Shahid Bilal Razzaq
11/22/2016
Assignment 12

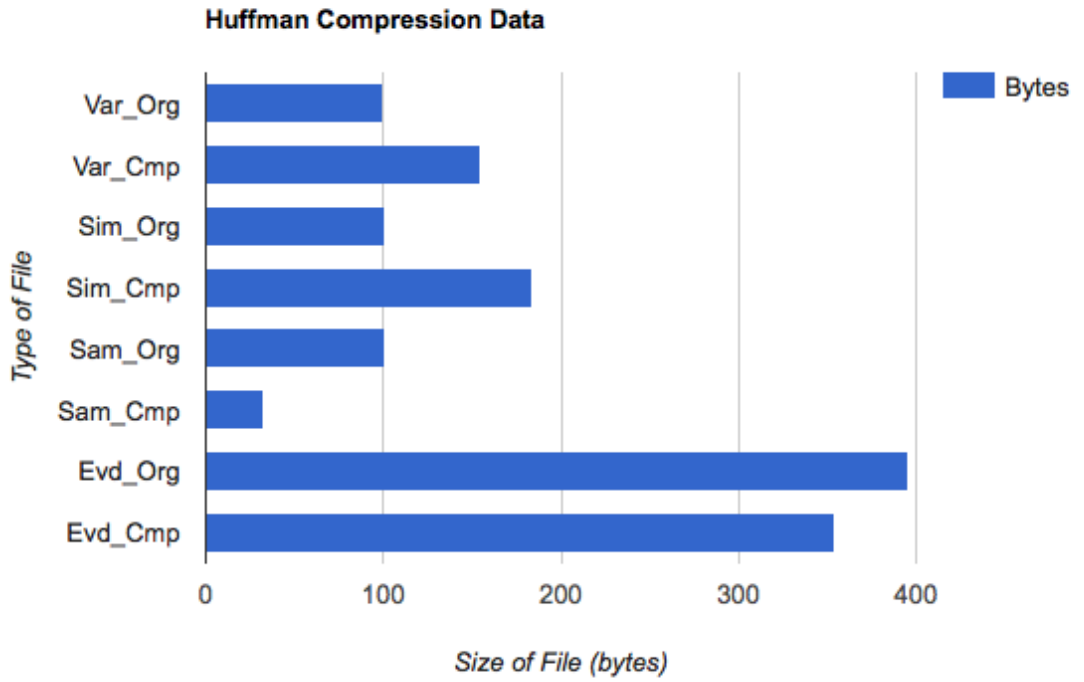
Analysis
Huffman Compression Algorithm

1. Design and conduct an experiment to evaluate the effectiveness of Huffman's algorithm. How is the compression ratio (compressed size / uncompressed size) affected by the number of unique characters in the original file and the frequency of the characters? Carefully describe your experiment, so that anyone reading this document could replicate your results. Submit any code required to conduct your experiment with the rest of your program and make sure that the code is well-commented.���Plot the results of your experiment. Since the organization of your plot(s) is not specified here, the labels and titles of your plots(s), as well as, your interpretation of the plots is critical.

```
* Methodology For Compression Experiment: Test a variety of files with the
* following content, and then test file size using file.length() method
* Criteria for Files:
* 1.File with 100 varying chars
* 2.File with 100 similar chars
* 3.File with 100 Same Char We are testing for frequency of
* characters, and how it affects compression.. the file size will be set to
* a standard of 100 chars, so to have a fair comparison. Chars can be
* anything from letters (upper/lower case) to symbols and spaces. Final
* test will be for a file containing an "everyday" writing type paragraph
* to simulate a real world scenario
*
* THIS EXPERIMENT REQUIRES THE FOLLOWING FILES: (included with assign.)
* 1."1_Original_Varying_Chars.txt"
* 2."1_Original_Similar_Chars.txt"
* 3."1_Original_Same_Chars.txt"
* 4."1_Original_EveryDay_Paragraph.txt"
```

This experiment was designed to assess the performance of the compression algorithm based on the frequency of the chars that occur in the file. Four different types of files were created to simulate a variety of files the compression algorithm can encounter.
All the files except for the last one were standardized to 100 chars to level the playing field of sorts. The first type of file created is a file of varying random chars (letters, numbers symbols etc.) with a random frequency. The second type of file is a file with mostly similar chars, the third has the exact same chars, and the final file simulates an everyday paragraph one would write.

Each of these files were compressed using an instance of the HuffmanTree class, and a corresponding compressed file was generated. Then using java's built in file.length(); method in the File object, I got the size of the files in bytes. Each original file is compared to the compressed file.



Acronyms: VAR = varying file, Org = original, Cmp = compressed, Sim = similar file, Sam= same File, Evd = everyday file

As we can see from the data, in the case of a varied char file, the compressed version is actually a larger file than the original file. This could be due to the fact, that because there are many unique characters, and not as many frequently occurring elements, the compressed file would then turn out to be larger than the original. The similar char file exhibited the same results, because of the same reason, but the same char file, with a high frequency of the same char effectively compressed to over 70% of its non compressed size. This is where the Huffman algorithm shines, as the level of compression will be dependent on the frequency and probability of the characters occurring in the file.

Finally, the file simulating the everyday paragraph shows a slight compression compared to its original file. This can vary depending, again, on the frequency of commonly used chars. In standard writing style, we can expect a large number of vowel chars, and less unique characters such as 'z' or 'x'. Thus compression can be really good, to just average depending on this factor. For my file, compression was slightly better than the original file.

2. For what input files will using Huffman's algorithm result in a significantly reduced number of bits in the compressed file? For what input files can you expect little or no savings?

Files in where there is a large number of frequently repeating characters, symbols, numbers, etc. will benefit from huffman's compression technique. Comparatively, files with a large number of unique characters, etc. will have the least benefit in terms of compression

3. Why does Huffman's algorithm repeatedly merge the two smallest-weight trees, rather than the two largest-weight trees?

We merge smallest weight trees because those trees are the ones that have the least frequency. This way when we build up the tree, the path to the lowest frequency will be more bits than the path to the higher frequency. This is how the algorithm works to "compress" the more frequently used "chars/words/numbers etc." by shortening the path to them, thus using less bits. If we merged the largest trees together, then we would not achieve this goal and the compression would not work.

4. Does Huffman's algorithm perform lossless or lossy data compression? Explain your answer. (A quick google search can define the difference between lossless and lossy compression).

In lossy data compression algorithms, not all of the original data can be regenerated during decompression. Lossless data compression is when all of the original data can be regenerated during decompression. Huffman's algorithm is an example of lossless data compression because when we decompress the file we get the original file back with all its original data.

5. How many hours did you spend on this assignment?

3-4 ish