COMP20290 - Compression

In this practical you will build your own command line utility for compressing text files, one you could practically use to compress your own files.

Your task for this practical is to implement a fully functional Huffman Compression command line tool that can both compress and decompress text files.

Huffman's algorithm is an example of a greedy. It is called greedy since the two smallest nodes are chosen at each step, and this local decision results in a globally optimal encoding tree. In general, greedy algorithms use small-grained, or local minimums/maximums to result in a global minimum/maximum.

If you encounter any difficulties with the practical, make sure to consult the lectures, look at resources online and ask questions to the demonstrators.

Tasks

1. Develop a Huffman tree by hand.
2. Implement the methods given in the starter code.
3. Test, analyse and compare your algorithm with various inputs.

NOTE: The implementations need to compress to binary codes, i.e. making heavy use of the BinaryStdIn / BinaryStdOut classes.

What is provided?

Some helper classes/files are provided. The starter code includes the methods necessary for a fully functioning Huffman coding suite.

Also provided is an implementation of Run Length Encoding.

Input files:

1. mobydick.txt - The text of the full book "Moby Dick"
2. medTale.txt - Some sample paragraphs
3. q32x48.bin - A bitmap file
4. genomeVirus.txt - Genetic code data

Task 1 - Huffman by Hand

Create a Huffman tree and codeword table for the phrase: “Home is where the heart is”.
You should follow the steps outlined in the lectures to create your Huffman tree by hand which include:

1. Count the characters in the input phrase (including the space character)
2. Build your tree from the bottom up beginning with the two characters with the lowest frequency
3. Merge these into a node / sub-trie with combined weight
4. Continue working through your character frequency table until you reach the root node
5. Encode each character (using either 0 or 1 for left forks in the tree etc.)

6. Write out your codeword table

In summary, the basic idea is that you take the two nodes with the lowest frequency and combine them with an internal-parent-node. The new parent-node takes the combined frequencies of its two sub-children as its frequency and is put back with all of the other nodes. This process is repeated until there is only one node left, which is the root-node. You then create the codes for each character by tracing a path from root to leaf node (aggregating the bits along the way).

Task 2 - Huffman Implementation

For this task you need to implement the compress and decompress methods for Huffman coding. There are several ways of doing this and how you do it is up to you. As long as it implements the methods and gets the result.

Your solution must work like the utility classes provided in the the startup files. E.g

```
java Huffman - < abra.txt > abra_compressed.bin
```

You will need to compile your code to class files so that they can be run using the `java` command.

For example

```
javac <you_file.java>
```

Task 3 - Analysis and Comparison to Run Length Encoding

Calculate the time to compress and compression ratios for each of the provided files using Huffman Compression. Include the results in a summary table and upload the resulting compressed files as part of your submission. You can use the tools provided to calculate the compression ratios and time taken to compress them.

Calculate the time to compress and compression ratios for each of the provided files using Huffman Compression. Include the results in a summary table and upload the resulting compressed files as part of your submission. You can use the tools provided to calculate the compression ratios and time taken to compress them.

Step 1: Calculate the compression ratio achieved for each file. Also, report the time to compress and decompress each file.

Step 2: Decompress the files you compressed
 Take the files you have just compressed and decompress them. Report the final bits of the decompressed files and the time taken to decompress each file.

Step 3: Analysis of your results
Assess the results of the above. What happens if you try to compress one of the already compressed files? Why do you think this occurs?

Step 4: Use the provided RunLengthEncoding file to compress the bitmap file q32x48.bin. Do the same with your Huffman algorithm. Compare your results. What reason can you give for the difference
in compression rates?