

Metodo di Compressione LZ78

Indice

1	Introduzione	2
2	Algoritmo di compressione LZ78	3
2.1	<i>Codifica</i>	3
2.1.1	Algoritmo di codifica	3
2.2	<i>Decodifica</i>	6
2.2.1	Algoritmo di decodifica	6
3	Conclusioni	7

1. Introduzione

LZ78 costituisce la tecnica fondamentale che sta alla base di un insieme di algoritmi di compressione di Lempel e Ziv. In LZ77 il dizionario è definito da una finestra di testo di dimensione pressata. Durante la codifica, la finestra scorre sul testo da comprimere. Quindi, la possibilità di trovare un match nel dizionario per la stringa corrente è legata ai dati contenuti nella window in quel momento. Infatti, se la stringa da codificare era già stata incontrata molto tempo prima, il codificatore non ha alcuna memoria di questo fatto in quanto la stringa in questione non è più nella window, cioè è uscita dal dizionario. In LZ77, inoltre, la lunghezza massima di un match che è possibile determinare è limitata dalla dimensione del buffer look-ahead. Sembrerebbe, allora, naturale risolvere i problemi sopra menzionati accrescendo la taglia e della sliding window e del buffer look-ahead. Tuttavia, questa soluzione non porta necessariamente a dei miglioramenti. Aumentando la dimensione della window, ad esempio, necessitiamo di un numero maggiore di bit per rappresentare un riferimento ad un match nella window. Per match corti, in particolare, potremmo avere una codifica più lunga del match stesso. Ma, il vero svantaggio proviene dall'aumento della taglia del buffer. Poiché le operazioni di confronto di stringhe tra il buffer look-ahead e le stringhe nella window procedono in maniera sequenziale, il run-time cresce in maniera direttamente proporzionale alla lunghezza del buffer.

LZ78 abbandona il concetto di sliding window e usa un differente approccio nel costruire e gestire il dizionario. Costruisce il dizionario inserendo, di volta in volta, stringhe composte da: una sottostringa, che individua un match nel dizionario, seguita da un carattere che rompe il match. In questo modo, non c'è alcuna restrizione su quanto indietro può essere cercato un match (back-reference), cioè non c'è un limite prestabilito alla dimensione del dizionario. Ciò incrementa la possibilità di trovare un match. In più il dizionario ha una caratteristica: se una stringa è memorizzata nel dizionario, allora lo è anche un suo prefisso. Usando questo approccio, la lunghezza possibile di un match non è limitata dalla dimensione di un buffer come in LZ77. Inoltre, il codificatore non ha necessità di mandare esplicitamente il dizionario al decodificatore. Quest'ultimo, infatti, è in grado di ricostruirlo automaticamente. Vedremo come.

2 Algoritmo di compressione LZ78

2.1 Codifica

All'inizio della codifica il dizionario è vuoto. Per definizione, il dizionario vuoto contiene un'unica stringa codificata: la stringa vuota. Per spiegare il principio di codifica, supponiamo che una parte dell'input sia già stata codificata. Il dizionario, quindi, contiene già qualche stringa e assumiamo che ciascuna di queste sia identificata da un indice. L'algoritmo, all' i -esimo passo, cerca nella sequenza che rimane da codificare, il più lungo prefisso che corrisponda a qualche indice nel dizionario. Cerca un nuovo prefisso partendo dal prefisso vuoto. Sia Pref il prefisso finora determinato. Legge il carattere successivo C nella sequenza. Se Pref + C (con + indichiamo l'operatore binario che concatena due stringhe) è presente nel dizionario, il prefisso Pref è esteso con il carattere C. Il processo di estensione continua fino a quando Pref + C è una stringa che non è presente nel dizionario. A questo punto, il passo si conclude. L'algoritmo manda in output la coppia costituita dall'indice nel dizionario corrispondente a Pref e dall'ultimo carattere letto C. Quindi, aggiunge l'intera stringa Pref + C al primo indice libero nel dizionario. Pref diventa nuovamente la stringa vuota, e comincia la ricerca di un nuovo prefisso. Si noti che, il caso in cui Pref è la stringa vuota e il dizionario non contiene la stringa con il solo carattere C (per esempio, questo accade sempre nel primo passo di codifica) viene considerato come match con la stringa nulla. Perciò viene mandato in output '0', che è l'indice nel dizionario per la stringa nulla, seguito dal carattere letto C. Quest'ultimo, coerentemente alla modalità di codifica, è aggiunto al dizionario. Riassumendo, l'output di questo algoritmo è una serie di coppie indice-carattere (I,C). Ogni volta che una coppia (I,C) è data in output, la stringa del dizionario corrispondente a I è estesa con il carattere C e la stringa risultante da questa estensione è aggiunta al dizionario.

2.1.1 Algoritmo di codifica

Illustriamo di seguito l'algoritmo tramite pseudo-codice, assumendo che un'entry nel dizionario sia composta da un numero intero e da una stringa. Definiamo, inoltre, un tipo di dati coppia con un campo indice di tipo intero ed un campo char di tipo carattere. L'output dell'algoritmo è una lista di coppie. La funzione *Cerca(x,Diz)*

restituisce l'indice della stringa x nel dizionario Diz se presente, altrimenti 0. La funzione $Inserisci(s,Diz)$ aggiunge la stringa s nella prima entry libera in Diz e la funzione $In_lista(L,cop)$ mette in coda a L la coppia cop . S è la sequenza da codificare e con Λ indichiamo la stringa vuota.

Alg_Codifica(S)

```

L = lista_vuota; Pref =  $\Lambda$ ;
Diz = diz_vuoto; cop = coppia_vuota;
while S  $\neq$   $\Lambda$ 
  do
    C = carattere successivo in S;
    if Cerca(Pref + C,Diz)  $\neq$  0
      then Pref = Pref + C;
    else cop.indice = Cerca(Pref,Diz);
         cop.char = C;
         In_lista(L,cop);
         Inserisci(Pref + C,Diz);
         Pref =  $\Lambda$ ;
         cop = coppia_vuota;
    end_if;
  end_while;
if Pref  $\neq$   $\Lambda$ 
  then cop.indice = Cerca(Pref,Diz);
       cop.char =  $\Lambda$ ;
       In_lista(L,cop);
end_if;
return L;
end;
```

Nota che, il secondo *if* gestisce il caso in cui il processo di compressione si concluda determinando un match nel dizionario. Alla lista di output sarà aggiunta, in questo caso, la coppia che ha come indice quello nel dizionario dell'ultimo prefisso determinato e come carattere la stringa vuota.

Esempio: Codifica della sequenza "DAD DADA DADDY"

Passo	C	Pref	Dizionario		Coppia aggiunta ad L
			Indice	Stringa	
1	D	Λ	1	D	(0,D)
2	A	Λ	2	A	(0,A)
3	D	D	-	-	-
3	" "	D	3	D" "	(1," ")
4	D	D	-	-	-
4	A	D	4	DA	(1,A)
5	D	D	-	-	-
5	A	DA	-	-	-
5	" "	DA	5	DA" "	(4," ")
6	D	D	-	-	-
6	A	DA	-	-	-
6	D	DA	6	DAD	(4,D)
7	D	D	-	-	-
7	Y	D	7	DY	(1,Y)

Il primo carattere letto è D e Pref = Λ . La stringa Pref + C = Λ + D = D non è nel dizionario per cui Pref rimane il prefisso vuoto. Il carattere D è aggiunto al dizionario all'indice 1 e la coppia (0,D) è concatenata alla lista di output. Il primo passo si è concluso senza aver determinato alcun prefisso. Andiamo un po' oltre e vediamo cosa accade al passo 6. Pref = Λ perché siamo all'inizio di un nuovo passo. Il codificatore legge D. Pref + C = "D" è nel dizionario all'indice 1. Pref è esteso con il carattere D, ossia Pref = "D". Nessuna stringa è inserita nel dizionario e la ricerca continua. Il carattere successivo è A. La stringa Pref + C = "DA" è presente nel dizionario all'indice 4 e Pref è esteso con A (Pref = "DA"). Quindi, legge D e la stringa Pref + C ="DAD" non ha alcuna corrispondenza nel dizionario. Il passo 6 termina inserendo "DAD" nel dizionario e aggiungendo alla lista di output la coppia (4,D). Il prefisso che è stato trovato in questo passo è la stringa "DA".

La sequenza completa di output è: (0,D), (0,A), (1," "), (4," "), (4,D), (1,Y).

2.2 Decodifica

Come nella codifica, all'inizio il dizionario contiene solo la stringa vuota. Sarà ricostruito dal processo di decodifica. Ad ogni passo, una coppia indice-carattere (I,C) è letta dall'input. L'indice I si riferisce sempre ad una stringa già presente nel dizionario. La stringa nel dizionario corrispondente a I ed il carattere C sono mandati in output e la stringa risultante dalla loro fusione è aggiunta al dizionario. Dopo la decodifica, il dizionario sarà esattamente uguale a quello della codifica.

2.2.1 Algoritmo di decodifica

La funzione *Seleziona(i,Diz)* restituisce la stringa dell'entry di indice i nel dizionario Diz. Se l'indice è 0, restituisce la stringa vuota. La funzione *Estrai(L)* restituisce la coppia successiva nella lista L cancellandola da essa. L'input L è una lista di coppie.

```
Alg_dec ( L )
```

```
Diz = diz_vuoto; c = coppia_vuota;  
str , s_dec =  $\Lambda$ ;
```

```
while L  $\neq$  lista_vuota  
    c = Estrai(L);  
    str = Seleziona(c.indice,Diz);  
    if str  $\neq$   $\Lambda$   
        then s_dec = s_dec + str;  
    end_if;  
    Inserisci(str + c.char,Diz);  
    s_dec = s_dec + c.char;  
    c = coppia_vuota;  
    str =  $\Lambda$ ;  
end_while;
```

```
return s_dec;
```

```
end;
```

Esempio: Decodifica della sequenza (0,D), (0,A), (1," "), (1,A), (4," "), (4,D), (1,Y)

Passo	Coppia	Entry		s_dec	Dizionario
		Indice	Stringa		
1	(0,D)	0	Λ	D	D
2	(0,A)	0	Λ	A	A
3	(1," ")	1	D	D" "	D" "
4	(1,A)	1	D	DA	DA
5	(4," ")	4	DA	DA" "	DA" "
6	(4,D)	4	DA	DAD	DAD
7	(1,Y)	1	D	DY	DY

Al primo passo, l'algoritmo legge la coppia (0,D). L'entry nel dizionario, il cui indice è 0, contiene la stringa vuota. Il carattere, pertanto, è aggiunto al dizionario e diventa il primo carattere nella sequenza decompressa. Il passo 2 è del tutto analogo al passo. Al passo 4, legge la coppia (1,A). La stringa reperita dal dizionario è "D". Aggiunge alla sequenza decompressa il carattere D, quindi il carattere A. Infine, inserisce la stringa "DA" nel dizionario. La sequenza di output completa è: "DAD DADA DADDY".

3. Conclusioni

Riassumendo, con LZ78 ed il suo nuovo modo di concepire il dizionario, non c'è alcuna restrizione su quanto indietro nel testo può essere cercato un match. Come conseguenza diretta di ciò, utilizziamo un unico indice per sottostringhe identiche. Inoltre, l'abolizione dell'uso del buffer look-ahead non pone alcun limite sulla lunghezza di un match determinato e riduce notevolmente il numero di confronti tra stringhe nel processo di codifica. Il tasso di compressione, tuttavia, è superiore a quello di LZ77 solo per testi molto grandi. In questi, infatti, si evincono i miglioramenti apportati dall'assenza di limiti per le back-reference.