Trees, codes, information and entropy

1. Codes and English

A code is an alphabet or a list of words which are used in place of another alphabet or group of letters. All of our encryption systems are examples codes whose purpose is to hide the meaning of a message from an unintended listener.

Code books which give a list of words or numbers which are used in place of words or phrases to hide the meaning have been common method of encryption. The security of a code of this type is obviously dependent on the security of the codebook.

However, codes are used for many other reasons. For example we use common abbreviations (e.g IMHO, LMAO, NSA and CIA) as a code for either a phrase or a name. Pictures and text are often digitized to be stored in a computer and this is a code for the information that it represents. Radio and television signals are encoded into a signal which can be broadcast or transmitted by cable. Codes are commonly used with text because our usual alphabet or system of letters is not appropriate for storage or transfer of information and so they are encoded in order to store the information and decoded when that data is retreived.

For instance, when the telegraph was invented around 1836, it crudely allowed the transmission of on and off signals. The transmission of long and short pulses over a wire was a breakthrough for its time, but the technology did not allow for a lot of variations of signals. An agreed upon system distinguished between short (a dot) and long (a dash) 'on' signals. The 'off' then represented separators either between dots and dashes, or there was a longer pause between groups of dots and dashes. The groups of dots and dashes were a code which represented letters, numbers or symbols.

Morse code is named after one of the inventors of the telegraph, Samuel Morse, an American artist. The code is chosen so that common letters like E, T, A, I, N require a small number of pulses, but it is not necessarily optimal.

A •−	B −•••	C −•−•	D −••	E •	F ••−•
G ––•	H ••••	I ••	J •–––	K −•−	L •−••
M ––	N −•	O –––	P •––•	Q ––•−	R •−•
S •••	T −	U ••−	V •••−	W •––	X −••−
Y −•––	Z ––••				

This code requires that we put a pause in between the letter that we send. If not, it would be impossible to distinguish between the code for O, MT, TM and TTT because they are all represented by three dashes.

In computer languages, there are codes which are used to represent characters and digits with 0s and 1s. The standard code used in most languages today is known as American Standard Code for Information Interchange (ASCII). The computer represents each letter internally as a sequence of 8 bits (a byte). The standard ASCII table has only 128 characters (note $128 = 2^7$ and $2^8 = 256$) so letters are generally represented by a 0 plus 7 other bits,

although extensions of the ASCII code usually represent 256 or more characters and take advantage of all 8 bits in a byte. This standard ASCII table for alpha-numeric characters handles only the standard roman alphabet, but there are other versions of this table for other languages. The upper case letters A through Z are represented by the numbers 65 through 90 and the lowercase alphabet are represented by 97 through 122. Often this code is given in hexadecimal or (as I have done in the table below) binary.

A 01000001	B 01000010	C 01000011	D 01000100	E 01000101	F 01000110
G 01000111	H 01001000	I 01001001	J 01001010	K 01001011	L 01001100
M 01001101	N 01001110	O 01001111	P 01010000	Q 01010001	R 01010010
S 01010011	T 01010100	U 01010101	V 01010110	W 01010111	X 01011000
Y 01011001	Z 01011010				

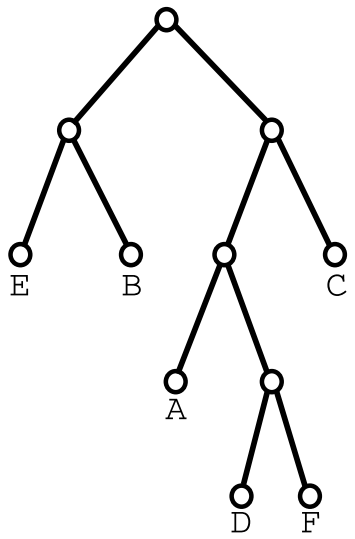
There is a lot of redundancy in this code as a way of representing letters with binary digits because they all begin with 010. Even if we threw the first three bits away, a more efficient code for representing the alphabet would take into account that some letters are more frequent than others and so E and T would have a shorter code than Q, X and Z.

2. Codes and binary trees

We call a binary code *comma-free* if no prefix of the code of a letter is the code of another letter. Morse code is not comma-free, but the ASCII code is because the prefix of any of the codes of a letter is not a code for another letter.

A *complete binary tree* is a data structure consisting of nodes and each node is either a *label* or has two *pointers* (a left and a right) each which point to another node. One node at the top of the tree is referred to as the *root* of the tree. The nodes which contain a label are called the *leaves* of the tree.

If we have a binary tree with labels $S = \{\ell_1, \ell_2, \ldots, \ell_r\}$ then we can make a comma-free code for S by assigning a binary word to each label ℓ_i by following the branches from the root to the leaf labeled by ℓ_i and recording a 1 for each left branch that we follow and a 0 for each right branch.



For instance in the example we have a binary tree with labels $\{A, B, C, D, E, F\}$. From the root to E we must take two left branches and so the code for the letter E will be 11. From the root to the branch labeled B we take a left and then a right pointer so the code for B will be 10. Similarly the code for A is 011, for D is 0101, the code for F is 0100, the code for C is 00.

Notice that in order for a label to have a word which is the prefix of the word of another label it must be that the label has at least one child, but this doesn't happen in our trees since only the leaves are labeled.

Proposition 1. *If $h_1, h_2, \ldots, h_r$ are the heights of the leaves $\ell_1, \ell_2, \ldots, \ell_r$ of a complete binary tree, then*

$$\sum_{i=1}^{r} \frac{1}{2^{h_i}} = 1 \ .$$

Proof: We prove this by induction on the number of leaves in the tree. Assume that the sum of $1/2^{h_i'}$ is equal to 1 for all sets of heights h_i' of a complete binary tree with less than r leaves.

Say that the left branch of the root has the labels $\ell_{i_1}, \ell_{i_2}, \ldots, \ell_{i_d}$ and the right branch of the tree has the labels $\ell_{i_{d+1}}, \ell_{i_{d+2}}, \ldots, \ell_{i_r}$. The left branch of this tree is a complete binary tree with heights of labels equal to $h_{i_1} - 1, h_{i_2} - 1, \ldots, h_{\ell_{i_d}} - 1$, hence

$$\sum_{k=1}^{d} \frac{1}{2^{h_{i_k}-1}} = 1.$$

The right branch of this tree is also a complete binary tree with heights equal to

$$h_{i_{d+1}} - 1, h_{i_{d+2}} - 1, \ldots, h_{i_r} - 1$$

so

$$\sum_{j=d+1}^{r} \frac{1}{2^{h_{i_j}-1}} = 1.$$

Therefore

$$\begin{aligned}
\sum_{i=1}^{r} \frac{1}{2^{h_i}} &= \sum_{k=1}^{d} \frac{1}{2^{h_{i_k}}} + \sum_{j=d+1}^{r} \frac{1}{2^{h_{i_j}}} \\
&= \frac{1}{2} \sum_{k=1}^{d} \frac{1}{2^{h_{i_k}-1}} + \frac{1}{2} \sum_{j=d+1}^{r} \frac{1}{2^{h_{i_j}-1}} \\
&= \frac{1}{2} + \frac{1}{2} = 1
\end{aligned}$$

3. The relationship between entropy and information

Say that we have a file with characters $\ell_1, \ell_2, \ldots, \ell_r$ which are generated at random with respective probabilities $p_1, p_2, \ldots, p_r$. If we encode these characters with a comma-free code of respective lengths $h_1, h_2, \ldots, h_r$, then the expected number of bits that will be required on average per letter is equal to

$$p_1 h_1 + p_2 h_2 + \cdots + p_r h_r \ .$$

This quantity is the expected value of the random variable representing the lengths of the code which occur with the probability that the letters are generated.

When we introduced the notion of entropy, we declared it to be the amount of information in the outcome of a random variable. We should expect this to be related to the expected number of bits per letter. This connection between a numerical quantity associated and what we refer to as the 'information' in a random variable is because of the following theorem.

Theorem 2. *Given a comma-free code with respective codes lengths $h_1, h_2, \ldots, h_r$ which occur with probabilities $p_1, p_2, \ldots, p_r$, the expected number of bits per letter is greater than or equal to the entropy of the random variable $= \sum_{i=1}^{r} p_i log_2(1/p_i)$.*

Proof: We can assume that our comma-free code comes from the code of a complete tree. If not then we can shorten the code by deleting unused leaves. Set $q_i = \frac{1}{2^{h_i}}$, then $\sum_{i=1}^{r} q_i = \sum_{i=1}^{r} \frac{1}{2^{h_i}} = 1$ by Proposition 1 and $h_i = log_2(1/q_i)$. Recall that we have previously shown that for any two finite sets of probabilities such that $\sum_i p_i = \sum_i q_i = 1$ that we have

$$\sum_i p_i log_2(q_i) \leq \sum_i p_i log_2(p_i)$$

and hence

$$\sum_i p_i log_2(1/q_i) \geq \sum_i p_i log_2(1/p_i) \ .$$

The expected number of bits per letter is equal to

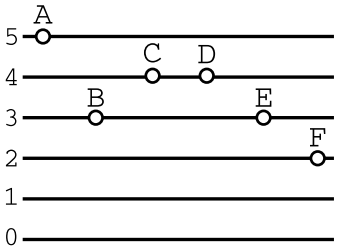
$$\begin{aligned} p_1h_1 + p_2h_2 + \cdots + p_rh_r &= p_1 log_2(1/q_1) + p_2 log_2(1/q_2) + \cdots + p_r log_2(1/q_r) \\ &\geq p_1 log_2(1/p_1) + p_2 log_2(1/p_2) + \cdots + p_r log_2(1/p_r) \\ &= \text{ entropy} \end{aligned}$$

In other words this theorem says that the entropy is a lower bound for the number of bits per letter for a given code. The question is, is it possible to find a code which gives an expected number of bits per letter equal to the entropy? In general, the answer to this question is 'no,' but it is possible to construct a code where the expected number of bits per letter is within one of the entropy.

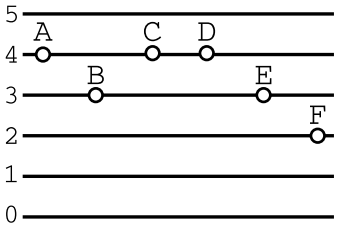
4. Trees from heights

A simple way of constructing this code is to place the alphabet $\ell_1, \ell_2, \ldots, \ell_r$ which occur with probabilities $p_1, p_2, \ldots, p_r$ at heights $h_i = \lceil log_2(1/p_i) \rceil$ and then order these leaves from left to right from the largest height to the smallest. Next, create a tree by grouping pairs of nodes at a given height together to create a branch. If there aren't two nodes at a given height, then the node and all branches attached to it are lowered. This process is not unique because it only specifies that any sets of pairs of nodes are joined. But what is true is that every node will finish with height less than or equal to h_i.

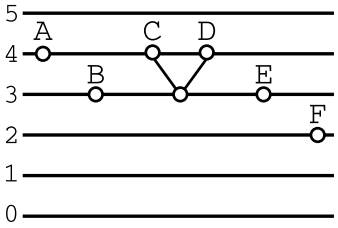
For example if we have letters A, B, C, D, E, F which occur with probability 1/24, 5/24, 1/12, 1/12, 5/24, 3/8 then we can place these letters at respective heights $5, 3, 4, 4, 3, 2$. I will construct this tree in a sequence of steps starting with the following picture.



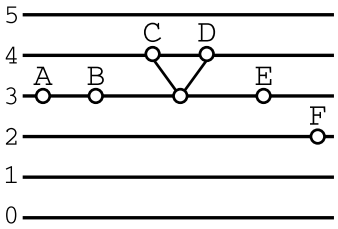
The first step will be to move the node at level 5 to level 4 since there is nothing to pair it with.



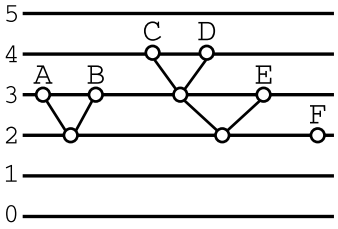
Next group the C and D together because they are on the same level and close to each other (alternatively, we could switch the A and B and then group A and C or A and D together).



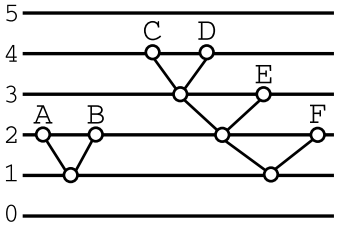
Next since the A is all by itself, we can lower it to level 3.



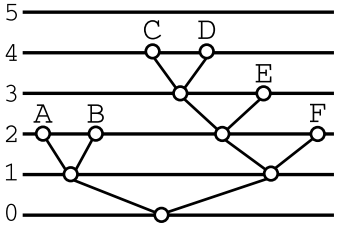
Next, because they are close together, group the A and B node together and the CD node with the E.



Next, we choose to group the branch with CDE and the node labeled with F together (this choice is arbitrary, we could choose to group the branch with AB and CDE or with AB and F together). When we do this, the branch with AB is left on a level by itself and so the branch is lowered a level.



Finally with two nodes on level 1, they can be joined to make a complete tree. The result is a tree from heights.



If we make a tree from heights it is always the case that expected number of bits per letter will be within 1 of the entropy.

Theorem 3. *Let $p_1, p_2, p_3, \ldots, p_r$ be the respective probabilities of a random variable X with outcomes of letters $\ell_1, \ell_2, \ldots, \ell_r$. Say that random words are generated by repeatedly recording the outcomes of this random variable. If the letters are encoded using a tree from heights with respective code lengths $h_1, h_2, \ldots, h_r$, then*

$$p_1 h_1 + p_2 h_2 + \cdots + p_r h_r \leq H(X) + 1.$$

Proof. The key to this theorem is noticing that since the heights h_i are from a tree from heights, it must be that $h_i \leq \lceil log_2(1/p_i) \rceil$. Therefore, the expected number of bits per

letter is equal to

$$\begin{aligned} p_1h_1 + p_2h_2 + \cdots + p_rh_r &\leq p_1\lceil log_2(1/p_1)\rceil + p_2\lceil log_2(1/p_2)\rceil + \cdots + p_r\lceil log_2(1/p_r)\rceil \\ &\leq p_1(log_2(1/p_1)+1) + p_2(log_2(1/p_2)+1) + \cdots + p_r(log_2(1/p_r)+1) \\ &= (p_1log_2(1/p_1) + p_2log_2(1/p_2) + \cdots + p_rlog_2(1/p_r)) + 1 \\ &= H(X) + 1 \end{aligned}$$

□

Consider for example the tree from heights that we created above. The expected number of bits per letter is equal to

$$2 \cdot \frac{1}{24} + 2 \cdot \frac{5}{24} + 4 \cdot \frac{1}{12} + 4 \cdot \frac{1}{12} + 3 \cdot \frac{5}{24} + 2 \cdot \frac{3}{8} \approx 2.54$$

while the entropy of the random variable with those probabilities is

$$\frac{1}{24}log_2(24) + \frac{5}{24}log_2(\frac{24}{5}) + \frac{1}{12}log_2(12) + \frac{1}{12}log_2(12) + \frac{5}{24}log_2(\frac{24}{5}) + \frac{3}{8}log_2(\frac{8}{3}) \approx 2.26\ .$$

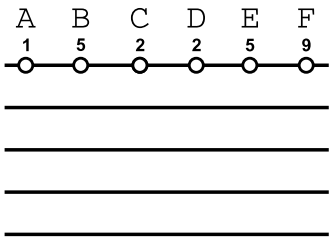
5. The Huffman tree

It turns out that there is an algorithm for producing the 'best' tree possible with the shortest expected number of bits per letter. Like the tree from heights, the code is not (in general) unique, but in this case the expected number of bits per letter is minimized and is as close to the entropy as possible. The only way of getting a shorter number of bits per letter is to encode more than one letter at a time.

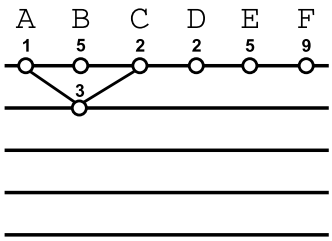
The Huffman tree by starting off with labels $\ell_1, \ell_2, \ldots, \ell_r$ with respective probabilities $p_1, p_2, \ldots, p_r$ and then grouping the pair of nodes together with the smallest two probabilities and creating a new node with two branches and associating the sum of the probabilities to this new node.

Lets create the Huffman tree from the labels A, B, C, D, E, F with the same respective probabilities 1/24, 5/24, 1/12, 1/12, 5/24, 3/8 as in our last example. We will see that the expected number of bits per letter from this new code is better than the one that we created using the method of the tree from heights but (again) larger than the entropy.

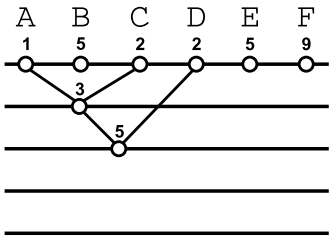
To start, we list the probabilities with a common denominator of 24 so they can be easily compared. Since we don't need the denominators we give them respective weight 1, 5, 2, 2, 5, 9.



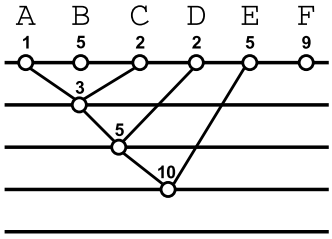
We then group the two nodes with the smallest weight (with 1 and 2) and we give that new node weight $3 = 1 + 2$.



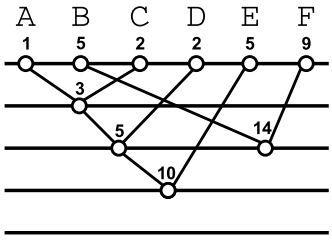
We then group the two nodes with the smallest weight which are now 2 and 3 and give that node a weight $5 = 2 + 3$.



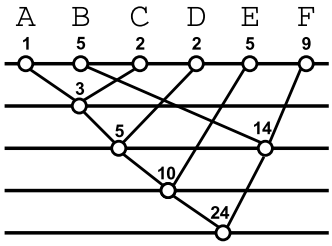
Since there are now three nodes of weight 5, we have a choice. Since it doesn't matter we will group the rightmost two nodes with weight 5 together and give it weight $10 = 5+5$.



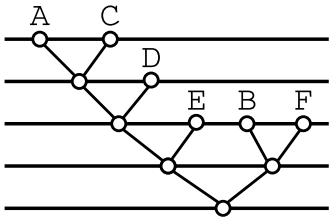
Next we group the node of weight 9 with the node of weight 5 and give it weight $14 = 9+5$. It is often the case that branches will cross over and the last step will be to reorder the nodes to straighten them out.



Since there are only two nodes which are left to be grouped, the last step is to connect them.



Although it is not absolutely necessary, we can then redraw the tree to avoid crossing branches.



Using the probabilities from before we calculate that the expected number of binary registers using this code is equal to

$$4 \cdot \frac{1}{24} + 4 \cdot \frac{2}{24} + 3 \cdot \frac{2}{24} + 2 \cdot \frac{5}{24} + 2 \cdot \frac{5}{24} + 2 \cdot \frac{9}{24} \approx 2.33$$

Again, notice that this is slightly less than the expected number of bits per letter from than the tree from heights, but slightly larger than the entropy.

This is summarized in the following theorem:

Theorem 4. (see [1] for proof) *If the Huffman encoding for a set of probabilities $p_1, p_2, \ldots, p_r$ has heights $h_1, h_2, \ldots, h_r$, then for any other encoding with heights $h'_1, h'_2, \ldots, h'_r$,*

$$p_1 h_1 + p_2 h_2 + \cdots + p_r h_r \leq p_1 h'_1 + p_2 h'_2 + \cdots + p_r h'_r.$$

[1] D.A. Huffman, *A Method for the Construction of Minimum-Redundancy Codes*, Proceedings of the I.R.E., September 1952, pp. 1098–1102.

Exercises:

(1) Say that the letters A, B, C and D are generated randomly and occur with probabilities 1/16, 3/16, 5/16, 7/16 respectively.
   (a) Draw the Huffman tree which has the minimum expected code length.
   (b) For each branch of this tree, assume that if the branch is towards the letter A it is labelled with 1 and other branches labelled with 0. What word does the message 10111011111011110 correspond to?
   (c) What is the expected number of bits needed to encode a letter on average using this code?
   (d) Now encode letters two at a time. Draw the tree with the minimum expected code length using the encoding of letters two at at time.
   (e) What is the expected number of bits *per letter* needed to encode on average using this code?

(2) A random procedure for choosing a three letter word is determined from the charts below and the outcome is a random variable X. The first letter is chosen with probability given by the first table, the second and third letters are chosen using the table below where the entries represent the the number of times the second letter appears (in the column) given the previous letter (in the row).

a	4
s	2
d	1
f	3

	a	s	d	f
a	5	0	2	3
s	2	4	4	0
d	1	5	3	1
f	6	1	1	2

(a) Find $H(X|$ the third letter is 'f')
(b) Encode the set of three letter words that end in an 'f' by using a Huffman tree with one word per leaf of the tree.
(c) What is the expected number of bits required to store a single word using this coding scheme?