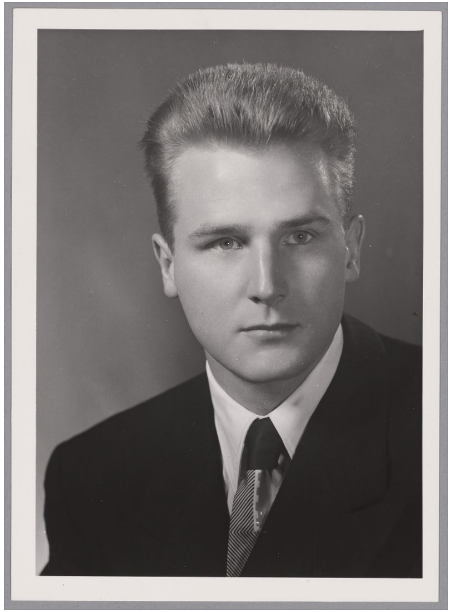
# Huffman Coding

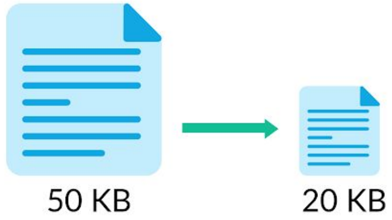
By Aaron Partridge

# History

In 1951, David A. Huffman was a Sc.D. student at MIT and he and his classmates were given the choice by professor, Robert M. Fano, to either take a final exam or write a term paper on the problem of finding the most efficient binary code. Huffman chose the term paper, but was unable to prove any code was the most efficient method. He almost gave up when he had the idea of using a frequency-sorted binary tree. This turned out to be the most efficient and ended up beating the algorithm his professor wrote called the, Shannon-Fano coding algorithm. Huffman's method focused on a bottom up approach, which proved to be much more efficient than the top down approach of his professor's algorithm.



# Why Huffman Coding is important




If it wasn’t for compression algorithms, we wouldn’t have companies like Youtube, Netflix, Amazon, or even Google. These companies rely on compression because it makes sending information fast and efficient and also makes storing information at scale even possible. Imagine if Youtube had to store it’s estimated 5.1 billion videos in the original file size. Or imagine if Netflix had to send uncompressed, full-length movies to it’s estimated 300 million users. That would be an incredibly slow and costly process. Compression is vital because it reduces the amount of space needed to store the information and makes transmitting the data much more efficient. This is why Huffman coding is involved in compression algorithms such as .zip, .gzip, and multimedia compression such as .jpeg, .png, and .mp3. Huffman coding is also a form of lossless data compression, which means that no data is lost between compression and decompression. Some forms of multimedia compression are lossy because the human eye and ear can always perceive small amounts of data loss in photo, video, and audio quality.

# Intuition

Step 1: read in the data

Step 2: count up the number of each character

Step 3: give each character a frequency (# of occurrences) and order them from greatest occurring to least.

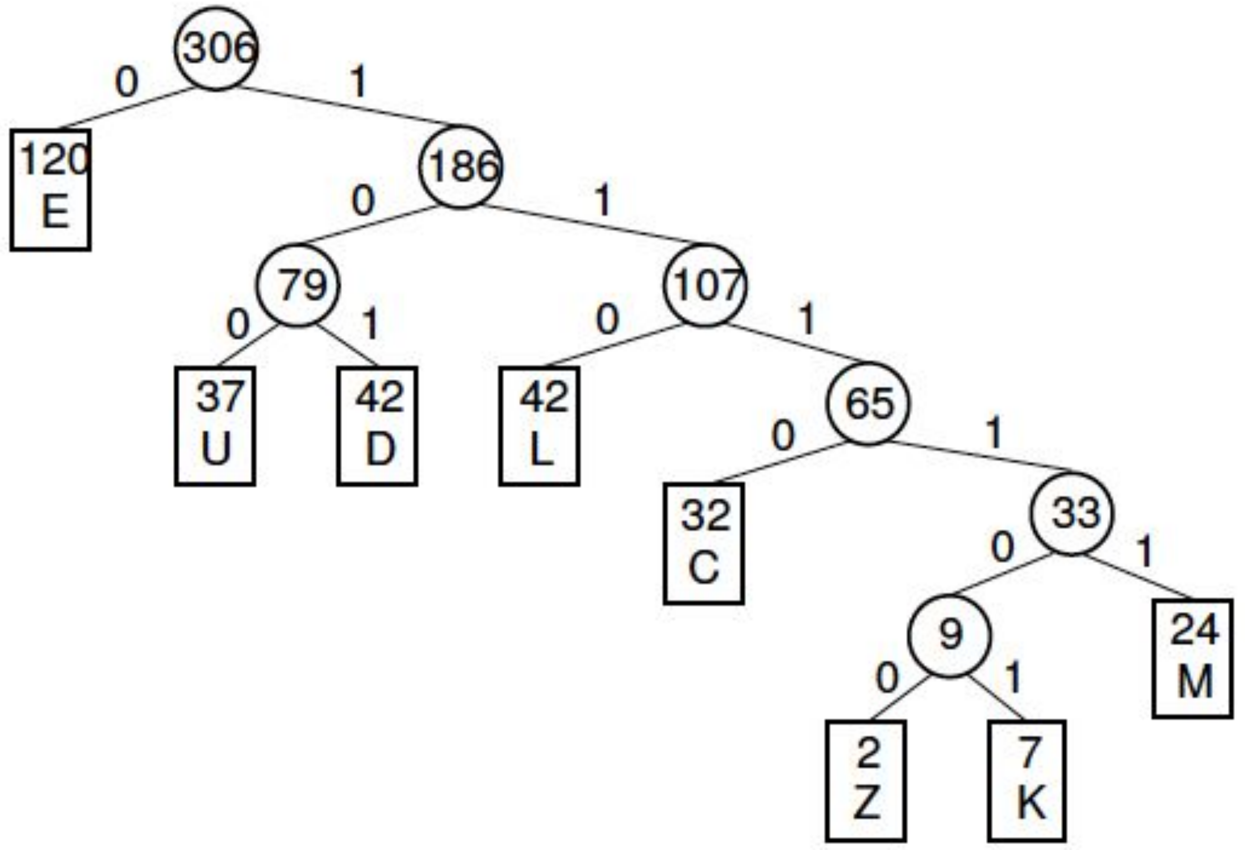
Step 4: Create a tree where the parent node is the sum of the two child node frequencies.

Step 5: assign edge weights of 0 and 1. 0 if going to the left (higher frequency) and 1 if going to the right (lower frequency).

Step 6: Traverse the tree for each character and record the 0's and 1's required to reach it.

Step 7: assign that new binary value to the character.

Step 8: assemble all the new binary values. This is the compressed data.




Source: http://homes.sice.indiana.edu/yye/lab/teaching/spring2014-C343/huffman.php

# Pseudocode

main()

getFrequency()

makeHuffmanTree()

getHuffmanCodes()

encode()

```
getFrequency(originalMessage, nodes)
    For every c in originalMessage
        found = false
        for each node in nodes
            if node.ch == c
                node.frq++
                found = true
        if !found
            nodes.append(new Node(c, 1))
```

```
makeHuffmanTree(nodes)
	PQ[] //min-heap priority queue
	for each node in nodes
		PQ.push(node)
	while PQ.size > 1
		temp1 = PQ.top
		PQ.pop
		temp2 = PQ.top
		PQ.pop
		tempNode = new Node('\0', temp1.frq + temp2.frq)
		tempNode.left = temp1
		tempNode.right = temp2
		PQ.push(tempNode)
	root = PQ.top
	return root
```

```
getHuffmanCodes(node, currentCode)
    if !node
        return
    if node.ch != ‘\0’
        node.code = currentCode

    getHuffmanCodes(node.left, currentCode + “0”)
    getHuffmanCodes(node.right, currentCode + “1”)
```

```
encode(originalMessage, nodes)
    compressedMessage = “”
    for each c in originalMessage
        for each node in nodes
            if node.ch == c
                compressedMessage += node.code
                break
    return compressedMessage
```

```
decode(compressedMessage, root)
    result = “”
    currNode = root
    for each c in compressedMessage
        if c == ‘0’
            currNode = currNode.left
        else if c == ‘1’
            currNode = currNode.right
        if currNode.ch != ‘\0’
            result += currNode.ch
            currNode = root
    return result
```

# Runtime Analysis

getFrequency() - O(n * v)

makeHuffmanTree() - O(v log(v))

getHuffmanCodes() - O(v)

encode() - O(n * v)

For n = total number of characters in original message

And v = number of nodes (unique ASCII characters)

Therefore, the overall runtime is O(v log(v)) because this will dominate the the rest of the algorithm.

```
0) The letter 'a', repeated 100,000 times.
1) Enough repetitions of the alphabet to fill 100,000 characters
2) 100,000 characters, randomly selected from [a-z|A-Z|0-9|!| ] (alphabet size 64)
3) The King James version of the bible
4) Complete genome of the E. Coli bacterium
5) The first million digits of pi
6) The CIA world fact book

Choose a test case 0-6: 0

-----------------------------------------------------------


Original File Size: 800008 bits
Compressed Size: 100001 bits
Percent Difference: 87.5% improvement

Decompressing Message
Decompression Successful: Messages Are Identical

Total Time: 14377 microseconds

-----------------------------------------------------------
```

```
0) The letter 'a', repeated 100,000 times.
1) Enough repetitions of the alphabet to fill 100,000 characters
2) 100,000 characters, randomly selected from [a-z|A-Z|0-9|!| ] (alphabet size 64)
3) The King James version of the bible
4) Complete genome of the E. Coli bacterium
5) The first million digits of pi
6) The CIA world fact book

Choose a test case 0-6: 2

-----------------------------------------------------------


Original File Size: 800008 bits
Compressed Size: 601479 bits
Percent Difference: 24.8% improvement

Decompressing Message
Decompression Successful: Messages Are Identical

Total Time: 77518 microseconds

-----------------------------------------------------------
```

```
0) The letter 'a', repeated 100,000 times.
1) Enough repetitions of the alphabet to fill 100,000 characters
2) 100,000 characters, randomly selected from [a-z|A-Z|0-9|!| ] (alphabet size 64)
3) The King James version of the bible
4) Complete genome of the E. Coli bacterium
5) The first million digits of pi
6) The CIA world fact book

Choose a test case 0-6: 3

-----------------------------------------------------------


Original File Size: 32379136 bits
Compressed Size: 17747595 bits
Percent Difference: 45.2% improvement

Decompressing Message
Decompression Successful: Messages Are Identical

Total Time: 1592824 microseconds

-----------------------------------------------------------
```

```
0) The letter 'a', repeated 100,000 times.
1) Enough repetitions of the alphabet to fill 100,000 characters
2) 100,000 characters, randomly selected from [a-z|A-Z|0-9|!| ] (alphabet size 64)
3) The King James version of the bible
4) Complete genome of the E. Coli bacterium
5) The first million digits of pi
6) The CIA world fact book

Choose a test case 0-6: 4


-----------------------------------------------------------


Original File Size: 37109528 bits
Compressed Size: 10418207 bits
Percent Difference: 71.9% improvement

Decompressing Message
Decompression Successful: Messages Are Identical

Total Time: 440030 microseconds


-----------------------------------------------------------
```

```
0) The letter 'a', repeated 100,000 times.
1) Enough repetitions of the alphabet to fill 100,000 characters
2) 100,000 characters, randomly selected from [a-z|A-Z|0-9|!| ] (alphabet size 64)
3) The King James version of the bible
4) Complete genome of the E. Coli bacterium
5) The first million digits of pi
6) The CIA world fact book

Choose a test case 0-6: 5


-----------------------------------------------------------


Original File Size: 8000008 bits
Compressed Size: 3498617 bits
Percent Difference: 56.3% improvement

Decompressing Message
Decompression Successful: Messages Are Identical

Total Time: 166120 microseconds


-----------------------------------------------------------
```

```
0) The letter 'a', repeated 100,000 times.
1) Enough repetitions of the alphabet to fill 100,000 characters
2) 100,000 characters, randomly selected from [a-z|A-Z|0-9|!| ] (alphabet size 64)
3) The King James version of the bible
4) Complete genome of the E. Coli bacterium
5) The first million digits of pi
6) The CIA world fact book

Choose a test case 0-6: 6


-----------------------------------------------------------


Original File Size: 19787200 bits
Compressed Size: 12468759 bits
Percent Difference: 37.0% improvement

Decompressing Message
Decompression Successful: Messages Are Identical

Total Time: 1377603 microseconds


-----------------------------------------------------------
```

# Thanks!