Huffman Coding

Huffman suggested an encoding technique for compression. Assume that we have a text file, and we want to compress the size of file. As per the standard a character takes a size of 8bits in memory. If we want to compress the file size then definitely, we need to store each character in minimum number of bits. **The question is what the size of each bit will be and what will be criteria of assigning the bits to each character**.

Let's take an example where we have set of characters and their frequency in a text file.

characters	**frequency**
a	45
b	13
c	12
d	16
e	9
f	5

There is total 100 characters in the text file and if each character requires 8 bits, then the total file size would be 800 bits.

If we randomly assign unique binary codes to each character for encoding purpose e.g., (a = 0, b = 1, c = 01, d = 10, etc). Don't you think it's the best approach? We can achieve maximum compression but there will be an issue in decoding part.

Consider this sample example: Input text: "**cd**", since "**01**" is the code of "**c**" and "**10**" is the code of "**d**" so the given text will be encoded as: "**0110**". Now when it comes to decoding part the given encoded string will be decoded as "**abba**" instead of "cd". **Why?** Basically, during the process of decoding it reads the contents of encoded string letter by letter and replace the codes with the matching characters. So, the first letter in the encoded string is "**0**" which will be replaced with "**a**" similarly the second letter "**1**" will be replaced with "**b**", third letter "**1**" will be replaced with "**b**" and fourth letter "**0**" will be replaced with "**a**". The encoded string is not converted back into the original input text, so the shortcoming of this encoding technique is that it is not able to decode the original input string.

Criteria to assign the code to each character:

While assigning the code to each character we need to make sure that the prefix part should not be the code of any other character e.g., if we are going to assign the binary code of 1010 to any character then the first three letters of the code i.e., "101" is the prefix part of "1010". So, if any part of prefix code is the code of any other character, then the encoding scheme will be failed to decode the original text. e.g., in the above-mentioned example, the code of "c" was "01" where "0" was the code of "a" and "1" was the code of "b" so the encoding technique failed to decode the original text back because prefix part of the code was the code of any other character. The best encoding technique will be the one which provides prefix free encoding. The encoding technique suggested by Huffman is prefix free encoding technique.

Idea: first create a Huffman tree and then assign codes to each character.

How to create Huffman tree?

Since the frequency of each character is given so **create a min heap** from the given set of characters.

Now iterate from 1 to N and in each iteration perform the following tasks:

1. **Create a "temp" node**.
2. **Extract two nodes from the min heap and make them left and right child nodes of the "temp" node.**
3. **Push the "temp" node back into the min heap.**

Huffman Algorithm:

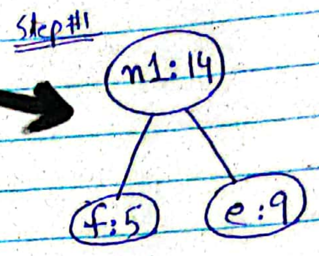
C[]: An array having characters and their frequencies at each index.

```
Huffman(C[], N)
{
   Create a minHeap
   Q = minHeap(C)  //here Q is basically representing the min-Heap
   for(i=1 to N-1)
   {
      Create a "temp" Node
      temp.Left = ExtractMin(Q)   //Extract min and make the left child
      temp.Right = ExtractMin(Q)  //Extract min and make the right child
      temp.freq = temp.Left.freq + temp.Right.freq    //update the freq of "temp" node
      Insert(Q,temp)  //Insert the "temp" node in the minHeap.
   }
}
//Time Complexity: O(N*logN)
```

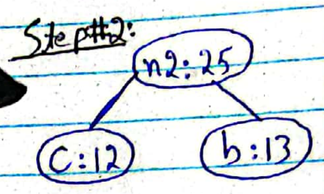
```
void heapifyUp(int index) {
        while (index > 0 && heap[parent(index)] > heap[index])
{
            swap(heap[parent(index)], heap[index]);
            index = parent(index);
        }
    }

void insert(int value) {
        heap.push_back(value);
        heapifyUp(heap.size() - 1);
    }

MinHeap minHeap;

minHeap.insert(5);
```

Q / Part No. Rough Work

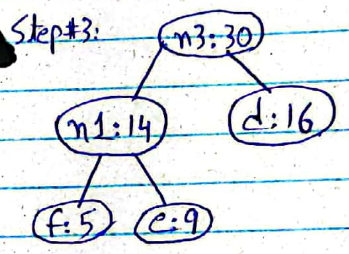
$a:45, b:13, c:12, d:16, \boxed{\cancel{e:9}}, \boxed{\cancel{f:5}}$



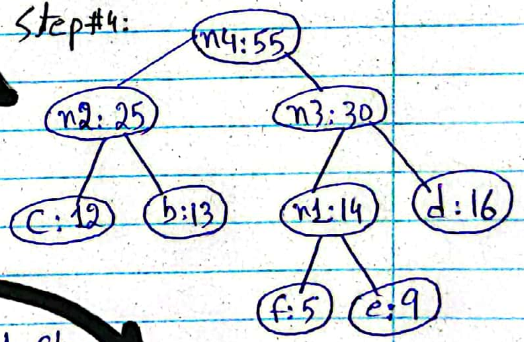
$a:45, \boxed{\cancel{b:13}}, \boxed{\cancel{c:12}}, d:16, n1:14$



$a:45, \boxed{\cancel{d:16}}, \boxed{\cancel{n1:14}}, n2:25$

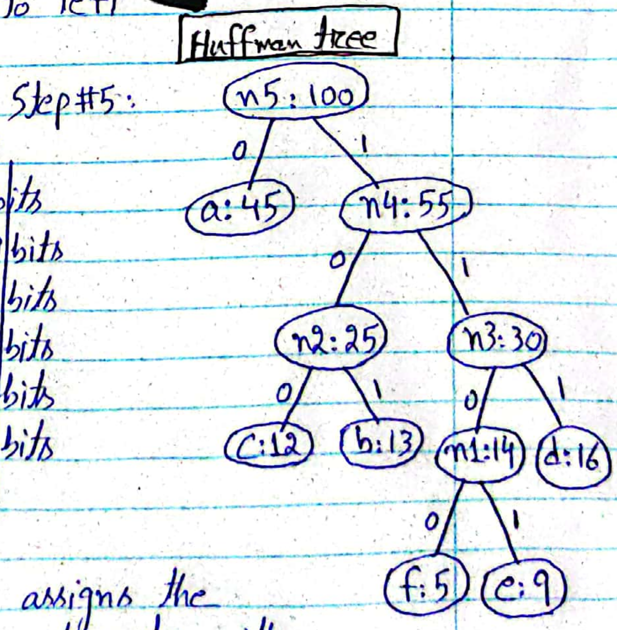


$a:45, \boxed{\cancel{n2:25}}, \boxed{\cancel{n3:30}}$



$\boxed{\cancel{a:45}}, \boxed{\cancel{n4:55}}$

We are assigning the value "0" to left child and "1" to right child



Characters	Code	# of bits	freq	Mem Req
a	0	1	45	45x1=45bits
b	101	3	13	13x3=39 bits
c	100	3	12	12x3=36 bits
d	111	3	16	16x3=48 bits
e	1101	4	9	9x4=36 bits
f	1100	4	5	5x4=20 bits
			Total	224bits

Notice that Huffman technique assigns the code of mimum length to the character with maximum frequency.

$\boxed{\text{Mem Req} = \text{\# of bits} * \text{freq}}$

Scanned with CamScanner