Huffman coding and its generalizations

Dmitriy Kovalev
Novosibirsk State University
Russia
Email: dmitriy.kovalev@gmail.com

Sergey Krendelev
Novosibirsk State University
Russia
Email: tak@mail.nsk.ru

Abstract

In this paper we try to discuss the Huffman coding from our point of view. As a result of this approach we came up with several interesting generalizations. We had also used the results of this work to create and implement lossless video coding algorithm [1].

I. Introduction

Huffman method [2] is an essential part of any university course related to information theory or data compression. Lections commonly begin with the formal definition of entropy, prefix codes, binary trees and finally with optimal Huffman compression. Majority of students only understand the formal side of all constructions without real understanding why the entropy is a right measure for information, why prefix codes are used, and why this process gives optimal representation.

II. Main part

Consider alphabet A that consists of m elements $a_1, a_2, \ldots, a_m$. Let A^n be a set of all strings of length n formed with elements of A. It is obvious that A contains m^n elements. Suppose that S is a subset of A^n and contains p elements. Since S is finite, all its elements can be numbered $0, 1, \ldots p-1$, and then each such number can be represented in binary numeral system. This binary representation requires $H = \log_2 p$ bits of information. Value H is called entropy, specifically the entropy of A^n is $H = \log_2 m^n = n \log_2 m$.

Typically the subset S can be distinguished from A^n by some conditions. A condition is a mapping from A^n to the set with just two elements: $true$ and $false$. Each string $s \in A^n$ that meets the condition $P(s)$ forms a subset of A^n (subset can be empty). Hence, if P is a condition, then set A^n is divided by it into two mutually exclusive subsets. The first subset contains all strings $s \in A^n$ when $P(s)$ is $true$ and the second contains all another strings when $P(s)$ is $false$.

Here are examples of some conditions:

P_1: element a_2 occurs in the string exactly 10 times.
P_2: a_7 is always followed by element a_3.
P_3: a_1 and a_7 do not occur in the string simultaneously.
P_4: a_{12} occurs no more than 30 times.

Let's suppose that some string meets conditions $P_1, P_2, \ldots, P_r$. Suppose that the addressee knows these conditions. How can we transmit this string? Theoretically, this can be done as follows. First, search through all m^n strings in some fixed order and check which strings satisfy the conditions $P_1, P_2, \ldots, P_r$. When we find a string that meets all the conditions, a sequential number is assigned to that string. As a result we can assign a number to each string from S that satisfies all of the above conditions and, thus, find the number of elements in S. That way to transmit the string we only need to send the sequential number of the string to our addressee, since our $P_1, P_2, \ldots, P_r$ conditions are known and, therefore, the original string can be reconstructed by its number. In real situations it's commonly required to send all conditions $P_1, P_2, \ldots, P_r$ to the addressee along with the message, thus increasing its size.

Unfortunately this method is not applicable to real life situations since it uses enumeration of all combinations. Thus, our task is to understand what conditions can be used to describe all combinations relatively quickly.

Consider the following example. Suppose that some subset $S \in A^n$ is defined by conditions:

P_1: a_1 occurs n_1 times.
P_2: a_2 occurs n_2 times.
...
P_m: a_m occurs n_m times.

Then, $n_1 + n_2 + \ldots + n_m = n$. In this case set S is a multiset and it contains

$$\frac{n!}{n_1!n_2!\ldots n_m!}$$

elements. This expression is a multinomial coefficient and it commonly written as

$$\binom{n}{n_1, n_2, \ldots, n_m}.$$

It means that the entropy of set S is

$$H = \log_2 \frac{n!}{n_1!n_2!\ldots n_m!}$$

and each element from this set can be represented as an integer number N:

$$0 \le N \le \binom{n}{n_1, n_2, \ldots, n_m} - 1.$$

Multinomial coefficients has a useful property:

$$\begin{aligned}\binom{n_1+n_2+\ldots+n_m}{n_1, n_2, \ldots, n_m} &= \binom{(n_1+n_2+\ldots+n_k)+(n_{k+1}+\ldots+n_m)}{n_1+n_2+\ldots+n_k, n_{k+1}+\ldots+n_m} \times \\ &\times \binom{n_1+n_2+\ldots+n_k}{n_1, n_2, \ldots, n_k} \times \binom{n_{k+1}+\ldots+n_m}{n_{k+1}, \ldots, n_m}\end{aligned}$$

Each multinomial coefficient can be interpreted as a number of strings with a given length and with a given number of every element. In particular, the previous formula can be interpreted as follows. The alphabet set is divided into two groups of elements, first group contains all elements of alphabet with numbers $1, 2, \ldots, k$, and second group includes all alphabet elements with numbers $k+1, \ldots, m$. This partition can be represented as binary string with length m. After that we can add two strings: the first contains k different elements and its length is $n_1 + n_2 + \ldots + n_k$, the second contains $m - k$ elements and its length is $n_{k+1} + \ldots + n_m$. So, each element from the alphabet can be represented as a pair (i, j) where i defines a group number and j defines the number of an element in group. This is nothing other than prefix coding.

This observation allows us to make a generalization. Suppose that alphabet A is divided into mutually exclusive subsets $A_0, A_1, \ldots, A_r$, then the union of all these sets is A. Let's take any A_i and divide it again. We can do this process recursively until we get one-element sets. After that each element of original alphabet can be represented as tuple $a = (\lambda_1, \lambda_2, \ldots)$ where λ_1 is the number of the first set from the fist partition that contains element a, λ_2 is the number of the second partition that contains element a, and so on. Evidentially, if the partition is fixed, then each element of the alphabet is uniquely defined. This is what is called prefix coding. If at every partition we would divide each set into two subsets, we would come up with bit representation of each element of the alphabet.

It is useful to consider partition process as a tree where each root represents a subset of alphabet. In that case, if at each step we divide a set in half, we will get a binary tree. Partition into d parts will result in d-ary tree. The generalization of both cases is when we divide each time into different number of parts.

Notice that approach of different set partition is very similar to an approach of representing an integer number as a summation of integer numbers.

Thus, each set partition allows us to represent a multinomial coefficient as a set of multipliers where each multiplier is also a multinomial coefficient:

$$\binom{n_1 + n_2 + \ldots + n_m}{n_1, n_2, \ldots, n_m} = B_0 B_1 \ldots B_r.$$

It is possible to create many set partitions, which is why the set of multipliers is not unique. However, based on logarithm qualities, regardless of partition, the number of bits necessary to represent the string will be the same.

Using our approach, number N can be represented in positional numeral system with the mixed radix $B_0, B_1, \ldots, B_r$:

$$N = \lambda_0 + \lambda_1 B_0 + \lambda_2 B_1 B_0 + \ldots + \lambda_{r+1} B_0 B_1 \ldots B_r, \quad 0 \le \lambda_k < B_k.$$

Each λ_k is mapped to some string $s(\lambda_k)$. If we will use string representation, then each element from the set S can be represented as sequence of strings $s(\lambda_0), s(\lambda_1), \ldots, s(\lambda_{r+1})$.In this case, $s(\lambda_0)$ is a string of length n that consists of first elements of a prefix code. All other strings have similar meaning.

The described encoding method of all elements of S has its drawbacks. It is necessary to remember the number of occurrences of each element of the alphabet (message statistics). Another problem is that computation of λ_k is quite complex because of manipulations with binomial coefficients. These are the reasons why enumeration encoding is rarely used.

The amount of computations can be reduced, but with a loss of coding efficiency. Let each element of alphabet has be numbered $0, 1, \ldots, m-1$. Then each string of length n can be represented as digits of a number from a numeral system with base m:

$$K = a_0 + a_1 m + a_2 m^2 + \ldots + a_{n-1} m^{n-1}.$$

It is obvious that $K < m^n$ and $\log_2 m^n = n \log_2 m$ bits should be used to save any such number. The minimal deviation between K and appropriate multinomial coefficient occurs when $n_1 = n_2 = \ldots = n_m$.

The next construction is analogous to construction with binomial coefficients. First, we need to create a set partition. Then each number $B_0, B_1, \ldots, B_r$ should be replaced with the appropriate power using the following rule:

$$B_k = \binom{u_1 + n_2 + \ldots + u_s}{u_1, u_2, \ldots, u_s} \leftrightarrow D_k = s^h,$$

where the length of the string is $h = u_1 + u_2 + \ldots + u_s$ and s is a number of different elements. All other reasonings remain true. In particular, each such string of length n can be mapped to the number:

$$N = \lambda_0 + \lambda_1 D_0 + \lambda_2 D_0 D_1 + \ldots + \lambda_{r+1} D_0 D_1 \ldots D_r, \quad 0 \le \lambda_k < D_k.$$

III. Conclusion

From this it follows that for efficient representation of data it is necessary to use arbitrary trees where each node can contain different number of children. Besides, there are statements similar to Kraft-MacMillan inequality. Methods defined above can be also used for Markov chain coding.

References

[1] http://videosoft.org/codecs/fastcodec/

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