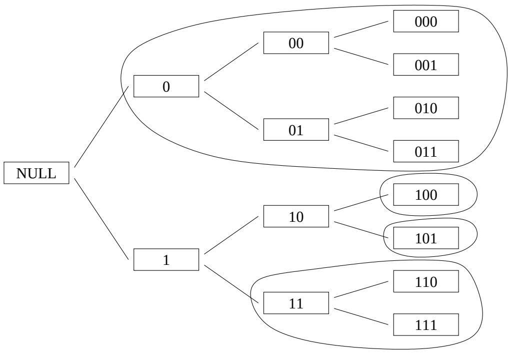
Another Look at Code Trees

Any instantaneous code can be represented by a tree such as the following, with subtrees for codewords circled:



Rather than concentrate on the codewords that head each subtree, let's concentrate on the rightmost column...

Viewing a Code as a Way of Dividing up a "Codespace"

Here's the right column from the code tree, divided up according to codeword:

Symbol s_1, Codeword 0	000
	001
	010
	011
Symbol s_2, Codeword 100	100
Symbol s_3, Codeword 101	101
Symbol s_4, Codeword 11	110
	111

If we view $\{000, 001, 010, 011, 100, 101, 110, 111\}$ as an available "codespace", we see that this code divides it up so that symbol s_1 gets 1/2 of it, symbols s_2 and s_3 get 1/8, and symbol s_4 gets 1/4.

Can We Use Other Divisions?

We know that this code is optimal if the fraction of codespace assigned to a symbol is equal to the symbol's probability.

But suppose the symbol probabilities were $3/8, 1/8, 1/8, 3/8$. We would then like to divide up codespace as follows:

	000
Symbol s_1, probability 3/8	001
	010
Symbol s_2, probability 1/8	011
Symbol s_3, probability 1/8	100
	101
Symbol s_4, probability 3/8	110
	111

Unfortunately, these divisions don't correspond to subtrees — so there's no code like this.

Viewing the Codespace as the Interval From 0 to 1

Let's ignore this problem of how to generate codewords for the moment.

Instead, let's ask how we could handle symbols that have probabilities like 1/3, which aren't multiples of 1/8.

A solution: Consider the codespace to be the interval of real numbers between 0 and 1.

For example:

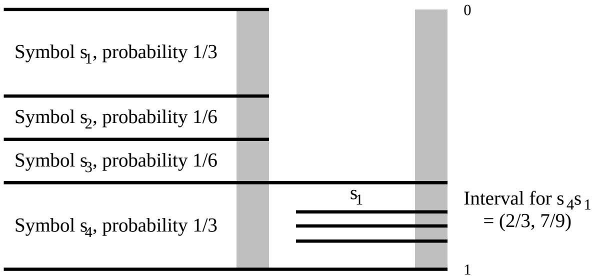
	0
Symbol s_1, probability 1/3	1/6
	1/3
Symbol s_2, probability 1/6	
	1/2
Symbol s_3, probability 1/6	
	2/3
Symbol s_4, probability 1/3	5/6
	1

A Key Concept: We Can Encode Blocks by Subdividing Further

Suppose we want to encode blocks of two symbols from this source.

We can do this by just subdividing the interval corresponding to the first symbol in the block, in the same was as we subdivided the original interval.

Here's, how we encode the block $s_4 s_1$:



Encoding Large Blocks as Intervals

Here's a general scheme for encoding a block of n symbols, $s_{i_1}, \ldots, s_{i_n}$:

1) Initialize the interval to $\left[l^{(0)},\ u^{(0)}\right)$, where $l^{(0)} = 0$ and $u^{(0)} = 1$.

2) For $k = 1, \ldots, n$:

   Let $l^{(k)} = l^{(k-1)} + \left(u^{(k-1)} - l^{(k-1)}\right) \sum_{j=1}^{i_k - 1} p_j$.

   Let $u^{(k)} = l^{(k)} + \left(u^{(k-1)} - l^{(k-1)}\right) p_{i_k}$.

3) Output a codeword that corresponds (somehow) to the final interval, $\left[l^{(n)},\ u^{(n)}\right)$.

This scheme is known as *arithmetic coding*, since codewords are found using arithmetic operations on the probabilities.

Finding a Codeword for an Interval

The last step requires that we be able to find a codeword for the final interval. We'll insist on an instantaneous code, for which no codeword is a prefix of another codeword.

Any binary codeword defines a number in $[0, 1)$, found by putting a "binary point" at its left end. Eg, the codeword 101 defines the number $1 \times (1/2) + 0 \times (1/4) + 1 \times (1/8)$.

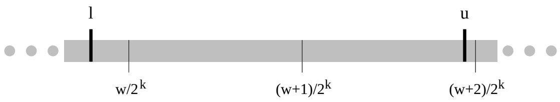
We'll choose a codeword such that:

- The codeword defines a point in the final interval.
- If we added any string of bits to the end of the codeword, it would still define a point in the final interval.

Codewords chosen in this way will form a prefix code for the blocks.

How Long Will the Codewords Be?

Here's a picture of how we pick a codeword for an interval:



Here, the interval $[w/2^k,\ (w+1)/2^k)$ fits entirely within $[l,\ u)$, the final interval found when encoding the block. We can therefore use the k-bit binary representation of w as the codeword for this block.

This can only be true if $u - l \geq 1/2^k$. Also, we will always be able to find such a codeword of length k if $u - l \geq 2/2^k = 1/2^{k-1}$ (as above).

Conclusion: We can pick a codeword of length k for a block of probability p $(= u - l)$ if $k \geq \log(1/p) + 1$. So codewords need be no longer than $\lceil \log(1/p) \rceil + 1$.