TEST COMPRESSION ASSIGMENT

IMPLEMENTATION

In the implementation, two python files have been created, huff-compress.py and huff-decompress.py.

In the huff-compress.py, it mainly does compression for a text type file in two methods (character based and word based). Firstly, it needs to read file and count the number of the characters or the words and calculate the probability in all. Secondly, a Huffman Model, also called Huffman Tree is built, done by the function build_huffman() according to the probability of each term and the Huffman Model is stored in the dictionary data structure. Finally, the original text need to be encoded according to the Huffman Tree and meanwhile the code is converted in the binary type and store in the new file. This step is completed by the function convert_binary() and store().

In the huff-decompress.py, firstly, it need to read the .bin file and .plk file which store the encoding content and Huffman Tree respectively. Secondly, putting the model into the class Huffman_model and converting the binary code into String. Finally, reading the binary string char by char and iterating the Huffman Tree to decode the content is completed by the function iterate_() in class Huffman_model. Finally, the binary string would be completely decoded into readable content and stored in a new file.

PERFORMANCE

The program has successfully pass the evaluation program (which is test-harness.py). For the performance, the result would be shown in the below.

ENCODE

Symbol	Origin file size	Encode file(.bin) size	Model file size(.plk)	Cost of building model	Cost of encoding
char	1,220,150 bytes	690,360 bytes	1,389 bytes	0.15s	0.32s
word	1,220,150 bytes	392,562 bytes	556,815 bytes	0.75s	14.46s

DECODE

Symbol	Origin file size	Decode file Size	Cost
char	1,220,150 bytes	1,220,150 bytes	5.43s
word	1,220,150 bytes	1,220,150 bytes	2.83s

IMPROVEMENT

According to the result in the above, it could be found that using char-based method to do compression for the text the model file would be smaller than using word-base, but the encode file would be larger than using word-based. So, choosing a method is dynamic which means if the text size is not big (e.g. less than 10M), the char-based method could be chosen. If the text size is large (e.g. more than 10M), the word-based method could be chosen, because when the size of the text increase, the occurrence frequency of the same word would be increase so that the size of the model file would be increase slowly, then the total size of model file and encode file would be exceed by using char-based.