Joshua Rodstein

1501 – Summer 2017

jor94@Pitt.edu

**Project 3**: LZW Compression Analysis

The goal of project 3 was to modify the provided LZW algorithm to allow variable size code words, which in the long run would (in most cases) provide a better compression ration and a faster run time. The challenge for this project was conceptualizing the syncing of reading byte in both the expand and compress portions of the algorithm. The actual compression algorithm was relatively straight forward once the slogging through understanding was over. What follows is observational analysis.

Compress Was a simple nested while loop structure that allowed for the appending of characters until the prefix length was different than that of the characters, which meant that the last character appending to the codeword had not been added to the symbol table as of yet. This allow for an incremental code word process that made it easy to expand the code word size once the maximum number of words for the bit width (2^ current bit width) had been reached.

The expand portion of this algorithm was a bi trickier in that it the syncing of when to add and when to increase had to be precise, otherwise files were corrupted very easily. This was especially evident in the jpgs. A corrupt jpg would either not open if the ID bits had been changed. A jpg might open but might not look identical to the original, either leaving out pixels altogether or showing differences in the colors. For archives, a file might compress and show a good ration, however upon expand the file would provide a message stating that it had been corrupted and thus the information had been lost. One can see the importance of developing sound and stable compression algorithms for instances when important information needs to be compressed and uncompressed.

**Compression Ratios:**

I will start with the worst compression ratios. It seems that with JPG and GIF files, the UNIX compression offered no compression at all. This gave a 1/1 ratio. LZW actually expanded the frosty.jpg at a ratio of .7/1, with LZWmod’s variable codewords providing a slight improvement over LZW at .75/1 but still obviously expanding the size. This is no doubt due to more entropy with image files.

Moderate compression ratios were shown for archives and text files. The texts.tar archive provided a 2.2 ratio with UNIX, 2.17 with LZW, and the same with LZWmod. Reducing the size of the file by more than 50%. This favorable ratio is due to high entropy in an archive containing only text files. Following that logic, text files gave moderate to goo compression.

Excellent Compression was seen with bmp (bit maps). This is due to VERY high entropy. The best compression ratio was with wacky.bmp. A 900kb file compressed to 4kb. For a 225/1 compression ratio. This file indeed had the highest entropy. I tested my .java files as well PRE comments and they seemed to stay about the same throughout the different compressions, giving about a 2/1 ration for most. If I had added comments and tested, I probably would have gained compression due to the presence of character patterns.