**huffman\_encoding()**

the function is pretty complicated and include number of helper function, different objets and inner recursive function:

•pre case- cases with 1 or 0 str ( or number of identical letter) – O(1)

1. create frequency table in dictionary structure ( dictionary is the most efficient way to find the key and the freq of the key) the function run once on all the letters in the string O(n)

2. create min heap of nodes – I used mean heap in order to take the min value of the data structure (while I insert items in during the step 3 process) mean heap is the most efficient way. there are n insertion , every insertion is take O(logn) time ->O(logn)\* O(n)->O(nlogn)

3. main function- create the Hoffman code structure- single loop(O(n)) + remove min heap O(logn) and insert (O(logn) -> O(n)\*2\*O(logn) -> O(n\*logn)

4. Generate the Encoded Data dictionary- dict which the key is the letter and the value is the code of the letter- O(n)

• again, the dictionary is powerful and efficient data structure for ne next part

5. Generate the Encoded Data— helper function that take str and return code according to the dictionary code -log (n)

Efficiency

When I sum and neglect the item the big o notation is O(n\*logn)

**huffman\_decoding(data,tree)**

It’s simpler function comparing the encoding function:

Precase- cases with 1 or 0 str ( or number of identical letter) – O(1)

1. Generate the Decoded Data dictionary- dict which the key is the code and the value is the letter - O(n)

2. decode the data – the step included loop and sub loop. The first loop check all the letters in the code(O(n/))) the sub loop check if the code id found in the dictionary(O(n)).

•explanation- The time complexity of both of the loops is function of the diversity of the letter- if there are high diversity of letters-> long code of most of the letters-> high time complexity for the second loop but low time complexity of the first loop and vice versa. So actually even thought there are nested loop , the time complexity is O(n) and not O(n^2)

Efficiency

When I sum and neglect the item the big o notation is O(n)

Test case:

I choose the regular case ( simple sentence) and case which there aren’t letters , single letter or sentence with one kind letter .

Reference

Object MinHeap was taken from : [www.educative.io](http://www.educative.io) and modify it for node list