Chapter 5

Probability

We have been considering the model of Figure 5.1, an information handling system in which symbols from an input are encoded into bits, which are then sent across a \channel" to a receiver and get decoded back into symbols. In earlier chapters of these notes we have looked at various components in this model. Now we return to the source and model it more fully, in terms of probability distributions.

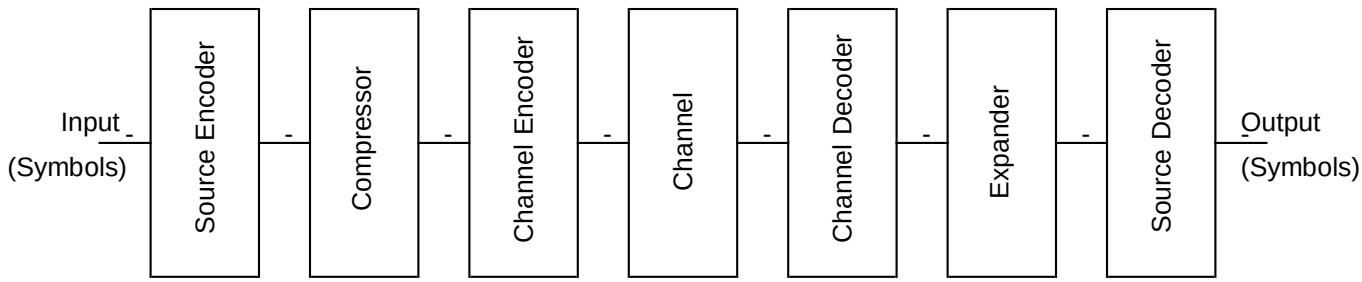

Figure 5.1: Communication system

5.1 Sources

A source provides a symbol or a sequence of symbols, selected from some set. The selection process might be an experiment, such as ipping a coin or rolling dice. Or it might be the observation of actions not caused by the observer. Or the sequence of symbols could be from a representation of some object, such as characters from text, or pixels from an image.

We consider only cases with a nite number of symbols to choose from, where the symbols are both mutually exclusive (only one can be chosen at a time) and exhaustive (one is actually chosen). Each choice constitutes an outcome . If we want to trace the sequence of outcomes, and the information that travels with them from the input to the output, we need to be able to say what the outcome is.

If we know the outcome, we have a perfectly good way of denoting it. We can simply name the symbol chosen, and ignore all the rest of the symbols, which were not chosen. But what if we do not yet know the outcome, or are uncertain to any degree? How are we supposed to express our state of knowledge if there is

Author: Paul Pen eld, Jr.
This document: http://www.mtl.mit.edu/Courses/6.050/2013/notes/chapter5.pdf
Version 1.7, February 28, 2013. Copyright c 2013 Massachusetts Institute of Technology
Start of notes back next j 6.050J/2.110J home page j Search j Comments and inquiries

The notion here is that the amount of information we learn upon hearing the outcome is the minimum number of bits that could have been used to tell us, i.e., to specify the symbol. This approach has some merit but there are two problems.

First, an actual specification of one symbol by means of a sequence of bits requires an integral number of bits. What if the number of symbols is not an integral power of two? For a single selection, there may not be much that can be done, but if the source makes repeated selections and these are all to be specified, they can be grouped together to recover the fractional bits. For example if there are five possible symbols, then three bits would be needed for a single symbol, but the 25 possible combinations of two symbols could be communicated with five bits rather than six, and the 125 combinations of three symbols could get by with seven bits (2.33 bits per symbol). This is not much greater than $\log_2(5)$ which is 2.32 bits. We would like our definition of information to apply even if we group symbols in a sequence. In other words, we need to deal with fractional bits..

Second, different events may have different likelihoods of being selected. This may be inherent in the experiment or in our choice of a partition. We may know that our dice are loaded, or, as in the game of blackjack, we may want to use a partition where all face cards are grouped together. Or perhaps we already know the result (one $p(A_i)$ equals 1 and all others equal 0), so no further information is gained. Our definition of information should apply to all these cases.

To illustrate the second problem, consider a class of 32 students, of whom two are women and 30 are men. If one student is chosen and we want to know which one, our uncertainty is initially five bits, since that is what would be necessary to specify the outcome. If a student is chosen at random, the probability of each being chosen is 1/32. The choice of student also leads to a gender event, either "woman chosen" with probability $p(W) = 2/32$ or "man chosen" with probability $p(M) = 30/32$.

How much information do we gain if we are told that the choice is a woman but not told which one? Our uncertainty is reduced from five bits to one bit (the amount necessary to specify which of the two women it was). Therefore the information we have gained is four bits. What if we are told that the choice is a man but not which one? Our uncertainty is reduced from five bits to $\log_2(30)$ or 4.91 bits—we have learned only 0.09 bits of information.

The point here is that if we have a partition whose events have different probabilities, we learn different amounts from different outcomes. If the outcome was a likely one we learn less than if the outcome was unlikely. We showed this idea by describing each outcome in terms of the selection of an event from an underlying, fundamental partition, but the principle applies even if we don't care about or don't know about the fundamental partition. The information learned from outcome i is $\log_2(1/p(A_i))$ bits. As a special case, if $p(A_i) = 1$ for some i, then the information learned from that outcome is 0 bits since $\log_2(1) = 0$; this is what we would expect because we already are certain of the outcome and cannot learn anything more.

If we want to quantify our uncertainty before learning an outcome, we cannot use the information gained by any specific outcomes, because we would not know which to use. Instead, we average over all possible outcomes, i.e., over all events in the partition with nonzero probability. The average information per outcome is found by multiplying the information, in bits, for each event A_i by $p(A_i)$ and adding them up:

$$I = \sum_i p(A_i) \log_2 \left(\frac{1}{p(A_i)} \right) \tag{5.14}$$

This quantity, which is of fundamental importance for characterizing the information of sources, is called the **entropy** of a source. The formula applies if the probabilities are all equal and also if they are not; it applies if the outcome is known and therefore one of the probabilities is 1 and all the others 0; it works for any partition, not just the fundamental partition.

In this and other formulas for information, care must be taken with events A_i that have zero probability $p(A_i)$. These cases can be treated as though they have a very small but nonzero probability. Yes, the logarithm $\log_2(1/x)$ approaches infinity if its argument $1/x$ approaches infinity, but it does so very slowly. The product $x \log_2(1/x)$ actually approaches zero. Therefore all terms in the sum in Equation 5.14 with $p(A_i) = 0$ can be directly set to zero even though the formula might suggest an indeterminate result, or a calculating procedure might encounter a "divide by zero" error.

5.7.1 Information is Not Negative

The entropy of a source is the information we do not have about an unknown outcome. It is reasonable to expect that this is positive (or zero), because otherwise we might know less as a result of learning something. The formula for I, Equation 5.14, guarantees this property. Each term in the summation is the product of two positive (or 0) numbers: $p(A_i)$ is positive (or 0), and so is the logarithm because its argument, being the reciprocal of $p(A_i)$, is greater than (or equal to) 1.

5.7.2 Two Ways to Write the Formula

Equation 5.14 is written showing the reciprocal of $p(A_i)$ explicitly. This was done because most people find it easier to think about positive than negative numbers, and especially negative logarithms. Because

$$\log_2\left(\frac{1}{x}\right) = -\log_2(x) \tag{5.15}$$

the formula for entropy can also be written with terms that are inherently negative,

$$I = -\sum_i p(A_i)\log_2\left(p(A_i)\right) \tag{5.16}$$

Some people will find formulas like Equation 5.16 with minus signs and negative logarithms preferable to ones like Equation 5.14 with fraction bars. Take your choice.

5.7.3 Units of Information

Like other physical quantities, information is expressed in units. The formulas above are for the number of bits. This is consistent with the use of logarithms with base 2.

Sometimes logarithms with other bases are used. Formulas like Equation 5.14 or Equation 5.16 with natural logarithms (base e) give the information in **natural bits** or **nits**. One nit is $\log_2(e)$ bits (1.443 bits).

Other units for information are possible. Most computers deal with information eight bits at a time. A **byte** is equal to 8 bits.

Memory devices can handle larger amounts of data, and it is customary for metric-system prefixes to be used for approximate multiples of a thousand, million, and so forth. Because 2^{10} (1024) is close to 10^3 (1000), 1024 bytes is defined as a **kilobyte** or **kB**. Then 1024 kB is a **megabyte** or **MB**, and so on to **gigabytes GB** (2^{30} bytes or approximately 10^9 bytes), **terabytes TB** (2^{40} bytes), **petabytes PB** (2^{50} bytes), **exabytes EB** (2^{60} bytes), and beyond.

In physical systems the information we do not have is very large, and generally entropy is expressed in **Joules/Kelvin**. One J/K is about 1.045×10^{23} bits.

5.7.4 Dimensions of Information

It is convenient to think of physical quantities as having dimensions. For example, the dimensions of velocity are length over time; velocity can be expressed in meters (a unit of length) per second (a unit of time). Numbers are dimensionless, as are probabilities and the base, argument, and result of the logarithm function. Two quantities with different dimensions cannot be added, subtracted, or compared to see which is larger, though they can be multiplied or divided to produce products or ratios with other dimensions.

The results of algebraic derivations must be dimensionally consistent. Keeping track of dimensions during derivations is an excellent way of detecting errors early, so they can be corrected quickly and easily.

Traditional lists of physical dimensions start with a few basic ones, usually including length, time, mass, and temperature. Then dimensions of other physical quantities are expressed in terms of these. Information does not appear on such lists because it is only recently that a fundamental role for information in physics has been envisioned. Entropy in thermodynamic systems is not a basic dimension but one derived from temperature and energy, which is itself expressed in terms of mass, length, and time.

In the future information might replace temperature on the list of basic dimensions, but even if that never happens, information can and should be included in dimension analyses.

5.7.5 Examples

If a partition has two events with probabilities p and $(1-p)$, the information in bits per symbol is

$$I = p\log_2\left(\frac{1}{p}\right) + (1-p)\log_2\left(\frac{1}{1-p}\right) \tag{5.17}$$

which is shown, as a function of p, in Figure 5.3. It is largest (1 bit) for $p = 0.5$. Thus the information is a maximum when the probabilities of the two possible events are equal. Furthermore, for the entire range of probabilities between $p = 0.4$ and $p = 0.6$ the information is close to 1 bit. It is equal to 0 for $p = 0$ and for $p = 1$. This is reasonable because for these two values of p the outcome is certain, so no information is gained by learning it.

For partitions with more than two possible events the information per symbol can be higher than 1 bit. If there are n possible events the information per symbol lies between 0 and $\log_2(n)$ bits, the maximum value being achieved when all probabilities are equal.

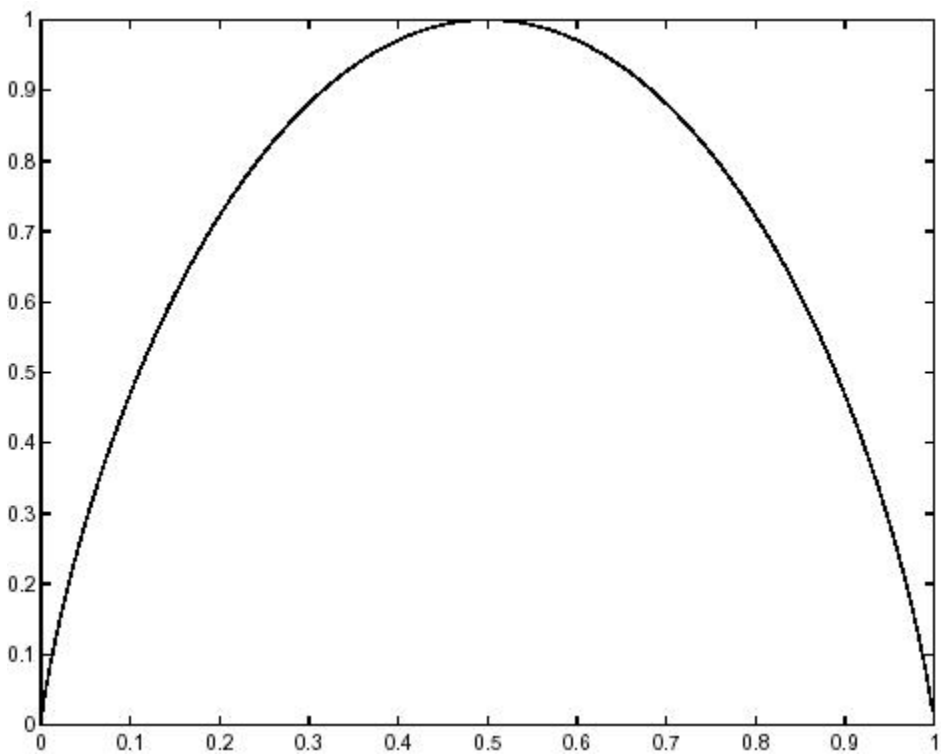

Figure 5.3: Entropy of a source with two symbols as a function of p, one of the two probabilities

5.8 Efficient Source Coding

If a source has n possible symbols then a fixed-length code for it would require $\log_2(n)$ (or the next higher integer) bits per symbol. The average information per symbol would be smaller if the symbols have significantly different probabilities. In this case, a fixed-length code would be inefficient. Is it possible to

encode a stream of symbols from such a source with fewer bits on average, by using a variable-length code with fewer bits for the more probable symbols and more bits for the less probable ones?

Certainly. Morse Code is an example of a variable length code which does this quite well. There is a general procedure for constructing codes of this sort which are very efficient (in fact, they require an average of less than $I + 1$ bits per symbol, even if I is considerably below $\log_2(n)$. The codes are called Huffman codes after MIT graduate David Huffman (1925–1999), and they are widely used in communication systems. See Section 5.10.

5.9 Detail: Life Insurance

An example of statistics and probability in everyday life is their use in life insurance. We consider here only one-year term insurance (insurance companies are very creative in marketing more complex policies that combine aspects of insurance, savings, investment, retirement income, and tax minimization).

When you buy a life insurance policy, you pay a premium of so many dollars and, if you die during the year, your beneficiaries are paid a much larger amount. Life insurance can be thought of in many ways.

From a gambler's perspective, you are betting that you will die and the insurance company is betting that you will live. Each of you can estimate the probability that you will die, and because probabilities are subjective, they may differ enough to make such a bet seem favorable to both parties (for example, suppose you know about a threatening medical situation and do not disclose it to the insurance company). Insurance companies use mortality tables such as Table 5.2 (shown also in Figure 5.4) for setting their rates. (Interestingly, insurance companies also sell annuities, which from a gambler's perspective are bets the other way around—the company is betting that you will die soon, and you are betting that you will live a long time.)

Another way of thinking about life insurance is as a financial investment. Since insurance companies on average pay out less than they collect (otherwise they would go bankrupt), investors would normally do better investing in another way, for example by putting their money in a bank, depending, of course, on what rates are available.

Most people who buy life insurance, of course, do not regard it as either a bet or an investment, but rather as a safety net. They know that if they die, their income will cease and they want to provide a partial replacement for their dependents, usually children and spouses. The premium is small because the probability of death is low during the years when such a safety net is important, but the benefit in the unlikely case of death may be very important to the beneficiaries. Such a safety net may not be as important to very rich people (who can afford the loss of income), single people without dependents, or older people whose children have grown up.

Figure 5.4 and Table 5.2 show the probability of death during one year, as a function of age, for the cohort of U. S. residents born in 1988 (data from The Berkeley Mortality Database[3]).

Note that for almost all ages, males are more likely to die than females. If you were running an insurance company, would you set different life-insurance premiums for men and women? If your competitors did, you would have to. But some people argue that charging different premiums is a type of sexual discrimination that should not be permitted. Since December 21, 2012 the European Union has had gender-neutral premiums. American insurance companies point to statistics such as Table 5.2 to justify different rates. The other side of the argument is that much of the difference is due to risky behavior by some but not all men (eating poorly, drinking to excess, speeding, smoking, and even being more successful in suicide attempts due to use of more violent techniques) and that it might be fair to charge different premiums based on individual behavior patterns, but not on gender alone.

[3]The Berkeley Mortality Database can be accessed online: http://www.demog.berkeley.edu/~bmd/states.html

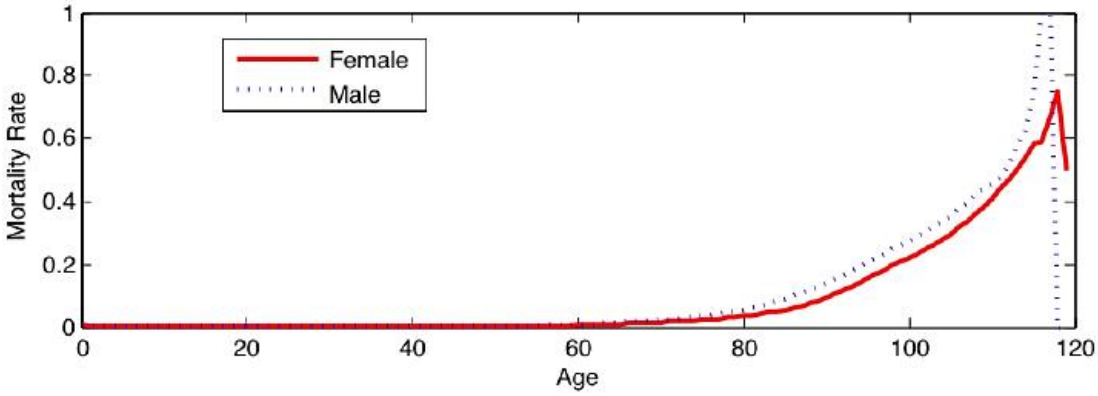

Figure 5.4: Probability of death during one year for U. S. residents born in 1988.

Age	Female	Male	Age	Female	Male	Age	Female	Male
0	0.008969	0.011126	40	0.000945	0.002205	80	0.035107	0.055995
1	0.000727	0.000809	41	0.001007	0.002305	81	0.038323	0.061479
2	0.000384	0.000526	42	0.00107	0.002395	82	0.041973	0.067728
3	0.000323	0.000415	43	0.001144	0.002465	83	0.046087	0.074872
4	0.000222	0.000304	44	0.001238	0.002524	84	0.050745	0.082817
5	0.000212	0.000274	45	0.001343	0.002605	85	0.056048	0.091428
6	0.000182	0.000253	46	0.001469	0.002709	86	0.062068	0.100533
7	0.000162	0.000233	47	0.001616	0.002856	87	0.06888	0.110117
8	0.000172	0.000213	48	0.001785	0.003047	88	0.076551	0.120177
9	0.000152	0.000162	49	0.001975	0.003295	89	0.085096	0.130677
10	0.000142	0.000132	50	0.002198	0.003566	90	0.094583	0.141746
11	0.000142	0.000132	51	0.002454	0.003895	91	0.105042	0.153466
12	0.000162	0.000203	52	0.002743	0.004239	92	0.116464	0.165847
13	0.000202	0.000355	53	0.003055	0.00463	93	0.128961	0.179017
14	0.000263	0.000559	54	0.003402	0.00505	94	0.142521	0.193042
15	0.000324	0.000793	55	0.003795	0.005553	95	0.156269	0.207063
16	0.000395	0.001007	56	0.004245	0.006132	96	0.169964	0.221088
17	0.000426	0.001161	57	0.004701	0.006733	97	0.183378	0.234885
18	0.000436	0.001254	58	0.005153	0.007357	98	0.196114	0.248308
19	0.000426	0.001276	59	0.005644	0.008028	99	0.208034	0.261145
20	0.000406	0.001288	60	0.006133	0.008728	100	0.220629	0.274626
21	0.000386	0.00131	61	0.006706	0.009549	101	0.234167	0.289075
22	0.000386	0.001312	62	0.007479	0.010629	102	0.248567	0.304011
23	0.000396	0.001293	63	0.008491	0.012065	103	0.263996	0.319538
24	0.000417	0.001274	64	0.009686	0.013769	104	0.280461	0.337802
25	0.000447	0.001245	65	0.011028	0.015702	105	0.298313	0.354839
26	0.000468	0.001226	66	0.012368	0.017649	106	0.317585	0.375342
27	0.000488	0.001237	67	0.013559	0.019403	107	0.337284	0.395161
28	0.000519	0.001301	68	0.014525	0.020813	108	0.359638	0.420732
29	0.00055	0.001406	69	0.015363	0.022053	109	0.383459	0.439252
30	0.000581	0.001532	70	0.016237	0.023393	110	0.408964	0.455882
31	0.000612	0.001649	71	0.017299	0.025054	111	0.437768	0.47619
32	0.000643	0.001735	72	0.018526	0.027029	112	0.466216	0.52
33	0.000674	0.00179	73	0.019972	0.029387	113	0.494505	0.571429
34	0.000705	0.001824	74	0.02163	0.032149	114	0.537037	0.625
35	0.000747	0.001859	75	0.023551	0.035267	115	0.580645	0.75
36	0.000788	0.001904	76	0.02564	0.038735	116	0.588235	1
37	0.00083	0.001961	77	0.027809	0.042502	117	0.666667	1
38	0.000861	0.002028	78	0.030011	0.046592	118	0.75	0
39	0.000903	0.002105	79	0.032378	0.051093	119	0.5	0

Table 5.2: Mortality table for U. S. residents born in 1988

5.10 Detail: Efficient Source Code

Sometimes source coding and compression for communication systems of the sort shown in Figure 5.1 are done together (it is an open question whether there are practical benefits to combining source and channel coding). For sources with a finite number of symbols, but with unequal probabilities of appearing in the input stream, there is an elegant, simple technique for source coding with minimum redundancy.

5.10.1 Example of a Finite Source

Consider a source which generates symbols which are MIT letter grades, with possible values A, B, C, D, and F. You are asked to design a system which can transmit a stream of such grades, produced at the rate of one symbol per second, over a communications channel that can only carry two boolean digits, each 0 or 1, per second.[4]

First, assume nothing about the grade distribution. To transmit each symbol separately, you must encode each as a sequence of bits (boolean digits). Using 7-bit ASCII code is wasteful; we have only five symbols, and ASCII can handle 128. Since there are only five possible values, the grades can be coded in three bits per symbol. But the channel can only process two bits per second.

However, three bits is more than needed. The entropy, assuming there is no information about the probabilities, is at most $\log_2(5) = 2.32$ bits. This is also $\sum_i p(A_i)\log_2(1/p(A_i))$ where there are five such p_i, each equal to 1/5. Why did we need three bits in the first case? Because we had no way of transmitting a partial bit. To do better, we can use "block coding." We group the symbols in blocks of, say, three. The information in each block is three times the information per symbol, or 6.97 bits. Thus a block can be transmitted using 7 boolean bits (there are 125 distinct sequences of three grades and 128 possible patterns available in 7 bits). Of course we also need a way of signifying the end, and a way of saying that the final word transmitted has only one valid grade (or two), not three.

But this is still too many bits per second for the channel to handle. So let's look at the probability distribution of the symbols. In a typical "B-centered" MIT course with good students, the grade distribution might be as shown in Table 5.3. Assuming this as a probability distribution, what is the information per symbol and what is the average information per symbol? This calculation is shown in Table 5.4. The information per symbol is 1.840 bits. Since this is less than two bits perhaps the symbols can be encoded to use this channel.

A	B	C	D	F
25%	50%	12.5%	10%	2.5%

Table 5.3: Distribution of grades for a typical MIT course

Symbol	Probability	Information	Contribution to average
	p	$\log\left(\frac{1}{p}\right)$	$p\log\left(\frac{1}{p}\right)$
A	0.25	2 bits	0.5 bits
B	0.50	1 bit	0.5 bits
C	0.125	3 bits	0.375 bits
D	0.10	3.32 bits	0.332 bits
F	0.025	5.32 bits	0.133 bits
Total	1.00		1.840 bits

Table 5.4: Information distribution for grades in an average MIT distribution

[4]Boolean digits, or binary digits, are usually called "bits." The word "bit" also refers to a unit of information. When a boolean digit carries exactly one bit of information there may be no confusion. But inefficient codes or redundant codes may have boolean digit sequences that are longer than the minimum and therefore carry less than one bit of information per bit. This same confusion attends other units of measure, for example meter, second, etc.

5.10.2 Huffman Code

David A. Huffman (August 9, 1925–October 6, 1999) was a graduate student at MIT. To solve a homework assignment for a course he was taking from Prof. Robert M. Fano, he devised a way of encoding symbols with different probabilities, with minimum redundancy and without special symbol frames, and hence most compactly. He described it in *Proceedings of the IRE*, September 1962. His algorithm is very simple. The objective is to come up with a "codebook" (a string of bits for each symbol) so that the average code length is minimized. Presumably infrequent symbols would get long codes, and common symbols short codes, just like in Morse code. The algorithm is as follows (you can follow along by referring to Table 5.5):

1. **Initialize:** Let the partial code for each symbol initially be the empty bit string. Define corresponding to each symbol a "symbol-set," with just that one symbol in it, and a probability equal to the probability of that symbol.

2. **Loop-test:** If there is exactly one symbol-set (its probability must be 1) you are done. The codebook consists of the codes associated with each of the symbols in that symbol-set.

3. **Loop-action:** If there are two or more symbol-sets, take the two with the lowest probabilities (in case of a tie, choose any two). Prepend the codes for those in one symbol-set with 0, and the other with 1. Define a new symbol-set which is the union of the two symbol-sets just processed, with probability equal to the sum of the two probabilities. Replace the two symbol-sets with the new one. The number of symbol-sets is thereby reduced by one. Repeat this loop, including the loop test, until only one symbol-set remains.

Note that this procedure generally produces a variable-length code. If there are n distinct symbols, at least two of them have codes with the maximum length.

For our example, we start out with five symbol-sets, and reduce the number of symbol-sets by one each step, until we are left with just one. The steps are shown in Table 5.5, and the final codebook in Table 5.6.

Start: (A='-' p=0.25) (B='-' p=0.5) (C='-' p=0.125) (D='-' p=0.1) (F='-' p=0.025)
Next: (A='-' p=0.25) (B='-' p=0.5) (C='-' p=0.125) (D='1' F='0' p=0.125)
Next: (A='-' p=0.25) (B='-' p=0.5) (C='1' D='01' F='00' p=0.25)
Next: (B='-' p=0.5) (A='1' C='01' D='001' F='000' p=0.5)
Final: (B='1' A='01' C='001' D='0001' F='0000' p=1.0)

Table 5.5: Huffman coding for MIT course grade distribution, where '-' denotes the empty bit string

Symbol	Code
A	0 1
B	1
C	0 0 1
D	0 0 0 1
F	0 0 0 0

Table 5.6: Huffman Codebook for typical MIT grade distribution

Is this code really compact? The most frequent symbol (B) is given the shortest code and the least frequent symbols (D and F) the longest codes, so on average the number of bits needed for an input stream which obeys the assumed probability distribution is indeed short, as shown in Table 5.7.

Compare this table with the earlier table of information content, Table 5.4. Note that the average coded length per symbol, 1.875 bits, is greater than the information per symbol, which is 1.840 bits. This is because the symbols D and F cannot be encoded in fractional bits. If a block of several symbols were considered

Symbol	Code	Probability	Code length	Contribution to average
A	01	0.25	2	0.5
B	1	0.50	1	0.5
C	001	0.125	3	0.375
D	0001	0.1	3.32	0.4
F	0000	0.025	5.32	0.1
Total		1.00		1.875 bits

Table 5.7: Huffman coding of typical MIT grade distribution

together, the average length of the Huffman code could be closer to the actual information per symbol, but not below it.

The channel can handle two bits per second. By using this code, you can transmit over the channel slightly more than one symbol per second on average. You can achieve your design objective.

There are at least six practical things to consider about Huffman codes:

- A burst of D or F grades might occur. It is necessary for the encoder to store these bits until the channel can catch up. How big a buffer is needed for storage? What will happen if the buffer overflows?
- The output may be delayed because of a buffer backup. The time between an input and the associated output is called the "latency." For interactive systems you want to keep latency low. The number of bits processed per second, the "throughput," is more important in other applications that are not interactive.
- The output will not occur at regularly spaced intervals, because of delays caused by bursts. In some real-time applications like audio, this may be important.
- What if we are wrong in our presumed probability distributions? One large course might give fewer A and B grades and more C and D. Our coding would be inefficient, and there might be buffer overflow.
- The decoder needs to know how to break the stream of bits into individual codes. The rule in this case is, break after '1' or after '0000', whichever comes first. However, there are many possible Huffman codes, corresponding to different choices for 0 and 1 prepending in step 3 of the algorithm above. Most do not have such simple rules. It can be hard (although it is always possible) to know where the inter-symbol breaks should be put.
- The codebook itself must be given, in advance, to both the encoder and decoder. This might be done by transmitting the codebook over the channel once.

5.10.3 Another Example

Freshmen at MIT are on a "pass/no-record" system during their first semester on campus. Grades of A, B, and C are reported on transcripts as P (pass), and D and F are not reported (for our purposes we will designate this as no-record, N). Let's design a system to send these P and N symbols to a printer at the fastest average rate. Without considering probabilities, 1 bit per symbol is needed. But the probabilities (assuming the typical MIT grade distribution in Table 5.3) are $p(P) = p(A) + p(B) + p(C) = 0.875$, and $p(N) = p(D) + p(F) = 0.125$. The information per symbol is therefore not 1 bit but only $0.875 \times log_2(1/0.875) + 0.125 \times log_2(1/0.125) = 0.544$ bits. Huffman coding on single symbols does not help. We need to take groups of bits together. For example eleven grades as a block would have 5.98 bits of information and could in principle be encoded to require only 6 bits to be sent to the printer.