Data Compression
Huffman Coding Algorithm

Problem Statement

In this problem, you are tasked with implementing a Huffman Coding System that efficiently encodes text by constructing a binary tree (called Huffman tree) based on character frequencies. Huffman Coding has applications in data compression, as it represents more frequent characters in a text using fewer bits.

Your goal is to write C++ classes/functions that implement Huffman Coding tasks like generating a frequency table, constructing a Huffman Tree, and providing character encodings as well as decodings. The encoder maps each distinct character in the input to a unique codeword (a sequence of bits) based on its frequency, with more frequent characters getting mapped to shorter codewords. Let C denote the number of distinct characters in the input text. Then the collection of C codewords is called the codebook. Additionally, you should represent the Huffman tree in Newick format to enable conversions between tree structures and the codebook. Note that a prefix code is an encoding under which no codeword is a prefix of any other codeword in the codebook. Huffman code is a prefix code.

Background: Huffman (Tree Construction) Algorithm, Optimal Prefix Code, and Tie-Breaking Rule

A binary tree with all C characters at its leaves can be used to derive a (prefix code's) codebook as follows. The codeword of each character c_i at a leaf is found by starting from the root, and recording the path to the leaf, using a 0 for indicating a left branch and 1 for a right branch. The cost of the code is given by $\sum_{i=1}^{C} d_i f_i$, where d_i is the depth of the leaf with character c_i in the binary tree (i.e., length of the codeword for c_i) and f_i is the number of occurrences (frequency) of c_i in the input text.

The above binary tree based encoding yields a prefix code, and the prefix code with optimal (minimum) cost can be obtained using the greedy Huffman algorithm as follows. Huffman algorithm maintains a forest of trees. The weight/frequency of a tree is equal to the sum of the frequencies of its leaves. In each of $C - 1$ iterations, select the two trees, T_1 and T_2, of smallest weight, breaking ties as mentioned below, and form a new tree with T_1 and T_2 as left and right subtrees. At the beginning of the algorithm, there are C single-node trees – one for each character, constructed in a lexicographic order (e.g., the single node for 'a' is constructed earlier than for 't'). At the end of the algorithm, there is one tree, and this is the Huffman tree corresponding to the optimal prefix code.

To break ties when two or more trees in the forest have the same weight, we choose the tree whose root was constructed earlier in the above algorithm. Also, T_1 and T_2, the trees in the forest with the two smallest weights in the above algorithm are merged such that – the tree with the lower of these two weights is set as the left subtree, and the other tree the right subtree of the merged tree (if both T_1 and T_2 have the same weight, the ties are again broken by setting the tree whose root was constructed earlier as the left subtree).

Summary of Tasks, and Expected Running Time

You are tasked with implementing various aspects of a Huffman Coding System, with each functionality triggered by a specific command and its appropriate inputs. Your implementation will handle 4 main categories of tasks and have expected running time complexity as indicated:

- **Q1: Encoder** — Commands to build frequency tables ($O(M \log C)$ time), and construct Huffman trees ($O(C \log C)$ time).
- **Q2: Decoder** — Commands to derive Huffman trees from codeword tables ($O(CH)$ time) and decode encoded messages ($O(NH)$ time).
- **Q3: Enhanced Encoding** — Commands to further optimize the tree construction ($O(C)$ time) and count distinct Huffman trees ($O(C \log C)$ time).
- **Q4: Length-limited Encoding** — Command to limit maximum codeword length.

In the expected running time complexity indicated above,

- C is the number of unique characters in input text/message,
- M is the length of the input message in characters,
- N is the length of the encoded message in bits, and
- H is the height of the Huffman tree.

Constraints on the inputs/parameters

Each command is triggered by its "Command Name", followed by its appropriate inputs, and then terminated by "Quit" command. For the parameters defined above,

- $2 \leq C \leq 63$ (i.e., 2 to 63 possible characters in message, all of which will be either lower-case or upper-case alphabets or digits (0-9) or '_' (underscore).
- $2 \leq M \leq 10^7$ characters.
- $2 \leq N \leq 10^5$ bits.