

# huffman data compression :

huffman data compression is an efficient way of compressing text data so that the bits required to store or send a particular data is highly reduced. simply, huffman data compression values space, better say memory. In short, huffman data compression consists of the following steps.

1. Examine text to be compressed to determine the relative frequencies of individual letters.
2. Assign a binary code to each letter using shorter codes for the more frequent letters. This is the heart of the Huffman algorithm.
3. Encode normal text into its compressed form. We'll see this just as a string of '0's and '1's. This will turn out to be quite easy.
4. Recover the original text from the compressed. This will demonstrate a nice use of recursive traversal of a binary tree, but will still remain fairly simple.

now let's take an example and see how huffman data compression method deals with it.

assuming we have to send a string of data, say "mississippi river". If we are sending it in the general way each letter will take 8bits. Here there are 17 letters for this word. hence it will take a total of 136 bits ( $17 * 8$ ).

now coming to huffman's method, here initially we are finding the relative frequencies of each letters. Considering our example. We have to note that in "mississippi river", space between these two words is also considered as a letter. Hence here am using "\_" to represent the white space :

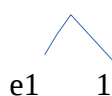
```
m ----> 1
i ----> 5
s ----> 4
p ----> 2
r ----> 2
v ----> 1
e ----> 1
_ ----> 1
```

now we have to assign codes. And as a first step, we have to sort these letters in the frequency of its occurrence. That is, "i" in this example comes first with occurrence 5. similarly s(4) and so on. So the respective sorted order will be :

```
i5    s4    p2    r2    m1    v1    e1    _1
```

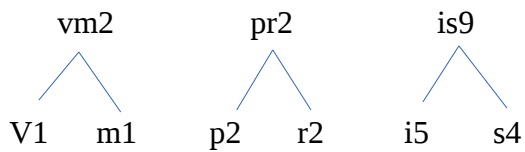
now our next step towards assigning codes is to add the respective frequencies. We start from the words with the shortest frequencies and we build the tree with the above data as leaf nodes. So here first we add

e2      here first we added e and space "\_" and their frequencies. And we got e\_2, where 2 is their added frequency.

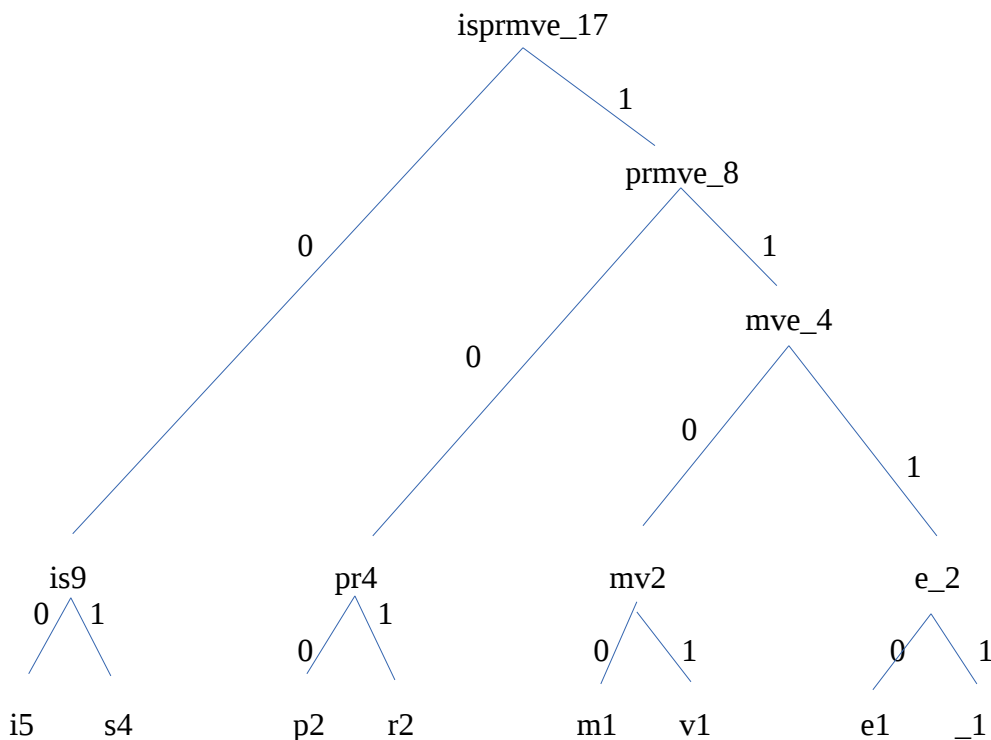


```
e1  _1
```

similarly we add (v1, m1) (p2, r2) (i5, s4) to get vm2, pr4, is9.



similarly we keep adding to the top as shown in the figure below to construct the tree. After constructing the tree we label the left branch with a “0” and right branch with “1” :



now the tree structure is complete. from this tree we get the codes for each letter .

For that we have to view the tree from the top to leaf for each respective letter. For eg taking the letter i. Starting from the root “ isprmve\_ “ it goes to “ is9 ” and then reaches leaf i5 through the branch 00 . thus assigned code for the letter “a” is 00 .

similarly we get codes for each respective letters by traversing the tree.

```
m ----> 1 ----> 1100
i ----> 5 ----> 00
s ----> 4 ----> 01
p ----> 2 ----> 100
r ----> 2 ----> 101
v ----> 1 ----> 1101
e ----> 1 ----> 1110
_ ----> 1 ----> 1111
```

and checking the obtained result we can see that the letter with least frequencies like “v” and “e”

need more bits to represent than letters with high frequency like “i” and “s” (2 bits each) .  
So now when we try to send the word “mississippi river” it needs only 46 bits instead of 136 bits

m	i	s	s	i	s	s	i	p	p	i	“_”	r	i	v	e	r
1100	00	01	01	00	01	01	00	100	100	00	1111	101	00	1101	1110	101

= 46

concluding we can say huffman algorithm compresses the bits required to represent a text data in an efficient way.

The following post will be explaining steps to implement huffman using python and java script based on this algorithm..