Java Programming assignment #4: Huffman Coding and Data compression

due by May 30th, 2019

Introduction

We developed encoding/decoding classes library. Codec (coder and decoder) is all about data presentation. An application of encoding /decoding of information is data compression. Data compression is reducing the size of data by changing the data presentation.

There are generally two kinds of compression methods: lossy compression and lossless compression. Lossy compression tries to reduce data size, although we lose some information. Most multimedia file formats (JPEG, avi, MPEG, mp3, mp4, mpg, etc.) optimizes the size using lossy data compression. Because adjacent pixel would have similar pixel value, so we can remove redundant information as much as possible.

On the other hands, lossless compression is data compression method that preserves the original information and only reduces the size of data. Gzip or rar are the compressed file formats that we use today. The compressed data file should be decompressed, identically reversed back as before.

Huffman coding

One of key algorithms that reduces the size of file is using Huffman encoding. Huffman coding is variable-length data encoding scheme. According to the frequency in the text, we can allocate different length symbols, instead of having the fixed-sized same length symbols. For example, we map more frequently used code word with shorter length symbol. Some less frequently used symbol can have longer length. However, the average data size could be smaller because shorter symbols appear more frequently.

Huffman coding is based upon the Huffman tree. According to the frequency of code word, the Huffman tree guides you to allocate symbols. To construct a Huffman tree, first, you need to sort the code word in frequency order. You need to keep the frequency along with the code word. Second, find the least frequent two code words, and make a parent node. Third, The parent node should have the frequency of the sum of children. Now that the code word for children is removed. Then, repeat from Second and Third operation, building up a tree structure until all the nodes are getting into the constructed tree. Once the tree is constructed, we can allocate symbols to code word. In the tree structure, the lower level children gets one more symbol. The left hand child gets 0, and the right hand child gets 1. Finally, we get Huffman table from the Huffman tree. The Huffman table has a map between the code word and symbols. Now, it is ready to encode/decode.

Your Goal

Very similarly to what we did in the encoding/decoding, change the code word into symbols. Note that the symbols are bits; it can be grouped into bytes unit, and saved in a file.

The goal of this homework is encoding/decoding data with Huffman coding.

Requirements to do

There are some options in implementation, you can use pre-built Huffman table, and you can build from scratch.

Req1. Design and implement Huffman encoding class:

a. Huffman

Req2. Make class hierarchy, along with your encoding library. Design a parent class or interface.

Req3. Test your class with the homework description text. Here are some statistics for the file. Compare the data size.

Optional Req4. Make a decoding method that gets you back the original text.

Optional Req5. Construct a Huffman Tree & Huffman Table from the original text.

We will build the Huffman tree in the class time, so that you can use it. (You can use it on your hand).

Seehwan Yoo