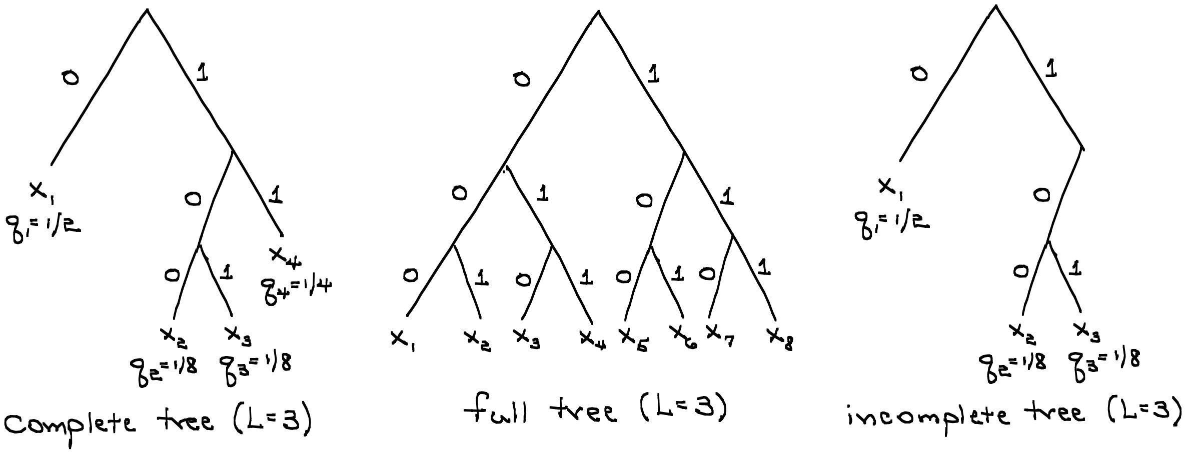
Phys 572 Quantum Information Theory Fall 2014

Homework Problem 1.6 Discussion Friday, September 5

1.6 **Shannon information and yes/no questions.** In this problem, we formulate a rigorous way of thinking of the Shannon information as the number of yes/no questions required to identify a particular alternative.

Consider a random variable X that can take on values x_j, $j = 1, \ldots, D$, and let $p_j = p_X(x_j)$ be the probability for x_j. Imagine a sequence of yes(1)/no(0) questions, at the termination of which one identifies a particular value of X. The sequence of questions and their ultimate answers can be depicted as a binary tree, an example of which is shown on the left of the figure below for a case of four values. A (potential) branch point is called a *node*, a path descending from a node is a *branch*, and a terminal node is a *leaf*. Each alternative thus sits at a leaf. The sequence of answers leading to a leaf is a code word for the value at that leaf. For the tree on the left, the four code words are $c_1 = 0$, $c_2 = 100$, $c_3 = 101$, and $c_4 = 11$. The entries within each code word are called *letters*.



Complete tree (L=3) full tree (L=3) incomplete tree (L=3)

I am in the habit of drawing trees with the branches extending downward, which means the branches might better be called roots. My way does have the advantage that descendants of nodes actually descend.

The collection of code words is called a *prefix-free* code, because by construction no code word is a prefix of any other code word. The prefix-free property guarantees that in a message consisting of a sequence of code words, each code word can be decoded as soon as it is identified, without having to worry that some other code word has the identified code word as a prefix. For this reason prefix-free codes are also called *instantaneous* codes. In a message in an instantaneous code, there is no need for an additional symbol to separate code words, like the space in ordinary English text.

The code word for x_j has length ℓ_j, which is the number of yes/no questions leading to x_j. For the tree on the left, we have $\ell_1 = 1$, $\ell_2 = 3$, $\ell_3 = 3$, and $\ell_4 = 2$. The longest length in a code is the *order* L of the tree.

A *full* tree of order L is a tree that has 2^L leaves, all of length L. The full tree of order $L = 3$ is shown in the middle of the figure. Any tree can be extended to a full tree.

Given a particular tree, the number of descendants leaf j has in the full-tree extension of order L is $2^{L-\ell_j}$. The quantity $q_j = 2^{-\ell_j}$ is the fraction of leaves in a full-tree extension that descend from leaf j, and it is also the sum of the q-values for those descendants.

We are now ready to formulate the condition for a prefix-free code: *There is a prefix-free code with code-word lengths ℓ_j, $j = 1, \ldots, D$, if and only if*

$$Q = \sum_{j=1}^{D} q_j = \sum_{j=1}^{D} 2^{-\ell_j} \leq 1 \ .$$

The inequality $Q \leq 1$ is called the *Kraft inequality.*

Since Q is a fraction of the leaves in a full-tree extension, it is clear that $Q \leq 1$ for a prefix-free code. What remains is to show that if $Q \leq 1$, then a prefix-free code exists. To show this, order the code-word lengths from biggest to smallest, i.e., $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_D$. Consider the quantity

$$Q_k = \sum_{j=1}^{k} q_j = \sum_{j=1}^{k} 2^{-\ell_j} \ .$$

For $k < D$, it is clear that Q_k is strictly less than 1. Moreover, the binary expansion of Q_k terminates after ℓ_k digits, so $Q_k \leq 1 - 2^{-\ell_k}$ for $k < D$. We assign code words of appropriate length successively, and we know we can do this for the following reason. After assigning the first $k - 1 \leq D - 1$ code words, the number of unassigned code words of length ℓ_k is

$$2^{\ell_k} - 2^{\ell_k - \ell_1} - 2^{\ell_k - \ell_2} - \cdots - 2^{\ell_k - \ell_{k-1}} = 2^{\ell_k}(1 - Q_{k-1}) \geq 2^{\ell_k} 2^{-\ell_{k-1}} \geq 1 \ ,$$

thus allowing us always to find the needed code word of length ℓ_k.

If $Q = 1$, the tree is called a *complete* tree; if $Q < 1$, an *incomplete* tree. An incomplete tree can always be completed in the following way. The fraction Q has a binary expansion that terminates after L digits, meaning that $Q = 1 - n2^{-L}$, where n is an integer. We add n code-word lengths L to the list of code-word lengths in the incomplete tree. The proof of the Kraft inequality shows that we can find n additional code words of length L, thus completing the tree.

Enough of introduction. Let's turn now to the average code-word length—i.e., the average number of yes/no questions—given probabilities p_j:

$$\bar{\ell} = \sum_{j=1}^{D} p_j \ell_j \ .$$

(a) *Show* that the average code-word length satisfies

$$\bar{\ell} \geq H(\mathbf{p}) \ .$$

(b) *Show* that it is possible to find code words such that

$$\bar{\ell} < H(\mathbf{p}) + 1 \; .$$

Unless you do something really clever, the code you introduce to prove this is called a *Shannon-Fano* code.

The slop of 1 bit in the relation between average number of yes/no questions and Shannon information is clearly due to the mismatch between the q_j's, which are inverse powers of 2, and the probabilities, which generally aren't. We can get a tight relation between Shannon information and the number of yes/no questions by considering N trials drawn from the distribution $\mathbf{p}$. Now there are D^N alternatives, and the Shannon information for the N trials is $NH(\mathbf{p})$. The average code-word length satisfies

$$\sum_{j=1}^{D^N} p_j \ell_j \ge NH(\mathbf{p}) \; ,$$

so the average code-word length per trial is bounded below as for a single trial,

$$\frac{1}{N} \sum_{j=1}^{D^N} p_j \ell_j \ge H(\mathbf{p}) \; ,$$

but there exists a code satisfying

$$\sum_{j=1}^{D^N} p_j \ell_j \ge NH(\mathbf{p}) + 1 \; ,$$

which gives an average code-word length per trial such that

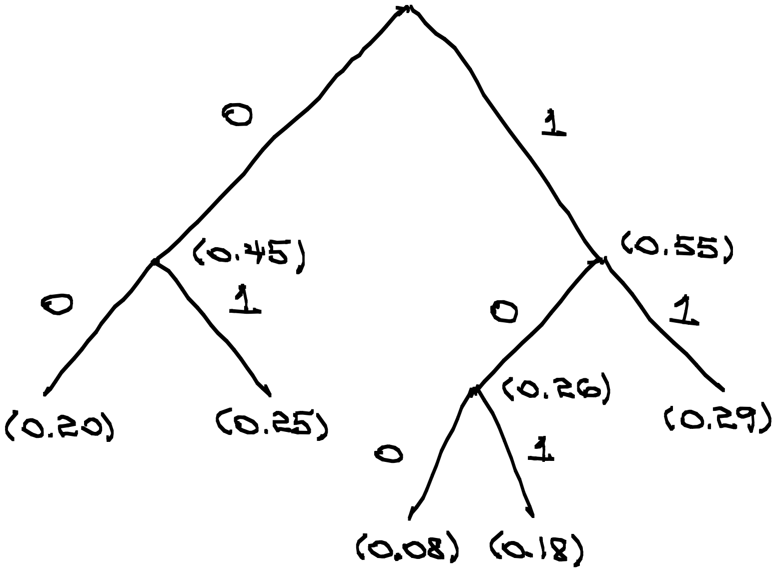
$$\frac{1}{N} \sum_{j=1}^{D^N} p_j \ell_j \ge H(\mathbf{p}) + \frac{1}{N} \; .$$

In the limit $N \to \infty$, we obtain a tight relation between the average code-word length per trial and the Shannon information.

Of course, when N is large, the law of large numbers guarantees that in each realization, the code-word length per trial is nearly certainly the average code-word length per trial, so this becomes a particular form of block coding.

An *optimal code* is one that minimizes the average length $\bar{\ell} = \sum_j p_j \ell_j$. It turns out that an optimal code can be constructing using a procedure called *Huffman coding.* Codes other than Huffman codes can be optimal, but you will show below that they can never have shorter average length than a Huffman code.

Here's how Huffman coding works. Take the two least likely alternatives, and make the last letter in their code words 0 and 1. Now amalgamate these two alternatives into a single alternative whose probability is the sum of the probabilities for the two original alternatives. Repeat the process for this new situation. A particular case of Huffman coding is illustrated in the figure below.



Huffman code for five alternatives with probabilities 0.29, 0.25, 0.20, 0.18, and 0.08. The figures in parentheses are probabilities. The average code-word length is 2.26.

(c) *Show* that Huffman coding is optimal, i.e., that no code has an average code-word length smaller than that for Huffman coding. (Hint: There are many ways to do this, but one way is first to show that any optimal code can be converted to an equivalent optimal code in which the two least likely alternatives have code words of the same (maximum) length, these two code words differing only in the final letter. This allows you to show that any optimal code can be converted to a code constructed by the Huffman procedure.)