Assignment Report

ZHUANGSHENG LI 201769230

Part I
First, the input on the first line is read in isolation to determine whether to expand or compress.

For the expansion operation, the variable "next_ch" is declared to record the first time a character differs from the current character "ch", and "dup" is used to record the number of consecutive occurrences of the current character "ch".

Print directly when the number of repetitions is less than three times, such as reaching the end of the file directly out of the loop.

If the number of repetitions reaches three times, the last two characters are a two-digit hexadecimal count. Converts the count to decimal and prints it.

At the end of the extended loop, the first character different from the meta "ch" is assigned to "ch" to enter the next loop.

For compression, the variable "flag" records whether the file has been read through.

The "next ch" variable records the first character that is different from the current character.

Print directly when the number of repetitions is less than 3, set "flag" to 1 at the end of the file, and then jump out of the counting loop. If the number of repetitions is greater than or equal to 3, the number of repetitions is printed in hexadecimal.

At the end of the extended loop, the first character different from the meta "ch" is assigned to "ch" to enter the next loop.

Part II (reported by function)

Main():

Define "MAX LEN OF LINE" as the maximum input length OF a line and "MAX LEN of WORD" as the maximum word length.

Define a tree node. Nodes can store words, counts, and subtrees.
Get the first line of target word input, copy it and capitalize it for comparison.

The tree structure required by the problem is a binary search tree. The tree is sorted lexicographically. The node generated by the target word is set as the root.

Get the input line by line, split the input by "strtok" and add the obtained word to the tree.

After the input is complete, calculate the number of target words according to the "count" value of the root node.

Finally, the resources occupied by the tree are released

addTree():

By comparing the lexicographic order of the current word and the current node representing the word, the smaller lexicographic order is added recursively to the left subtree, and the larger lexicographic order is added recursively to the right subtree. If the lexicographic order is the same as the current node representing the word, the count is added by one.

freeTree():

Recursively releases each node and pointer.

toUpper():

Convert all letters to uppercase and non-letters to Spaces.