Compressione dati

Capitolo 11

Leonora Bianchi
IDSIA-DTI-SUPSI

La compressione è *ovunque*!

- Nelle reti di computer: compressione pacchetti inviati
- file di backup
- FAX
- …

N.B.: trasmettere qui è sinonimo di comprimere

Indice

L'idea

IEEE ➔ "I3E"

"Vrbl-lngt-ncdng"

"v. sopra" (dizionario)

Come fareste voi?

Testo non compresso = 312 car.

"Si sentì meglio, dopo aver preso quella decisione di comportarsi come un gabbiano qualsiasi. Basta! Non avrebbe più dovuto dar retta a quel demone che l'istigava a imparare nuove cose. Basta d'ora in poi con le sfide, basta con i fallimenti. Ah, era bello smettere di pensare, e volare tranquilli nel buio, verso le luci occhieggianti della costa."*

Testo compresso = ?

- IEEE ➔ "I3E"
- "S snt mgl, dp vr prs …"
- "v. sopra" (dizionario)

* Il gabbiano Jonathan Livingston, R.Bach

Run-length encoding (IEEE ➔ "I3E")

L'idea

AAABBBBAAABBCCCCC
3A4B3ABB5C

Limitazioni

• Comprime solo se ci sono tante sequenze con più di 2 lettere uguali

• Se ci sono cifre, come distinguo tra caratteri di compressione e caratteri del testo?

• tanto più è "ricco" il testo da comprimere, tanto meno efficace è la compressione

Run-length encoding - stringhe qualunque

Migliorie rispetto all'idea base

- faccio precedere il contatore da un carattere di "escape"

- per i contatori uso *lettere* anziché *cifre*

Esempio

Carattere di escape = Q
Contatori: A=1, B=2, C=3, D=4,…, J=10, L=11

AAABBBBAAABBCCCCCCCCCC
AAAQDBAAABBQJC

Run-length encoding - sequenze binarie

L'idea Alternanza di zero e uno: il primo contatore sia per lo zero

Originale	01111001010001100110011000000	29 bit ≈ 4 byte
Cifre decimali	1-4-2-1-1-1-3-2-2-2-2-2-6	25•8 = 200 byte
Cifre con numero fissato di bit (qui 3)	001100010001001001011010010010010010110	3•13 = 39 bit ≈ 5 byte
Con 2 bit: contatore ≤ 3	(1 3 0 1 2 1 1 1 3 2 2 2 2 2 3 0 3) 011100011001010111101010101011100 11	2•17 = 34 bit ≈ 4.2 byte

Quando il run-length comprime veramente

Quando i contatori contengono spesso valori vicino a $2^{\text{numero bit contatore}} - 1$

Originale	0000001111000001111111000000 01111111111	42 bit ≈ 5.3 byte
Cifre decimali	6-4-5-7-6-7-0-7	15•8 = 120 byte
Con 3 bit: contatore ≤ 7	110100101111110111000111	3•8 = 24 bit = 3 byte

Variable-length encoding ("Vrbl-lngt-ncdng")

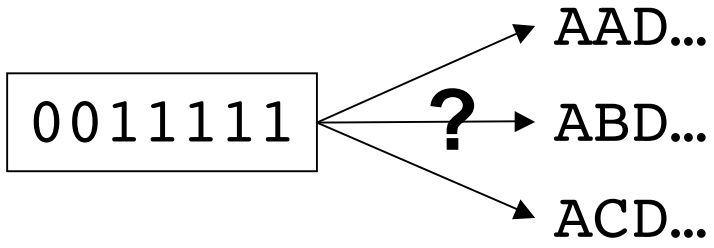
L'idea: riscrivere le stesse cose, ma risparmiando spazio

Un carattere di solito occupa 8 bit = 1 byte
Se uso *meno bit* per i caratteri più frequenti, risparmio spazio!

A	0
B	01
C	011
D	111

Emerge un problema:

Riconoscere un codice dal successivo:



Due soluzioni non soddisfacenti

1. Usare un separatore (ma anche il separatore va codificato)

2. Leggere tutta la sequenza per discriminare fra le varie possibilità (ma l'albero che si costruisce può occupare molto spazio!)

Una soluzione soddisfacente: la regola del prefisso

Un codice ***non sia prefisso*** di un altro codice

NO

A	0
B	01 = [A]1
C	011 = [B]1
D	111

SI

A	11
B	00
C	010
D	10
E	011

Esempio

A	11
B	00
C	010
D	10
E	011

11000101010111011000111 1

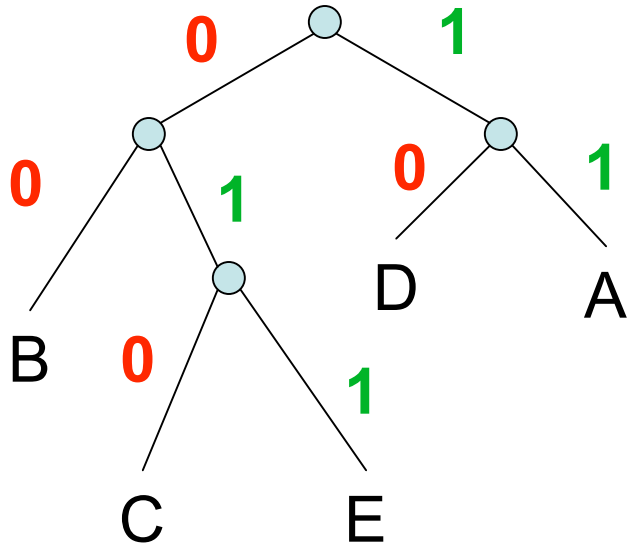
24 bit = 3 byte

11 00 010 10 10 11 10 11 00 011 11

ABCDDADABEA

11×8 = 88 byte

Come costruire un codice che soddisfa la regola del prefisso



- Più bassa è l'altezza dell'albero, più efficace è la compressione
- Ma a parità di albero, l'efficacia dipende dal file da comprimere

Misura dell'efficacia: cammino ponderato

Definizione: **cammino ponderato**

$$\sum_{i \in foglie} frequenza(i) \times livello(i)$$

La codifica di Huffman (1952)

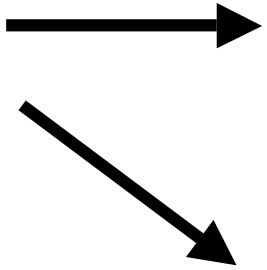
Ovvero, come creare un codice di compressione “su misura”, che minimizza il cammino ponderato

1. Calcolo la frequenza con cui ogni carattere compare nel file da comprimere

2. Costruisco l’albero binario corrispondente al codice di compressione, partendo dai caratteri con frequenza minore

Diversi codici di Huffman

	frequenza
A	0.4
B	0.2
C	0.2
D	0.1
E	0.1



A	1
B	010
C	011
D	000
E	001

c.p.= 2.2

A	1
B	01
C	000
D	0010
E	0011

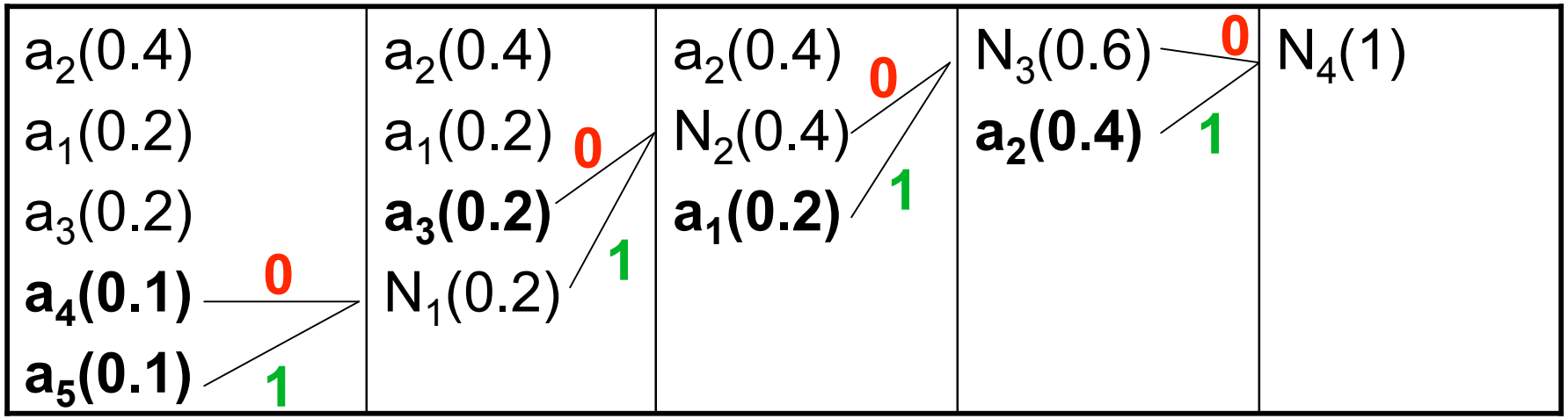
c.p.= 2.2

Teorema: nessun albero di codifica ha cammino ponderato < del cammino di Huffman.

➔ Huffman è ottimale

Costruzione del codice di Huffman

Lettera	Frequenza	Codice bin.
a_2	0.4	1
a_1	0.2	01
a_3	0.2	000
a_4	0.1	0010
a_5	0.1	0011



Codifica di Shannon-Fano

(Simile a Huffman, scoperto 2 anni prima, ma subito rimpiazzato da Huffman)

1. Ordinamento dell'insieme dei simboli in funzione delle frequenze
2. Divisione ricorsiva in due sottoinsiemi con frequenza approssimativamente uguale

Limitazioni della codifica di Huffman

• nel file compressone all'inizio bisogna includere l'albero di decodifica

• in files disomogenei, la codifica potrebbe rispecchiare solo una piccola parte del file

• codifica e decodifica "run-time" impossibile, perché bisogna disporre della statistica (frequenze) completa

➔ Codifica di Huffman **adattativa**

Indice

L'idea

IEEE ➔ "I3E"

"Vrbl-lngt-ncdng"

"v. sopra" (dizionario)

Medodi di compressione basati su dizionario
ovvero
algoritmi “LZ”

Vantaggi:
- ok per dati eterogenei (immagini, testo, database)
- no info a priori necessarie per la codifica e decodifica

Rilevanza:
LZ77 + Huffman ➔ zip
➔ gzip
➔ png
➔ jpeg

Due tipi di dizionario

Dizionario *im*plicito

"v. sopra"

LZ77

(Lempel - Ziv, 1977)

LZSS

(Storer - Szymanski, 1982)

.....

Dizionario *es*plicito

"v. pag.256, riga 9"

LZ78

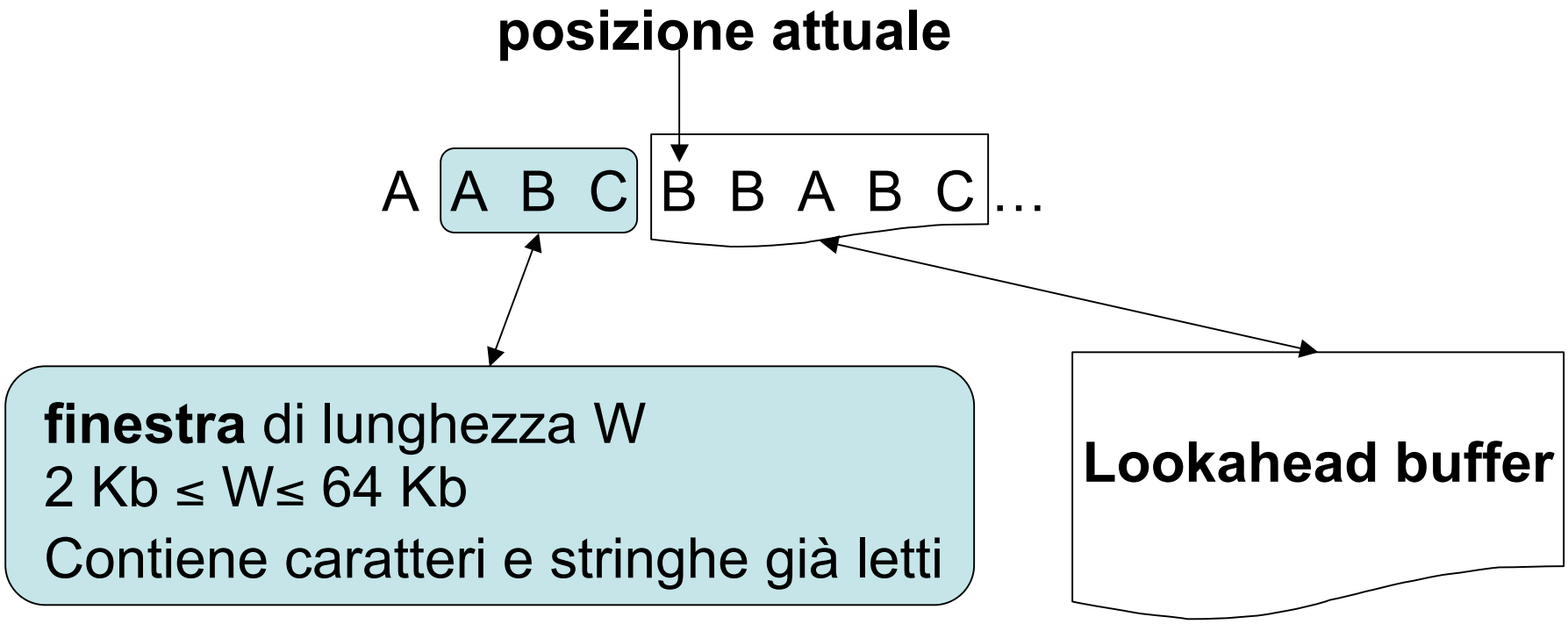
(Lempel - Ziv, 1978)

LZW

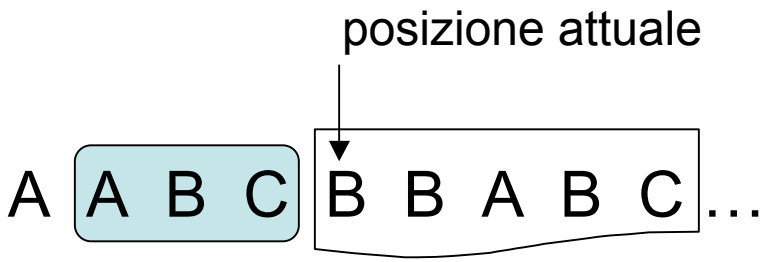
(Welch, 1984)

.....

LZ77 - definizioni



LZ77 - codifica



Carattere successivo alla sequenza codificata

(b, l) C

Numero di passi indietro da fare nella finestra dalla posizione corrente per arrivare all'inizio della sequenza identificata nel lookahead buffer

Lunghezza della sequenza identificata

LZ77 - esempio

1	2	3	4	5	6	7	8	9
A	A	B	C	B	B	A	B	C

Passo	Posizione attuale	Corrispondenza	(b,l)c
1	1	-	(0,0)A
2	2	A	(1,1)B
3	4	-	(0,0)C
4	5	B	(2,1)B
5	7	ABC	(5,3)C

Oss: ogni volta W := W+(l+1)

pa:= pa + (l+1)

LZ77 - migliorìe

1. Compressione delle triplette (b,l)C

Ad esempio usando Huffman

2. (0,0)C è uno spreco, usare ad esempio <escape>C [LZSS]

3. Pkzip, Zip, gzip, ARJ = LZ77 + Huffman + dettagli

Limitazione della finestra

Assume implicitamente che parole simili siano vicine, ma a volte no!

esempio

Due tipi di dizionario

Dizionario *im*plicito

"v. sopra"

LZ77

(Lempel - Ziv, 1977)

LZSS

(Storer - Szymanski, 1982)

.....

Dizionario *es*plicito

"v. pag.256, riga 9"

LZ78

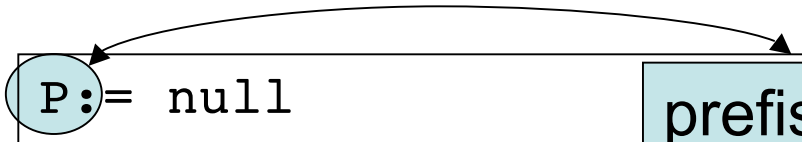
(Lempel - Ziv, 1978)

LZW

(Welch, 1984)

.....

LZ78, Con dizionario *esplicito*




```
P:= null
Loop
      exit when {non c'è un prossimo carattere}
      C:= {prossimo carattere}
      if P + C {nel dizionario} then
            P := P + C
      else
            {scrivi l'indice di P}
            {scrivi C}
            {aggiungi P + C al dizionario}
            P := null
      end if
End loop
If P /= null then
      {scrivi l'indice di P}
End if
```


➔ esempio

LZ78 - **de**codifica

Per ogni codice ricevuto <i C>:

- leggo nel dizionario la stringa corrispondente ad i, s(i)
- aggiungo nel dizionario s(i) + C

Passo	Codice	Stringa ricostruita	Dizionario
1	0 A	A	1 A
2	0 B	AB	2 B
3	2 C	ABC	3 BC
4	3 A	ABCBCA	4 BCA
5	2 A	ABCBCABA	5 BA

LZ78 - migliorìe

La più famosa:
Ridurre la codifica da coppie di codice <i C> ad <i>

LZW

Due tipi di dizionario

Dizionario *im*plicito

"v. sopra"

LZ77

(Lempel - Ziv, 1977)

LZSS

(Storer - Szymanski, 1982)

.....

Dizionario *es*plicito

"v. pag.256, riga 9"

LZ78

(Lempel - Ziv, 1978)

LZW

(Welch, 1984)

.....

LZW, Con dizionario esplicito

- trasmette solo codici
- il dizionario deve essere inizializzato con tutti i singoli caratteri presenti nel file

- L'idea:
  - continuo ad estendere la stringa letta P con un nuovo carattere finché trovo che P+C $\notin$ dizionario
  - trasmetto idice(P)
  - aggiungo P+C al dizionario

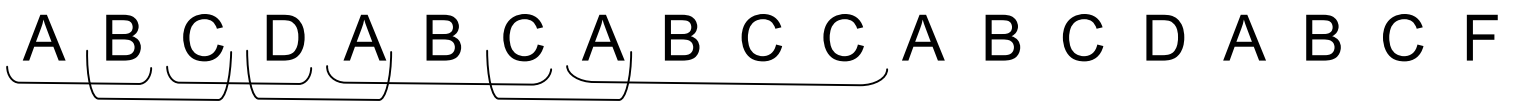
LZW - pseudocodice

```
P:= null
Loop
      exit when {non c'è un prossimo carattere}
      C:= {prossimo carattere}
      if P + C {nel dizionario} then
            P := P + C (*)
      else
            {scrivi l'indice di P}
            {scrivi C}
            {aggiungi P + C al dizionario}
            P := null    P := C (**)
      end if
End loop
If P /= null then
      {scrivi l'indice di P}
End if
```


➔ esempio

esempio

Nel dizionario, dopo A B C D E F



Invece, con LZ78:

LZW - decodifica

Dizionario iniziale

1	A
2	B
3	C
4	D
5	E
6	F

Messaggio codificato

1	2	3	4	7	3	11	12

LZW - decodifica 1

```
{leggi primo codice}
P := {stringa corrispondente al codice}
{scrivi P}
Loop
       exit when {non esiste prossimo codice}
       {leggi prossimo codice}
       N := {stringa corrispondente al codice}
       {scrivi N}
       C := {primo carattere di N}
       {aggiungi P+C al dizionario}
       P := N
End loop
```

Limitazione della decodifica 1

Il dizionario viene creato in ritardo rispetto alla codifica

codifica		decodifica	
codice creato	agg. diz.	codice letto	agg. diz.
1	7 AB	1	
2	8 BC	2	7 AB
3	9 CD	3	8 BC
4	10 DA	4	9 CD
7	11 ABC	7	10 DA
3	12 CA	3	11 ABC
11	13 ABCC	11	12 CA

→ esempio

LZW - decodifica 2

```
{leggi primo codice}
P := {stringa corrispondente al codice}
{scrivi P}
Loop
       exit when {non esiste prossimo codice}
       {leggi prossimo codice}
       if {codice non nel dizionario} then
              N := P + C
       else
              N := {stringa corrispondente al codice}
       end if
       {scrivi N}
       C := {primo carattere di N}
       {aggiungi P+C al dizionario}
       P := N
End loop
```