Assignment: Text Compression

Author: SIFAN WU

This work is to compress a text with Huffman Coding text compression algorithm in Python and investigate the advantages and disadvantages of character-based vs word-based encoding. I have done the job and make a table to show different parameters including the size of the symbol model and compressed file, the time of building the symbol model, encoding and decoding file.

There is some brief about compress and decompress processes:

Compress:
In the beginning, I should read each character or word from the input file. In this part, I choose *re* module (using *pattern. findall()*) to find them, then the original file has been spitted into characters or words. At the same time, count each character and word and save them in a list. Finally, I get a list which contains the symbol frequency and each symbol.
The main part is building a Huffman code-tree. I create a new class for nodes. Each node has four parameters such as left-child, right-child, father node and frequencies. In this case, I use frequency of symbol as its weight. Creating Huffman tree is to select the two lowest weight nodes and combine them to form a new subtree from the list has been made. In the next step, adding this new node into the list and taking the two lowest weight nodes out is necessary, then two new lowest weight nodes appear and keep repeating the last step until there is only one node left, then the Huffman code tree has been built. If the node is not the root node, then its left-child is given with "0" and the right-child with "1". Saving the symbol model as a key to decode the compressed file. Changing the original characters or words with the Huffman code. Now, all the content has been replaced with "0" or "1" strings. If I want to compress them into binary file, the total number of characters must be multiples of eight, so I add "0" in the end of string to make sure strings can be write into binary file, and add a byte that represents the number of adding "0" in the beginning of characters at the same time, the reason is for decoding the compressed file.

Decompress:
Firstly, open the compress file and read with binary way. I choose to read by one byte to make sure no item is lost, because I put a number at the beginning of string, the first few characters are '0'. The next step of decompressing is converting numbers to binary character strings by using *bin()* function. Deleting the first two characters that is "0b" and adding "0" at the beginning of binary character string to make sure each string has 8 bits is needed before combing all strings as the final decompressed string. After getting the final string, deleting the first 8 bits and the "0" characters that were added in the compress process. Th last step is converting character strings to the original characters or words by using a dictionary saved in the *infile-symbol-model.pk*. The

decompress process has done until now.

Some improvement:
In the process of program, I find that the system need to traverse the loop, so if I choose the method called dichotomy find method, that will be faster when dealing with a big file.

Compress types	*Size of original file*	*build the symbol model*	*Size of the symbol model*	*encode*	*decode*	*Size of compressed file*
character-based	1214(KB)	0.27(s)	2(KB)	2.35(s)	4.48(s)	675(KB)
word-based	1214(KB)	19.56(s)	846(KB)	0.89(s)	2.11(s)	383(KB)

Character-based:

Advantages:

- ✓ Building the symbol model is fast
- ✓ The size of the symbol model is small

Disadvantages:

- o Need more time for encode and decode
- o Th compressed file is bigger

Word-based:

Advantages:

- ✓ The size of compressed file is smaller
- ✓ The time for encode and decode is smaller

Disadvantages:

- o Th size of the symbol model is much bigger
- o Need more time to build symbol model

Compared to each other, It is obvious that the effectiveness of word-based is higher because of the compressed file size and time of encode and decode. The most time is for building the symbol model, so if we already have a symbol model of word-based that contains usual words and signs, we do not need to waste time for building the symbol model, that will make this compress way is effective and good.
Under configuration of character-based, it cost less time in total because its symbol model is simpler, most time is used for encoding and decoding. The compressed file is bigger with the simple symbol model. In other words, the compress way of character-based is good at saving time instead of effectiveness of compressing.
In my opinion, if the original files are not big, choosing character-based is a good idea. When the original files are big and there are some requirements about the size of compressed file, selecting word-based is much better.