Uniquely Decodable Codes

Minimizing the average code length of a source is not the only point in compression. We have to decode it, too. There is a famous joke about the fundamental limits and design strategies for compression algorithms:

- Anonymous Engineer : "I have obtained an excellent compression algorithm that encodes **any** source down to 1 bit!!! Well... I am still working on the decoding algorithm."

This engineer will not be able to obtain the decoder for his/her compressor :-)

The radical solution of unique decodability is putting separations between symbols in a message. For example, if we have a messagege like 01, 0, 1, 11, 1011, 110, 0, 11, ... then each symbol can easily be separated without any ambiguity. However, This requires a new symbol (the coma symbol here) in out alphabet, and it adds one extra symbol for each symbol in a message. Obviously, this is very inefficient, therefore undesirable. However, we can do something else:

Consider a source alphabet consisting of 4 symbols : a_1, a_2, a_3, a_3, each with occurence probabilities: $P(a_1)$=0.5, $P(a_2)$=0.25, $P(a_3)$=0.125, and $P(a_4)$=0.125. If you use Eq. 3, you see that the entropy for this source is 1.75 bits/symbol. Let us make some bit sequence assignments for these 4 letters, and see which assignment sets are decodable, which ones are not, and why:

Letters	Probability	Code 1	Code 2	Code 3	Code4
a_1	0.5	0	0	0	0
a_2	0.25	0	1	10	01
a_3	0.125	1	00	110	011
a_3	0.125	10	11	111	0111
Average Length:		1.125	1.25	1.75	1.875

As you see, the first two codes have an average rate less than the entropy 1.75. It is not possible that they are uniquely decodable. Just try to write down some of the symbol representations one after another, and then try to read it. You will see that you may read it in many different ways (similar to the Morse code example).

Code 3 is in one of the most desirable forms. It has exactly the minimum rate (equal to the entropy) and it is uniquely decodable. Again, try to write a message using the representations of these 4 letters, and then try to read it. You will come up with only one way to decode the message. It is indeed uniquely decodable.

Code 4 is also uniquely decodable. However it has slight differences. First of all, its rate is a little higher than Code 3, threfore it is not as efficient as Code 3. Secondly, the decoding is slightly more involved. The decoder cannot determine the point where the last bit of a symbol is encountered before the next symbol starts. Whenever the next symbol starts, it starts with a 0, and according to the amount of 1's in the symbol, you can determine which symbol was sent.

The structure of Code 3 implies an important property for code generation. Notice that Code 3 is not only uniquely decodable, but it is also "instantly" decodable. In other words, we can decode the message while the symbols are arriving, without any need of waiting for the end of message. These sorts of codes are called **prefix codes**.

Test for unique decodability: It is easy do determine whether a code is uniquely decodable, or not. Alternatively, we can test if it is a prefix code, or not. A uniquely decodable code is a code which satisfies the following conditions for all of its binary codewords. The prefix code test simpler.

Consider two binary codewords a and b, where a is k bits long, and b is n bits long. Also assume that n>k (therefore b is a longer codeword). If the first k bits of b is exactly equal to the codeword of a, we say that a is a **prefix** of b. The rest of the bits of b (which have a length of n-k bits) are called the **dangling suffix**. As an example, if a=01001 and b=010011001, then a is a prefix of b and 1001 is the dangling suffix. The test for suffix code proceeds as follows:

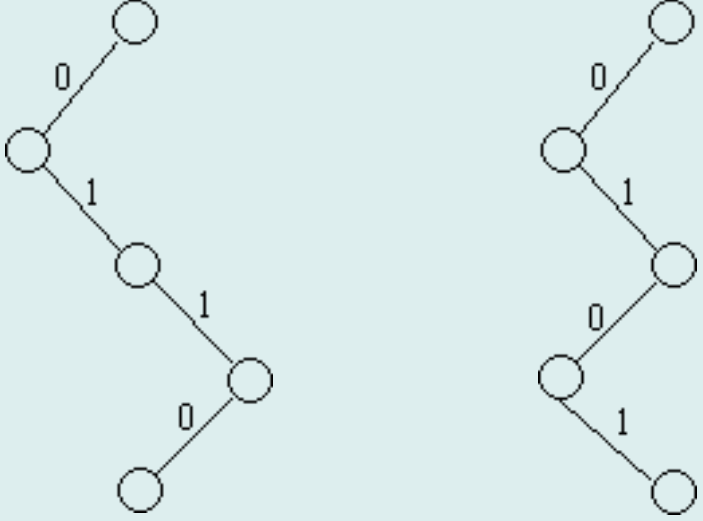
- Construct a list of all codewords in your alphabet.
- Examine all the codewords and see if any code is a prefix of another code or not.
- If you find such a prefix case, obtain the dangling suffix, and add it to your list.
- Continue examining your list and adding the dangling suffixes until one of the following two things happen:
    1. You come up with a dangling suffix which is a codeword in the list.
    2. There are no more unique dangling suffixes
- If you get the first outcome, then the code is not uniquely decodable. Otherwise, the code is uniquely decodable.

Ex. 5 : Consider a code list {0, 01, 11}. 0 is the prefix of 01, and the dangling suffix is 1. 1 is not a codeword. There are no more dangling suffixes, so stop. The code is uniquely decodable.

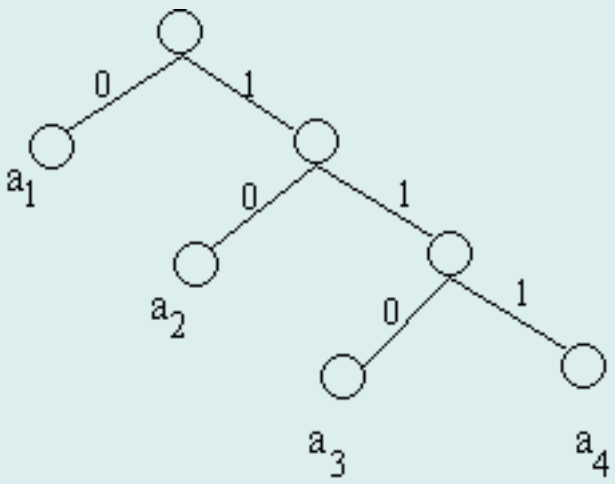
Ex. 6 : Consider a code list {0, 01, 10}. 0 is the prefix of 01, and the dangling suffix is 1. 1 is not a codeword, so add to the list. The new entry 1 is a prefix of 10 and 0 is the dangling suffix which is a codeword. So the condition "1" is satisfied, and the code is not uniquely decodable.

The way to test if the code is a prefix code is simple: No code in the list should be a prefix of another.

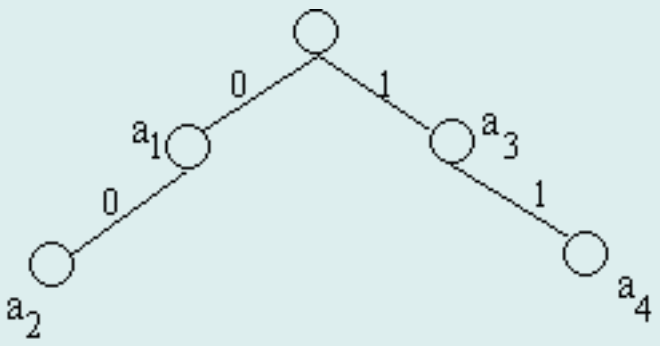
There is an alternative way of testing prefix codes and unique decodability. Form a **tree** using your codewords. For example, 0 may corespond to a tree branch to the left, 1 may correspond to a branch to the right. In this way, as an example, codewords 0110 and 0101 produce the following trees:



Form a tree that describes all codewords. A prefix code corresponds to having codewords only on the leaves of the tree. Therefore, we should no have a codeword corresponding to any internal node. For example, The following code is a prefix code: (corresponding to {0, 10, 110, 111})



On the other hand, the following code is not a prefix code: (corresponding to {0, 00, 1, 11})



The Counting Argument: The compression methods are limited. No method can compress all of the possible sources. The argument is: No lossless compression algorithm can compress all possible sources of size N to some value less than N.

Proof:

- There are 2^N different sources of size N.
- Assume that we can compress all of these sources down to some size less than N.
- Now let us calculate how many different sources there are which have size less than N:

  There are 2^{N-1} different sources of size N-1, there are 2^{N-2} different sources of size N-2, ... , there are 2 different sources of size 2, there is one different source of size 1.

- Let us take the summation of all these sources: It makes $2^{N-1}+2^{N-2}+\cdots+1=2^{N}-1$ different sources of size less than N.

As we see, the total number of different sources with size less than N is less than the total number of different sources with size N. This means that if we can compress **all** of the sources with size N, then at least two of the sources should produce the same compressed source. When we try to decompress that source, we cannot decide which original source (with size N) it should correspond to. Therefore, we conclude that we cannot compress all of the sources. In fact, sometimes, we expand the source size instead of compression.

Exercise: Compress a file in your computer using winzip or a similar compression utility. Now try to compress the zip file once more. In some cases, you may observe that the file size expands after the second compression. Repeat the exercise by trying to compress MPEG movie files that you can download from the internet. You may notice that you sometimes get data expansion instead of compression. This phenomenon is more common if you try to compress a source which has already been compressed (MPEG and JPEG files are compressed files).

Turning back to the topic of uniquely decodable codes; the uniquely decodable codes (prefix codes are a subset of them) has the following important property.

Kraft-McMillan Ineqality: Let *C* be a code with N codewords with lengths $l_1, l_2, \cdots, l_N$. If *C* is uniquely decodable, then we have:

$$K(C)=\sum_{i=1}^{N}2^{-l_i}\leq 1$$

This inequality gives a lower bound on the codeword lengths similar to the entropy expression in Eq. 3. The proof is rather involved, and is beyond the scope of the course.

Lemma: If the set of integers satisfy the Kraft McMillan inequality $\sum_{i=1}^{N}2^{-l_i}\leq 1$, then we can always find a prefix code with codeword lengths $l_1, l_2, \cdots, l_N$. This is also an important observation wich indicates that, we can really come up with codeword assignments to alphabet symbols as long as we obey the Kraft-McMillan inequality.

In the next section, we will develop strategies to assign codewords which are as near to the entropy expression (Eq. 3) as possible, and at the same time satisfy the Kraft-McMillan inequality.