Next we turn our attention to encoding data as sequences of 0's and 1's, a string of bits.

An encoding is an unambiguous mapping between bit strings and the members of the set of data to be encoded.

For example, suppose we have a set of four symbols -- A, B, C, and D – and we want to use bit strings to encode messages constructed of these symbols, for example "ABBA".

If we choose to encode the message one character at a time, our encoding would assign a unique bit string to each symbol.

Since we have four symbols, we might choose a unique two-bit string for each: "A" could be "00", B = "01", C = "10", and D = "11".

This would be called a "fixed-length encoding" since the bit strings used to represent the symbols all have the same length.

The encoding for the message "ABBA" would be "00-01-01-00".

And we can run the process backwards: given a bit string and the encoding key, we can look up the next bits in the bit string, using the key to determine the symbol they represent.

"00" would be decoded as "A", "01" as B and so on.

We can also use bit strings of different lengths to encode symbols -- this is called a variable-length encoding.

So A could be "01", B = "1", C = "000" and D = "001".

"ABBA" would be encoded as "01-1-1-01".

We'll see that carefully constructed variable-length encodings are useful for the efficient encoding of messages where the symbols occur with different probabilities.

We have to be careful that the encoding is unambiguous!

Suppose we had decided to encode A as "0", B as "1", C as "10" and D as "11".

The encoding for "ABBA" would be "0-1-1-0".

Looking good since that encoding is shorter than either of the previous two encodings.

Now let's try to decode this bit string -- oops.

Using the encoding key, we can unfortunately arrive at several decodings: "ABBA" of course, but also "ADA" or "ABC" depending on how we group the bits.

This attempt at specifying an encoding has failed since the message cannot be interpreted unambiguously.

Graphically we can represent an unambiguous encoding as a binary tree, labeling the branches from each tree node with "0" and "1", placing the symbols to be encoded as the leaves of the tree.

If you build a binary tree for a proposed encoding and find that there are no symbols labeling interior nodes and exactly one symbol at each leaf, then your encoding is good to go!

For example, consider the encoding shown on the left.

It takes just a second to draw the corresponding binary tree.

The symbol B is distance 1 from the root of the tree, along an arc labeled "0".

A is distance two, and C and D are distance 3.

If we receive an encoded message, for example "01111", we can decode it using successive bits of the encoding to identify a path from the root of tree, descending step-by-step until we come to leaf, then repeating the process starting at the root again, until all the bits in the encoded message have been consumed.

So the message from the sheep is: "0" takes us from the root to the leaf B, which is our first decoded symbol.

Then "1-1" takes us to A and the next "1-1" results in a second A.

The final decoded message -- "BAA" -- is not totally unexpected, at least from an American sheep.