Registration number: 180128228

Text Processing: Text Compression

Introduction

The purpose of this report is to highlight the implementation to a brief extent that I have achieved for the purpose of this assignment and I will demonstrate the Huffman compression and Huffman decompression implementation I have achieved throughout this assignment. The purpose of assignment was to implement Huffman Coding text compression algorithm in Python.

Implementation stages

As demonstrated in coding and files I have constructed and implemented two Python files as per requested which are huff-compress and huff-decompress which were tested against the tes-harness.py that was provided to us.

Huff-compress implementation:

1. I have implemented regarding the huff-compress two different file methods which are for word based and character based.
2. This is achieved by the frequencies are calculated for each unique characters and words.
3. After I have achieved this implementation I have constructed the Huffman Tree which calculates the likelihood of each unique word which is kept in a dictionary data structure.
4. Another implementation to be achieved is that making a binary conversion and then printing out the results to an output binary file.
5. As you can see in the coding regarding the word based that it finds all the words and also it finds all the symbols. Where wording_array refers to the frequency of each word in the text.
6. Regarding the pseudo-EOF this is where the binary encoding concerning the text is a random number of bits.

Huff-decompress implementation:

1. The huff-decompress Python file is an easy Python file to read which highlights and demonstrates regarding the .bin file and regarding the. pkl file which keeps the text in the Huffman Tree.
2. I have also constructed and implemented a way that the binary code can be converted into a string.
3. Furthermore, regarding the Huffman_model and converting the binary into strings. Moreover, this reads the binary string via character and regarding the iterations it decodes the entire document content.
4. The binary string has now been fully decoded so it can be read easily without any difficulties.
5. As highlighted the infile-symbol-model.pkl and regarding the infile.bin which is an encoded file.

Registration number: 180128228

Results/performance:

Below are the results from the implementations that I have achieved:

	File size	.bin file size	. pkl file size	Encoding	Decoding
Char	1,243,083 bytes	692,224 bytes	1,387 bytes	1.7 seconds	3.26 seconds
Word	1,243,083 bytes	392,562 bytes	556,815 bytes	1.0 seconds	2.37 seconds

Conclusion:

As you can see from the results above that the file sizes and regarding the encoding and decoding it could be stated that the turnaround time it much quicker for word-based characters. But in fact, if you look at the results displayed above it could be said that using the character-based model regarding the compression for the file can be considered smaller compared with working with word based. Moreover, regarding the encode file it can be assumed safely that it can be bigger compared to consuming. The program was tested against the test-harness.py and passed the tests.