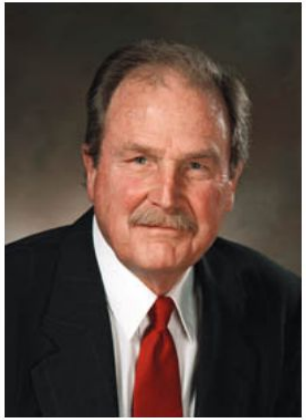
Huffman Encoding

Chad Germany

History

- Huffman was an electrical engineering student of Fano
- In 1951 as a student. Huffman found the most efficient binary code.
- As a result he did not have to take his final.

Robert Fano

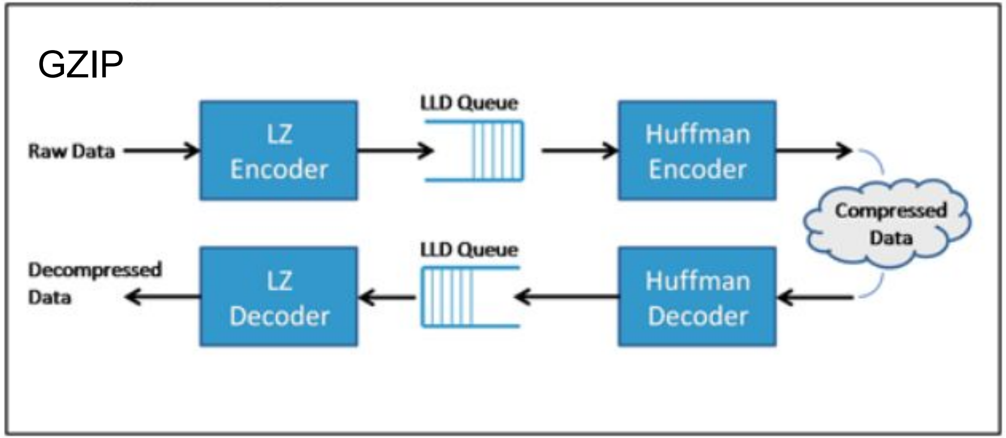


David A. Huffman

https://www.maa.org/press/periodicals/convergence/discovery-of-huffman-codes

Why is it important ?

- Huffman coding is used in conventional compression formats like GZIP, BZIP2, PKZIP, etc.
    - gzip is based on the DEFLATE algorithm, which is a combination of LZ77 and Huffman coding



https://en.wikipedia.org/wiki/Gzip

https://www.chipestimate.com/Unzipping-the-GZIP-compression-protocol/Altior/Technical-Article/2010/03/23

What is huffman encoding

- Huffman Encoding is a technique of compressing data to reduce its size without losing any of the details.
- Huffman Coding is generally useful to compress the data in which there are frequently occurring characters.
- The most frequent character gets the smallest code and the least frequent character gets the largest code
- he variable-length codes assigned to input characters are Prefix Codes
    - [5,9,59] is not a prefix code

Algorithm for Huffman Encoding

1. Create dictionary of characters with frequencies
2. create a priority queue Q consisting of each unique character.
3. sort them in ascending order of their frequencies.
4. for all the unique characters:
   a. create a newNode
   b. extract minimum value from Q and assign it to leftChild of newNode
   c. extract minimum value from Q and assign it to rightChild of newNode
   d. calculate the sum of these two minimum values and assign it to the value of newNode
   e. insert this newNode into the tree
5. return rootNode

https://www.programiz.com/dsa/huffman-coding

Building a Huffman Tree

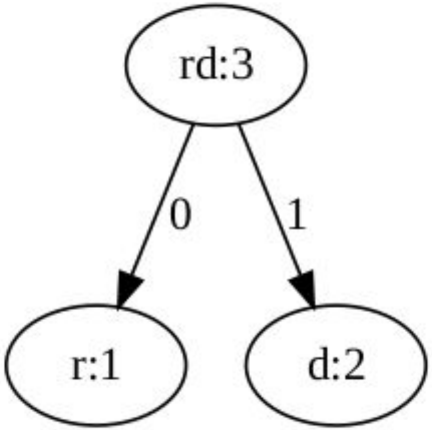
Want to encode: **“feed me more food”**

Step 1: Calculate frequency of every character in the text, and order by increasing frequency. Store in a queue which is a minimum heap.

r : 1 | d : 2 | f : 2 | m : 2 | o : 3 | 'SPACE' : 3 | e : 4

https://courses.engr.illinois.edu/cs225/sp2019/labs/huffman/

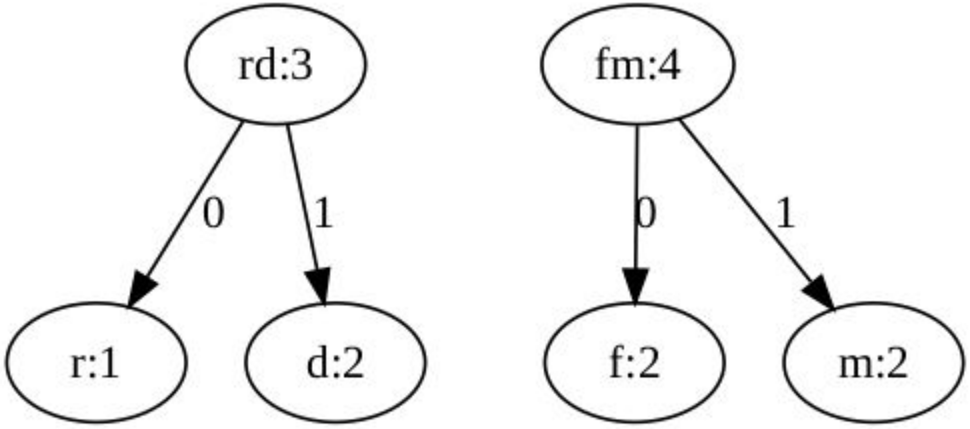
Step 2: Build the tree from the bottom up. Start by taking the two least frequent characters and merging them (create a parent node for them). Store the merged characters in a new queue:



SINGLE: f : 2 | m : 2 | o : 3 | 'SPACE' : 3 | e : 4

MERGED: rd : 3

Step 3: Repeat Step 2 this time also considering the elements in the new queue. ‘f’ and ‘m’ this time are the two elements with the least frequency, so we merge them

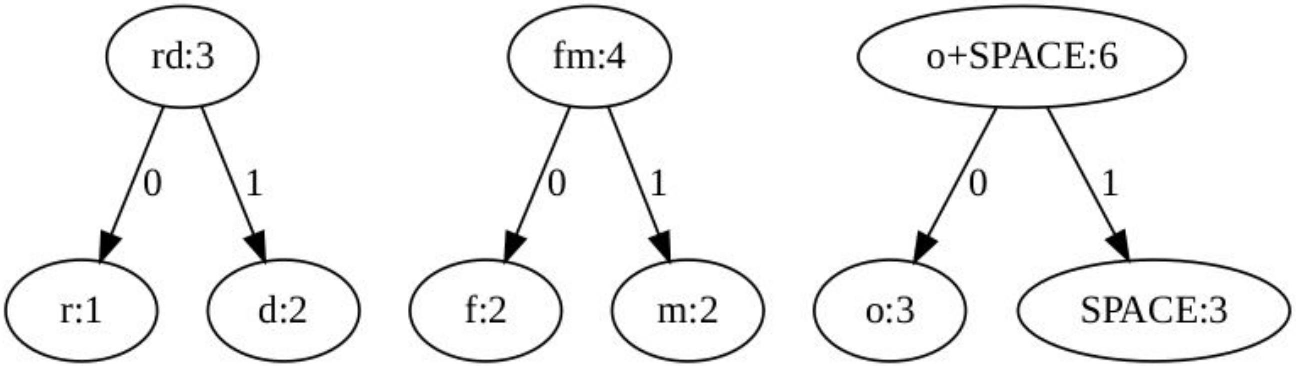


SINGLE: o : 3 | 'SPACE' : 3 | e : 4

MERGED: rd : 3 | fm : 4

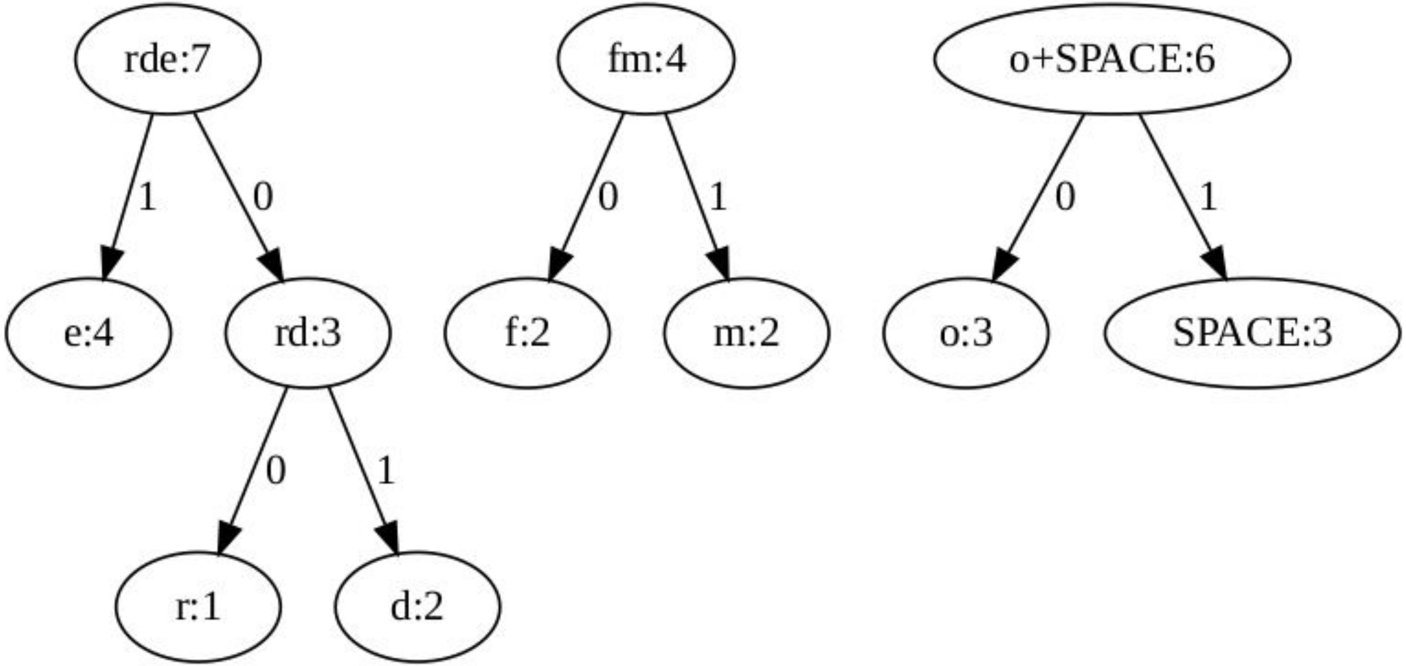
Step 4: Repeat Step 3 until there are no more elements in the SINGLE queue, and only one element in

the MERGED queue:



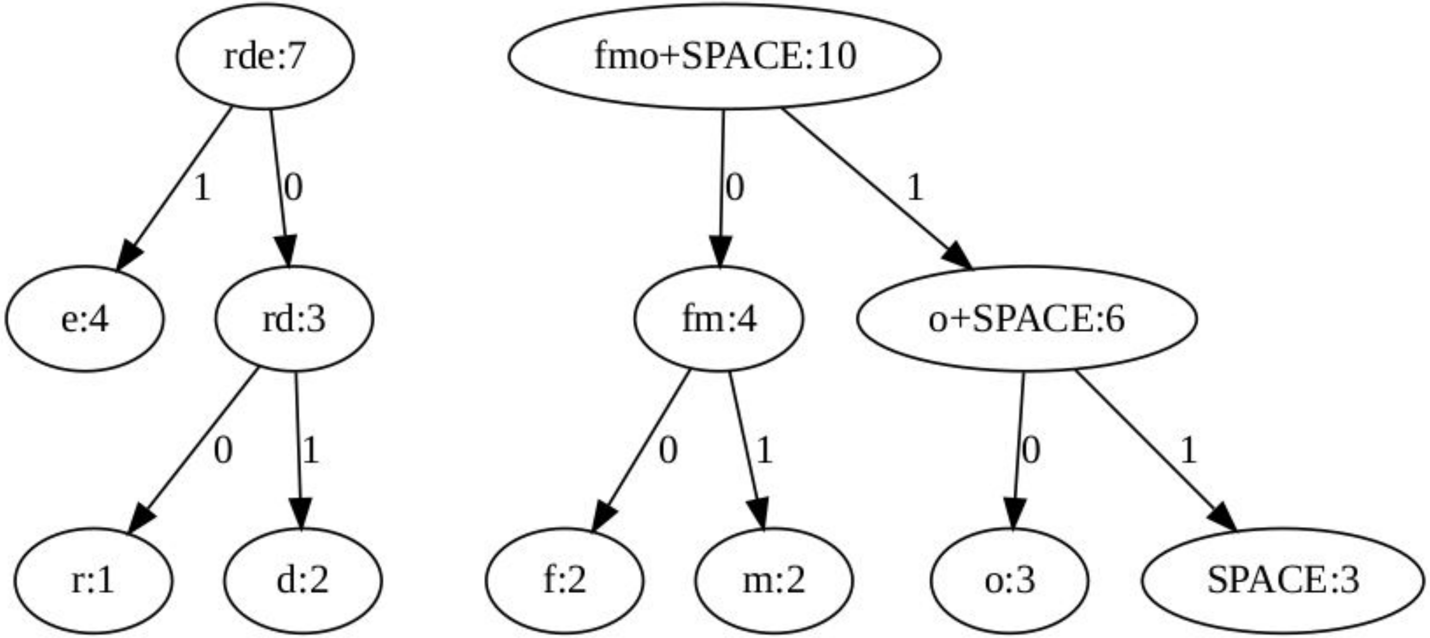
SINGLE: e : 4

MERGED: rd : 3 | fm : 4 | o+SPACE : 6



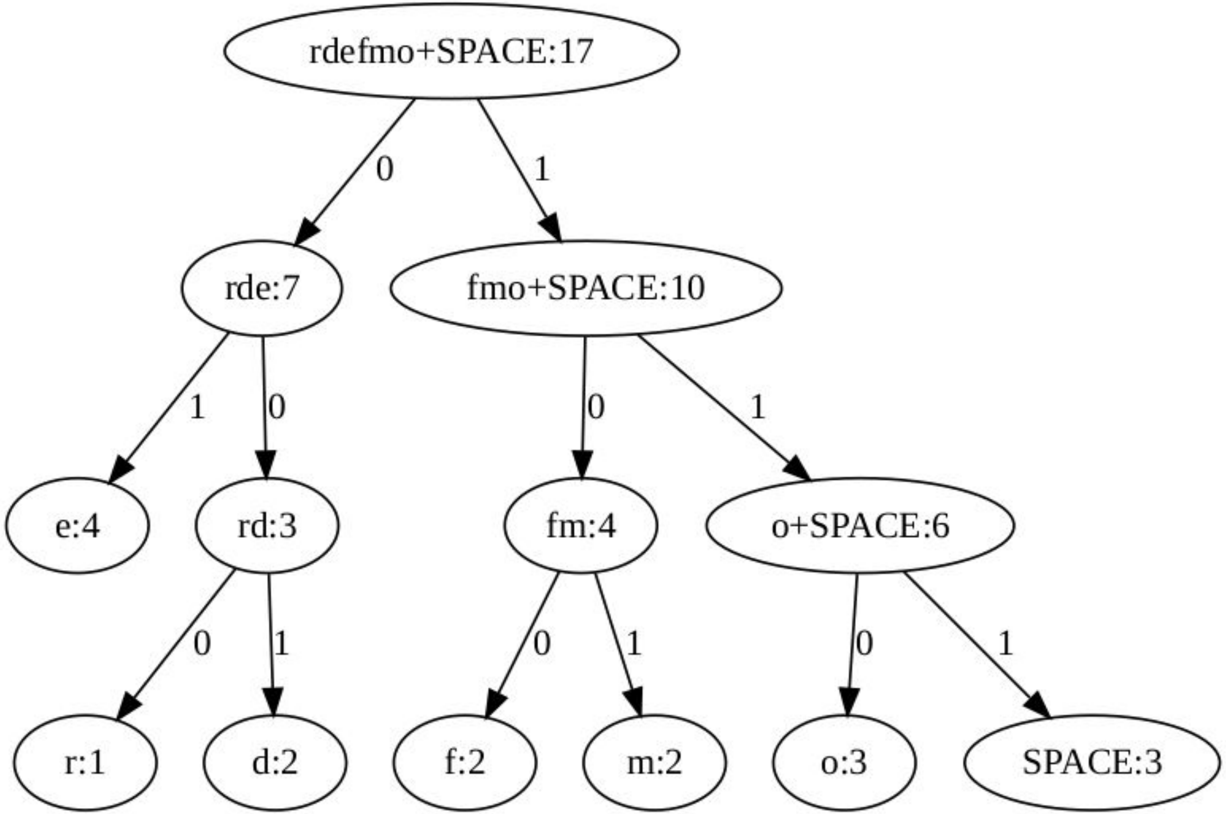
SINGLE:

MERGED: fm : 4 | o+SPACE : 6 | rde: 7



SINGLE:

MERGED: rde: 7 | fmo+SPACE: 10



SINGLE:

MERGED: rdefmo+SPACE: 17

In addition to saving the compress message we need to also save the tree itself.

Algorithm:

1) Start at the root

2) If the current node is a leaf:

    a) Write a “1” to the output file

    b) Write the character that the leaf node represents to the output file

3) Else (the current node is an internal nod):

    a) Write a “0” to the output file

    b) Recurse on the left subtree, then the right subtree

For the tree in the example the code is:

001 1e001d1o001r101f1m

Huffman Table:

Character	Frequency	Code	Size
e	4	01	4*2=8
	3	00	3*2=6
o	3	101	3*3=9
d	2	100	2*3=6
r	1	1100	1*4=4
\n	1	1101	1*4=4
f	2	1110	2*4 =8
m	2	1111	2*4=8
8*8=64			55

The total after compression is 119 bits

Before compression

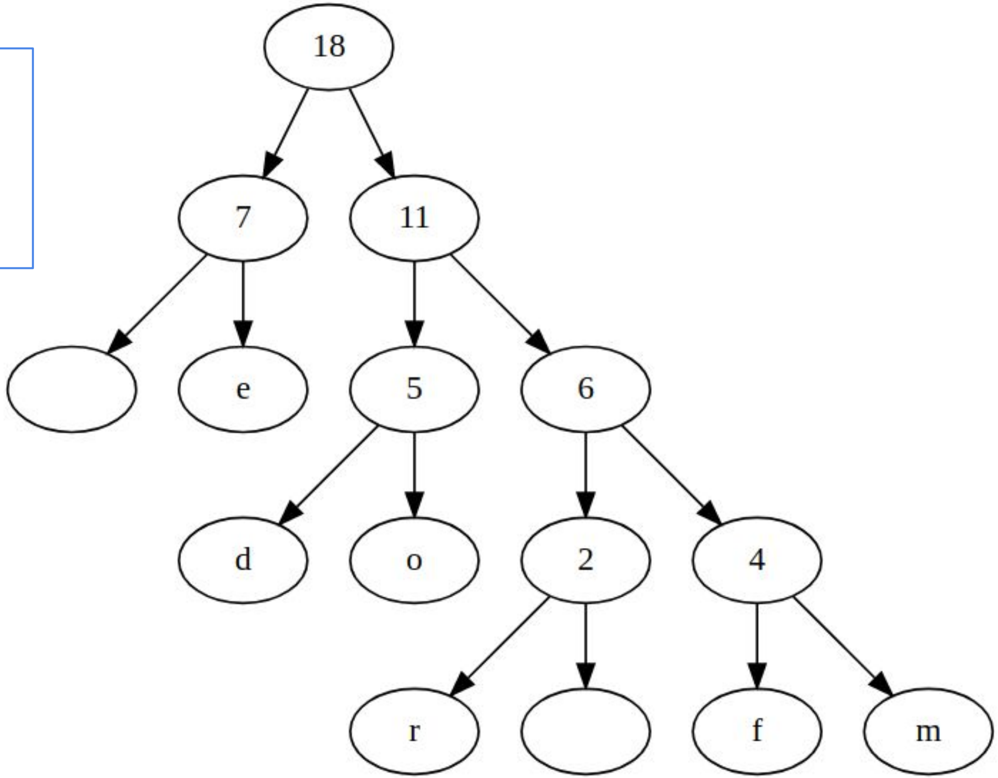
18*8 = 144 bits

So we saved 25 bits

Huffman Tree

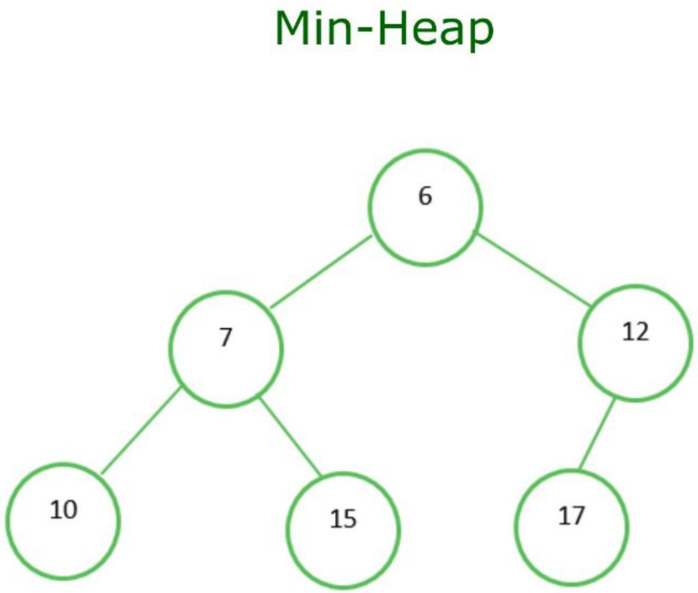
For the tree in the example the code is:

001 1e001d1o001r101f1m



Some words on Minimum Heap

- In a Min-Heap the key present at the root node must be less than or equal to among the keys present at all of its children.
- In a Min-Heap the minimum key element present at the root.
- A Min-Heap uses the ascending priority.
- In the construction of a Min-Heap, the smallest element has priority.
- In a Min-Heap, the smallest element is the first to be popped from the heap.
- For restructuring after popping O(Logn) time complexity



https://www.geeksforgeeks.org/difference-between-min-heap-and-max-heap/

Time Complexity

- To encode message length n, with c possible characters
  - Count frequencies: O(n)
  - Build tree: O(clogc) (with priority queue)
  - Encode: O(n)

Resources to read

- https://www.cs.utoronto.ca/~brudno/csc373w09/huffman.pdf
- https://www.chipestimate.com/Unzipping-the-GZIP-compression-protocol/Altior/Technical-Article/2010/03/23
- https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Storer%E2%80%93Szymanski
- https://www.huffmancoding.com/my-uncle/david-bio (Ken Huffman's website about his uncle)
- https://www.maa.org/press/periodicals/convergence/discovery-of-huffman-codes
- https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/
- https://courses.cs.washington.edu/courses/cse326/10wi/lectures/lec24/lec24-10wi-Huffman.pdf
- https://www.programiz.com/dsa/huffman-coding
- https://www.youtube.com/watch?v=fWk6Y8Rd6bs (youtube video)

Lz77 dictionary