# Huffman codes

Last modified: 2026-11-20 15:19:13 (PST)

Source coding shows that the entropy of a source is the least average length of a code for it, up to one bit. This page asks which codes a decoder can read one codeword at a time, and how long such a code must be.

## Prefix codes

**Definition 0.1** (Prefix code)**.** A binary code for a discrete random variable $X$ is a **prefix code**, also called a **prefix-free code** or an **instantaneous code**, if no codeword is the beginning of another codeword. Writing $c(x)$ for the codeword of the value $x$, the condition is that, for every two different values $x$ and $x'$, the string $c(x')$ does not start with the string $c(x)$.

**Theorem 0.1** (Kraft's inequality)**.** *Let $X$ be a discrete random variable that takes a finite number of values, and let a prefix code for $X$ have codeword lengths $\ell(x)$. Then*

$$s_x 2^{-\ell(x)} \leq 1. \tag{1}$$

Proof

*Proof.* Let $m$ be the largest codeword length. It is finite because $X$ takes a finite number of values.
For each value $x$, the strings of $m$ bits that start with the codeword $c(x)$ are those that choose the remaining $m - \ell(x)$ bits freely. So there are $2^{m-\ell(x)}$ of them.
No string of $m$ bits starts with two different codewords. If it started with $c(x)$ and with $c(x')$, where $\ell(x) \leq \ell(x')$, then $c(x)$ would be the first $\ell(x)$ bits of $c(x')$, which the prefix code forbids. So the strings that start with different codewords form groups with no string in two groups.
There are $2^m$ strings of $m$ bits in all, so the groups together hold at most that many:

$$s_x 2^{m-\ell(x)} \le 2^m. \tag{2}$$

Dividing both sides by $2^m$ gives the claim. □

**Theorem 0.2** (Prefix codes exist for lengths that satisfy Kraft's inequality)**.** *Let* $\ell_1 \le \ell_2 \le \cdots \le \ell_n$ *be positive integers with*

$$\sum_{i=1}^{n} 2^{-\ell_i} \le 1. \tag{3}$$

*Then some prefix code on* $n$ *values has codeword lengths* $\ell_1, \ldots, \ell_n$.

Proof

*Proof.* For each $i$, let

$$a_i = s_{j<i} 2^{-\ell_j}, \tag{4}$$

so that $a_1 = 0$. Let the codeword $c_i$ be the $\ell_i$ bits that spell $N_i = 2^{\ell_i} a_i$ as a base-2 number, with zeros added at the front to reach $\ell_i$ digits.

This is possible if $N_i$ is an integer with $0 \le N_i < 2^{\ell_i}$. Each term of $N_i = s_{j<i} 2^{\ell_i - \ell_j}$ is an integer, because $\ell_j \le \ell_i$. By the assumption, $a_i + 2^{-\ell_i} \le s_{j \le i} 2^{-\ell_j} \le 1$, so $N_i \le 2^{\ell_i} - 1 < 2^{\ell_i}$.

Suppose that $c_i$ were the beginning of $c_k$ for some $i < k$. The first $\ell_i$ digits of $c_k$ are the base-2 digits of the integer part of $N_k / 2^{\ell_k - \ell_i} = 2^{\ell_i} a_k$. Because $c_i$ is these digits, this integer part is $N_i$.

But $a_k = a_i + s_{i \le j < k} 2^{-\ell_j} \ge a_i + 2^{-\ell_i}$, so $2^{\ell_i} a_k \ge N_i + 1$, and the integer part of $2^{\ell_i} a_k$ is at least $N_i + 1$, not $N_i$. This contradiction shows that no $c_i$ with $i < k$ begins $c_k$. Nor can $c_k$ be the beginning of $c_i$ with $i < k$: that would give $\ell_k \le \ell_i$, so $\ell_k = \ell_i$ and $c_k = c_i$, which the first case excludes. □

**Example 0.1** (A prefix code and a code that is not)**.**
Take four values with the codeword lengths $(1, 2, 3, 3)$. The code $0, 10, 110, 111$ is a prefix code. The code $0, 01, 011, 111$ has the same lengths, but the codeword 01 starts with the codeword 0, so it is not a prefix code. The code below tests the definition on every pair of codewords, and adds up $2^{-\ell(x)}$ for each code, as in Theorem 0.1.

```
is_prefix_code <- function(codewords) {
  index <- seq_along(codewords)
  pairs <- expand.grid(shorter = index, longer = index)
  distinct_pairs <- pairs[pairs$shorter != pairs$longer, ]
  shorter_words <- codewords[distinct_pairs$shorter]
  longer_words <- codewords[distinct_pairs$longer]
  begins <- startsWith(longer_words, shorter_words)
  !any(begins)
}
kraft_sum <- function(codewords) {
  lengths <- nchar(codewords)
  sum(2^(-lengths))
}
prefix_code <- c("0", "10", "110", "111")
other_code <- c("0", "01", "011", "111")
codes <- list(prefix_code, other_code)
data.frame(
  code = c("0, 10, 110, 111", "0, 01, 011, 111"),
  is_prefix_code = vapply(codes, is_prefix_code, logical(1)),
  kraft_sum = vapply(codes, kraft_sum, numeric(1))
)
#>              code is_prefix_code kraft_sum
#> 1 0, 10, 110, 111           TRUE         1
#> 2 0, 01, 011, 111          FALSE         1
```

Both codes have $s_x 2^{-\ell(x)} = 1$, the largest value Theorem 0.1 allows. The sum depends on the lengths alone, so it is the same for the second code, which is not a prefix code. Theorem 0.2 gives a prefix code with these lengths, and the first code is one.

*Remark* 0.1 (Where Kraft's inequality comes from). Cover and Thomas (2006, sec. 5.2) prove Theorem 0.1 and Theorem 0.2. They show in section 5.5 that the lengths of every uniquely decodable code have the same sum at most 1, so restricting to prefix codes costs nothing in expected length. MacKay (2003, sec. 5.2) states the result for prefix codes as well.

## The Huffman algorithm

**Definition 0.2** (Huffman code)**.** Let $X$ be a discrete random variable that takes $n \geq 2$ values, with probability mass function $p$. A **Huffman code** for $X$ is a binary code built as follows:

1. Start with $n$ nodes, one for each value $x$, each carrying the number $p(x)$.
2. Repeat $n-1$ times: take two nodes that carry the smallest numbers, make them the two children of a new node that carries the sum of theirs, and label the edge to one child 0 and the edge to the other child 1.
3. Give each value $x$ the codeword made of the labels on the path from the last node down to the node for $x$.

Ties between nodes that carry equal numbers may be broken in any way, so $X$ can have several Huffman codes.

**Theorem 0.3** (Huffman codes are prefix codes)**.** *Every Huffman code is a prefix code.*

Proof

*Proof.* The construction in Definition 0.2 builds a tree whose nodes for the values $x$ are leaves: each of them is made first, and only the new nodes get children. The codeword $c(x)$ is the list of edge labels on the path from the last node to the leaf for $x$.
Suppose that $c(x)$ were the beginning of $c(x')$ for two different values $x$ and $x'$. Both paths start at the last node, and the same labels lead to the same child at every step, so the path to $x$ would be the first part of the path to $x'$. Then the leaf for $x$ would lie on the way to the leaf for $x'$, and the leaf for $x$ would have a child. No leaf has a child, so no codeword is the beginning of another. □

**Example 0.2** (A Huffman code for a five-valued variable)**.**
Let $X$ have five values with probability mass function $p = (0.4, 0.2, 0.2, 0.1, 0.1)$. The code below carries out the construction of Definition 0.2, breaking ties in favor of the node listed first. It then computes the expected length $L$ of the code, the entropy $\mathbb{H}(X)$ with base-2 logarithms, and the sum $s_x 2^{-\ell(x)}$ of Theorem 0.1.

```r
huffman_code <- function(p) {
  codewords <- rep("", length(p))
  weights <- p
  members <- as.list(seq_along(p))
  while (length(weights) > 1) {
    ord <- order(weights)
    first <- ord[1]
    second <- ord[2]
    low <- members[[first]]
    high <- members[[second]]
    codewords[low] <- paste0("0", codewords[low])
    codewords[high] <- paste0("1", codewords[high])
    merged_weight <- weights[first] + weights[second]
    merged_members <- c(low, high)
    two_smallest <- 1:2
    rest <- ord[-two_smallest]
    weights <- c(weights[rest], merged_weight)
    members <- c(members[rest], list(merged_members))
  }
  codewords
}
p <- c(0.4, 0.2, 0.2, 0.1, 0.1)
codewords <- huffman_code(p)
len <- nchar(codewords)
surprisal <- -log2(p)
expected_length <- sum(p * len)
entropy <- sum(p * surprisal)
kraft_sum <- sum(2^(-len))
data.frame(p = p, codeword = codewords, length = len)
#>     p codeword length
#> 1 0.4       11      2
#> 2 0.2       00      2
#> 3 0.2       01      2
#> 4 0.1      100      3
#> 5 0.1      101      3
c(expected_length = expected_length, entropy = entropy, kraft_sum = kraft_sum)
#> expected_length         entropy       kraft_sum
#>        2.200000        2.121928        1.000000
```

The code has expected length $L = 2.2$ bits and the source has entropy $\mathbb{H}(X) = 2.122$ bits, so $L - \mathbb{H}(X) = 0.078$ bits. The sum $s_x 2^{-\ell(x)} = 1$ is the largest value Theorem 0.1 allows, and no codeword is the beginning of another, as Theorem 0.3 says.

## Optimality

**Theorem 0.4** (Huffman codes are optimal)**.** *Let $X$ be a discrete random variable that takes $n \geq 2$ values, each with probability $\mathbb{p}(x) > 0$. Then a Huffman code for $X$ has the least expected length $L$ among all prefix codes for $X$.*

Proof

*Proof.* We use induction on $n$. For $n = 2$, every prefix code gives each value a codeword of at least one bit, and the Huffman code gives them the codewords 0 and 1, so no prefix code is shorter.
Let $n > 2$, and suppose the claim holds for $n - 1$ values. Let $x$ and $y$ be two values with the smallest probabilities, the pair that the first step of the construction merges.
First, some prefix code with the least expected length has $x$ and $y$ as siblings, meaning that their codewords have the same length $m$ and differ only in the last bit. To see this, take a prefix code with the least expected length.

- If $\mathbb{p}(u) > \mathbb{p}(v)$, then $\ell(u) \leq \ell(v)$. Otherwise, exchanging the codewords of $u$ and $v$ changes $L$ by $(\mathbb{p}(u) - \mathbb{p}(v))(\ell(v) - \ell(u)) < 0$.
- A longest codeword $c$ has a sibling. Otherwise, deleting the last bit of $c$ leaves a string that neither begins another codeword nor is begun by one, which gives a prefix code with a smaller $L$.

Let $u$ be a value with a longest codeword, and let $v$ be the value of its sibling. Both codewords have length $m$, the largest codeword length. By the first bullet, exchanging the codeword of $x$ with that of $u$ does not increase $L$, because $\mathbb{p}(x) \leq \mathbb{p}(u)$ and $\ell(u) = m$ is the largest length. The same holds for exchanging $y$ with $v$. If $x$ or $y$ is already one of $u$ and $v$, leave it where it is, and exchange the other with the remaining one of the pair. If $x = v$ and $y \neq u$, for example, exchange $y$ with $u$. Ties among the smallest probabilities do not matter, because every value other than $x$ and $y$ has probability at least $\mathbb{p}(y)$, and the exchanges use only that. These exchanges give a prefix code with the least expected length in which $x$ and $y$ are siblings.
Next, let $X'$ be the random variable that merges $x$ and $y$ into one value $z$ with $\mathbb{p}'(z) = \mathbb{p}(x) + \mathbb{p}(y)$ and keeps every other value and probability. Any prefix code for $X$ in which $x$ and $y$ are siblings turns into a prefix code for $X'$ by deleting the last bit of their common codeword to make the codeword of $z$. The two codes differ only in the lengths of $x$, $y$ and $z$, so their expected lengths satisfy

$$L = L' + \mathbb{p}(x) + \mathbb{p}(y). \tag{5}$$

The Huffman code for $X$ is built from a Huffman code for $X'$ by replacing the codeword of $z$ with two codewords, one that ends in 0 and one that ends in 1. By the induction

hypothesis, a Huffman code for $X'$ has the least expected length $L'$ among prefix codes for $X'$. So no prefix code for $X$ in which $x$ and $y$ are siblings has an expected length below that of the Huffman code for $X$. By the first part, a prefix code with the least expected length is among these, so the Huffman code has the least expected length among all prefix codes. □

**Example 0.3** (Three prefix codes for one source)**.**
Take the source of Example 0.2, with probabilities $(0.4, 0.2, 0.2, 0.1, 0.1)$. Its Huffman code has the codeword lengths $(2, 2, 2, 3, 3)$. A fixed-length code gives every value three bits. Giving each value $x$ the length $\lceil -\log_2 \mathbb{p}(x) \rceil$ is a third choice of lengths. All three choices satisfy the sum in Theorem 0.1, so Theorem 0.2 gives a prefix code for each. The code below computes their expected lengths.

```
p <- c(0.4, 0.2, 0.2, 0.1, 0.1)
huffman_lengths <- c(2, 2, 2, 3, 3)
fixed_lengths <- rep(3, length(p))
shannon_lengths <- ceiling(-log2(p))
length_sets <- list(huffman_lengths, fixed_lengths, shannon_lengths)
data.frame(
  code = c("Huffman", "fixed length", "Shannon lengths"),
  kraft_sum = vapply(length_sets, function(l) sum(2^(-l)), numeric(1)),
  expected_length = vapply(length_sets, function(l) sum(p * l), numeric(1))
)
#>              code kraft_sum expected_length
#> 1         Huffman     1.000             2.2
#> 2    fixed length     0.625             3.0
#> 3 Shannon lengths     0.625             2.8
```

The Huffman code has the least expected length of the three, as Theorem 0.4 says it must have among all prefix codes.

**Theorem 0.5** (A Huffman code is within one bit of the entropy)**.** *Let $X$ be a discrete random variable that takes $n \geq 2$ values, each with probability $\mathbb{p}(x) > 0$. Then a Huffman code for $X$ has expected length $L$ with*

$$\mathbb{H}(X) \leq L < \mathbb{H}(X) + 1, \tag{6}$$

*where $\mathbb{H}(X)$ is the entropy of $X$ with base-2 logarithms.*

Proof

*Proof.* Every prefix code is uniquely decodable, so the source coding theorem gives $L \geq \mathbb{H}(X)$.
For the other half of the claim, give each value $x$ the length $\ell(x) = \lceil -\log_2 \mathbb{p}(x) \rceil$. Then $2^{-\ell(x)} \leq \mathbb{p}(x)$, so adding $2^{-\ell(x)}$ for every value gives at most the sum of their probabilities, which is 1. By Theorem 0.2, some prefix code has these lengths. Its expected length is below $\mathbb{H}(X) + 1$, because $\ell(x) < -\log_2 \mathbb{p}(x) + 1$ for each value.
By Theorem 0.4, a Huffman code has an expected length no larger than this prefix code, so $L < \mathbb{H}(X) + 1$. □

**Example 0.4** (The bound for the Huffman code of the earlier example)**.**
The Huffman code of Example 0.2 has the codeword lengths $(2, 2, 2, 3, 3)$ for the probabilities $(0.4, 0.2, 0.2, 0.1, 0.1)$. The code below computes its entropy and expected length.

```
p <- c(0.4, 0.2, 0.2, 0.1, 0.1)
huffman_lengths <- c(2, 2, 2, 3, 3)
entropy <- -sum(p * log2(p))
expected_length <- sum(p * huffman_lengths)
data.frame(entropy, expected_length, upper_bound = entropy + 1)
#>    entropy expected_length upper_bound
#> 1 2.121928             2.2    3.121928
```

The expected length is 2.2 bits. It lies between the entropy, 2.122 bits, and the entropy plus one bit, 3.122 bits, as Theorem 0.5 says.

## References


Cover, Thomas M., and Joy A. Thomas. 2006. *Elements of Information Theory.* 2nd ed. Wiley-Interscience. https://doi.org/10.1002/047174882X.

MacKay, David J. C. 2003. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press.