Algorithmics I – Assessed Exercise

Status and Implementation Reports

Robert Thomson
2314819 here

November 15, 2019

Status report

I believe that my programs are working correctly. This is due to that the original file size is stated, the reduced file size is stated and is less than the original file size. The compression ratio has then been stated, and when manually calculated from the file sizes outputted it produces the same result. These results also align with the sample results supplied when the "small.txt" file is applied to the algorithms. The algorithms also indicate the time taken to compute the compression ratio. The times are as expected and gradually increase as the file size increases.

Implementation report

Compression ration for the Huffman algorithm

The Huffman algorithm begins by using a file reader to to get access to the file data. A character array list is then initialised to store all the characters in the file. A character hash set is then created to store all the unique characters in the file. An Integer Priority Queue is then initialised to store the different frequencies of each character. The algorithm then iterates through the unique characters. For each unique character, it iterates through the list of all characters and counts the number of times it appears. It then adds the number of frequencies to the Priority Queue. The new file size is then initialised to 0 bits.

It then loops through the Priority Queue merging two of the last frequencies and adding that value to the compressed file size and then inputs that value back into the Priority queue. It does this until there is only 1 entry in the Priority Queue.

The original file size is then calculate in bits by taking the size of the array of all the characters (the size in bytes) and multiplying it by 8. The compressed file size has been calculated and the compression ratio if calculated by taking the new file size and dividing it by the old file size.

Compression ration for the LZW algorithm

The LZW algorithm begins by using a file reader to to get access to the file data. A character array list is then initialised to store all the characters in the file. A Trie is then is then initialised. It is filled with 128 ascii values by iterating 128 times and calling the insert function of the trie on the characterized value of the iteration number. Next it initialises empty characters that will be used as the previous and next character codes, along with dictionary size of the trie(set at 128 for thee 128 ascii characters it starts with), the Code length(set at 8) and initialises the compressed file size value(0).

It then iterates through the characters in the array list. At each iteration it sets the next code to equal the previous code concatenated to the character of the current iteration. It then checks if this new code is in the trie. If it is not, the new code is inserted into the trie and the file size is increased by the current code word length. It then checks if the word size needs to be increased by comparing its square to the dictionary size. In either case, the current code is then set to be the previous code in preparation for the next iteration. If the new code is in the trie, the previous code is set as the new code in preparation for the next iteration.

The algorithm then checks if the new code is not empty, if it is not, the new code is inserted into the trie and the compressed file size is incremented a final time by the code length. The original file size is then calculate in bits by taking the size of the array of all the characters (the size in bytes) and multiplying it by 8.The compressed file size has been calculated and the compression ratio if calculated by taking the new file size and dividing it by the old file size.

Empirical results

```
Input file small.txt Huffman algorithm

Original file length in bits = 46392
Compressed file length in bits = 26521
Compression ratio = 0.57167184
Elapsed time: 32 milliseconds

Input file medium.txt Huffman algorithm

Original file length in bits = 7096792
Compressed file length in bits = 4019468
Compression ratio = 0.5663782
Elapsed time: 360 milliseconds

Input file large.txt Huffman algorithm

Original file length in bits = 25632616
Compressed file length in bits = 14397675
Compression ratio = 0.56169355
Elapsed time: 963 milliseconds


Input file small.txt LZW algorithm

Original file length in bits = 46392
Compressed file length in bits = 24832
Compression ratio = 0.53526473
Elapsed time: 31 milliseconds

Input file medium.txt LZW algorithm

Original file length in bits = 7096792
Compressed file length in bits = 2620426
Compression ratio = 0.3692409
Elapsed time: 656 milliseconds
```

```
Input file large.txt LZW algorithm

Original file length in bits = 25632616
Compressed file length in bits = 9158986
Compression ratio = 0.35731766
Elapsed time: 2239 milliseconds
```