CEN 350 - Theory of Computation
Louis Alban Ziko
Assignment Report - Compression Theory

1 Abstract

This assignment had the topic of compression theory. A compression algorithm and an alphabet were randomly assigned and using those a program was to be written which would compress a file and then decompress it. The algorithm I was assigned is LZW and the alphabet was the protein alphabet.

2 Introduction

LZW algorithm is a lossless compression algorithm which uses reoccurring patterns to save data space. The alphabet is stored into a dictionary at the start of compression and new values are added to it from the input file until it reaches 4096 entries. The input file is turned into 12 bit integers which store the index of the dictionary entry. The same dictionary is reconstructed at decompression.

Since we only go through each character once during execution the time complexity, for both compression and decompression, is $\mathcal{O}(n)$ where n is the length of the input file in bytes.

3 Dataset

The dataset was downloaded from 'http://pizzachili.dcc.uchile.cl/texts.html' as suggested. I used a smaller sample file of about 380KB instead of the 50MB file found on the website so that the program didn't take too long to execute.

4 Experiment and Result

As mentioned before, this assignment required a program to be written which would compress and decompress an input file. To do this two programs were written:

1. One would compress a given file and output the compressed file
2. The other would decompress a given file and output the original file

Consult the README.md file to find how the program should be compiled and run.

4.1 Task 1

This task involved computing the tuples of the input file and counting them. The tuples in the case of LZW are just one value which is the index of the dictionary entry. The values in the output file are in bit form (12 bit each). To output the values in human readable format we can use the option '-t <filename>' where filename can also be stdout to output to the console. In the case of the 380KB file, the output was comprised of 156910 tuples which was found at the end of the tuples.txt file.

4.2 Task 2

This task involved compressing and decompressing the input file and checking whether any information was lost. To do this the file was first compressed using the following command:

```
bin/compress -o res/output.lzw res/test_file.txt
```

Then the output file was decompressed:

```
bin/decompress res/output.lzw -o res/output.txt
```

And finally, using the unix command cmp the files were checked for any changes:

```
cmp res/test_file.txt res/output.txt
```

The command stops and prints the current line and byte index when it finds a difference between the two input files. Since the command didn't output anything the two files are the same.

4.3 Task 3

This task involved changing the tuples to bit representation and calculate the compression ratio. The tuples are already in bit representation from the previous tasks so the compression ratio can be found. The size of the input file is 382314B and the size of the output file is 234880B. The compression ratio then is:

$$CompressionRatio = \frac{UncompressedSize}{CompressedSize} = \frac{382314B}{234880B} = 1.62767 \tag{1}$$

4.4 Task 4

This task involved doing the previous tasks with the input file reversed. To do this the option -r is added. This will the same algorithm however the input will be reversed.

In this case the program generated 155191 tuples which is different from the 156910 tuples which were generated before. The size of the output file now is 232304B. The compression ratio has changed also:

$$CompressionRatio = \frac{UncompressedSize}{CompressedSize} = \frac{382314B}{232304B} = 1.64575 \tag{2}$$

So in this case using the reverse of the input would be more efficient.