Alex Visbisky

Assignment 3

CS1501

LZW Technical Write-up

The first thing to do with this program was to read over the code and understand how the methods worked and what it all relied upon. I downloaded everything needed to implement the code and decided to stick with the TST implementation since it seemed to be simple and worked as is with the code given. Starting off on this program I spent far too long trying to get the byte-by-byte streaming input to work and in the end I was unable to implement it so I focused most of my efforts on getting the variable length codebook and resetting the codebook to work. A lot of the issues in the byte-by-byte implementation came in NullPointerErrors that I could not resolve. It would read in a character, append it to the symbol table and then once it went to look for the StringBuilder built from the characters it would return that error. To make sure that the program knows how the file is encoded I added the type of compression stored as a char to the front of the file after compression, n for “Do Nothing” and r for “Reset Mode”. Then when the file is being uncompressed the program reads in the first char to see what kind of compression was used to compress the file. To begin, getting the variable length codebook to work was not as complicated as I initially assumed. First off, I began by changing the values of R, number of input characters, L, number of codewords, and W, codeword width from final variables to regular variables and changed the value of W to 9 and L to 512 since that was what the specification called for. The algorithm works as practically the same as the authors with a few changes. So, I added conditionals that would evaluate to true if the current codebook was full, i.e. L being at its max value. If this evaluated to true it would also check if W was less than 16, meaning that the codebook was not already at its max length of 16 as defined in the specifications. If this was true, then W would be incremented by 1 and L would be multiplied by 2 since the math to calculate L is 2^W. This ensures that if the current size of the codebook is full and the codebook is not already at its maximum size the program would add a new character to the length of codewords and increase the size. The next step was to get resetting the codebook to work when the codebook was filled. It seemed easy enough to just add a conditional check in the algorithm that would test to see if W was equal to 16 and that the codebook was equal to the maximum value of L (65535). When this was equal to true it would then just create a new TST and set all of the values of W, L and code, R + 1, back to their original values. That worked as is, so I decided to work on the extra credit portion of the assignment. For the monitored compression aspect of the project I added a few variables to the program; compression, read, newR and oldR. Compression is a variable that continually increments by W and read increments by the length of what it’s reading for each codeword. The ratios are stored in newR and oldR, newR being the value of current read divided by compression and then we compare the total rate by dividing oldR by newR to ensure that the compression is still at a reasonable rate. If the rate is higher than 1 to 1 the codebook will reset back to the initial values of W and L. I couldn’t necessarily find a “sweet spot” for the ratio to monitor for. Most of the cases given the value would be the same for monitored compression or just simply resetting the codebook. In some cases, a kilobyte or two was saved but other than that it typically just resets when the codebook is too full.