Science Fair Project Write-Up

GitHub Repository:

Question: Which type of file can my Huffman tree algorithm compress the most effectively and/or quickest?

Hypothesis: The sparse file is the quickest, and the most compressible. The uniform files will be the least compressible.

Materials:

- A Computer (My computer is an Apple MacBook with Mac OS X 10.6.8 - see below for more specs)
- The Programs I wrote (avaliable on my GitHub repository - see top)

Computer Specifications:

- Processor: Intel Core 2 Duo
- Memory: 2GB 800MHz DDR2 SDRAM
- OS: Mac OS X 10.6.8 Snow Leopard
- Apple MacBook

Procedure

1. Gather materials
2. Generate the files to test
3. Close all other programs
4. Run the compression algorithm on all of your files
5. Record results

Variables:

Constants:

- Computer run on

- Number of times in a row run

Responding Variable:

- Speed/Rate of Compression

Manipulated Variable:

- File Run On

Background Information:

In my experiment I will be testing a compression algorithm I wrote. This uses a Huffman tree based method. This method uses the frequency of each character to optimize compression. The way most compression algorithms work is to shorten the string of bits used to represent a character. Other binary tree algorithms use fixed length codes. This means that all the codes are the same length for each character, no matter the frequency. The special thing about a Huffman tree technique is that it has variable length codes, and uses frequency to determine which characters should have the shorter codes than others. Therefore, it should have a better compression rate.

Background Research Citations:

1. "Data Compression." *Wikipedia*. Wikimedia Foundation, <http://en.wikipedia.org/wiki/Data_compression>.
2. "Huffman Coding." *Wikipedia*. Wikimedia Foundation, <http://en.wikipedia.org/wiki/Huffman_coding>.

Conclusion:

Based on the data in my experiment, my hypothesis is partially correct. The sparse file was compressed more than the other files, at 86.9246%. However, it was incredibly slow compared to the other files. This is surprising, and further work could be done on that.

My hypothesis was also correct that the uniformly distributed files were the least compressed. The reason the uniform files are compressed at a low rates is because the Huffman tree has to give long codes to frequent characters. Therefore, the compression on the file isn't very high, and could become negative if the file has enough size and diversity. Negative compression is making the file bigger rather than smaller.

Based on the data the larger files seem to take a shorter time per bit. This is due to the fact that there is a certain time for "setup" at the beginning of the program, that is about the same for all the programs. The large file has more bits to spread that time across and therefore has a shorter time per bit.

The Huffman tree algorithm has a maximum compression rate of 87.5%. The only way to achieve this is to have a file that has all the same characters in it. The sparse file had close to maximum compression, and for a good reason. The sparse file should be about 99.3333% zeros and 0.6666% other characters. The percent of maximum compression for sparse file was 99.3424%. The maximum compression is 87.5% because the most a Huffman tree can compress an 8 bit character down to is 1 bit. One divided by eight is .125, converted to a percent and subtracted from 100 you are left with 87.5. That's only if there is one kind character in the file. So to get maximum compression you would have to have a file of all a's.

In this experiment I only tested the performance of the Huffman tree, and limited file types. I wonder how the Huffman tree algorithm would hold up against another compression algorithm? Test it with more file types? Further work could be done testing a wide variety of files. It's not conclusive if adding more files would improve results. Overall I think my project went well and produced good results about the Huffman tree algorithm.