**UNIT V**

**Source coding**

Coding theory is the study of the properties of [codes](https://en.wikipedia.org/wiki/Code) and their respective fitness for specific applications. Codes are used for [data compression,](https://en.wikipedia.org/wiki/Data_compression) [cryptography,](https://en.wikipedia.org/wiki/Cryptography) [error-correction,](https://en.wikipedia.org/wiki/Error-correction) and [networking.](https://en.wikipedia.org/wiki/Network_coding) Codes are studied by various scientific disciplines—such as [information theory,](https://en.wikipedia.org/wiki/Information_theory) [electrical engineering,](https://en.wikipedia.org/wiki/Electrical_engineering) [mathematics,](https://en.wikipedia.org/wiki/Mathematics) [linguistics,](https://en.wikipedia.org/wiki/Linguistics) and [computer](https://en.wikipedia.org/wiki/Computer_science) [science](https://en.wikipedia.org/wiki/Computer_science)—for the purpose of designing efficient and reliable [data transmission](https://en.wikipedia.org/wiki/Data_transmission) methods. This typically involves the removal of redundancy and the correction or detection of errors in the transmitted data.

The aim of source coding is to take the source data and make it smaller.

All source models in information theory may be viewed as random process or random sequence models. Let us consider the example of a discrete memory less source (DMS), which is a simple random sequence model.

A DMS is a source whose output is a sequence of letters such that each letter is independently selected from a fixed alphabet consisting of letters; say a1, a2

Independent of each other. A fixed probability assignment for the occurrence of each letter is also assumed. Let us, consider a small example to appreciate the importance of probability assignment of the source letters.

Let us consider a source with four letters a1, a2, a3 and a4 with P(a1)=0.5, P(a2)=0.25, P(a3)= 0.13, P(a4)=0.12. Let us decide to go for binary coding of these four

Source letters While this can be done in multiple ways, two encoded representations are shown below:

*Code Representation#1:*

a1: 00, a2:01, a3:10, a4:11

*Code Representation#2:*

a1: 0, a2:10, a3:001, a4:110

It is easy to see that in method #1 the probability assignment of a source letter has not been considered and all letters have been represented by two bits each. However in

The second method only a1 has been encoded in one bit, a2 in two bits and the remaining two in three bits. It is easy to see that the average number of bits to be used per source letter for the two methods is not the same. ( *~~a~~* for method #1=2 bits per letter and *~~a~~* for method #2 < 2 bits per letter). So, if we consider the issue of encoding a long sequence of

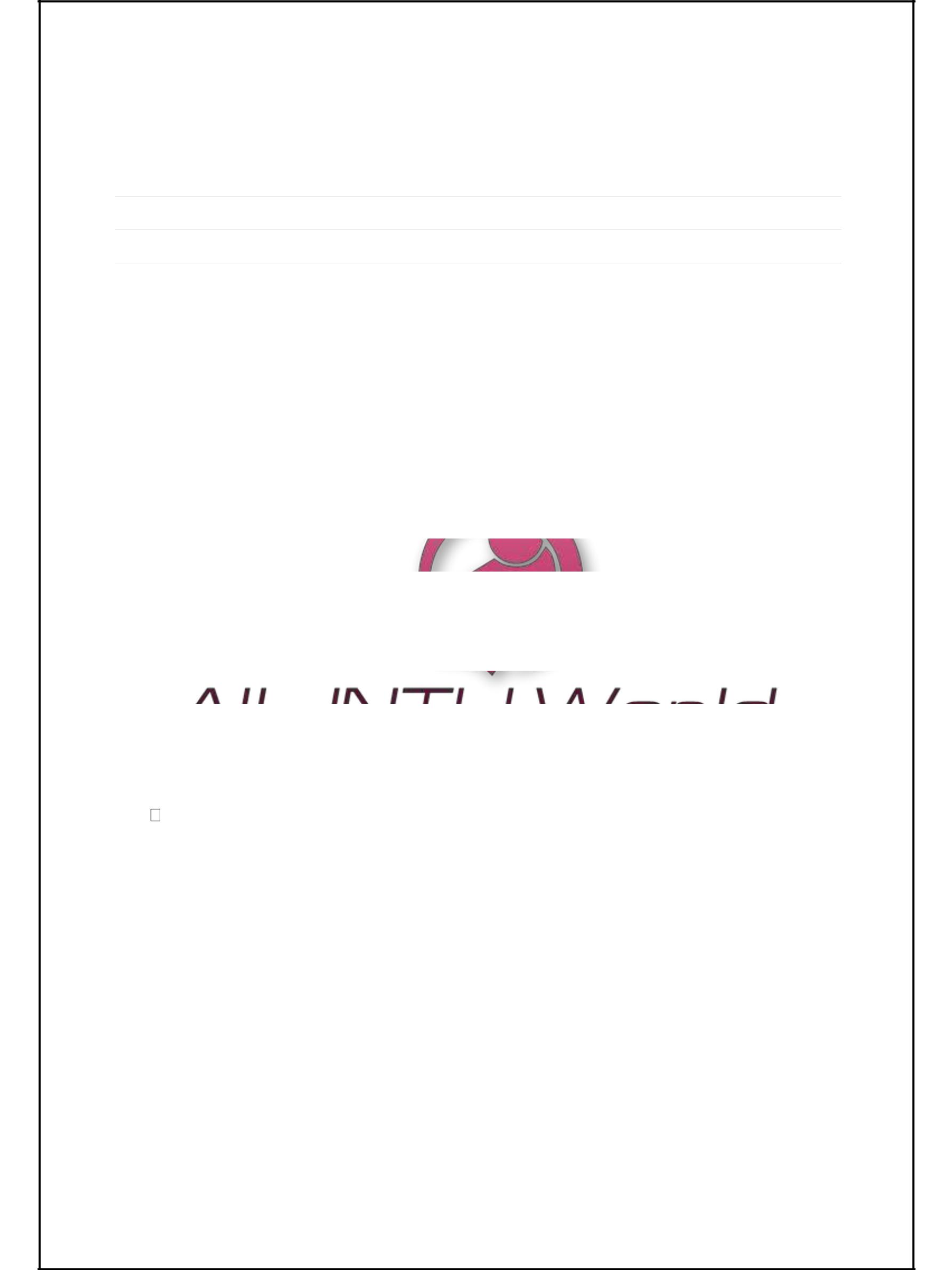
Letters we have to transmit less number of bits following the second method. This is an important aspect of source coding operation in general. At this point, let us note

1. We observe that assignment of small number of bits to more probable letters and assignment of larger number of bits to less probable letters (or symbols) may lead to efficient source encoding scheme.

1. However, one has to take additional care while transmitting the encoded letters. A careful inspection of the binary representation of the symbols in method #2 reveals that it may lead to confusion (at the decoder end) in deciding the end of binary representation of a letter and beginning of the subsequent letter.

So a source-encoding scheme should ensure that

1. The average number of coded bits (or letters in general) required per source letter is as small as possible and
2. The source letters can be fully retrieved from a received encoded sequence.



Shannon-Fano Code

Shannon–Fano coding, named after Claude Elwood Shannon and Robert Fano, is a technique for constructing a prefix code based on a set of symbols and their probabilities. It is suboptimal in the sense that it does not achieve the lowest possible expected codeword length like Huffman coding; however unlike Huffman coding, it does guarantee that all codeword lengths are within one bit of their theoretical ideal *I(x) =−log* *P(x).*

In Shannon–Fano coding, the symbols are arranged in order from most probable to least probable, and then divided into two sets whose total probabilities are as close as possible to being equal. All symbols then have the first digits of their codes assigned; symbols in the first set receive "0" and symbols in the second set receive "1". As long as any sets with more than one member remain, the same process is repeated on those sets, to determine successive digits of their codes. When a set has been reduced to one symbol, of course, this means the symbol's code is complete and will not form the prefix of any other symbol's code.

The algorithm works, and it produces fairly efficient variable-length encodings; when the two smaller sets produced by a partitioning are in fact of equal probability, the one bit of information used to distinguish them is used most efficiently. Unfortunately, Shannon–Fano does not always produce optimal prefix codes.

For this reason, Shannon–Fano is almost never used; Huffman coding is almost as computationally simple and produces prefix codes that always achieve the lowest expected code word length. Shannon–Fano coding is used in the IMPLODE compression method, which is part of the ZIP file format, where it is desired to apply a simple algorithm with high performance and minimum requirements for programming.

Shannon-Fano Algorithm:

A Shannon–Fano tree is built according to a specification designed to define an effective code table. The actual algorithm is simple:

For a given list of symbols, develop a corresponding list of probabilities or frequency counts so that each symbol’s relative frequency of occurrence is known.

 Sort the lists of symbols according to frequency, with the most frequently occurring

Symbols at the left and the least common at the right.

 Divide the list into two parts, with the total frequency counts of the left part being as

Close to the total of the right as possible.

 The left part of the list is assigned the binary digit 0, and the right part is assigned the digit 1. This means that the codes for the symbols in the first part will all start with 0, and the codes in the second part will all start with 1.

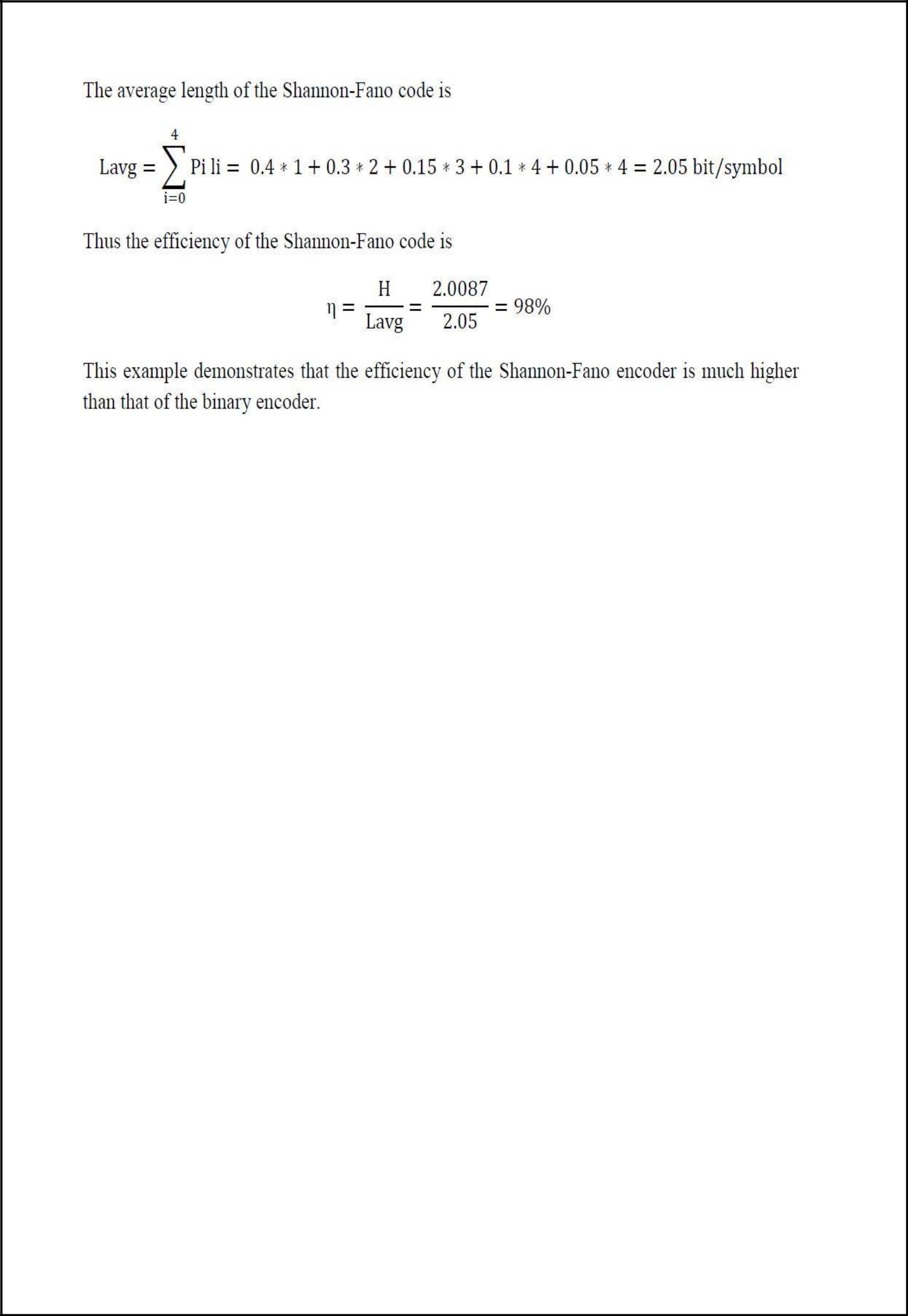
Recursively apply the steps 3 and 4 to each of the two halves, subdividing groups

and adding bits to the codes until each symbol has become a corresponding code leaf on the tree.

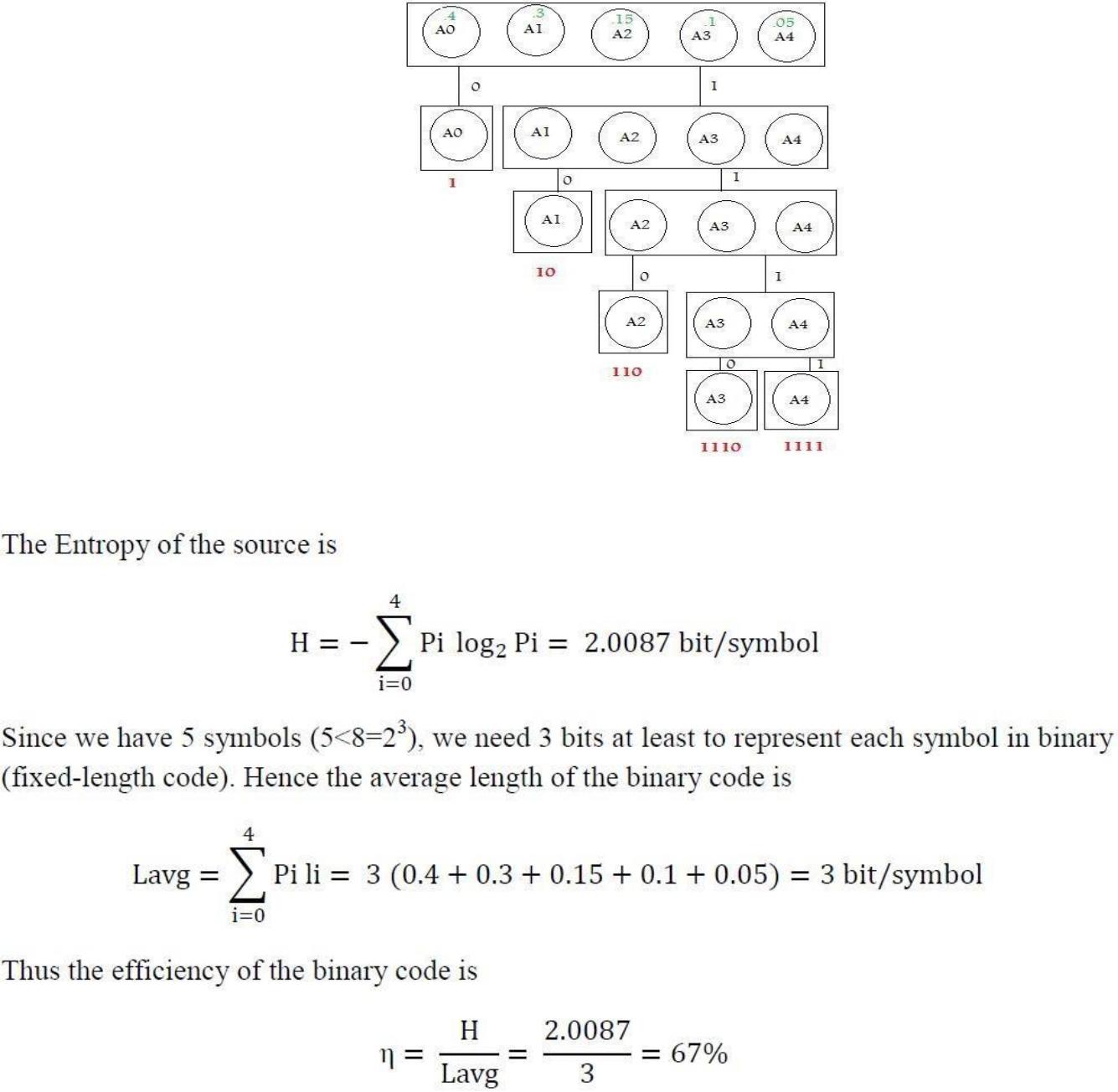
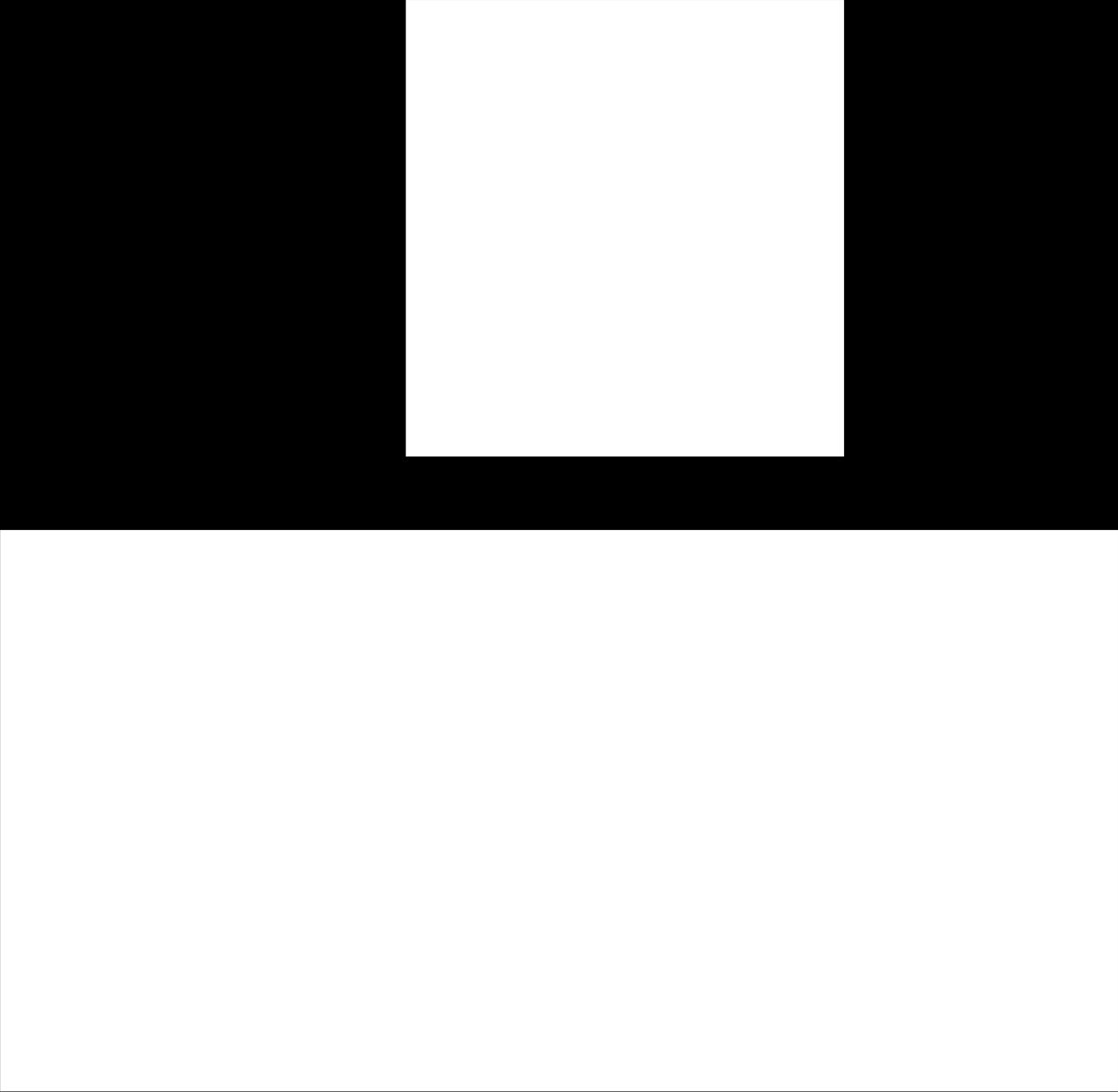
Example:

The source of information A generates the symbols {A0, A1, A2, A3 and A4} with the corresponding probabilities {0.4, 0.3, 0.15, 0.1 and 0.05}. Encoding the source symbols using binary encoder and Shannon-Fano encoder gives

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Source Symbol | Pi |  | Binary Code | Shannon-Fano |
|  |  |  |  |  |
| A0 | 0.4 | 000 |  | 0 |
|  |  |  |  |  |
| A1 | 0.3 | 001 |  | 10 |
|  |  |  |  |  |
| A2 | 0.15 | 010 |  | 110 |
|  |  |  |  |  |
| A3 | 0.1 | 011 |  | 1110 |
|  |  |  |  |  |
| A4 | 0.05 | 100 |  | 1111 |
|  |  |  |  |  |
| Lavg | H = 2.0087 | 3 |  | 2.05 |
|  |  |  |  |  |



Shanon-Fano code is a top-down approach. Constructing the code tree, we get



Binary Huffman Coding (an optimum variable-length source coding scheme)

In Binary Huffman Coding each source letter is converted into a binary code word. It is a prefix condition code ensuring minimum average length per source letter in bits.

Let the source letters a1, a 2, ……….aK have probabilities P(a1), P(a2),………….

P(aK) and let us assume that P(a1) ≥ P(a2) ≥ P(a 3)≥…. ≥ P(aK).

We now consider a simple example to illustrate the steps for Huffman coding.

Steps to calculate Huffman Coding

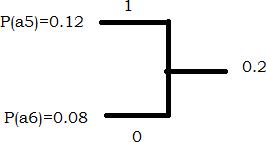
Example Let us consider a discrete memory less source with six letters having

P(a1)=0.3,P(a2)=0.2, P(a 3)=0.15, P(a 4)=0.15, P(a5)=0.12 and P(a6)=0.08.

Arrange the letters in descending order of their probability (here they are arranged).



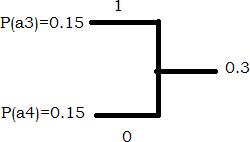
Consider the last two probabilities. Tie up the last two probabilities. Assign, say, 0 to the last digit of representation for the least probable letter (a6) and 1 to the last digit of representation for the second least probable letter (a5). That is, assign ‘1’ to the upper arm of the tree and ‘0’ to the lower arm.



(3) Now, add the two probabilities and imagine a new letter, say b1, substituting for a6 and a5. So P(b1) =0.2. Check whether a4 and b1are the least likely letters. If not, reorder the letters as per Step#1 and add the probabilities of two least likely letters. For our example, it leads to:

P(a1)=0.3, P(a2)=0.2, P(b1)=0.2, P(a3)=0.15 and P(a4)=0.15

(4) Now go to Step#2 and start with the reduced ensemble consisting of a1 , a2 , a3 ,



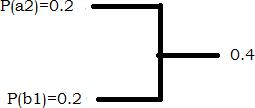
a4 and b1. Our example results in:

Here we imagine another letter b1, with P(b2)=0.3.

Continue till the first digits of the most reduced ensemble of two letters are assigned a ‘1’ and a ‘0’.

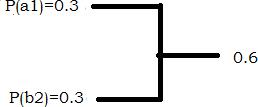
Again go back to the step (2): P(a1)=0.3, P(b2)=0.3, P(a2)=0.2 and P(b1)=0.2.

Now we consider the last two probabilities:

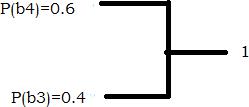


So, P(b3)=0.4. Following Step#2 again, we get, P(b3)=0.4, P(a1)=0.3 and P(b2)=0.3.

Next two probabilities lead to:



With P(b4) = 0.6. Finally we get only two probabilities



1. Now, read the code tree inward, starting from the root, and construct the code words. The first digit of a codeword appears first while reading the code tree inward.

Hence, the final representation is: a1=11, a2=01, a3=101, a4=100, a5=001, a6=000.

A few observations on the preceding example

1. The event with maximum probability has least number of bits
2. Prefix condition is satisfied. No representation of one letter is prefix for other. Prefix condition says that representation of any letter should not be a part of any other letter.
3. Average length/letter (in bits) after coding is
   * ∑*P* (*ai* )*ni* = 2.5 bits/letter.
4. Note that the entropy of the source is: H(X)=2.465 bits/symbol. Average length per source letter after Huffman coding is a little bit more but close to the source entropy. In fact, the following celebrated theorem due to C. E. Shannon sets the limiting value of average length of code words from a DMS.

Shannon–Hartley theorem

In [information theory,](https://en.wikipedia.org/wiki/Information_theory) the Shannon–Hartley theorem tells the maximum rate at which information can be transmitted over a communications channel of a specified [bandwidth](https://en.wikipedia.org/wiki/Bandwidth_%28signal_processing%29) in the presence of [noise.](https://en.wikipedia.org/wiki/Noise_%28electronics%29) It is an application of the [noisy-channel coding theorem](https://en.wikipedia.org/wiki/Noisy-channel_coding_theorem) to the archetypal case of a [continuous-time](https://en.wikipedia.org/wiki/Continuous-time) [analog](https://en.wikipedia.org/wiki/Analog_signal) [communications channel](https://en.wikipedia.org/wiki/Communications_channel) subject to [Gaussian](https://en.wikipedia.org/wiki/Gaussian_noise) [noise.](https://en.wikipedia.org/wiki/Gaussian_noise) The theorem establishes Shannon's [channel capacity](https://en.wikipedia.org/wiki/Channel_capacity) for such a communication link, a

bound on the maximum amount of error-free [information](https://en.wikipedia.org/wiki/Information) per time unit that can be transmitted with a specified [bandwidth](https://en.wikipedia.org/wiki/Bandwidth_%28signal_processing%29) in the presence of the noise interference, assuming that the signal power is bounded, and that the Gaussian noise process is characterized by a known power or power spectral density.

The law is named after [Claude Shannon](https://en.wikipedia.org/wiki/Claude_Elwood_Shannon) and [Ralph Hartley.](https://en.wikipedia.org/wiki/Ralph_Hartley)

Hartley Shannon Law

The theory behind designing and analyzing channel codes is called Shannon’s noisy channel coding theorem. It puts an upper limit on the amount of information you can send in a noisy channel using a perfect channel code. This is given by the following equation:



where C is the upper bound on the capacity of the channel (bit/s), B is the bandwidth of the channel (Hz) and SNR is the Signal-to-Noise ratio (unit less).

Bandwidth-S/N Tradeoff

The expression of the channel capacity of the Gaussian channel makes intuitive sense:

1. As the bandwidth of the channel increases, it is possible to make faster changes in the information signal, thereby increasing the information rate.

2 As S/N increases, one can increase the information rate while still preventing errors due to noise.

3. For no noise, S/N tends to infinity and an infinite information rate is

possible irrespective of bandwidth.

Thus we may trade off bandwidth for SNR. For example, if S/N = 7 and B = 4kHz, then the channel capacity is C = 12 ×103 bits/s. If the SNR increases to S/N = 15 and B is decreased to 3kHz, the channel capacity remains the same. However, as B tends to 1, the channel capacity does not become infinite since, with an increase in bandwidth, the noise power also increases. If the noise power spectral density is ɳ/2, then the total noise power is N = ɳB, so the Shannon-Hartley law becomes

