Chapter 6

Dictionary Coding

Coding procedure:

- Construct a dictionary
- Send the index corresponding to the entry in the dictionary for each symbol

Static dictionary

- Prior knowledge about the source is available for constructing a fixed dict.
- Especially suitable for use in specific application
- Work well only for applications and data they were designed for

索引值	縣市名稱
00001	台北市
00010	台北縣
…	…
01110	台南市
…	…
10001	高雄市
…	…

台北市 (6 Bytes = 48 bits)

00001 (5 bits)

台南市 (6 Bytes = 48 bits)

01110 (5 bits)

Example for Static Dictionary

Example 6.1: (page 146, textbook)

Encode the sequence a b d a e e a c d d b c

索引值 (3 bits)	字 (6bits)
000	ab (000001)
001	ac (000010)
010	ba (001000)
011	bd (001011)
100	ee (100100)
101	cd (010011)
110	da (011000)
111	dc (011010)

讀入: ab　　編碼成: 000
讀入: da　　編碼成: 110
讀入: ee　　編碼成: 100
讀入: ac　　編碼成: 001
讀入: dd　　編碼成: ???

Solutions:

1. 使用一個額外的旗標位元來表示要找的資料是否存在字典中，若資料在字典中(又稱匹配，match)，匹配旗標設為1，送出"1"，再送出和輸入資料相匹配那個字的索引值；若資料不存在字典中(又稱不匹配，no match)，匹配旗標設為0，送出"0"，再送出資料真正的代碼。
2. 建構字典時，先確保輸入符號集合A內的所有符號都存在字典中(如此例中的a, b, c, d, e)，字典內剩下的空間才存放那些較常出現的字。

Example for Static Dictionary

Encode the sequence a b d a e e a c d d b c

加上旗標位元來編碼此輸入序列的動作如下：

讀入: ab　在字典中　編碼成: 1000(第一個位元是旗標位元)

讀入: da　在字典中　編碼成: 1110

讀入: ee　在字典中　編碼成: 1100

讀入: ac　在字典中　編碼成: 1001

讀入: dd　不在字典中 編碼成: 0011011(d的代碼為011)

讀入: bc　不在字典中 編碼成: 0001010(b是001，c是010)

因此編碼後的結果為: 10001110110010010011010001010

索引值 (3 bits)	字
000	a (000)
001	b (001)
010	c (010)
011	d (011)
100	e (100)
101	ab (000001)
110	da (011000)
111	dc (011010)

讀入a ，因為字典中索引000和索引101的內容都是以a起頭，故需再讀入下一個符號。下一個是b所以可將這兩個符號一起用索引101來表示(其壓縮效果比使用索引000好)。

讀入下一個符號d，因為字典中索引011、110與111的內容都是以d起頭，故需再讀入下一個符號。下一個是a，所以可將這兩個符號一起用索引110來表示(其壓縮效果比使用索引011好)。

LZ77

LZ77 (adaptive dictionary, 1977, Jacob Ziv and Abraham Lempel)

使用先前編碼過的輸入序列(encoded input sequence)來當作字典。經由一個滑動視窗(sliding window)來編碼資料，視窗包含兩個區域：搜尋緩衝區(search buffer)和前看緩衝區(look-ahead buffer)。搜尋緩衝區存放剛編碼過的輸入序列(本方法中的字典)，前看緩衝區則存放即將要編碼的輸入序列(sequence to be encoded)。

LZ77編碼演算法：

步驟一： 移動一個指向搜尋緩衝區內資料的指標(pointer)，一直到該指標所指的符號等於前看緩衝區內的第一個符號才停止移動。檢查指標所指符號後面的那個符號，看它的內容是否等於前看緩衝區內的第二個符號。如果相等，再往下各看一個符號，重複比對，一直到不相等為止。那些相等符號的總個數，稱為子序列匹配長度(length of match)或符號串匹配長度。

步驟二： 重複步驟一，找出搜尋緩衝區內存在的所有匹配子序列，並決定出最長(longest)的匹配子序列才停止。

步驟三： 送出三個欄位的編碼結果[offset, length, char]，其中offset：從前看緩衝區到指標位置的距離(指標指到最長匹配子序列中的第一個符號)，length：最長匹配子序列的長度(也就是符號相等的個數)，char：緊接在最長匹配子序列後面第一個不匹配的編碼符號。

Example for LZ77 (Encoding)

Example 6.2: (page 153, textbook)

Encode the sequence RRSTRRSTSRRSRRSRRSTSR with LZ77

search buffer=9 characters, look-ahead buffer=8 characters

[offset, length, char]

編碼步驟一：

	搜尋緩衝區	前看緩衝區	輸 入 序 列	編碼結果
編碼前		RRSTRRST	SRRSRRSRRSTSR	[0,0,R]
編碼後	R	RSTRRSTS	RRSRRSRRSTSR	

★ **How many bits for offset, length, char ?**

編碼步驟二:

	搜尋緩衝區	前看緩衝區	輸 入 序 列	編碼結果
編碼前	R	RSTRRSTS	RRSRRSRRSTSR	[0,1,S]
編碼後	RRS	TRRSTSRR	SRRSRRSTSR	

• • • •

Example for LZ77 (Encoding)

[offset, length, char]

		搜尋緩衝區	前看緩衝區	輸 入 序 列	編碼結果
步驟一			RRSTRRST	SRRSRRSRRRSTSR	[0,0,R]
步驟二		R	RSTRRSTS	RRSRRSRRRSTSR	[0,1,S]
步驟三		RRS	TRRSTSRR	SRRSRRSTSR	[0,0,T]
步驟四		RRST	RRSTSRRS	RRSRRSTSR	[3,4,S]
步驟五		RRSTRRSTS	RRSRRSRR	STSR	[8,3,R]
步驟六	..RST	RRSTSRRSR	RSRRSTSR		[2,5,T]
步驟七	..TSR	RSRRSRRST	SR		[7,2,“E”]

Disadvantage of LZ77:

當輸入符號在搜尋緩衝區內找不到時，三個欄位的資料量卻只編碼了一個符號。

Example for LZ77 (Decoding)

[offset, length, char]

	輸入	解碼結果		搜尋緩衝區	前看緩衝區
步驟一	[0,0,R]	R		R	
步驟二	[0,1,S]	RS		RRS	
步驟三	[0,0,T]	T		RRST	
步驟四	[3,4,S]	RRSTS		RRSTRRSTS	
步驟五	[8,3,R]	RRSR	RRST	RRSTSRRSR	??
步驟六	[2,5,T]	RSRRST	..STSR	RSRRSRRST	
步驟七	[7,2,“E”]	SR	..SRRS	RRSRRSTSR	

LZSS

LZSS編碼演算法：

步驟一：移動指向搜尋緩衝區內資料的指標(pointer)，直到該指標所指符號等於前看緩衝區內第一個符號才停止移動。檢查指標所指符號後面的那個符號，看它的內容是否等於前看緩衝區內的第二個符號。如果相等，再往下各看一個符號，重複比對，直到不相等為止。那些相等符號的總個數，稱為匹配長度(length of match)。

步驟二：重複步驟一，找出搜尋緩衝區內存在的所有匹配子序列，並決定出最長的匹配子序列才停止。

步驟三：要編碼的輸入符號在搜尋緩衝區內找的到，匹配旗標flag設為1，送出三個欄位的編碼結果[flag, offset, length]，其中offset：從前看緩衝區到指標位置的距離，length：最長匹配子序列的長度。

要編碼的輸入符號在搜尋緩衝區內找不到，匹配旗標位元flag設為0，送出兩個欄位的編碼結果[flag, char]，其中char：代表這個要編碼的輸入符號。

步驟四： 重複步驟一到步驟三直到待編碼資料都處理完畢才停止。

Example for LZSS (Encoding)

Example 6.3: (page 160, textbook)

Encode the sequence RRSTRRSTSRRSRRSRRSTSR with LZSS

search buffer=9 characters, look-ahead buffer=8 characters

MATCH: [flag, offset, length]; NO MATCH: [flag, char];

編碼步驟一:

	搜尋緩衝區	前看緩衝區	輸 入 序 列	編碼結果
編碼前		RRSTRRST	SRRSRRSRRSTSR	[0,R]
編碼後	R	RSTRRSTS	RRSRRSRRSTSR	

編碼步驟二:

	搜尋緩衝區	前看緩衝區	輸 入 序 列	編碼結果
編碼前	R	RSTRRSTS	RRSRRSRRSTSR	[1,0,1]
編碼後	RR	STRRSTSR	RSRRSRRSTSR	

• • • •

Example for LZSS (Encoding)

[flag, offset, length];
[flag, char];

		搜尋緩衝區	前看緩衝區	輸 入 序 列	編碼結果
步驟一			RRSTRRST	SRRSRRSRRSTSR	[0,R]
步驟二		R	RSTRRSTS	RRSRRSRRSTSR	[1,0,1]
步驟三		RR	STRRSTSR	RSRRSRRSTSR	[0,S]
步驟四		RRS	TRRSTSRR	SRRSRRSTSR	[0,T]
步驟五		RRST	RRSTSRRS	RRSRRSTSR	[1,3,4]
步驟六		RRSTRRST	SRRSRRSR	RSTSR	[1,5,1]
步驟七		RRSTRRSTS	RRSRRSRR	STSR	[1,8,3]
步驟八	RRS	TRRSTSRRS	RRSRRSTS	R	[1,2,6]
步驟九	..STS	RRSRRSRRS	TSR		[0,T]
步驟十	..TSR	RSRRSRRST	SR		[1,7,2]

Example for LZSS (Decoding)

[flag, offset, length];
[flag, char];

解碼步驟一:

	輸入	搜尋緩衝區	前看緩衝區	解碼結果
解碼前	[0,R]			R
解碼後		R		

解碼步驟二:

	輸入	搜尋緩衝區	前看緩衝區	解碼結果
解碼前	[1,0,1]	R		R
解碼後		RR		

• • • •

解碼步驟七:

	輸入	搜尋緩衝區	前看緩衝區	解碼結果
解碼前	[1,8,3]	RRSTRRSTS		RRS
解碼後		RRS TRRSTSRRS		

解碼步驟八:

	輸入	搜尋緩衝區	前看緩衝區	解碼結果
解碼前	[1,2,6]	RRS TRRSTSRRS	???	RRSRRS
解碼後		..STS RRSRRSRRS		

Why ?

The details can be found in page 167 of textbook.

LZ78

LZ78 (adaptive dictionary, 1978, Jacob Ziv and Abraham Lempel)

LZ77字典編碼法假設大部份的待編碼資料和最近剛編碼過的資料很接近，所以待編碼資料的子序列通常在搜尋緩衝區內找的到。但是如果資料重複出現的週期(period)大於搜尋緩衝區的話，壓縮效果會明顯的降低。LZ78編碼法與LZ77完全不同，它會建立一個真正的字典，當輸入資料重複出現週期大於搜尋緩衝區時，LZ78仍能達到有效的壓縮。

LZ78編碼演算法：

步驟一： 建立一本初始空字典。

步驟二： 找出目前待編碼資料中和字典的字相同(相匹配)之最長子序列(符號串)。將此匹配子序列與其後的一個符號編碼，送出兩個欄位的編碼結果[index, char]，其中index：最長匹配子序列在字典中的索引位置，char：緊接在此匹配後面的第一個不匹配符號。最後，再把此匹配子序列與char結合成為一個新序列，將此新序列加入字典中成為一個新字。

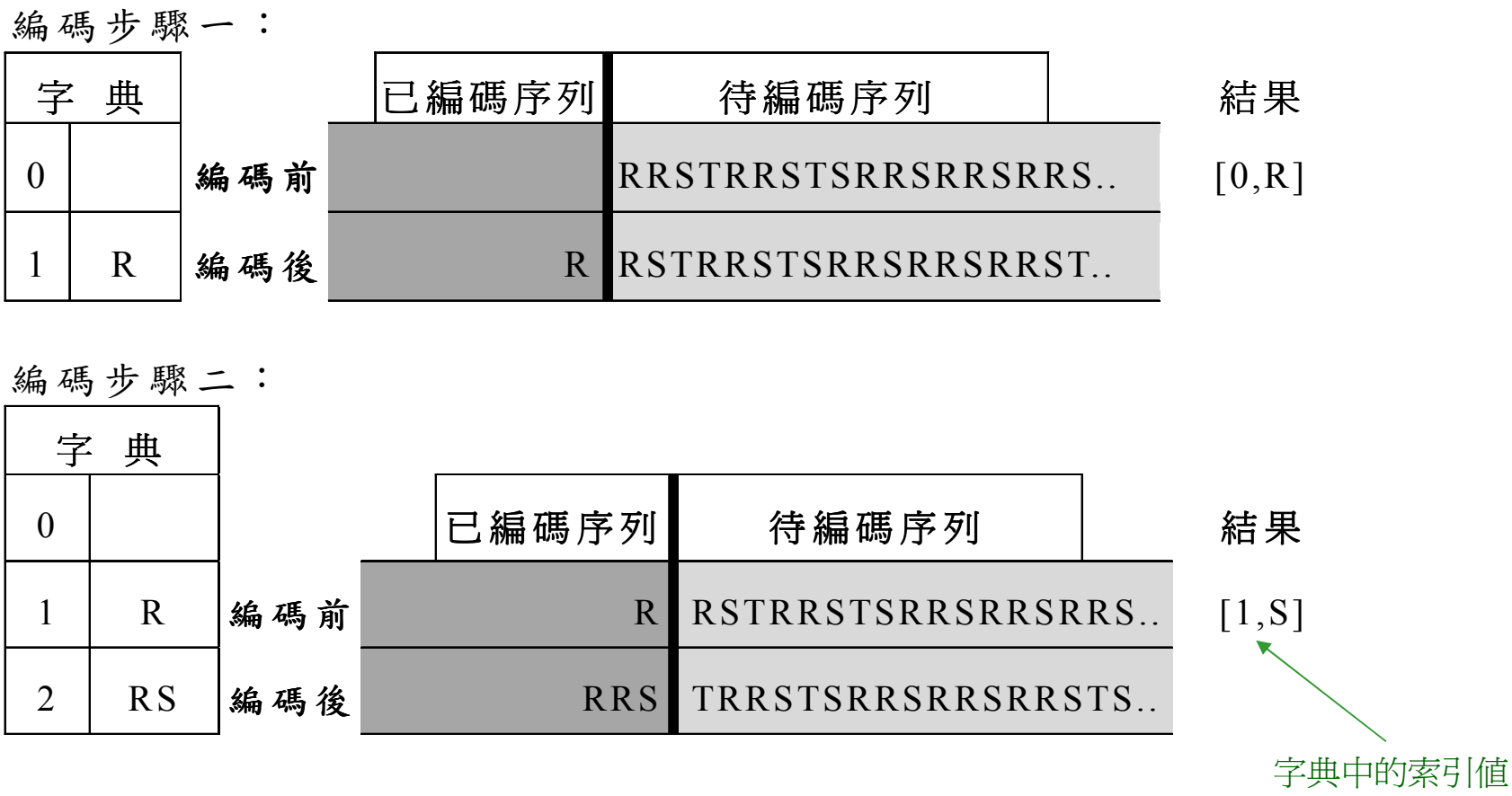
步驟三： 重複步驟二直到待編碼資料都處理完畢才停止。

Example for LZ78 (Encoding)

Example 6.4: (page 169, textbook)

Encode the sequence RRSTRRSTSRRSRRSRRSTSR with LZ78

[index, char]



Example for LZ78 (Encoding)

字 典			已編碼序列	待編碼序列	結果
		步驟一		RRSTRRSTSRRSRRSRR..	[0,R]
1	R	步驟二	R	RSTRRSTSRRSRRSRRS..	[1,S]
2	RS	步驟三	RRS	TRRSTSRRSRRSRRSTSR	[0,T]
3	T	步驟四	RRST	RRSTSRRSRRSRRSTSR	[1,R]
4	RR	步驟五	RRSTRR	STSRRSRRSRRSTSR	[0,S]
5	S	步驟六	RRSTRRS	TSRRSRRSRRSTSR	[3,S]
6	TS	步驟七	RRSTRRSTS	RRSRRSRRSTSR	[4,S]
7	RRS	步驟八	TRRSTSRRS	RRSRRSTSR	[7,R]
8	RRSR	步驟九	TSRRSRRSR	RSTSR	[2,T]
9	RST	步驟十	RSRRSRRST	SR	[5,R]

Example for LZ78 (Decoding)

	輸入	解碼結果	字 典		已解碼序列
步驟一	[0,R]	R	1	R	R
步驟二	[1,S]	RS	2	RS	RRS
步驟三	[0,T]	T	3	T	RRST
步驟四	[1,R]	RR	4	RR	RRSTRR
步驟五	[0,S]	S	5	S	RRSTRRS
步驟六	[3,S]	TS	6	TS	RRSTRRSTS
步驟七	[4,S]	RRS	7	RRS	..TRRSTSRRS
步驟八	[7,R]	RRSR	8	RRSR	..TSRRSRRSR
步驟九	[2,T]	RST	9	RST	..RSRRSRRST
步驟十	[5,R]	SR	10	SR	..RRSRRSTSR

LZW

LZW字典編碼法的主要特色是：

一開始就將待壓縮資料源的輸入符號集合*A*中的所有符號字元都放入到字典中，如此一來，編碼過程就不會發生待編碼符號不存在字典中的情況。換句話說，不需要像LZ78用“0”這個index值來代表要編碼的符號不在字典中。

LZW編碼演算法：

步驟一： 使用資料源的輸入符號集合*A*中的所有符號字元來建立一本初始的字典，其中每一個符號被當成字典中的一個字。

步驟二： 找出目前待編碼資料中和字典的字相同(相匹配)之最長子序列(符號串)。將此匹配子序列編碼，送出一個欄位的編碼結果[index]，其中index：最長匹配子序列在字典中的索引位置。此**匹配子序列**與**其後的一個符號**連結成一個新子序列，將此新序列加入字典中成為一個新字。

步驟三： 重複步驟二直到待編碼資料都處理完畢才停止。

Example for LZW (Encoding)

Example 6.5: (page 176, textbook)

Encode the sequence RRSTRRSTSRRSRRSRRSTSR with LZW

[index]

初 始 字 典	
1	R
2	S
3	T

編碼步驟一：

字 典			已編碼序列	待編碼序列	結果
1~3	R, S, T	編碼前		RRSTRRSTSRRSRRSR..	[1]
4	RR	編碼後	R	RSTRRSTSRRSRRSRR..	

編碼步驟二：

字 典			已編碼序列	待編碼序列	結果
1~3	R, S, T				
4	RR	編碼前	R	RSTRRSTSRRSRRSR..	[1]
5	RS	編碼後	RR	STRRSTSRRSRRSRR..	

• • • •

Example for LZW (Encoding)

字	典
1	R
2	S
3	T
4	RR
5	RS
6	ST
7	TR
8	RRS
9	STS
10	SR
11	RRSR
12	RRSRR
13	RST
14	TS

	已編碼序列	待編碼序列	結果
步驟一		RRSTRRSTSRRSRRSRRS..	[1]
步驟二	R	RSTRRSTSRRSRRSRRST..	[1]
步驟三	RR	STRRSTSRRSRRSRRSTSR	[2]
步驟四	RRS	TRRSTSRRSRRSRRSTSR	[3]
步驟五	RRST	RRSTSRRSRRSRRSTSR	[4]
步驟六	RRSTRR	STSRRSRRSRRSTSR	[6]
步驟七	..RSTRRST	SRRSRRSRRSTSR	[2]
步驟八	..STRRSTS	RRSRRSRRSTSR	[8]
步驟九	..RSTSRRS	RRSRRSTSR	[11]
步驟十	..RRSRRSR	RSTSR	[5]
步驟十一	..SRRSRRS	TSR	[3]
步驟十二	..RRSRRST	SR	[10]

Example for LZW (Decoding)

解碼步驟一：

字典	
1~3	R, S, T

輸入 [1]

解碼前

解碼後

已解碼序列
R

解碼結果：R

讀入下一個索引[1]，因為index為1，解碼出符號R並將R移入已編碼序列，**步驟一解出R而此步驟也解出R，故將RR加入字典成為新字**，如下圖步驟二所示：

解碼步驟二：

字典	
1~3	R, S, T
4	RR

輸入 [1]

解碼前

解碼後

已解碼序列
R
RR

解碼結果：R

讀入下一個索引[2]，因為index為2，解出符號S並將S移入已編碼序列，**步驟二解出R而此步驟解出S，故將RS加入字典成為新字**，如下圖步驟三所示：

解碼步驟三：

字典	
1~3	R, S, T
4	RR
5	RS

輸入 [2]

解碼前

解碼後

已解碼序列
RR
RRS

解碼結果：S

Example for LZW (Decoding)

• • • •

接下來讀入[8]，因為index為8，解出符號RRS，**步驟七解出S而此步驟解出RRS，故將SR(前一步驟解出的序列加上本步驟解出序列的前面符號)加入字典成為新字**，如下圖步驟八所示：

解碼步驟八：

字典	
1~3	R, S, T
4~5	RR, RS
6~7	ST, TR
8~9	RRS, STS
10	SR

解碼前　解碼後

輸入 [8]

已解碼序列	解碼結果
RRSTRRSTS	RRS
..RRSTSRRS	

接下來讀入[11]，然而這裏出現一個嚴重的解碼問題，因為第11個字根本尚未加入字典中，如何解碼呢？

Example for LZW (Decoding)

這個LZW的解碼問題是因為--解碼端的字典總是比編碼端的字典延後一筆(慢一步)才建立，如果在某一步驟時，編碼端的字典已經建立*k*個字而且此時編碼所得的索引值也正好是*k*，就會在解碼時出現問題，因為解碼端此時只建立到第*k*-1個字，如何能解出第*k*個字？此範例的編碼步驟八才將RRSR加入到字典中成為第11個字，而在步驟九馬上用RRSR這個字來編碼輸入序列，故送出編碼值[11]。然而如上所述，解碼端的字典總是比編碼端的字典慢一步，所以解碼端在步驟八才將SR加到字典中成為第10個字，下一步驟要解的是[11]，而此時第11個字根本不在解碼字典中，該如何解碼呢？一旦發生這種情況時，通常是呼叫一個特別處理程序(或程式)來解決，該程序簡述如後。

如前所述，步驟八解出RRS，步驟九要解索引值[11]。假設索引值[11]解出的符號串是*c*，我們知道接下來要將RRS與*c*的前面符號相結合，加入字典成為第11個新字。因此，加入字典的第11個字一定是 RRS?(以RRS開頭的符號串)，故[11]解碼出的字*c*一定也是RRS?。所以第11個字應該是{RRS}+{R}(步驟八解碼出的RRS加上步驟九解碼出的開頭字母R)，即RRSR。特別處理程序若能找出RRSR並把它加入解碼字典，解碼便能順利進行。