文字壓縮 － LZW 演算法

黃育銘
國立暨南國際大學資訊工程系助理教授
E-Mail : ymhuang@csie.ncnu.edu.tw

假設我們想用電腦來儲存以下一段 9 個字元之文字資料，ABBBABAAB，如果每個字元用 7 個位元之 ASCII 碼（不含同位元）來儲存，則需 63 個位元之記憶體。然而，該電腦如果已擁有一個如表一之電腦字典，很幸運的，藉助該字典的資訊則僅需 10 個位元的記憶體，亦即 AB、BB、AB、A、及 AB 等文字資料分別會先被編碼成 10、11、10、00、及 10 等十個位元，然後再存入記憶體，其省下的儲存空間高達 84% 。之後，如果一次以兩個位元的方式來讀取資料，同樣地透過該字典的查尋，則記憶體內所儲存的資料 1011100010，很快地就會被解讀出原來之文字資料 ABBBABAAB。

字典索引	索引之對應編碼值	字典內容
1	00	A
2	01	B
3	10	AB
4	11	BB

表一. 電腦字典 1

J. Ziv 及 A. Lempel 兩位博士，於西元 1977 年及 1978 年〔參考文獻 1 及 2〕，陸續提出了兩篇有關文字資料壓縮的論文，該論文演算法在之後的文獻裏分別被稱作 LZ77 及 LZ78，奠定了爾後文字資料壓縮研究的良好根基。其中 LZ78 演算法的精神，相似於上述字典編碼法的觀念。西元 1984 年，T.A. Welch 博士〔參考文獻 3〕針對 LZ78 演算法作改進，提出所謂 LZW 演算法。至今，在處理文字資料壓縮的問題時，LZW 演算法一直是廣泛被使用的一大利器。

上述之字典編碼法，稱作靜態字典(Static Dictionary)編碼法，因為表一電腦字典的內容是固定的。然而，如果想要儲存的資料為 BAAABABBA，顯然透過表二之電腦字典，其資料壓縮效果會比透過表一之電腦字典來得好。

字典索引	索引之對應編碼值	字典內容
1	00	A
2	01	B
3	10	BA
4	11	AB

表二. 電腦字典 2

因此，如果根據所欲儲存文字資料的型態，動態地建構起電腦字典的內容，即所謂動態字典(Adaptive Dictionary)編碼法，能明確地反應出目前所處理之文字資料的型態，當然其資料壓縮效果會比用靜態字典編碼法所得的結果來得好 。LZW 演算法即是屬於動態字典編碼法。

範例

編碼器：輸入字元序列 ABBBABAAB , 輸出索引序列 124313
解碼器：輸入索引序列 124313 , 輸出字元序列 ABBBABAAB

字元： 例如: A 或 B
索引： 例如: 1、2、3、…
字串： 一個字元（含）以上所組成，例如: A、AB、BBB、…
空字串 ： 未含任一字元

LZW 演算法

編碼法則： (S, C) S : 字串 C : 字元

1. 先在字典裏存入輸入字元序列裏所有可能出現的字元，每一個字元在字典裏之對應索引值，即為該字元爾後會被編碼成的值。
2. 從目前字典裏的內容找出最長字串 S，假設其長度為 k，且該字串 S 與目前尚未被編碼之字元序列的前 k 個字元相同； C 為目前未被編碼字元序列之第 k+1 個字元。
3. 輸出字串 S 在字典裏的索引值。
4. 將新的字串（SC）存入字典裏，且爾後該字串 SC 會被編碼成其在字典裏之對應索引值。
5. 對 C 字元(含)以後之剩餘字元序列繼續作編碼(即重覆法則 2)。

解碼法則： (S1, S2(=CS3)) S1, S2: 字串 C : 字元 S3 : 字串或空字串

1. 如同編碼法則 1，先建構起含輸入字元序列裏所有可能出現字元的字典，每一個字元在字典裏之對應索引值，即為該字元在編碼器裏會被編碼成的值。
2. 依序從收到的索引值，根據目前字典裏的內容，可解碼出相對應的字串。
3. 假設目前收到的索引值，其被解碼出字串 S2(=CS3)，且上一個收到的索引值，其被解碼成字串 S1，則將新的字串（S1C）存入字典裏，而其對應之新索引值，爾後將會被解碼出 S1C。
4. 重覆法則 2。

範例之編碼過程 ： 詳如表三

步驟 0 : 將 A 及 B 字元存入字典裏，也就是 A 及 B 字元爾後分別會被編碼成索引

值 1 及 2。

步驟 1：從目前字典內容及未被編碼之序列 ABBBABAAB，得到 S=A，k=1，C=B，所以輸出 A 在字典裏的索引值 1，並將新的字串 AB 存入字典裏，其索引值等於 3。

步驟 2：從目前字典內容及未被編碼之序列 BBBABAAB，得到 S=B，k=1，C=B，所以輸出 B 在字典裏的索引值 2， 並將新的字串 BB 存入字典裏，其索引值等於 4。

步驟 3：從目前字典內容及未被編碼之序列 BBABAAB，得到 S=BB，k=2，C=A，所以輸出 BB 在字典裏的索引值 4， 並將新的字串 BBA 存入字典裏，其索引值等於 5。

步驟 4：從目前字典內容及未被編碼之序列 ABAAB，得到 S=AB，k=2，C=A，所以輸出 AB 在字典裏的索引值 3， 並將新的字串 ABA 存入字典裏，其索引值等於 6。

步驟 5：從目前字典內容及未被編碼之序列 AAB，得到 S=A，k=1，C=A，所以輸出 A 在字典裏的索引值 1， 並將新的字串 AA 存入字典裏，其索引值等於 7。

步驟 6：從目前字典內容及未被編碼之序列 AB，得到 S=AB，k=2，C 沒有值，所以輸出 AB 在字典裏的索引值 3。

編碼 步驟	(S,C)	被編碼 的字串	輸出	字典 索引	字典內容
0				1	A
0				2	B
1	(A,B)	A	1	3	AB
2	(B,B)	B	2	4	BB
3	(BB,A)	BB	4	5	BBA
4	(AB,A)	AB	3	6	ABA
5	(A,A)	A	1	7	AA
6	(AB,)	AB	3	8	

表 三 ：編碼過程 (輸入： ABBBABAAB)

範例之解碼過程 ： 詳如表四

步驟 0：將 A 及 B 字元存入字典裏，也就是 1 及 2 索引值爾後分別會被解碼出 A 及 B。

步驟 1：針對收到的索引值 1，根據目前字典內容，將解碼出 A。

步驟 2：針對收到的索引值 2 ，根據目前字典內容，將解碼出 B。此時 S1=A 且

S2=B，所以將新的字串 AB 存入字典裏，其索引值等於 3。

步驟 3 : 針對收到的索引值 4，但目前字典裏的內容只有三項，第四項仍在建構中此時似乎無法繼續解碼。但根據解碼法則 3，字典裏第四項的內容將會是 B*，*表示未知字串，所以索引值 4 將會被解碼出 B*，但再根據解碼法則 3，由索引值 2 及索引值 4 分別被解碼出的字串 B 及 B*，我們可以得知，字典裏第四項的內容將會是 BB，即 S1=B 及 C=B。所以此步驟裏將會解碼出 BB。

步驟 4 : 針對收到的索引值 3 ，根據目前字典內容，將解碼出 AB。此時 S1=BB 且 S2=AB，所以將新的字串 BBA 存入字典裏，其索引值等於 5。

步驟 5 : 針對收到的索引值 1 ，根據目前字典內容，將解碼出 A。此時 S1=AB 且 S2=A，所以將新的字串 ABA 存入字典裏，其索引值等於 6。

步驟 6 : 針對收到的索引值 3 ，根據目前字典內容，將解碼出 AB。此時 S1=A 且 S2=AB，所以將新的字串 AA 存入字典裏，其索引值等於 7。

解碼 步驟	被解碼 的索引	輸出	(S1,S2)	字典 索引	字典內容
0				1	A
0				2	B
1	1	A			
2	2	B	(A,B)	3	AB
3	4	BB	(B,BB)	4	BB
4	3	AB	(BB,AB)	5	BBA
5	1	A	(AB,A)	6	ABA
6	3	AB	(A, AB)	7	AA

表 四 ：解碼過程 (輸入： 124313)

註 1: 上述之每一個索引值，在編碼器裏，實際上會根據字典的大小(例如等於 8)，再進一步地被編碼成三個位元值；而解碼器會以一次三個位元方式讀取，先解碼出相對之索引值 (壓縮前資料量為 63 個位元，壓縮後資料量為 18 個位元，其壓縮率達 71%) 。一般原始碼程式或者英文文章的資料，如果利用 LZW 演算法對其作資料壓縮，其壓縮率約為 50-60% ，對於本篇文章，如果將其儲存成純文字檔再對這個檔案作資料壓縮，其壓縮率僅為 36.75% ，然而若將其儲存成 Microsoft word 檔再對這個檔案作資料壓縮，其壓縮率可達 67.73% 。由此可見，Microsoft word 檔案其資料重覆性相當高。

註 2: 字典的建立，對編碼器或解碼器而言是同步的。

註 3: UNIX 系統之 compress 指令，即是採用 LZW 演算法。其作法如下：一開始字典大小內定為 512 ，此時每個索引值會以 9 個位元來編碼。當字典內容均已填滿時

這時候字典大小增大為 1024，也就是說此時每個索引值會以 10 個位元來編碼，如果字典內容再填滿時，字典大小則再倍增(即等於 2048)，依此類推，字典大小最大可達 16384(16 乘上 1024)，也就是每個索引值至多以 16 個位元來編碼。一旦填滿 16384 個字典內容時，該編解碼器則改以靜態字典編碼法方式處理，並隨時監視往後之壓縮效率是否低於某個標準，如果是，則將字典之內容全部清除（除了字元序列裏所有可能出現之字元），重新建構起新的字典，這時候字典大小又回復成 512。

參考文獻

[1] J. Ziv and A. Lempel. A Universal Algorithm for Data Compression. IEEE Transactions on Information Theory, IT-23(3):337-343, May 1977.
[2] J. Ziv and A. Lempel. Compression of Individual Sequences via Variable-Rate Coding. IEEE Transactions on Information Theory, IT-24(5):530-536, September 1978.
[3] T.A. Welch. A Technique for High-Performance Data Compression. IEEE Computer, pages 8-19, June 1984.