da determinare:

funzione di encoding $e : C \to \{0,1\}^*$

proprietà che e deve avere:

- invertibile $\longrightarrow a \neq b \Rightarrow e(a) \neq e(b)$
- ammettere algoritmi efficienti, come e^{-1}

<u>Fixed-length encoding</u>: associa ad ogni carattere di C una stringa di 0 e 1 della stessa lunghezza

è l'idea più semplice possibile. Nota: ignora totalmente le frequenze

→ posso usare questa informazione per migliorare le cose?

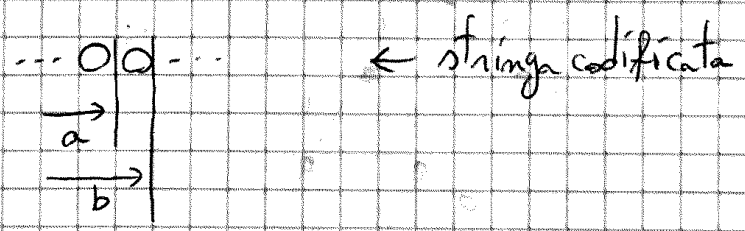
Idea: associare a caratteri più frequenti stringhe (codeword) più corte, e viceversa → <u>variable-length encoding</u>

Esempio:

	a	b	c	d	e	f
frequenza (%)	45	13	12	16	9	5
codeword	0	00	01	1	100	101

è un buon encoding?

No: $e(aa) = e(b)$



← stringa codificata

problema: dopo aver visto il 1° carattere (bit) non so se fermarmi perché ho visto una codeword completa oppure solo il <u>prefisso</u> di una codeword più lunga. Quindi il problema

è che ∃ codeword che sono prefissi di altre codeword!

Quindi, buone codifiche devono:

- lunghezza variabile delle codeword (per efficienza)
- il codice deve essere libero da prefissi ("prefix code"), cioè $\nexists\ a, b \in C$ tali che $e(a)$ è un prefisso di $e(b)$

Esempio:

a	b	c	d	e	f
45	13	12	16	9	5
0	101	100	111	1101	1100

→ codice a lunghezza variabile libero da prefissi

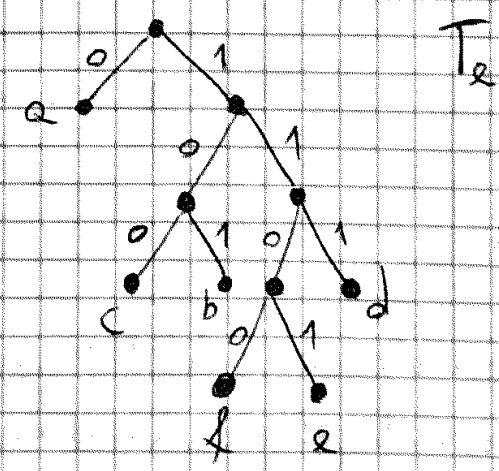
$$\text{n° di bit} = \frac{100K}{100}\left(45\cdot 1 + 13\cdot 3 + 12\cdot 3 + 16\cdot 3 + 9\cdot 4 + 5\cdot 4\right)$$

$$= 224K$$

risparmio di circa il 25% rispetto alla fixed-length

Decodifica:

uso un TRIE, cioè un albero binario che ha ogni lato etichettato con un valore binario (0 o 1)



Ogni codice binario può essere rappresentato in modo compatto con un albero binario T. Se il codice è libero da prefissi, le codeword corrispondono solo alle foglie.

File = bits + "albero di codifica"

Esempio:

stringa codificata: 1100|0|100|1101

stringa decodificata: f a c e

parto da radice, scendo seguendo la stringa fino ad una foglia, restituisco il carattere associato alla foglia; ripeto

algoritmo di decodifica

↳ è lineare nella lunghezza della stringa

Come stabilisco se un codice/albero è migliore di un altro?

$\forall c \in C$ ho $f(c)$ → frequenza di c

$d_T(c)$ = profondità di c in T → albero associato al codice

esempio: $d_T(a) = 1$

$d_T(c) = d_T(b) = d_T(d) = 3$

$d_T(e) = d_T(f) = 4$

$d_T(c)$ = lunghezza parola di codice del carattere c

funzione di costo:

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

somma pesata delle lunghezze delle codeword

perché è una buona funzione di costo?

perché la dimensione del file compresso è

$$B(T) \cdot \frac{|F|}{100}$$

$|F|$ → n° caratteri file non compresso

$\Rightarrow$ $B(T)$, a meno di una costante (100), è ~~il fattore~~ l'inverso del fattore di compressione: minore è $B(T)$ e migliore è il fattore di compressione.

(Osservazione: stiamo assumendo di dover associare una codeword a ogni singolo carattere, il che non è sempre la cosa migliore da fare:

abbabbabbabb....

Cioè, vedremo il codice libero da prefissi ottimo tra tutti i codici che associano una codeword a ogni singolo carattere)

Osservazione: un albero ottimo è sempre pieno (cioè i nodi interni hanno 2 figli)

$\Rightarrow$ spazio sottoproblemi: alberi pieni a n foglie

Goal: trovare un albero ottimo T, cioè che minimizza $B(T)$