1.4 Huffman Coding

Prefix codes

- Huffman coders are lossless data compression coders.
- Huffman coders are optimal in the sense that they can deliver codeword sequences which asymptotically approach the source entropy.
- Huffman coders are variable-length code words.
- Huffman coders are prefix coders. This kind of coders have the property of being self-punctuating. That means that the decoder can easily know the begining and ending of each code word. That makes decoding possible. In order to have that, the design must satisfy the following rule:

  No code word is a prefix of some other code word.

Such coders are prefix coders and instantaneously decodable.

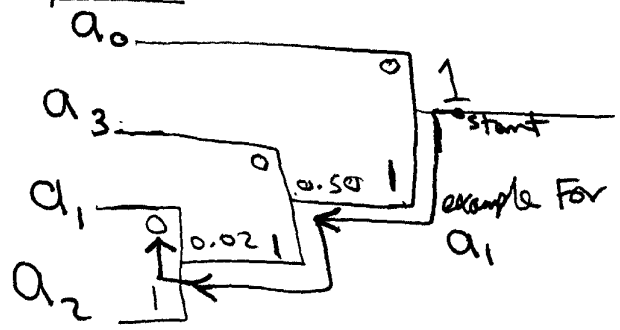
Huffman coding Construction

Example $A = \{a_0, a_1, a_2, a_3\}$

$P_A = \{0.5, 0.15, 0.05, 0.3\}$

First Arrange Symbols from High Probability to Low:

Prob.	Symbol	Codeword
0.5	a_0	0
0.3	a_3	10
0.15	a_1	110
0.05	a_2	111



Second Combine the probability of symbols starting from the Lowest probability and Lable the top one with "0" and the lower one with "1".

Third Codewords are assigned by reading the tree From right to left back to the original symbol. For example, for a_1, when we go from right to left, we get 1 1 0

Huffman Decoding

Example: Assume we received the following sequence from left to right

1001111001011 0

The decoder knows the codeword table.

The valid codewords in the sequence are

10, 0, 111, 10, 0, 10, 110

decoded sequence → $a_3\ a_0\ a_2\ a_3\ a_0\ a_3\ a_1$

1.5 Dictionary Codes and Lempel-Ziv Coding

- Huffman codes require us to find or estimate the probabilities of the source symbols. This might be practical for some applications. However, real-time applications, such as compressing computer files, are not tolerant to delays.

- Dictionary coders are compression coders that dynamically construct their coding and decoding tables "On the Fly" by looking at the data stream itself. No calculations of probabilities are required.
  - Disadvantage; dictionary coders are efficient only for long files or messages. Short files can result in the transmission of more bits, on average, than the original message.

— Lempel-Ziv (LZ) codes

- A class of dictionary codes

Advantages:

— Asymptotically approach the source entropy for long messages.

- The decoder doesn't require prior knowledge of the coding table constructed by the transmitter.
- The receiver construct it's own decoding table on the fly from the received code words.

— These codes initially expand rather than compress the data since extra information is sent to aid the receiver. Because of that, these codes don't work well for short files and we might get larger size than before compression.

— However, For long Files, As time progress, less data is sent and compression occurs.

— LZ codes have no decoding delays

1.5.2 A Linked-List LZ Algorithm

— Assume the source alphabet is $A=\{a_0, a_1, \cdots, a_{M-1}\}$ of cardinality M. The algorithm is initialized by constructing the first $M+1$ entries in the dictionary as follows:

	address	dictionary entry
$M+1$ entries	0	0 , null
	1	0, a_0
	2	0, a_1
	⋮	⋮
	m	0, a_{m-1}
	⋮	⋮
	M	0 , a_{M-1}

— Initialize Pointer variable $n=0$, ~~address pointer~~

Address pointer $m=M+1$ → next blank location in the dictionary.

- Algorithm

1. Fetch next source symbol a
2. If the ordered pair $\langle n,a\rangle$ is already in the dictionary then
   — n = dictionary address of entry $\langle n,a\rangle$

   else
   — transmit n
   — create new dictionary entry $\langle n,a\rangle$ at dictionary address m.
   — $m=m+1$
   — n = dictionary address of entry $\langle 0,a\rangle$
3. Return to step 1.

Example 1.5.1

A binary source emits the sequence of symbols

110 001 011 001 011 100 011 11

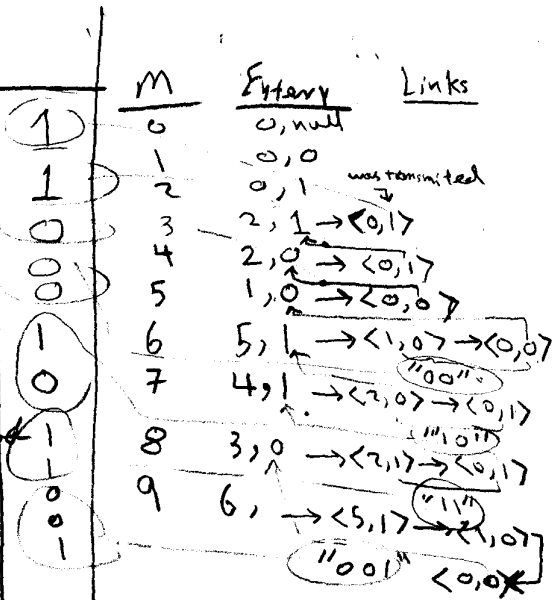
See the example in Text book Page 24

1.5.3 Decoding Process of LZ coder

- At the receiver, the decoder will construct a dictionary identical to the one at the transmitter
- it starts by the initial dictionary "root symbols" and build on it by examining the received pointer n.
- Operation of the decoder is governed by the following Rules:
  1. Reception of any code word means that a new dictionary entry must be constructed.
  2. Pointer n for this new dictionary entry is the same as the received code word n.
  3. Source symbol a for this entry is not yet known since it is the root symbol of the next string.

Example 1.5.2 : Read the explaination in Textbook P.26

Received (n)	decoded symbols
2	- we received n=2, add a new entry <2, ?>, still we don't know the source symbol a for this entry. However, we can find the source symbol of the previous source, we link the pointer 2 to the root symbol, we get symbol "1".
2	
1	
5	
4	- we receive n=2, link it to root symbol, we find the previous symbol to be "1".
3	- n=1, see text book
	- n=5, link to address 5 we have <1, ?>, link it again to address 1 we get a=0, So the previous symbols are "00". we decoded two symbols at the same time since we did two linking steps.
	- Read the explaination for the rest in the text book.



— Notice in example 1.5.1, the dictionary has more than 16 entries, therefore, n must consist of at least five bits. Thus, more bits are being transmitted than were in the source message up to that point.

— So, where is the compression?

Compression will happen when we process large amount of data.

Example of Huffman coding

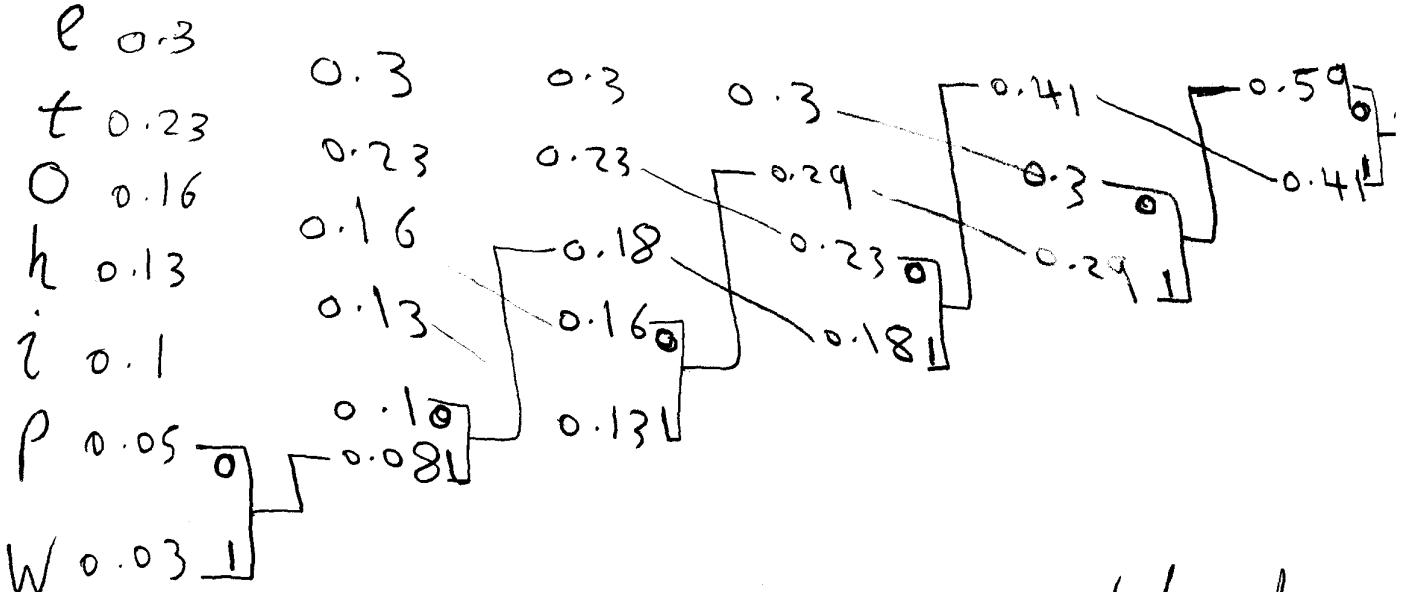
When we sum the lowest two probabilities, we might get a sum greater than some other symbol probability. In this case, we have to sort the probabilities again from Max to min and keep tracking the changes.

The following example will illustrate this:

Assume we have seven letters;

$$A = \{e, h, i, O, P, t, w\}$$

with Prob. $$P_A = \{0.3, 0.13, 0.1, 0.16, 0.05, 0.23, 0.03\}$$



Remember to sort every time and keep track of each symbol. Then trace the code of each symbol, we will get

e → 00
t → 10
O → 010
h → 011
i → 110
P → 1110
w → 1111