CS3302 Compression Practical

Sizhe Yuen

2016-10-07

1 Introduction

In this practical we were asked to implement the Adaptive Huffman algorithm to encode and decode files. My implementation was able to construct the code tree incrementally, swapping nodes when necessary to maintain the sibling property and use the code tree to both encode and decode text and image files.

Lorem Ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2 Design

2.1 Lookup table

In my `AHEncoder` class, I have the field:

```
private HashMap<Integer, Node> lookup;
```

This field is use to store a direct index to every Node in the Huffman tree. The integer is the integer representation of the symbol that was read from the file.

3 Outcome

3.1 Text files

3.2 Image files

For image files, I have found that the size of the output from the encoding depends on the image file type. For example, when I try to encode `.jpg` and `.png` files, the file size for the output from the encoding is larger than the original image itself. However, when encoding a `.bmp` image, the encoding works very well.

Original size	Encoded size	Compression	File extension
23538	23884	101%	`.png`
1523	1632	107%	`.jpg`
32886	5741	17.5%	`.bmp`