Today we will look at a method for constructing an optimal prefix code, called *Huffman coding*. To motivate this method, let's continue on with the ideas from the end of last class.

Given any alphabet of A_i's and probabilities $p(A_i)$, we can renumber these symbols in the alphabet in terms of their relative probabilities and relative codeword lengths. Once we do that, we have the following property:

Lemma 3.1 *Suppose we have an optimal prefix code. (Recall that it is non-unique.) Further, suppose the symbols are numbered such that*

$$p(A_1) \geq p(A_2) \geq p(A_3) \cdots \geq p(A_{N-1}) \geq p(A_N) \ > \ 0. \tag{1}$$

Then there exists an optimal prefix code C whose codeword lengths satisfy

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{N-1} \leq \lambda_N \tag{2}$$

Proof From last class, we know that such an ordering on the λ's automatically holds when the inequality on $p(A_i)$ and $p(A_j)$ is strict. Thus, the only concern is what to do when $p(A_i) = p(A_j)$ for some $i < j$, but $\lambda_i > \lambda_j$. In this case, we consider a new optimal prefix code, by swapping codewords $C(A_i)$ and $C(A_j)$. i.e. since the two symbols have the same probability, swapping their codewords does not change the average code length, and so the new code is also optimal. But it would ensure $\lambda_i < \lambda_j$, satisfying Eq. (2).

Similarly, if there are more than two symbols with equal probability, then we swap the codewords of all these symbols, such that we ensure the desired ordering. □

Lemma 3.2 *For any optimal prefix code whose symbols satisfy Eqs. (1) and (2), it must be that $\lambda_{N-1} = \lambda_N$.*

Proof We already know that $\lambda_{N-1} \leq \lambda_N$. If the inequality is strict ($\lambda_{N-1} < \lambda_N$) then $\lambda_i < \lambda_N$ for all $i < N$ because of Eq. (2). But then leaf A_N would have no sibling, contradicting a result from last class. □

Lemma 3.3 *Given any optimal prefix code C whose symbols are numbered according to Eq. (1), there exists an optimal prefix code C whose codeword lengths are ordered*

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{N-1} \leq \lambda_N. \tag{3}$$

and whose code is such that $C(A_N)$ and $C(A_{N-1})$ are identical except for the last bit. That is, they are siblings in the binary tree representation of the code.

Proof: Take any optimal prefix code. Both A_{N-1} and A_N must have siblings, as we saw last class. If A_{N-1} and A_N are already siblings, then the proof is done. Otherwise, observe that A_{N-1}, A_N, and their siblings all have the same codeword length. So, swap $C(A_N)$ with the codeword of the sibling of A_{N-1}. This does not change the average code length, but it does make A_{N-1} and A_N siblings. Thus, if the original code was an optimal prefix code, then the new code must be optimal as well. □

Huffman coding

The lemmas give us the following recursive algorithm (due to David Huffman in 1952) for building an optimal prefix code, given an alphabet and an probability function $p()$ defined on the alphabet.

Algorithm (recursive): *Make a Huffman code on an alphabet* $A_1, \ldots, A_N$

1.) Re-number symbols so that $p(A_1) \geq p(A_2) \geq \cdots \geq p(A_N)$
2.) Merge A_{N-1} and A_N into a new symbol A^*, such that $p(A^*) = p(A_{N-1}) + p(A_N)$
3.) Make a Huffman code C^* on the alphabet of $N-1$ symbols $\{A_1, A_2, \ldots, A_{N-2}, A^*\}$
4.) Define a code C for the N symbol alphabet, based on the code C^* for the $N-1$ symbol alphabet, such that:

$$C(A_{N-1}) = C^*(A^*)0, \qquad C(A_N) = C^*(A^*)1, \qquad C(A_i) = C^*(A_i) \text{ for } i \leq N-2.$$

Note, when we merge the symbols A^* might have higher probability than A_{N-2}. So, the ordering of decreasing probability no longer necessarily holds.

Example of constructing a Huffman code

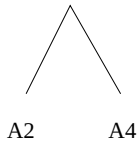
$$\begin{aligned} p(A_1) &= .25 \\ p(A_2) &= .2 \\ p(A_3) &= .4 \\ p(A_4) &= .15 \end{aligned}$$

To make the Huffman code, I like to by write an initial list of the symbols, from highest to lowest probability. (I find I make fewer careless mistakes when the symbols are ordered by probability at the initial stage. Note that I don't re-number the symbols. I merely order them.)

symbol	p()
A_3	.4
A_1	.25
A_2	.2
A_4	.15

We begin by merging A_2 and A_4 to get $A_{2,4}$, and we add this symbol to the list, putting an X beside A_2 and A_4, to indicate that we will not consider them further.

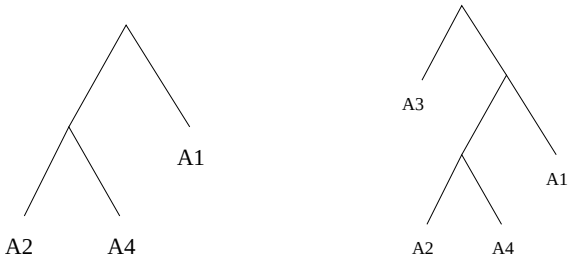
symbol	p()	
A_3	.4	
A_1	.25	
A_2	.2	X
A_4	.15	X
$A_{2,4}$	.35	



Then, we merge A_1 and $A_{2,4}$ to get $A_{1,2,4}$, and we add this symbol to the list, putting an X beside A_1 and $A_{2,4}$.

symbol	p()	
A_3	.4	
A_1	.35	X
A_2	.15	X
A_4	.1	X
$A_{2,4}$	.25	X
$A_{1,2,4}$	.6	

This gives giving us the tree shown on the left. Finally, we merge $A_{1,2,4}$ and A_3 to get the tree shown on the right.



This gives us the Huffman code:

$$\begin{aligned} C(A_1) &= 11 \\ C(A_2) &= 100 \\ C(A_3) &= 0 \\ C(A_4) &= 101 \end{aligned}$$

Note that this is not the only Huffman code we could have constructed. When merging two nodes in the tree, we chose which to put in the left child and which to put in the right child. These decisions affect the code, but they do not affect the code lengths.

Another choice we may sometimes have is which nodes to merge. This choice arises when there are more than two symbols that have the lowest probability. An example in which this ambiguous choice of symbols arises both at the beginning of the algorithm and at an intermediate step is the

following (check it out for yourself):

$$\begin{aligned} p(A_1) &= .1 \\ p(A_2) &= .3 \\ p(A_3) &= .2 \\ p(A_4) &= .1 \\ p(A_5) &= .2 \\ p(A_6) &= .1 \end{aligned}$$

We finish this lecture by proving that the Huffman code indeed yields an optimal prefix code.

Theorem 3.1 *For any set of N symbols and probabilities $p()$, the Huffman code is an optimal prefix code.*

Proof We prove the theorem by induction on N. That is, we prove it for $N = 2$. Then we show that if the theorem is true for $N = K$, it must also be true for $N = K + 1$.

The case of $N = 2$ symbols in the alphabet is trivial since each codeword must have length at least one, and so $C(A_1) = 0$ and $C(A_2) = 1$ must be optimal.

To prove the induction step, suppose we have an alphabet of $K + 1$ symbols and we have renumbered them from highest to lowest probability. The first step of the Huffman coding algorithm, we merge nodes A_K and A_{K+1} into a new node A^*, so that A_K and A_{K+1} are siblings in our Huffman code. We then continue by Huffman coding the reduced alphabet $\{A_1, A_2, \ldots, A_{K-1}, A^*\}$. Let λ_i denote the codewords lengths of the Huffman code on $\{A_1, A_2, \ldots, A_{K-1}, A_K, A_{K+1}.\}$ Note that $\lambda_K = \lambda_{K+1}$ since A_K and A_{K+1} are assumed to be siblings. Also note $p(A^*) = p(A_K) + p(A_{K+1})$.

The average code length of the Huffman code is:

$$\begin{aligned} \sum_{i=1}^{K+1} p(A_i)\ \lambda_i &= \sum_{i=1}^{K-1} p(A_i)\ \lambda_i + (p(A_K)\ +\ p(A_{K+1}))\ \lambda_K \\ &= \sum_{i=1}^{K-1} p(A_i)\ \lambda_i + p(A^*)\ \lambda_K, \\ &= \{\ \sum_{i=1}^{K-1} p(A_i)\ \lambda_i + p(A^*)\ (\lambda_K - 1)\ \}\ + p(A^*) \end{aligned}$$

The terms between the curly brackets on the right side is the average code length of the Huffman code on the reduced alphabet $\{A_1, A_2, \ldots, A_{K-1}, A^*\}$.

Now we apply the induction step. We assume that the Huffman code is an optimal prefix code for the reduced alphabet (N symbols). Therefore, we can replace the above terms between curly brackets as follows:

$$\sum_{i=1}^{K+1} p(A_i)\ \lambda_i \ \leq\ \{\ \sum_{i=1}^{K-1} p(A_i)\ \lambda'_i + p(A^*)\ \lambda'_K\ \}\ + p(A^*)$$

where λ'_i are the codelengths of *any other prefix code* C' on the reduced K symbol alphabet $\{A_1, A_2, \ldots, A_{K-1}, A^*\}$. That is, by the induction hypothesis, the statement of the theorem is true for an alphabet of $N = K$ symbols.

To show the statement holds for $N = K + 1$, we note that right side of the last line above is just the average code length of any prefix code on the $K + 1$ symbols, such that the codewords of A_K and A_{K+1} differ only their last bit. To see this, note:

$$p(A^*)\ \lambda'_K + p(A^*)\ =\ p(A^*)(\ \lambda'_K + 1)\ =\ p(A_K)(\ \lambda'_K + 1) + p(A_{K+1})(\ \lambda'_K + 1)$$

But from Lemma 3.3, there is an optimal prefix code on the $K+1$ symbols such that the codewords of A_K and A_{K+1} differ only in their last bit. Hence, the average length of the Huffman code on $K + 1$ symbols is less than or equal to the average length of some optimal prefix code. Hence the Huffman code must itself be optimal. $\square$