ABSTRACT

Huffman Coding is an approach to text compression originally developed by David A. Huffman while he was a Ph.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes". In computer science and information theory, it is one of many lossless data compression algorithms. It is a statistical compression method that converts characters into variable length bit strings and produces a prefix code. Most frequently occurring characters are converted to shortest bit strings; least frequent, the longest.

CONTENTS

1.Introduction to Huffman Coding:

Let us suppose, we need to store a string of length 1000 that comprises characters a, e, n, and z. To store it as 1-byte characters will require 1000 byte (or 8000 bits) of space. If the symbols in the string are encoded as (00=a, 01=e, 10=n, 11=z), then the 1000 symbols can be stored in 2000 bits saving 6000 bits of memory.

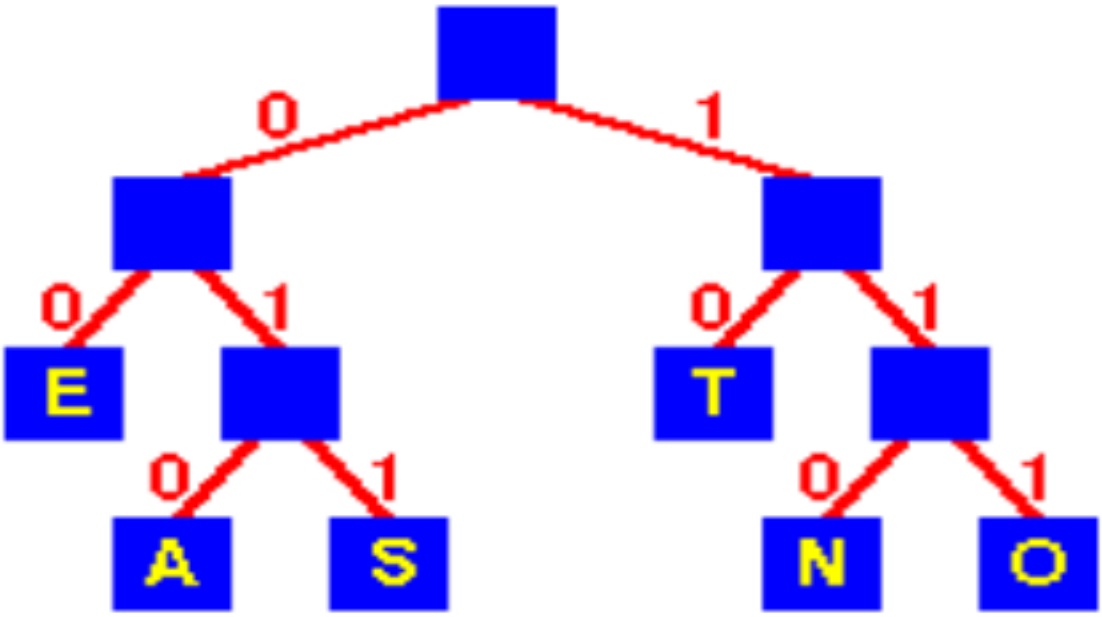
The number of occurrences of a symbol in a string is called its frequency. When there is considerable difference in the frequencies of different symbols in a string, variable length codes can be assigned to the symbols based on their relative frequencies. The most common characters can be represented using shorter codes than are used for less common source symbols. More is the variation in the relative frequencies of symbols, it is more advantageous to use variable length codes for reducing the size of coded string.

Since the codes are of variable length, it is necessary that no code is a prefix of another so that the codes can be properly decoded. Such codes are called prefix code (sometimes called "prefix-free codes", that is, the code representing some particular symbol is never a prefix of the code representing any other symbol). Huffman coding is so much widely used for creating prefix codes that the term "Huffman code" is sometimes used as a synonym for "prefix code" even when such a code is not produced by Huffman's algorithm.

Huffman was able to design the most efficient compression method of this type: no other mapping of individual source symbols to unique strings of bits(i.e. codes) will require less space for storing a piece of text when the actual symbol frequencies agree with those used to create the code.

2.Basic Technique:

In Huffman Coding , the complete set of codes can be represented as a binary tree, known as a **Huffman tree**. This Huffman tree is also a **coding tree** i.e. a full binary tree in which each leaf is an encoded symbol and the path from the root to a leaf is its codeword. By convention, bit '0' represents following the left child and bit '1' represents following the right child. One code bit represents each level. Thus more frequent characters are near the root and are coded with few bits, and rare characters are far from the root and are coded with many bits.



Huffman Tree

First of all, the source symbols along with their frequencies of occurrence are stored as leaf nodes in a regular array, the size of which depends on the number of symbols, n. A finished tree has up to n leaf nodes and n − 1 internal nodes.

PROBLEM DEFINITION:-

Given

A set of symbols and their weights (usually proportional to probabilities or equal to their frequencies).

Find

A prefix-free binary code (a set of codewords) with
minimum expected codeword length (equivalently, a tree with minimum weighted path length from the root).

The simplest construction algorithm uses a priority queue where the node with lowest probability is given highest priority:

Step 1:- Create a leaf node for each symbol and add it to the priority queue (i.e.Create a min heap of Binary trees and heapify it).

Step 2:- While there is more than one node in the queue (i.e. min heap):

i. Remove the two nodes of highest priority (lowest probability or lowest frequency) from the queue.
ii. Create a new internal node with these two nodes as children and with probability equal to the sum of the two nodes' probabilities (frequencies).
iii. Add the new node to the queue.

Step 3:- The remaining node is the root node and the Huffman tree is complete.

Joining trees by frequency is the same as merging sequences by length in optimal merge. Since a node with only one child is not optimal, any Huffman coding corresponds to a full binary tree.

Definition of optimal merge: Let D={n1, ... , nk} be the set of lengths of sequences to be merged. Take the two shortest sequences, ni, nj∈ D, such that n≥ ni and n≥ nj ∀ n∈ D. Merge these two sequences. The new set D is D' = (D - {ni, nj}) ∪ {ni+nj}. Repeat until there is only one sequence.
Since efficient priority queue data structures require O(log n) time per insertion, and a tree with n leaves has 2n−1 nodes, this algorithm operates in O(n log n) time.

The worst case for Huffman coding (or, equivalently, the longest Huffman coding for a set of characters) is when the distribution of frequencies follows the Fibonacci numbers.

If the estimated probabilities of occurrence of all the symbols are the same and the number of symbols is a power of two, Huffman coding is the same as simple binary block encoding, e.g., ASCII coding.

Although Huffman's original algorithm is optimal for a symbol-by symbol coding (i.e. a stream of unrelated symbols) with a known input probability distribution, it is not optimal when the symbol-by-symbol restriction is dropped, or when the probability mass functions are unknown, not identically distributed, or not independent (e.g., "cat" is more common than "cta").

3.Variations of Huffman Coding:

a) n-ary Huffman coding

The **n-ary Huffman** algorithm uses the {0, 1, ... , n − 1} alphabet to encode messages and build an n-ary tree.

b) Adaptive Huffman coding

It calculates the probabilities dynamically based on recent actual frequencies in the source string. This is somewhat related to the LZ family of algorithms.

c) Huffman template algorithm

The **Huffman template algorithm** enables one to use any kind of weights (costs, frequencies, pairs of weights, non-numerical weights) and one of many combining methods (not just addition).

d) Optimal alphabetic binary trees (Hu-Tucker coding) In the alphabetic version, the alphabetic order of inputs and outputs must be identical.This is also known as the **Hu**

Tucker problem, after the authors of the paper presenting the first linearithmic solution to this optimal binary alphabetic problem, which has some similarities to Huffman algorithm, but is not a variation of this algorithm. These optimal alphabetic binary trees are often used as binary search trees.

e) The canonical Huffman code

If weights corresponding to the alphabetically ordered inputs are in numerical order, the Huffman code has the same lengths as the optimal alphabetic code, which can be found from calculating these lengths, rendering Hu-Tucker coding unnecessary. The code resulting from numerically (re-)ordered input is sometimes called the canonical Huffman code and is often the code used in practice, due to ease of encoding/decoding. The technique for finding this code is sometimes called **Huffman-Shannon-Fano coding**, since it is optimal like Huffman coding, but alphabetic in weight probability, like Shannon-Fano coding.

4.Applications:

Arithmetic coding can be viewed as a generalization of Huffman coding; indeed, in practice arithmetic coding is often preceded by Huffman coding, as it is easier to find an arithmetic code for a binary input than for a nonbinary input. Also, although arithmetic coding offers better compression performance than Huffman coding, Huffman coding is still in wide use because of its simplicity, high speed and lack of encumbrance by patents.

Huffman coding today is often used as a "back-end" to some other compression method. DEFLATE (PKZIP's algorithm) and multimedia codecs such as JPEG and MP3 have a front-end model and quantization followed by Huffman coding.