WEEK 7

LZ Coding & Data compression

A drawback of the Huffman code is that it requires knowledge of a probabilistic model of the source; unfortunately, in practice, source statistics are not always known *a priori.* thereby compromising the efficiency of the code. To overcome these practical limitations, we may use the *Lempel-Ziv algorithm/* which is intrinsically *adaptive* and simpler to implement than Huffman coding.

Basically, encoding in the Lempel-Ziv algorithm is accomplished by *parsing the source data stream into segments that are the shortest subsequences not encountered previously.* To illustrate this simple yet elegant idea, consider the example of an input binary sequence specified as follows:

000101110010100101 ...

It is assumed that the binary symbols 0 and 1 are already stored in that order in the code book. We thus write

Subsequences stored: 0, 1

Data to be parsed: 000101110010100101 . . .

The encoding process begins at the left. With symbols 0 and 1 already stored, the *shortest subsequence* of the data stream encountered for the first time and not seen before is 00; so we write

Subsequences stored: 0, 1, 00

Data to be parsed: 0101110010100101 . . .

The second shortest subsequence not seen before is 01; accordingly, we go on to write

Subsequences stored: 0, 1, 00, 01

Data to be parsed: 01110010100101 . . .

The next shortest subsequence not encountered previously is 011; hence, we write

Subsequences stored: 0, 1, 00, 01, 011

Data to be parsed: 10010100101 . . .

We continue in the manner until the given data stream has been completely parsed. Thus, for the example at hand, we get the *code book* of binary subsequences shown in the second row below:

Numerical positions:	1	2	3	4	5	6	7	8	9
Subsequences:	0	1	00	01	011	10	010	100	101
Numerical representations:			11	12	42	21	41	61	62
Binary encoded blocks:			0010	0011	1001	0100	1000	1100	1101

The decoder is just as simple as the encoder. Specifically, it uses the *pointer* to identify the root subsequence and then appends the innovation symbol. Consider, for example, the binary encoded block 1101 in position 9. The last bit, 1, is the innovation symbol. The remaining bits, 110, point to the root subsequence 10 in position 6. Hence, the block 1101 is decoded into 101, which is correct.

♣----------♣

Another procedure (the o/p code here is variable length code), given that 0 & 1 are not stored in the encoder,
The method of compression is to replace a substring with a *pointer* to an earlier occurrence of the same substring.

Example:
If we want to encode the string:

1011010100010. . . ,

We parse it into an ordered dictionary of substrings that have not appeared before as follows:
λ, 1, 0, 11, 01, 010, 00, 10, ... We include the empty substring λ as the first substring in the dictionary and order the substrings in the dictionary by the order in which they emerged from the source. After every comma, we look along the next part of the input sequence until we have read a substring that has not been marked off before. A moment's reflection will confirm that this substring is longer by one bit than a substring that has occurred earlier in the dictionary. This means that we can encode each substring by giving a *pointer* to the earlier occurrence of that prefix and then sending the extra bit by which the new substring in the dictionary differs from the earlier substring. If, at the nth bit, we have enumerated s(n) substrings, then we can give the value of the *pointer* in ($\log_2$ s(n)) bits. The code for the above sequence is then as shown in the fourth line of the following table (with punctuation included for clarity), the upper lines indicating the source string and the value of s(n):

source substrings	λ	1	0	11	01	010	00	10
$s(n)$	0	1	2	3	4	5	6	7
$s(n)_{\text{binary}}$	000	001	010	011	100	101	110	111
(pointer, bit)		(, 1)	(0, 0)	(01, 1)	(10, 1)	(100, 0)	(010, 0)	(001, 0)

Notice that the first pointer we send is empty, because, given that there is only one substring in the dictionary - the string λ no bits are needed to convey the `choice' of that substring as the prefix. The encoded string is 100011101100001000010. The encoding, in this simple case, is actually a longer string than the source string, because there was no obvious redundancy in the source string.

In simple words ☺ (the idea is to check the first row (substring) , remove the least bit (innovation bit) and see the remaining bits (*pointer*) – which must point to an earlier code→ then we must take the binary equivalent of that code & add the innovation symbol.

For example the last substring is (10) (0) is innovation- to be added later, & we have remaining (1) which points to substring(1) with binary equivalent (001), so the sub-code should be constructed from the pointer & the innovation bit, which yields (001,0).

Exercises:

Endocde the following stream using LZ algorithm:

0 0 0 1 0 1 1 1 0 0 1 0 1 0 0 1 0 1 . . .

Endocde the following stream using LZ algorithm:

0011001111010100010001001 . . .