Huffman Coding Algorithm: Less Is More

Blair Kiel

Troy University

Abstract

The Huffman coding algorithm is the result of David Huffman's efforts to find an efficient, simple way to encode input characters as binary code representations. As a graduate student he was able to succeed in producing a solution to a challenge one of his professor's proposed to his class. Even though the Huffman coding algorithm was invented over 50 years ago, it still is used in modern applications. While many attempts have been made to find a more efficient way to encode data for a lossless output, none have been able to find a more simple way.

Huffman Coding: Less Is More

A cornerstone of Computer Science theory is finding efficient, creative solutions for runtimes and space. Upon the creation of the computer, this was a priority since memory and processing-power were relatively expensive compared to today's standards. Due to the high price-per byte for storage and processing it was obvious that engineers would have to become more creative. Several techniques were employed in order to overcome those problems, such as divide-and-conquer, dynamic-programming and data-compression algorithms. The point of all data compression techniques is to find a way to represent code in fewer values. The encoding process compacts and shortens the code. Later when it is needed, it is decoded into the original length and format. One example of a data-compression algorithm is the Huffman coding algorithm. The Huffman coding algorithm is a data-compression technique that, for a given set of characters, creates a representation of the characters as binary code. The Huffman coding algorithm sets itself apart by assigning the most frequently used characters, to the smallest binary code representation possible. The Huffman coding algorithm has been an important staple in the advancement of technology, but it is important to study its history to truly appreciate its value.

The Huffman coding algorithm was created by David Huffman as an Electrical Engineering graduate student at Massachusetts Institute of Technology in 1951. One of his professors proposed a problem to his class. The students were to find the most efficient way

to represent characters as binary code. Huffman reasoned that he needed to find a way to assign the most frequent numbers with the smallest binary values. The result of his efforts to find a solution to his professor's problem was what is now known as the Huffman coding algorithm.

Gary Stix, claims Huffman's "… insight was that by assigning the probabilities of the longest codes first and then proceeding along the branches of the tree toward the root, he could arrive at an optimal solution every time" (September 1991). This holds true for all code trees; the closer a leaf is to the root, the shorter of a "walk" it will take to get to that leaf. Although Huffman received credit by the name of his invented algorithm, he has humbly redirected praise, "It was my luck to be there at the right time and also not have my professor discourage me by telling me that other good people had struggled with this problem," (Stix, September 1991). David Huffman was a humble, hard-worker as his previous quote portrays; the fruition of his hard work has become one of the most famous, influential algorithms in the history of computer technology.

The Huffman coding algorithm initially processes a series of characters by counting the frequency of each character (Cormen, Leiserson, & Rivest, p. 431). Then the characters are arranged uniquely, as nodes from binary tree which contain a value and frequency, in increasing order by the frequency that they appear in a priority queue (Cormen, Leiserson, & Rivest, p. 431). The ordering of the character nodes determines the length of the each character's binary representation. The algorithm takes the next available two nodes, and

creates a new node as their parent; the new parent node's value is the sum of its children's frequencies (Cormen, Leiserson, & Rivest, p. 431). The new parent node is added to its appropriate position in the rest of the ordered list; this continues until the final two nodes are paired and a root of the tree is created (Cormen, Leiserson, & Rivest, p. 431). Due to each operation of "n" characters being "inserted" into the binary tree taking O (log n) each, the worst case run-time of the Huffman coding algorithm is O (n log n) (Cormen, Leiserson, & Rivest, p. 433).

The value of leaves, nodes that do not have children, in the resulting tree are the unique characters from the beginning of the algorithm. However, in order to retrieve a binary code representation of each character, the path from the root to the character's leaf must be traced. The binary representation is built by appending a 0 if the path goes left, or a 1 if the path goes right(Cormen, Leiserson, & Rivest, p. 431). Klein and Wiseman provide an example of the encoding's in action, "… Consider, for example, the simple Huffman code {00, 010, 011, 10, 11} for the characters A, B, C, D, E respectively. The encoding of the string BACEAD would then be the binary string 01000011110010…" (2003, p. 488). Further, by taking the binary string "111001101000" and following the indicated path until it reaches a leaf with a character value, the binary string can be decoded into { E, D, C, B, A }. By delaying the most frequently occurring characters, their leaves exists at higher levels of the tree. This is the genius idea that David Huffman realizing by using a binary tree.

One of the practical applications of the Huffman coding algorithm is used in VCR programming. The Scientific American Journal explains, "Instead of confronting the frustrating process of programming a VCR, the user simply types into the [VCR remote] a numerical code that is printed in the television listing" (September 1991). The remote takes the code input from the buttons, beams the code to the VCR where it is decoded and operates the VCR and TV. While this example uses VCR programming, it also is used in similar remote applications.

Another application of the Huffman coding algorithm is used in JPEG image formats. After an image has been split into 8 pixels by 8 pixels blocks, a discrete cosine transformation is applied to each block. The 64 pixels are then processed by quantization, then finally the Huffman coding algorithm is applied to the resulting coefficients as an entropy encoder (Klein, p. 492). It should be noted that, "The coefficient in position (0,0) (left upper corner) is called the DC coefficient and the 63 remaining values are called the AC coefficients" (Klein p. 492). Klein explains in more detail,

> Baseline JPEG uses two different Huffman trees to encode the data. The first encodes the lengths in bits (1 to 11) of the binary representations of the values in the DC fields. The second tree encodes information about the sequence of AC coefficients …
>
> The second tree also includes code words for EOB, which is used when no non-zero elements are left in the scanning order and for a sequence of 16 consecutive zeros in

> the AC sequence (ZRL), necessary to encode zero-runs that are longer than 15 (p. 492).

The default method of encoding JPEG images is Huffman coding, however another method of backend encoding for JPEG images is arithmetic coding.

For encoding JPEG images with the arithmetic coding technique, Neal and Clearly claim, “Arithmetic coding is superior in most respects to the better-known Huffman method. It represents information at least as compactly-sometimes considerably more so” (p. 520). They also admit the requirements of using Huffman coding, “[it] indeed achieves 'minimum redundancy'. In other words, it performs optimally if all symbol probabilities are integral powers of %. But this is not normally the case in practice; indeed, Huffman Coding can take up to one extra bit per symbol” (Neal and Clearly, p. 520). Even with these additional requirements, for most applications it seems as though the arithmetic coding technique is only marginally better, by that one less bit per symbol.

Other than arithmetic coding compression, more popular current data compression techniques include Lempel-Zip (LZ), Burrows-Wheeler Transformation (BWT), and Meridian Lossless Packaging. Lempel-Zip compression is used in other image formats such as PNG and GIF formats. When there is a need to store or deliver data to another recipient, it is sometimes beneficial to compress data into an archive structure. One of the preferred

archive formats bzip2, uses the Burrows-Wheeler Transformation. In other media formats such as DVD or Blu-ray, Meridian Lossless Packing is used.

These other data compression techniques have not altogether eliminated the Huffman coding algorithm. As previously stated, the Huffman coding algorithm is primarily used as the last step for the creation of a JPEG image. The belief that, "[standard arithmetic coding] is more complex than Huffman coding for certain implementations" (Wallace, p. 6), gives a convincing case for the place that the Huffman coding has for the future. The simplicity of Huffman's idea to use a binary tree to encode characters will preserve the Huffman coding algorithm for the future.

In conclusion, the Huffman coding algorithm was a groundbreaking data compression technique when David Huffman invented it in 1951, but has spurred on data compression innovations since its inception. Its simplicity allows for Computer Science and other engineering students to easily be introduced to data compression; it's also simple enough to be implemented in freelance applications, and efficient enough for professional applications. Technology continues to advance toward higher resolution displays, larger feature-packed applications, and an ever-increasing demand to have mobile data on hand at all times. Data compression will be needed in the future as much as it was needed in the past. The JPEG, as a staple of the Huffman coding algorithm, seems like it will remain in use for the foreseeable future.

References

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). *Introduction to algorithms*. Cambridge, MA: MIT Press.

Klein, S. T. (2003). Parallel Huffman Decoding with Applications to JPEG Files. *The Computer Journal, 46*(5), 487-497. doi:10.1093/comjnl/46.5.487

Marcellin, M., Gormish, M., Bilgin, A., & Boliek, M. (n.d.). An overview of JPEG-2000. *Proceedings DCC 2000. Data Compression Conference*. doi:10.1109/dcc.2000.838192

Stix, G. (1991, September). Encoding the "Neatness" of Ones and Zeroes. *Sci Am Scientific American, 265*(3), 54-58. doi:10.1038/scientificamerican0991-54

Wallace, G. (1992). The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics IEEE Trans. Consumer Electron., 38*(1), Xviii-Xxxiv. doi:10.1109/30.125072

Witten, I. H., Neal, R. M., & Cleary, J. G. (1987). Arithmetic coding for data compression. *Communications of the ACM Commun. ACM, 30*(6), 520-540. doi:10.1145/214762.214771