CS 2123 Programming Project 4
Huffman Coding
(100 pts)

Project Description
In this project, you will implement the Huffman Coding and apply it to compress and decompress files.

Create Huffman Coding and compress a text file
Your program must follow these steps:
(1) Analyze an input text file to count the occurrences of each symbol or character. Note that symbols or characters from text files are not always visible. For example, the newline character "\n". Your program must work for any character (visible or invisible, English or non-English). The text files used to test your program are downloaded from www.gutenberg.org (Project Gutenberg).
(2) Based on the statistics on symbols/characters, build the Huffman "trie" using the algorithm introduced in our lecture. As you build the "trie" in a bottom-up manner, a priority queue must be used to select group(s) of symbols with minimum total occurrences. In Python, priority queue is supported by the *heapq* module. In Java, you can use *java.util.PriorityQueue*.
(3) Read the Huffman "trie" to obtain the Huffman Coding (sequence of 0's and 1's) for each symbol/character. In Python, store the coding in a *dictionary* with symbols as the keys and sequences as the values. In Java, store the coding in a *java.util.HashMap* with symbols as the keys and sequences as the values.
(4) Read the input text file again symbol by symbol, translate each symbol into its sequence by checking the dictionary/HashMap from (3), then concatenate these sequences into a long sequence, and store it as a binary (bin) file.
For example, assume the following Huffman Coding for letters A, B and C:
A: 10, B: 0, C: 111
, then the text "ABCAABBBC" is translated into a long sequence "1001111010000111", which happens to contain 16 bits (2 bytes). When the sequence is stored as a bin file, the file should contain exactly two bytes:
Byte1: 10011110 or 0x9E
Byte2: 10000111 or 0x87
If the number of bits in the long sequence is not a multiple of 8, append extra 0's to make it so.
Hint: In Python, an integer variable can be converted to a byte using to_bytes() method. To open a bin file, use open("binFileName", "wb"). To write to it, use its write() method. In Java, a byte variable can be directly written to a bin file. To open a bin file, use *java.io.FileOutputStream*. To write to it, use its write() method.

(5) Calculate the compression ratio. Your program should count the total number of symbols in the input text. Suppose each symbol needs 1 byte to be stored, then the compression ratio is computed by:

$$compression\ ratio = \frac{bin\ file\ size\ in\ bytes}{number\ of\ symbols\ in\ text}$$

Note: the compression based on Huffman Coding should achieve a ratio of 50-75% for natural languages. Check if your results make sense.

Decompress a bin file to obtain a text file

Your program must follow these steps:

(1) Read the bin file byte by byte, and decode each byte to restore to the corresponding symbol(s), then write the symbol(s) to a new text file.

Your decoding program must be able to handle all the complications of the bin file. For example, the Huffman coding of a symbol may be wider than a byte. It is possible for one byte to represent multiple symbols. It is also not uncommon to see a symbol's coding crossing the boundary between bytes.

Hint: In Python, to open a bin file for reading, use open("binFileName", "rb"). To read the bytes, use the read() method. In Java, to open a bin file for reading, create an object with java.io.FileInputStream. To read the bytes, use the object's read() method.

(2) Compare the new text file with the original input file. The two files should be identical.

Input and Output Requirements

Your program must take a text file as the input file. The project provides six text files (book1-book6.txt) to test your program.

book1, 3, 6	book2	book4	book5
English	Germany	French	Spanish

When your submission is graded, in addition to these six files, there will be another four randomly selected texts from the Gutenberg project to be tested on.

Each execution of your program must generate two output files. One is the bin file encoding the input file. The other is the new text file, which is created by decompressing the bin file.

For example, suppose your program takes "book1.txt" as the input text, the two output files should be "book1.bin" and "book1copy.txt"

Useful Resources

(1) You may want to check the binary file created by your program. The following web tool can be used:

https://iamkate.com/code/binary-file-viewer/

Submission Requirements

Submit all your programs (.py or .java) and a two-page report (.pdf) to Harvey. The report must include:

(1) How you implement the project

(2) The compression ratios obtained for each book (book1-6). Are these ratios reasonable? Why?

(3) Which symbols are the top 5 most frequent in book1? What are their Huffman codes?

Academic Misconduct

The submission must be your own work. Making copies from online resources (ChatGPT included) or other students is not acceptable, and will be dealt with according to the university policy on academic misconduct.