Data (Text File) Compression - ECM1414

Compression Algorithms

1. *Huffman coding*

This is a lossless type of compression. I have used that type of compression in my continuous assessment. I believe it is the simplest to explain and implement. This definition describes it the best - “A minimal variable-length character coding based on the frequency of each character”. We have to analyse the text to work out the amount of unique characters and the total amount of them in the text. It works so that each character becomes a one-node binary tree. Then each node has the character frequency. The more same characters appear in the text the greater the frequency of the specific character. After we know all the frequencies, we can start to code the characters. For high frequency, we use smaller binary numbers, and then we code the rest

.

Example:

Let us say character “e” is the most frequent character in the text. We assign this character to binary number 0. Then we repeat this process until all characters have a relevant binary code. We put the text together but now using the binary numbers. It will compress and decrease the size of the original file.

Of course, to reach the original file we need the function to decompress the binary file. It will work the same way as the compress but the other way round. As I have said, easy and simple. This algorithm operates in O (n log (n)) time

2. *Run-length encoding*

This is a lossless type of compression. The definition is - Form of data compression in which a stream of data is given as the input (i.e. "AAABBCCCC") and the output is a sequence of counts of consecutive data values in a row (i.e. "3A2B4C"). It is usually used for image compression; nut also can be used for text. I believe if we use it with Huffman Coding (since we only have file with zero and one after Huffman Coding) it may be sufficiently decrease the original size of the file.

Example:

It is clear how this algorithm work on text; therefore let us look at the picture. Let us imagine a picture of two colors (black and white) 8x8. Each row has a pattern where black and white pixels are placed. To compress this file we will count how many whites and blacks are next to each other in a row. Instead of writing the more by one (w,w,b,b,b,b,w,w) we put a number of continuous whites and blacks (2w,4b,2w) which decreases our row from 8 bits to 6 bits. After we do the same for every row, we can get a sufficiently big compression.

3. *JPEG*

This is a lossy type of compression. To compress a file it will decrease its quality. After compression, the file cannot be restored to its original quality. The additional data is destroyed and deleted. The way JPEG works is by looking at RGB, and replaces more deep colors with more simple colors. The amount of compression can be regulated, therefore sometimes you won’t even be able to see the difference between the original and the compressed image.

Example of an original image and a JPEG image

As you can see, there are minor differences between two pictures

Data Structures and Algorithms Used

1. Lists - these data structures are mostly used to store data, in my case variable such as self.nodes is used to store the nodes

.

2. Dictionaries - in my code - self.indexes, self.indexes_for_decompression. In the dictionaries keys are related to vales, therefore it's easy to understand what character is related to which index.

3. Heap Tree - this data structure is a priority queue. In my code it is used because of one important property - when the pop method is called, it takes the smallest values in the queue. For the Huffman Coding it is an essential feature because to build the tree we need to start with characters that have the smallest frequency.

4. Search Algorithms - it is not an actual algorithm, but I thought i should mention it. In my code there are a lot of “for … in …” which are simply search functions. Look for element/s in this data structure.

Development Progress

Week 1: Look at the assessment and start solving the problem in mind. Set up the environment. Decide what books to choose.

Week 2: Research on Huffman Coding. Download the books (including the other languages) and put them in the correct form for the python reader. Pass the files into python and experiment with them.

Week 3: Make a plan on the development of the project. Look for information on how to analyse data files in python.

Week 4: Understand the Huffman Tree Algorithm. Make additional functions to implement the Huffman Tree.

Week 5: Finish the Huffman Tree. Look for information on compression. Start with compression.

Week 6: Make tests on compression and the code in general. Comment the code.

Week 7: Finish the implementation of the program.

Week 8: Start to write the documentation.

Week 9: Check everything for minor mistakes. Make final changes and submit the ca.

Performance analysis

I have choose two books:

1. “Metamorphosis” by Franz Kafka
2. “Sherlock Holmes” by Arthur-Conan Doyle

I have not found the original translation for French and Portuguese, therefore I have used an additional software to translate them online.

My results are:

“Metamorphosis” in **English** original size - 119 KB
“Metamorphosis” in **English** compressed - 65 KB
“Metamorphosis” in **French** original size - 132 KB
“Metamorphosis” in **French** compressed - 71 KB
“Metamorphosis” in **Portuguese** original size - 123 KB
“Metamorphosis” in **Portuguese** compressed - 66 KB

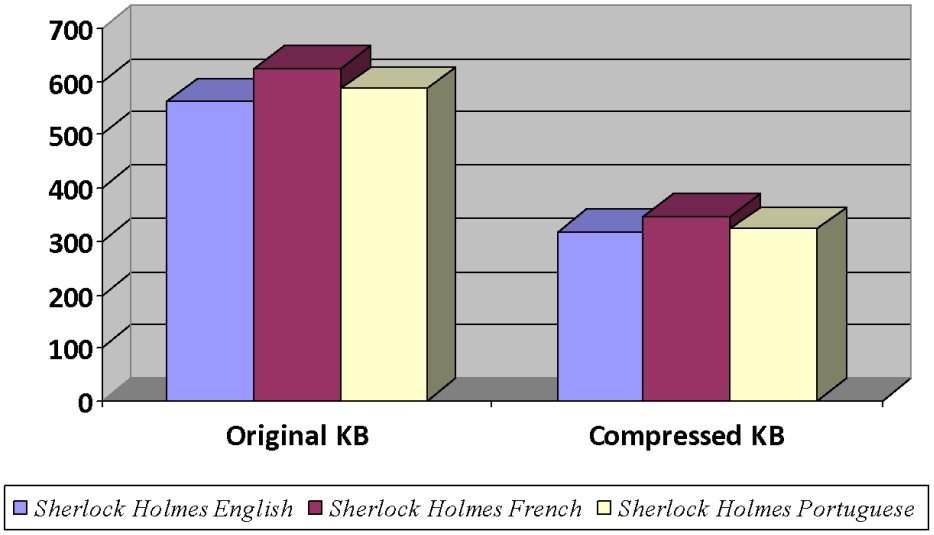
On average, the compression file of “Metamorphosis” is smaller than the original by 46%

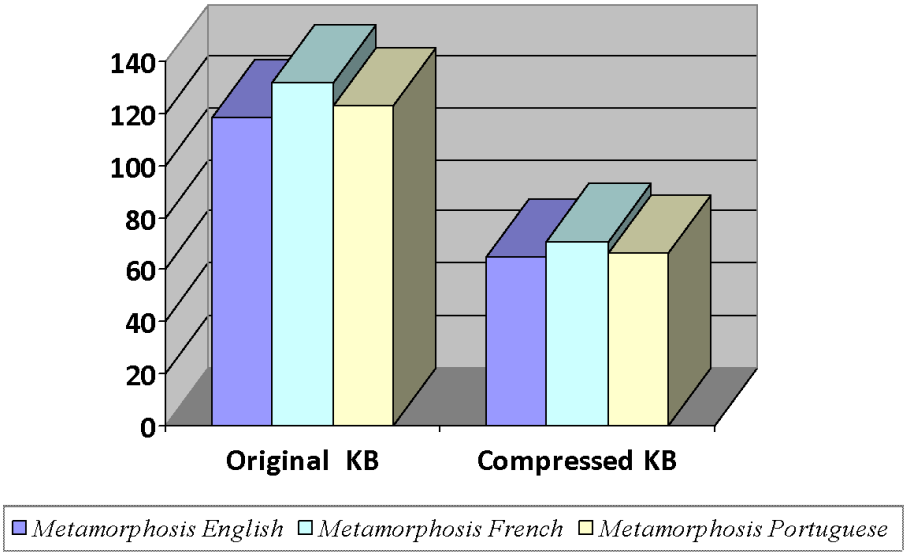
“Sherlock Holmes” in **English** original size - 562 KB
“Sherlock Holmes” in **English** compressed - 346 KB
“Sherlock Holmes” in **French** original size - 625 KB
“Sherlock Holmes” in **French** compressed - 346 KB
“Sherlock Holmes” in **Portuguese** original size - 586 KB
“Sherlock Holmes” in **Portuguese** compressed - 323 KB

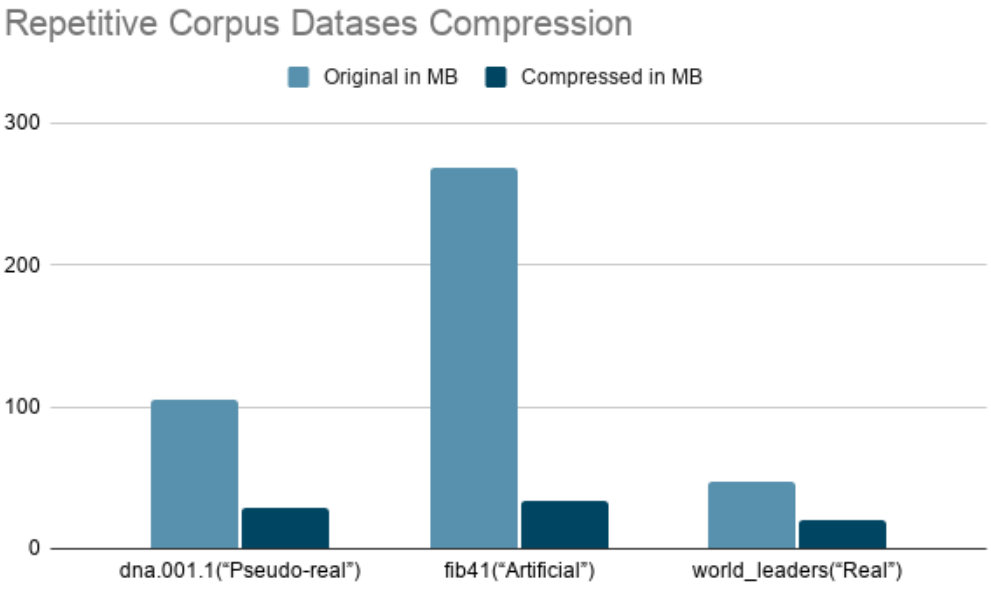
On average, the compression file of “Sherlock Holmes” is smaller than the original by 44.4%

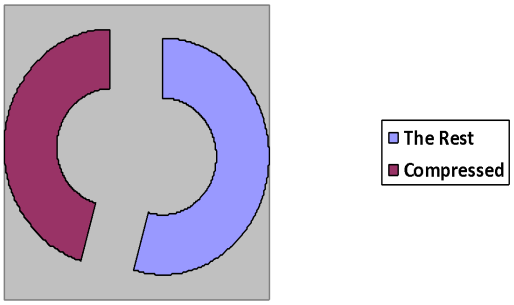
An obvious trend is that this compression is compressing the original file by almost a half.

Performance charts

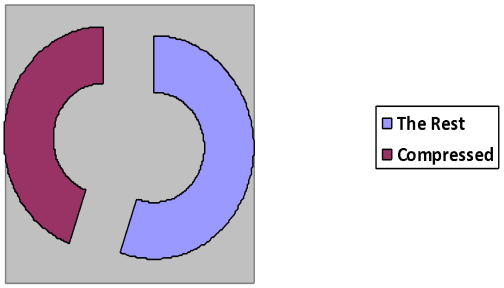




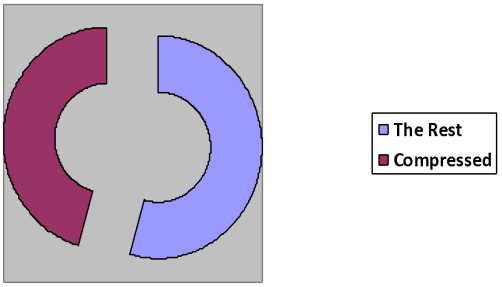




English books compression on average



French books compression on average



Portugese books compression on average

An obvious trend is that language does not really affect compression

References

https://stackoverflow.com - used for small problems within the code
https://www.youtube.com - watched many videos on the Huffman Encoding for better understanding
https://docs.python.org/3/tutorial - python documentation that helped me develop the Huffman Coding Algorithm

Most of the information was taken from the Continuous Assessment description