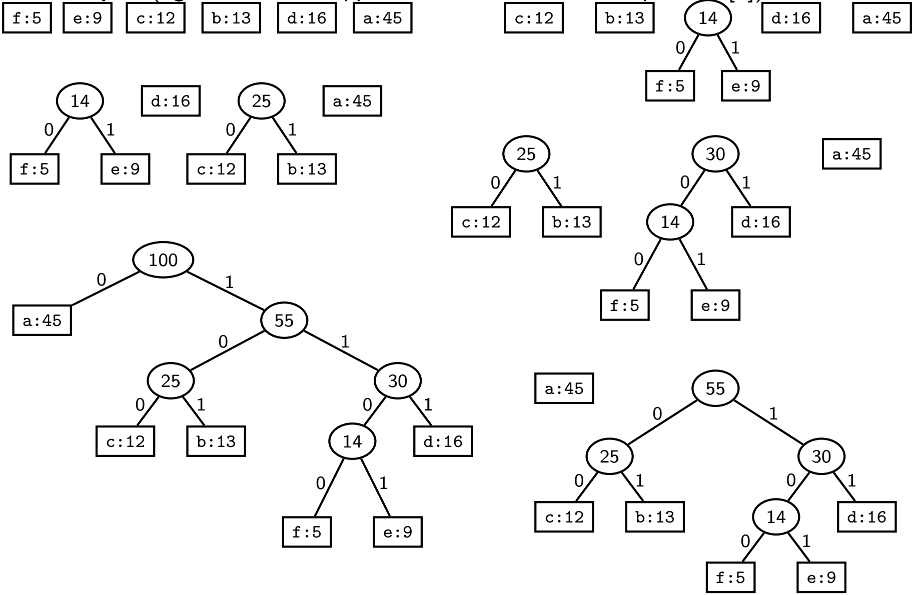
Esempio: (ogni carattere c appare con a fianco la sua frequenza $f[c]$)



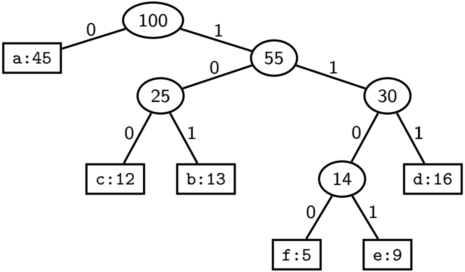
Esempio: (ogni carattere c appare con a fianco la sua frequenza $f[c]$)



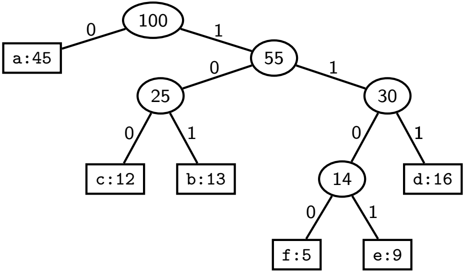
Esempio: (ogni carattere c appare con a fianco la sua frequenza $f[c]$)



Costruito l'albero

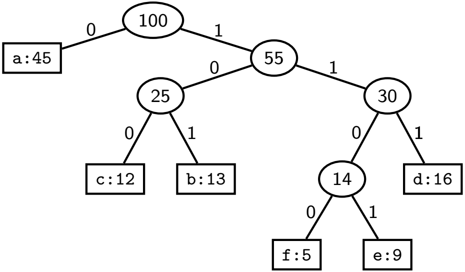


Costruito l'albero, la codifica dei caratteri è ottenuta leggendo la sequenza di bit che otteniamo andando dalla radice dell'albero alla foglia in cui il corrispondente carattere appare:



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

Costruito l'albero, la codifica dei caratteri è ottenuta leggendo la sequenza di bit che otteniamo andando dalla radice dell'albero alla foglia in cui il corrispondente carattere appare:

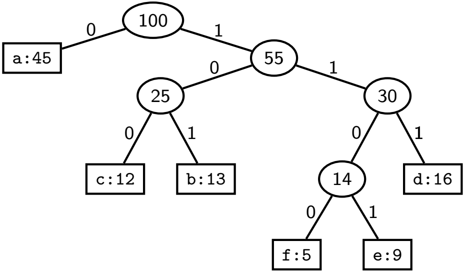


carattere	codifica
a	0
c	100
b	101
d	111
f	1100
e	1101

La lunghezza del file codificato (in bit) è :

45×1

Costruito l'albero, la codifica dei caratteri è ottenuta leggendo la sequenza di bit che otteniamo andando dalla radice dell'albero alla foglia in cui il corrispondente carattere appare:

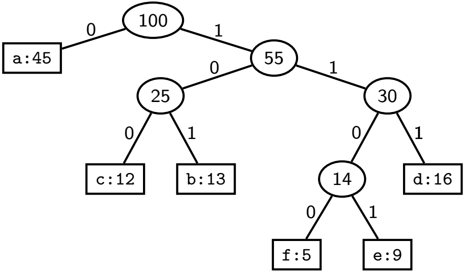


carattere	codifica
a	0
c	100
b	101
d	111
f	1100
e	1101

La lunghezza del file codificato (in bit) è :

$$45 \times 1 + 12 \times 3$$

Costruito l'albero, la codifica dei caratteri è ottenuta leggendo la sequenza di bit che otteniamo andando dalla radice dell'albero alla foglia in cui il corrispondente carattere appare:

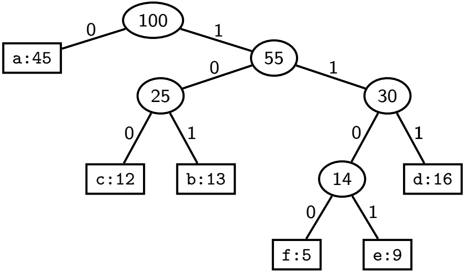


carattere	codifica
a	0
c	100
b	101
d	111
f	1100
e	1101

La lunghezza del file codificato (in bit) è :

$$45 \times 1 + 12 \times 3 + 13 \times 3$$

Costruito l'albero, la codifica dei caratteri è ottenuta leggendo la sequenza di bit che otteniamo andando dalla radice dell'albero alla foglia in cui il corrispondente carattere appare:

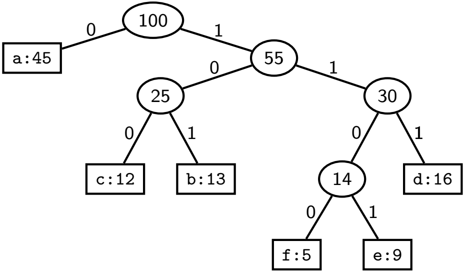


carattere	codifica
a	0
c	100
b	101
d	111
f	1100
e	1101

La lunghezza del file codificato (in bit) è :

$$45 \times 1 + 12 \times 3 + 13 \times 3 + 5 \times 4$$

Costruito l'albero, la codifica dei caratteri è ottenuta leggendo la sequenza di bit che otteniamo andando dalla radice dell'albero alla foglia in cui il corrispondente carattere appare:

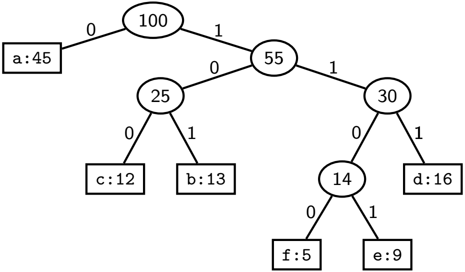


carattere	codifica
a	0
c	100
b	101
d	111
f	1100
e	1101

La lunghezza del file codificato (in bit) è :

$$45 \times 1 + 12 \times 3 + 13 \times 3 + 5 \times 4 + 9 \times 4$$

Costruito l'albero, la codifica dei caratteri è ottenuta leggendo la sequenza di bit che otteniamo andando dalla radice dell'albero alla foglia in cui il corrispondente carattere appare:

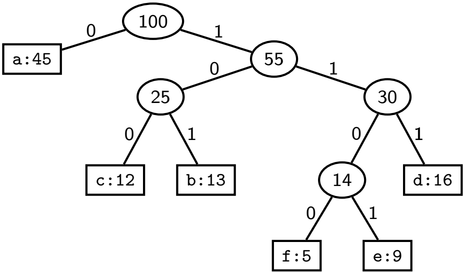


carattere	codifica
a	0
c	100
b	101
d	111
f	1100
e	1101

La lunghezza del file codificato (in bit) è :

$$45 \times 1 + 12 \times 3 + 13 \times 3 + 5 \times 4 + 9 \times 4 + 16 \times 3$$

Costruito l'albero, la codifica dei caratteri è ottenuta leggendo la sequenza di bit che otteniamo andando dalla radice dell'albero alla foglia in cui il corrispondente carattere appare:



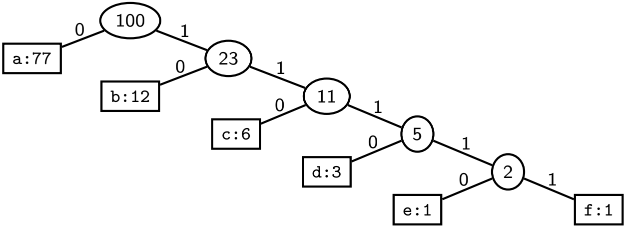
carattere	codifica
a	0
c	100
b	101
d	111
f	1100
e	1101

La lunghezza del file codificato (in bit) è :

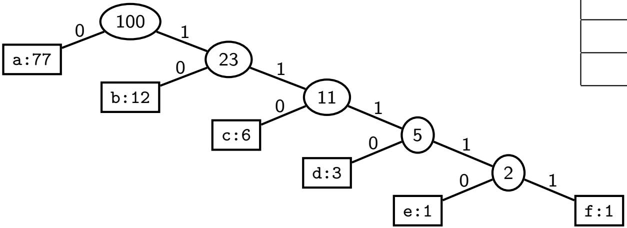
$$45 \times 1 + 12 \times 3 + 13 \times 3 + 5 \times 4 + 9 \times 4 + 16 \times 3 = 224$$

Nota: Se cambiano le frequenze dei caratteri, la codifica può cambiare!

Nota: Se cambiano le frequenze dei caratteri, la codifica può cambiare!



Nota: Se cambiano le frequenze dei caratteri, la codifica può cambiare!



carattere	codifica
a	0
b	10
c	110
d	1110
f	11110
e	11111

L'Algoritmo di Huffman

$\texttt{Huffman}(C = \{c_1, c_2, \ldots, c_n\},\ f(c_1), f(c_2), \ldots, f(c_n))$

L'Algoritmo di Huffman

Huffman($C = \{c_1, c_2, \ldots, c_n\}$, $f(c_1), f(c_2), \ldots, f(c_n)$)
1. metti gli elementi di C in un a coda Q

L'Algoritmo di Huffman

$\texttt{Huffman}(C = \{c_1, c_2, \ldots, c_n\},\ f(c_1), f(c_2), \ldots, f(c_n))$
1. metti gli elementi di C in un a coda Q
2. FOR $i \leftarrow 1$ TO $|C|$
3. DO alloca un nuovo nodo $\mathtt{z}$

L'Algoritmo di Huffman

$\texttt{Huffman}(C = \{\mathrm{c}_1, \mathrm{c}_2, \ldots, \mathrm{c}_n\},\ f(\mathrm{c}_1), f(\mathrm{c}_2), \ldots, f(\mathrm{c}_n))$
1. metti gli elementi di C in un a coda Q
2. FOR $i \leftarrow 1$ TO $|C|$
3. DO alloca un nuovo nodo $\mathtt{z}$
4. $\mathit{left}[\mathtt{z}] \leftarrow \mathtt{x} \leftarrow \texttt{Extract\_Min}(Q)$

L'Algoritmo di Huffman

$\texttt{Huffman}(C = \{c_1, c_2, \ldots, c_n\}, f(c_1), f(c_2), \ldots, f(c_n))$

1. metti gli elementi di C in un a coda Q
2. FOR $i \leftarrow 1$ TO $|C|$
3. DO alloca un nuovo nodo z
4. $\mathit{left}[\mathrm{z}] \leftarrow \mathrm{x} \leftarrow \texttt{Extract\_Min}(Q)$
5. $\mathit{right}[\mathrm{z}] \leftarrow \mathrm{y} \leftarrow \texttt{Extract\_Min}(Q)$

L'Algoritmo di Huffman

$\mathtt{Huffman}(C = \{\mathrm{c}_1, \mathrm{c}_2, \ldots, \mathrm{c}_n\},\ f(\mathrm{c}_1), f(\mathrm{c}_2), \ldots, f(\mathrm{c}_n))$
1. metti gli elementi di C in un a coda Q
2. FOR $i \leftarrow 1$ TO $|C|$
3. DO alloca un nuovo nodo $\mathtt{z}$
4. $\quad$ $\mathit{left}[\mathrm{z}] \leftarrow \mathrm{x} \leftarrow \mathtt{Extract\_Min}(Q)$
5. $\quad$ $\mathit{right}[\mathrm{z}] \leftarrow \mathrm{y} \leftarrow \mathtt{Extract\_Min}(Q)$
6. $\quad$ $f[\mathrm{z}] \leftarrow f[\mathrm{x}] + f[\mathrm{y}]$

L'Algoritmo di Huffman

```
Huffman(C = {c_1, c_2, ..., c_n}, f(c_1), f(c_2), ..., f(c_n))
1. metti gli elementi di C in un a coda Q
2. FOR i ← 1 TO |C|
3.    DO alloca un nuovo nodo z
4.       left[z] ← x ← Extract_Min(Q)
5.       right[z] ← y ← Extract_Min(Q)
6.       f[z] ← f[x] + f[y]
7.       Insert(Q, z)
```

L'Algoritmo di Huffman

```
Huffman(C = {c_1, c_2, ..., c_n}, f(c_1), f(c_2), ..., f(c_n))
1. metti gli elementi di C in un a coda Q
2. FOR i ← 1 TO |C|
3.    DO alloca un nuovo nodo z
4.       left[z] ← x ← Extract_Min(Q)
5.       right[z] ← y ← Extract_Min(Q)
6.       f[z] ← f[x] + f[y]
7.       Insert(Q, z)
```

Extract_Min(Q) restituisce il carattere in C che ha la frequenza minima, e lo toglie da Q.

L'Algoritmo di Huffman

```
Huffman(C = {c1, c2, ..., cn}, f(c1), f(c2), ..., f(cn))
1. metti gli elementi di C in un a coda Q
2. FOR i ← 1 TO |C|
3.    DO alloca un nuovo nodo z
4.       left[z] ← x ← Extract_Min(Q)
5.       right[z] ← y ← Extract_Min(Q)
6.       f[z] ← f[x] + f[y]
7.       Insert(Q, z)
```

`Extract_Min`(Q) restituisce il carattere in C che ha la frequenza minima, e lo toglie da Q.

Analisi: Vedremo in seguito come sarà possibile costruire la coda Q al passo 1. in modo tale che le operazioni al passo 4. e 5. si possono eseguire in tempo $O(\log n)$, dove n è il numero di caratteri in C.

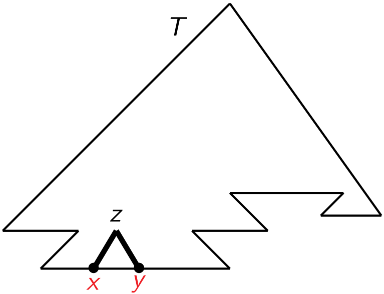
L'Algoritmo di Huffman

$\texttt{Huffman}(C = \{c_1, c_2, \ldots, c_n\},\ f(c_1), f(c_2), \ldots, f(c_n))$
1. metti gli elementi di C in un a coda Q
2. FOR $i \leftarrow 1$ TO $|C|$
3. DO alloca un nuovo nodo z
4. $\quad$ $\mathit{left}[\mathrm{z}] \leftarrow \mathrm{x} \leftarrow \texttt{Extract\_Min}(Q)$
5. $\quad$ $\mathit{right}[\mathrm{z}] \leftarrow \mathrm{y} \leftarrow \texttt{Extract\_Min}(Q)$
6. $\quad$ $f[\mathrm{z}] \leftarrow f[\mathrm{x}] + f[\mathrm{y}]$
7. $\quad$ $\texttt{Insert}(Q, \mathrm{z})$

$\texttt{Extract\_Min}(Q)$ restituisce il carattere in C che ha la frequenza minima, e lo toglie da Q.
<u>Analisi</u>: Vedremo in seguito come sarà possibile costruire la coda Q al passo 1. in modo tale che le operazioni al passo 4. e 5. si possono eseguire in tempo $O(\log n)$, dove n è il numero di caratteri in C. Pertanto, la complessità dell'algoritmo $\texttt{Huffman}(C, f)$ è $O(n \log n)$

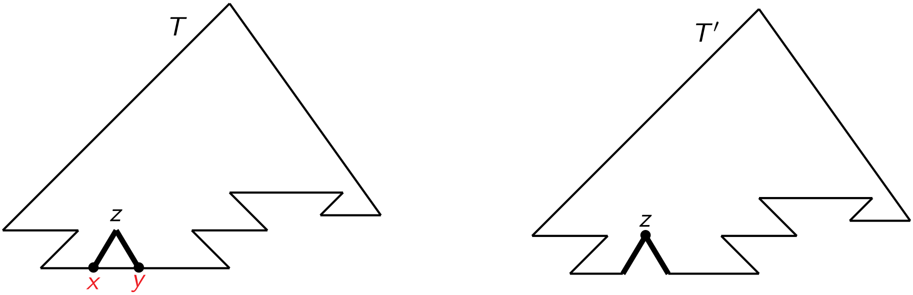
Ritorniamo per un momento al:

Fatto 1: Esiste *sicuramente* un albero ottimo T in cui i due caratteri $x, y \in C$ di frequenze minime appaiono nell'albero T alla profondità massima, e sono fratelli.



Ritorniamo per un momento al:

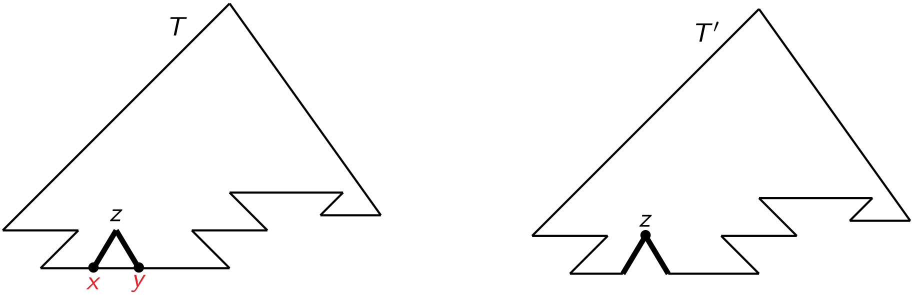
Fatto 1: Esiste *sicuramente* un albero ottimo T in cui i due caratteri $x, y \in C$ di frequenze minime appaiono nell'albero T alla profondità massima, e sono fratelli.



e chiediamoci che relazione esiste tra $B(T) = \sum_{c \in C} f(c) d_T(c)$ e $B(T') = \sum_{c \in C'} f(c) d_{T'}(c)$,

Ritorniamo per un momento al:

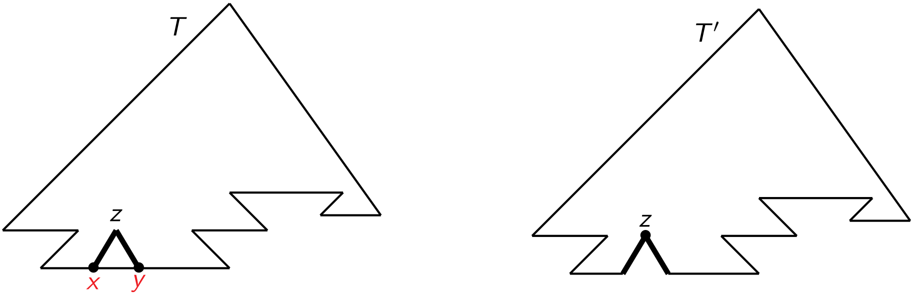
Fatto 1: Esiste *sicuramente* un albero ottimo T in cui i due caratteri $x, y \in C$ di frequenze minime appaiono nell'albero T alla profondità massima, e sono fratelli.



e chiediamoci che relazione esiste tra $B(T) = \sum_{c \in C} f(c) d_T(c)$ e $B(T') = \sum_{c \in C'} f(c) d_{T'}(c)$, dove T' è ottenuto da T eliminando x e y, rendendo quindi il loro padre z una foglia, con frequenza $f(z) = f(x) + f(y)$ (ricordiamoci come opera l'Algoritmo di Huffman)

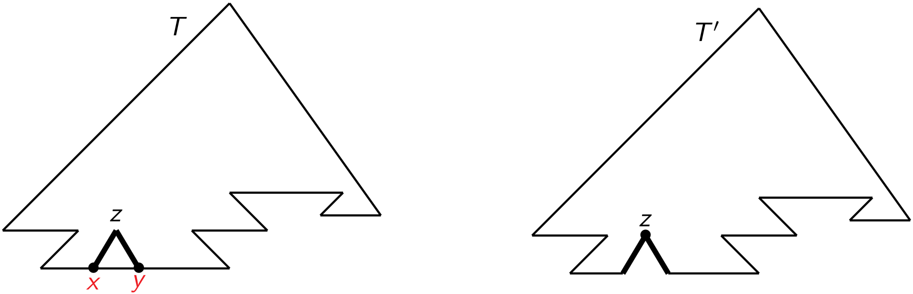
Ritorniamo per un momento al:

Fatto 1: Esiste *sicuramente* un albero ottimo T in cui i due caratteri $x, y \in C$ di frequenze minime appaiono nell'albero T alla profondità massima, e sono fratelli.

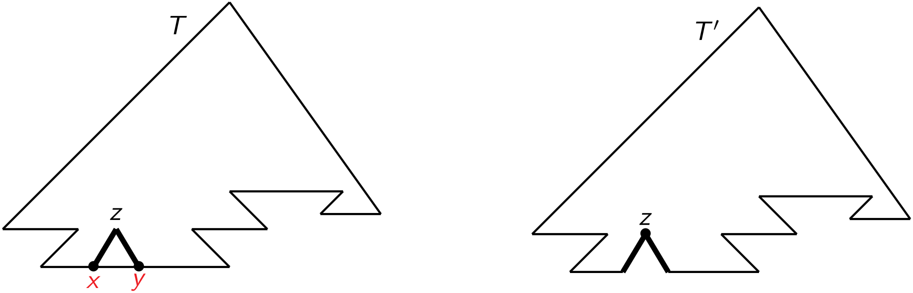


e chiediamoci che relazione esiste tra $B(T) = \sum_{\mathtt{c} \in C} f(\mathtt{c}) d_T(\mathtt{c})$ e $B(T') = \sum_{\mathtt{c} \in C'} f(\mathtt{c}) d_{T'}(\mathtt{c})$, dove T' è ottenuto da T eliminando x e y, rendendo quindi il loro padre z una foglia, con frequenza $f(\mathtt{z}) = f(\mathtt{x}) + f(\mathtt{y})$ (ricordiamoci come opera l'Algoritmo di Huffman) quindi $C' = (C \setminus \{\mathtt{x}, \mathtt{y}\}) \cup \{\mathtt{z}\}$

Calcoliamo, ricordando che $f(\mathtt{z}) = f(\mathtt{x}) + f(\mathtt{y})$:

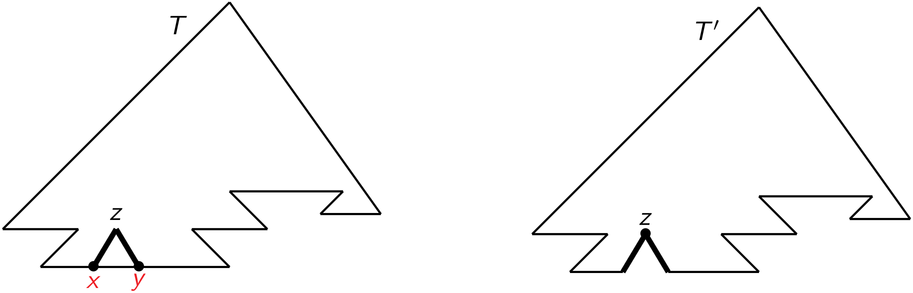


Calcoliamo, ricordando che $f(\mathtt{z}) = f(\mathtt{x}) + f(\mathtt{y})$:



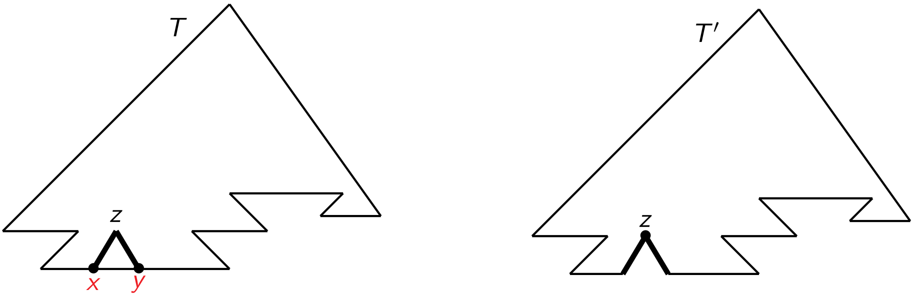
$B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C'} f(c) d_{T'}(c)$

Calcoliamo, ricordando che $f(\mathtt{z}) = f(\mathtt{x}) + f(\mathtt{y})$:



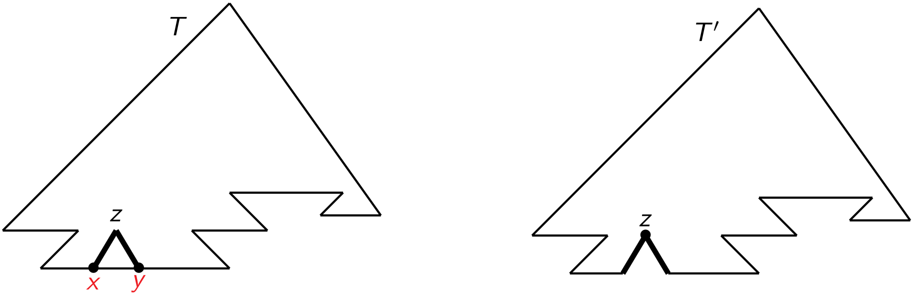
$$\begin{aligned} B(T) - B(T') &= \sum_{\mathtt{c}\in C} f(\mathtt{c}) d_T(\mathtt{c}) - \sum_{\mathtt{c}\in C'} f(\mathtt{c}) d_{T'}(\mathtt{c}) \\ &= f(\mathtt{x}) d_T(\mathtt{x}) + f(\mathtt{y}) d_T(\mathtt{y}) - f(\mathtt{z}) d_{T'}(\mathtt{z}) \end{aligned}$$

Calcoliamo, ricordando che $f(\mathtt{z}) = f(\mathtt{x}) + f(\mathtt{y})$:



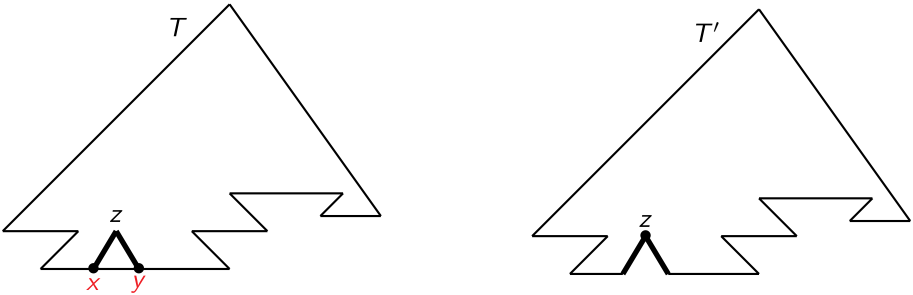
$$\begin{aligned} B(T) - B(T') &= \sum_{\mathtt{c}\in C} f(\mathtt{c})d_T(\mathtt{c}) - \sum_{\mathtt{c}\in C'} f(\mathtt{c})d_{T'}(\mathtt{c}) \\ &= f(\mathtt{x})d_T(\mathtt{x}) + f(\mathtt{y})d_T(\mathtt{y}) - f(\mathtt{z})d_{T'}(\mathtt{z}) \\ &= (f(\mathtt{x}) + f(\mathtt{y}))d_T(\mathtt{x}) - (f(\mathtt{x}) + f(\mathtt{y}))d_{T'}(\mathtt{z}) \end{aligned}$$

Calcoliamo, ricordando che $f(\mathtt{z}) = f(\mathtt{x}) + f(\mathtt{y})$:



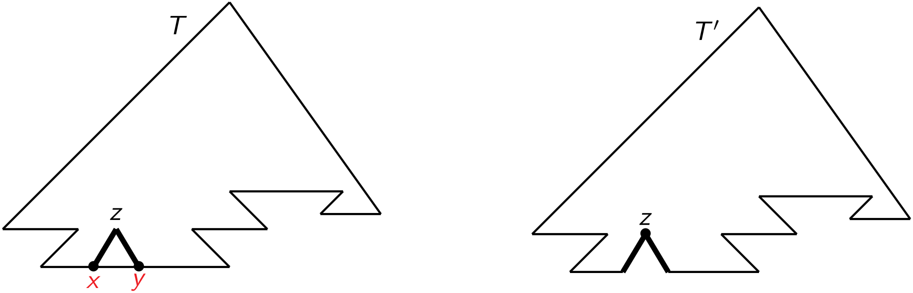
$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} f(\mathtt{c}) d_T(\mathtt{c}) - \sum_{c \in C'} f(\mathtt{c}) d_{T'}(\mathtt{c}) \\
&= f(\mathtt{x}) d_T(\mathtt{x}) + f(\mathtt{y}) d_T(\mathtt{y}) - f(\mathtt{z}) d_{T'}(\mathtt{z}) \\
&= (f(\mathtt{x}) + f(\mathtt{y})) d_T(\mathtt{x}) - (f(\mathtt{x}) + f(\mathtt{y})) d_{T'}(\mathtt{z}) \\
&= (f(\mathtt{x}) + f(\mathtt{y})) d_T(\mathtt{x}) - (f(\mathtt{x}) + f(\mathtt{y}))(d_T(\mathtt{x}) - 1)
\end{aligned}
$$

Calcoliamo, ricordando che $f(\mathtt{z}) = f(\mathtt{x}) + f(\mathtt{y})$:



$$
\begin{aligned}
B(T) - B(T') &= \textstyle\sum_{\mathtt{c}\in C} f(\mathtt{c})d_T(\mathtt{c}) - \sum_{\mathtt{c}\in C'} f(\mathtt{c})d_{T'}(\mathtt{c}) \\
&= f(\mathtt{x})d_T(\mathtt{x}) + f(\mathtt{y})d_T(\mathtt{y}) - f(\mathtt{z})d_{T'}(\mathtt{z}) \\
&= (f(\mathtt{x}) + f(\mathtt{y}))d_T(\mathtt{x}) - (f(\mathtt{x}) + f(\mathtt{y}))d_{T'}(\mathtt{z}) \\
&= (f(\mathtt{x}) + f(\mathtt{y}))d_T(\mathtt{x}) - (f(\mathtt{x}) + f(\mathtt{y}))(d_T(\mathtt{x}) - 1) \\
&= (f(\mathtt{x}) + f(\mathtt{y}))
\end{aligned}
$$

Calcoliamo, ricordando che $f(\mathrm{z}) = f(\mathrm{x}) + f(\mathrm{y})$:



$$
\begin{aligned}
B(T) - B(T') &= \textstyle\sum_{c \in C} f(c) d_T(c) - \sum_{c \in C'} f(c) d_{T'}(c) \\
&= f(\mathrm{x}) d_T(\mathrm{x}) + f(\mathrm{y}) d_T(\mathrm{y}) - f(\mathrm{z}) d_{T'}(\mathrm{z}) \\
&= (f(\mathrm{x}) + f(\mathrm{y})) d_T(\mathrm{x}) - (f(\mathrm{x}) + f(\mathrm{y})) d_{T'}(\mathrm{z}) \\
&= (f(\mathrm{x}) + f(\mathrm{y})) d_T(\mathrm{x}) - (f(\mathrm{x}) + f(\mathrm{y}))(d_T(\mathrm{x}) - 1) \\
&= (f(\mathrm{x}) + f(\mathrm{y})) \Rightarrow
\end{aligned}
$$

<u>Fatto 2</u>: Per alberi T e T' siffatti vale $\quad B(T) = B(T') + f(\mathrm{x}) + f(\mathrm{y})$

Proviamo che l'alg. $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero T con $B[T]$ *minimo*

Proviamo che l'alg. $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero T con $B[T]$ *minimo*

Per induzione sul numero n dei caratteri $\texttt{c}_1, \texttt{c}_2, \dots, \texttt{c}_n$.

Proviamo che l'alg. $\mathtt{Huffman(C, f)}$ produce un albero T con $B[T]$ *minimo*

Per induzione sul numero n dei caratteri $\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_n$.

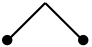
Per $n = 2$ l'algoritmo $\mathtt{Huffman(C, f)}$ produce un albero del tipo

che è chiaramente ottimo (è l'unico albero con due foglie!).

Proviamo che l'alg. $\mathtt{Huffman}(\mathtt{C},\mathtt{f})$ produce un albero T con $B[T]$ *minimo*

Per induzione sul numero n dei caratteri $\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_n$.

Per $n = 2$ l'algoritmo $\mathtt{Huffman}(\mathtt{C},\mathtt{f})$ produce un albero del tipo



che è chiaramente ottimo (è l'unico albero con due foglie!).

Supponiamo quindi che $\mathtt{Huffman}(\mathtt{C},\mathtt{f})$ produce un albero ottimo quando $|\mathtt{C}| = n$, e consideriamo l'esecuzione di $\mathtt{Huffman}(\mathtt{C}',\mathtt{f})$ su di un insieme di caratteri $\mathtt{C'} = \{\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_n, \mathtt{c}_{n+1}\}$.

Proviamo che l'alg. $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero T con $B[T]$ *minimo*

Per induzione sul numero n dei caratteri $\texttt{c}_1, \texttt{c}_2, \ldots, \texttt{c}_n$.

Per $n = 2$ l'algoritmo $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero del tipo

che è chiaramente ottimo (è l'unico albero con due foglie!).

Supponiamo quindi che $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero ottimo quando $|\texttt{C}| = n$, e consideriamo l'esecuzione di $\texttt{Huffman}(\texttt{C}', \texttt{f})$ su di un insieme di caratteri $\texttt{C'} = \{\texttt{c}_1, \texttt{c}_2, \ldots, \texttt{c}_n, \texttt{c}_{n+1}\}$. Siano $\texttt{x}$ e $\texttt{y}$ i due caratteri con frequenza minima in $\texttt{C}'$.

Proviamo che l'alg. $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero T con $B[T]$ *minimo*

Per induzione sul numero n dei caratteri $\texttt{c}_1, \texttt{c}_2, \ldots, \texttt{c}_n$.

Per $n = 2$ l'algoritmo $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero del tipo

che è chiaramente ottimo (è l'unico albero con due foglie!).

Supponiamo quindi che $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero ottimo quando $|\texttt{C}| = n$, e consideriamo l'esecuzione di $\texttt{Huffman}(\texttt{C}', \texttt{f})$ su di un insieme di caratteri $\texttt{C'} = \{\texttt{c}_1, \texttt{c}_2, \ldots, \texttt{c}_n, \texttt{c}_{n+1}\}$. Siano $\texttt{x}$ e $\texttt{y}$ i due caratteri con frequenza minima in $\texttt{C}'$. La *prima* cosa che $\texttt{Huffman}(\texttt{C}', \texttt{f})$ fa è di associare a $\texttt{x}$ e $\texttt{y}$ due nodi (foglie) fratelli, e poi di richiamare se stesso sull'insieme di n caratteri $\texttt{C} = (\texttt{C'} - \{\texttt{x}, \texttt{y}\}) \cup \{\texttt{z}\}$, dove il carattere $\texttt{z}$ ha frequenza `f(z)=f(x)+f(y)`.

Proviamo che l'alg. $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero T con $B[T]$ *minimo*

Per induzione sul numero n dei caratteri $\texttt{c}_1, \texttt{c}_2, \ldots, \texttt{c}_n$.

Per $n = 2$ l'algoritmo $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero del tipo

che è chiaramente ottimo (è l'unico albero con due foglie!).

Supponiamo quindi che $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero ottimo quando $|\texttt{C}| = n$, e consideriamo l'esecuzione di $\texttt{Huffman}(\texttt{C}', \texttt{f})$ su di un insieme di caratteri $\texttt{C'} = \{\texttt{c}_1, \texttt{c}_2, \ldots, \texttt{c}_n, \texttt{c}_{n+1}\}$. Siano `x` e `y` i due caratteri con frequenza minima in $\texttt{C}'$. La *prima* cosa che $\texttt{Huffman}(\texttt{C}', \texttt{f})$ fa è di associare a `x` e `y` due nodi (foglie) fratelli, e poi di richiamare se stesso sull'insieme di n caratteri $\texttt{C} = (\texttt{C'} - \{\texttt{x}, \texttt{y}\}) \cup \{\texttt{z}\}$, dove il carattere `z` ha frequenza `f(z)=f(x)+f(y)`. Per ipotesi induttiva `Huffman` chiamato su `C` produce un albero O *ottimo*.

Proviamo che l'alg. $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero T con $B[T]$ *minimo*

Per induzione sul numero n dei caratteri $\texttt{c}_1, \texttt{c}_2, \ldots, \texttt{c}_n$.

Per $n = 2$ l'algoritmo $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero del tipo

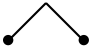
che è chiaramente ottimo (è l'unico albero con due foglie!).

Supponiamo quindi che $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero ottimo quando $|\texttt{C}| = n$, e consideriamo l'esecuzione di $\texttt{Huffman}(\texttt{C}', \texttt{f})$ su di un insieme di caratteri $\texttt{C'} = \{\texttt{c}_1, \texttt{c}_2, \ldots, \texttt{c}_n, \texttt{c}_{n+1}\}$. Siano $\texttt{x}$ e $\texttt{y}$ i due caratteri con frequenza minima in $\texttt{C}'$. La *prima* cosa che $\texttt{Huffman}(\texttt{C}', \texttt{f})$ fa è di associare a $\texttt{x}$ e $\texttt{y}$ due nodi (foglie) fratelli, e poi di richiamare se stesso sull'insieme di n caratteri $\texttt{C} = (\texttt{C'} - \{\texttt{x}, \texttt{y}\}) \cup \{\texttt{z}\}$, dove il carattere $\texttt{z}$ ha frequenza $\texttt{f(z)=f(x)+f(y)}$. Per ipotesi induttiva $\texttt{Huffman}$ chiamato su $\texttt{C}$ produce un albero O *ottimo*. Occorre provare che anche l'albero N prodotto da da $\texttt{Huffman}$ chiamato su $\texttt{C}'$ (con $n + 1$) caratteri è ottimo.

Proviamo che l'alg. $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero T con $B[T]$ *minimo*

Per induzione sul numero n dei caratteri $\texttt{c}_1, \texttt{c}_2, \ldots, \texttt{c}_n$.

Per $n = 2$ l'algoritmo $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero del tipo



che è chiaramente ottimo (è l'unico albero con due foglie!).

Supponiamo quindi che $\texttt{Huffman}(\texttt{C}, \texttt{f})$ produce un albero ottimo quando $|\texttt{C}| = n$, e consideriamo l'esecuzione di $\texttt{Huffman}(\texttt{C}', \texttt{f})$ su di un insieme di caratteri $\texttt{C'} = \{\texttt{c}_1, \texttt{c}_2, \ldots, \texttt{c}_n, \texttt{c}_{n+1}\}$. Siano x e y i due caratteri con frequenza minima in C'. La *prima* cosa che $\texttt{Huffman}(\texttt{C}', \texttt{f})$ fa è di associare a x e y due nodi (foglie) fratelli, e poi di richiamare se stesso sull'insieme di n caratteri $\texttt{C} = (\texttt{C'} - \{\texttt{x}, \texttt{y}\}) \cup \{\texttt{z}\}$, dove il carattere z ha frequenza f(z)=f(x)+f(y). Per ipotesi induttiva Huffman chiamato su C produce un albero O *ottimo*. Occorre provare che anche l'albero N prodotto da da Huffman chiamato su C' (con $n + 1$) caratteri è ottimo.

Intanto ricordiamo, dal Fatto 2, che per l'albero N prodotto da Huffman chiamato su C' vale $\boxed{B(N) = B(O) + (f(\texttt{x}) + f(\texttt{y}))}$

Che succederebbe invece se l'albero N *non* fosse ottimo?

Che succederebbe invece se l'albero N *non* fosse ottimo?

Succederebbe che esiste un'altro albero M (lui sì ottimo) sugli $n + 1$ caratteri $\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_n, \mathtt{c}_{n+1}$ per cui $B(M) < B(N) = B(O) + (f(\mathtt{x}) + f(\mathtt{y}))$

Che succederebbe invece se l'albero N *non* fosse ottimo?

Succederebbe che esiste un'altro albero M (lui sì ottimo) sugli $n+1$ caratteri $c_1, c_2, \ldots, c_n, c_{n+1}$ per cui $B(M) < B(N) = B(O) + (f(\mathrm{x}) + f(\mathrm{y}))$

Dal Fatto 1 (lo ricordate, vero?) abbiamo che: *esiste sicuramente un albero ottimo T in cui i due caratteri x, y di frequenze minime appaiono nell'albero T in due foglie alla profondità massima, e sono fratelli.*

Che succederebbe invece se l'albero N *non* fosse ottimo?

Succederebbe che esiste un'altro albero M (lui sì ottimo) sugli $n+1$ caratteri $\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_n, \mathtt{c}_{n+1}$ per cui $B(M) < B(N) = B(O) + (f(\mathtt{x}) + f(\mathtt{y}))$

Dal Fatto 1 (lo ricordate, vero?) abbiamo che: *esiste sicuramente un albero ottimo T in cui i due caratteri x, y di frequenze minime appaiono nell'albero T in due foglie alla profondità massima, e sono fratelli.*

Eliminando tali due foglie dall'albero T e rendendo il loro padre $\mathtt{z}$ foglia (con associata frequenza `f(z)=f(x)+f(y)`, e procedendo come nel Fatto 2, otteniamo un albero T'

Che succederebbe invece se l'albero N *non* fosse ottimo?

Succederebbe che esiste un'altro albero M (lui sì ottimo) sugli $n+1$ caratteri $\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_n, \mathtt{c}_{n+1}$ per cui $B(M) < B(N) = B(O) + (f(\mathtt{x}) + f(\mathtt{y}))$

Dal <u>Fatto 1</u> (lo ricordate, vero?) abbiamo che: *esiste sicuramente un albero ottimo T in cui i due caratteri x, y di frequenze minime appaiono nell'albero T in due foglie alla profondità massima, e sono fratelli.*

Eliminando tali due foglie dall'albero T e rendendo il loro padre `z` foglia (con associata frequenza `f(z)=f(x)+f(y)`, e procedendo come nel <u>Fatto 2</u>, otteniamo un albero T' (con una foglia in meno rispetto a T)

Che succederebbe invece se l'albero N *non* fosse ottimo?

Succederebbe che esiste un'altro albero M (lui sì ottimo) sugli $n+1$ caratteri $\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_n, \mathtt{c}_{n+1}$ per cui $B(M) < B(N) = B(O) + (f(\mathtt{x}) + f(\mathtt{y}))$

Dal Fatto 1 (lo ricordate, vero?) abbiamo che: *esiste sicuramente un albero ottimo T in cui i due caratteri x, y di frequenze minime appaiono nell'albero T in due foglie alla profondità massima, e sono fratelli.*

Eliminando tali due foglie dall'albero T e rendendo il loro padre `z` foglia (con associata frequenza `f(z)=f(x)+f(y)`, e procedendo come nel Fatto 2, otteniamo un albero T' (con una foglia in meno rispetto a T) per cui vale $B(T) = B(T') + f(x) + f(y)$.

Che succederebbe invece se l'albero N *non* fosse ottimo?

Succederebbe che esiste un'altro albero M (lui sì ottimo) sugli $n+1$ caratteri $\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_n, \mathtt{c}_{n+1}$ per cui $B(M) < B(N) = B(O) + (f(\mathtt{x}) + f(\mathtt{y}))$

Dal Fatto 1 (lo ricordate, vero?) abbiamo che: *esiste sicuramente un albero ottimo T in cui i due caratteri x, y di frequenze minime appaiono nell'albero T in due foglie alla profondità massima, e sono fratelli.*

Eliminando tali due foglie dall'albero T e rendendo il loro padre $\mathtt{z}$ foglia (con associata frequenza `f(z)=f(x)+f(y)`, e procedendo come nel Fatto 2, otteniamo un albero T' (con una foglia in meno rispetto a T) per cui vale $B(T) = B(T') + f(x) + f(y)$.
Riassumendo, abbiamo che
$B(T') + \mathtt{f(x)+f(y)} = B(T)$

Che succederebbe invece se l'albero N *non* fosse ottimo?

Succederebbe che esiste un'altro albero M (lui sì ottimo) sugli $n+1$ caratteri $\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_n, \mathtt{c}_{n+1}$ per cui $B(M) < B(N) = B(O) + (f(\mathtt{x}) + f(\mathtt{y}))$

Dal Fatto 1 (lo ricordate, vero?) abbiamo che: *esiste sicuramente un albero ottimo T in cui i due caratteri x, y di frequenze minime appaiono nell'albero T in due foglie alla profondità massima, e sono fratelli.*

Eliminando tali due foglie dall'albero T e rendendo il loro padre $\mathtt{z}$ foglia (con associata frequenza $\mathtt{f(z)=f(x)+f(y)}$, e procedendo come nel Fatto 2, otteniamo un albero T' (con una foglia in meno rispetto a T) per cui vale $B(T) = B(T') + f(x) + f(y)$.
Riassumendo, abbiamo che
$B(T') + \mathtt{f(x)+f(y)} = B(T) = B(M)$ (sia T che M sono ottimi)

Che succederebbe invece se l'albero N *non* fosse ottimo?

Succederebbe che esiste un'altro albero M (lui sì ottimo) sugli $n+1$ caratteri $\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_n, \mathtt{c}_{n+1}$ per cui $B(M) < B(N) = B(O) + (f(\mathtt{x}) + f(\mathtt{y}))$

Dal Fatto 1 (lo ricordate, vero?) abbiamo che: *esiste sicuramente un albero ottimo T in cui i due caratteri x, y di frequenze minime appaiono nell'albero T in due foglie alla profondità massima, e sono fratelli.*

Eliminando tali due foglie dall'albero T e rendendo il loro padre $\mathtt{z}$ foglia (con associata frequenza `f(z)=f(x)+f(y)`, e procedendo come nel Fatto 2, otteniamo un albero T' (con una foglia in meno rispetto a T) per cui vale $B(T) = B(T') + f(x) + f(y)$.
Riassumendo, abbiamo che

$$B(T') + \mathtt{f(x)+f(y)} = B(T) = B(M) \text{ (sia } T \text{ che } M \text{ sono ottimi)}$$
$$< B(N) = B(O) + (f(\mathtt{x}) + f(\mathtt{y}))$$

Che succederebbe invece se l'albero N *non* fosse ottimo?

Succederebbe che esiste un'altro albero M (lui sì ottimo) sugli $n+1$ caratteri $\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_n, \mathtt{c}_{n+1}$ per cui $B(M) < B(N) = B(O) + (f(\mathtt{x}) + f(\mathtt{y}))$

Dal Fatto 1 (lo ricordate, vero?) abbiamo che: *esiste sicuramente un albero ottimo T in cui i due caratteri x, y di frequenze minime appaiono nell'albero T in due foglie alla profondità massima, e sono fratelli.*

Eliminando tali due foglie dall'albero T e rendendo il loro padre $\mathtt{z}$ foglia (con associata frequenza `f(z)=f(x)+f(y)`, e procedendo come nel Fatto 2, otteniamo un albero T' (con una foglia in meno rispetto a T) per cui vale $B(T) = B(T') + f(x) + f(y)$.
Riassumendo, abbiamo che

$$\begin{aligned} B(T') + \mathtt{f(x)+f(y)} = B(T) &= B(M) \text{ (sia } T \text{ che } M \text{ sono ottimi)} \\ &< B(N) = B(O) + (f(\mathtt{x}) + f(\mathtt{y})) \\ &\Rightarrow B(T') < B(O) \end{aligned}$$

Che succederebbe invece se l'albero N *non* fosse ottimo?

Succederebbe che esiste un'altro albero M (lui sì ottimo) sugli $n+1$ caratteri $\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_n, \mathtt{c}_{n+1}$ per cui $B(M) < B(N) = B(O) + (f(\mathtt{x}) + f(\mathtt{y}))$

Dal Fatto 1 (lo ricordate, vero?) abbiamo che: *esiste sicuramente un albero ottimo T in cui i due caratteri x, y di frequenze minime appaiono nell'albero T in due foglie alla profondità massima, e sono fratelli.*

Eliminando tali due foglie dall'albero T e rendendo il loro padre $\mathtt{z}$ foglia (con associata frequenza `f(z)=f(x)+f(y)`, e procedendo come nel Fatto 2, otteniamo un albero T' (con una foglia in meno rispetto a T) per cui vale $B(T) = B(T') + f(x) + f(y)$.
Riassumendo, abbiamo che

$$\begin{aligned} B(T') + \mathtt{f(x)+f(y)} = B(T) &= B(M) \text{ (sia } T \text{ che } M \text{ sono ottimi)} \\ &< B(N) = B(O) + (f(\mathtt{x}) + f(\mathtt{y})) \\ &\Rightarrow B(T') < B(O) \text{ il che è chiariamente} \end{aligned}$$

assurdo, in quanto l'albero O era per ipotesi ottimo su n caratteri.

Che succederebbe invece se l'albero N *non* fosse ottimo?

Succederebbe che esiste un'altro albero M (lui sì ottimo) sugli $n+1$ caratteri $\mathtt{c}_1, \mathtt{c}_2, \ldots, \mathtt{c}_n, \mathtt{c}_{n+1}$ per cui $B(M) < B(N) = B(O) + (f(\mathtt{x}) + f(\mathtt{y}))$

Dal Fatto 1 (lo ricordate, vero?) abbiamo che: *esiste sicuramente un albero ottimo T in cui i due caratteri x, y di frequenze minime appaiono nell'albero T in due foglie alla profondità massima, e sono fratelli.*

Eliminando tali due foglie dall'albero T e rendendo il loro padre $\mathtt{z}$ foglia (con associata frequenza `f(z)=f(x)+f(y)`, e procedendo come nel Fatto 2, otteniamo un albero T' (con una foglia in meno rispetto a T) per cui vale $B(T) = B(T') + f(x) + f(y)$.
Riassumendo, abbiamo che

$$\begin{aligned} B(T') + \mathtt{f(x)+f(y)} = B(T) &= B(M) \text{ (sia } T \text{ che } M \text{ sono ottimi)} \\ &< B(N) = B(O) + (f(\mathtt{x}) + f(\mathtt{y})) \\ &\Rightarrow B(T') < B(O) \text{ il che è chiaramente} \end{aligned}$$

assurdo, in quanto l'albero O era per ipotesi ottimo su n caratteri. Ne segue che anche N è ottimo e pertanto `Huffman` produce sempre alberi ottimi per qualsiasi insieme di caratteri.