Submitted By - Sanyam Rajpal
CSCI-B-505 Applied Algorithms
(FA19 - BL - 11503)
Programming Assignment - IV

WRITE-UP

I used Python language for my code because it's easy to understand and reading the text files and storing it in an apt format is very simple. I printed out the sum of the optimal weights to be visible in the terminal. During my coding, I did make some syntax errors which I resolved and then ran the code perfectly.

I calculated Huffman code for characters in the following book -
The Project Gutenberg EBook of The Flame, by Gabriele D'Annunzio

As can be seen from the code I made an empty dictionary of all the ASCII values with keys set to 0. Then I iterated over all the elements in the list and updated frequency every time I encountered one of the 96 values because only those values had to be considered.
I used heapq library of python to use a pre-implemented version of the heap in python.

In the code, I called the huffmand_encode function which first converts the dictionary to a minimum heap considering its frequency and the respective symbol for the same. Initially, the number of characters in huffman_code is set to 0. Then iteratively 2 minimum values are called from the huffman_heap. Then '0' is added to the Huffman code of all the elements in the smallest one and '1' is added to all the elements in the second smallest one. Then we merge the frequency of the smallest and the second smallest into 1 and merge their list of character and Huffman code symbols and then push into the heap. This process is done until only a single element in left in the list. This algorithm has been inspired from the lecture slides. Then we store the heap node(children along with their Huffman code).
Then I merged the dictionary with characters and frequency and the huffman_code lists together with the character being the element in the dictionary and the list of frequency and Huffman code as the key. Then I sorted them in the order.

Then the characters are printed in tabular form along with there Huffman code and frequency.
Then the average length of the Huffman code sequence is calculated using the sum of (Number of bits in Huffman code x character frequency in the file/sum of all frequency of all the characters in the file.) which gives the result stated above.

Heapify has the time complexity of O(logN) and so the heapify function is called O(N) where n is the number of elements in the dictionary. Extracting minimum element twice requires O(1) time complexity each in the for a loop. Besides, there is an iteration of elements in the 2 extracted minimum elements so that requires constant time complexity as well. Overall the loop requires O(**NlogN**) time complexity. Then we sort the elements in the min_heap according to the smallest element in the list and pass along 1 element at a time.

The **Average Huffman_Code length** is **4.4283**
The **average saved bits per character** from the standard 7-bit encoding was **2.5717**
(7.0 - 4.4283) Since the 7-bit fixed-length encoding using 7-bit code representation for all the elements in the 96 characters. So it is fairly efficient than the standard 7-bit fixed-length encoding for the current file.

I referred to class slides for the Huffman code algorithm and the following 2 links for understanding which datatype to be used for feeding into the heapq.

https://docs.python.org/3/library/heapq.html
https://www.codespeedy.com/how-to-encode-a-string-in-huffman-coding-using-python
/