Project #3 (Huffman Coding)

CSC 172 (Data Structures and Algorithms), Fall 2017,
University of Rochester
Due Date: 11/21/2017 (11:59 pm)
Thanksgiving Gift: Submission allowed (without penalty) until 11/27/2017 (11:59 pm)

Objectives

This project covers a lot of topics, which includes but not limited to, priority queues, maps, and trees. Also you will learn how to read and write binary files. Most importantly, you are going to learn Huffman coding, a lossless compression algorithm.

Project Description

Huffman encoding is one of the most popular lossless compression algorithms that works fairly well on any text file. In fact, this algorithm can be applied to any file. It can reduce the storage required by a half or even more in some situations. At the end of the project, you will impress yourself implementing this excellent algorithm! This particular project can be used to even encrypt and decrypt files (provided you keep the frequency file hidden).

In this project, you have to write a program that allows the user to compress and decompress files using the standard Huffman algorithm for encoding and decoding.

Overview of the Program Structure

We will provide four Java files to start with:

BinaryIn.java

(source: https://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/BinaryIn.java.html)
This class provides methods for reading bits (one or many) from a binary input stream. The binary input stream can be produced from a filename.

The methods of interest:

```
public boolean readBoolean();
public char readChar();
```

BinaryOut.java

(source: https://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/BinaryOut.java.html)
This class provides methods for writing 1-bit boolean or 8-bit char to an output stream such as a file.

The methods of interest:

```
public void write(boolean x);
public void write(char x);
public void flush(); // Really important. Flushes the binary output
   stream, padding 0s if number of bits written so far is not a
   multiple of 8.
```

Huffman.java

`Huffman.java` contains Huffman interface which you must implement.
`You are not allowed to modify this file.`
This interface contains two methods:

```
public void encode(String inputFile,String outputFile,String freqFile);
public void decode(String inputFile,String outputFile,String freqFile);
```

HuffmanSubmit.java

This class implements Huffman interface. All modifications must be limited to this file. Feel free to add new methods and variables, and import any packages as required. The only requirement is implementing `encode()` and `decode()` methods correctly. We will test your submission on this two methods only.

source code

You can download the tar file from http://www.cs.rochester.edu/courses/172/fall2017/tar/proj3_upload.tar

Extract the tar file. The resulting directory should contain four java files and 2 sample files.

Your Task

HuffmanSubmit.java

This is the only file where you need to do any modification.
`Implement encode() and decode() methods.`

For `encode()` method:

- *inputFile*: The name of the file to be encoded
- *outputFile*: The name of the encoded file. This is the compressed file (produced after encoding)
- *freqFile*: The name of the frequency file. This file stores the frequency of each character present in the input file. Each row contains a 8 digit representation of the char and its frequency separated by ':'. A sample row may look like: `11001010:40` where '11001010' is the character which has appeared 40 times in the input file.

For `decode()` method:

- *inputFile*: The name of the already encoded file
- *outputFile*: The name of the decoded file you want to produce. This is the file to be written after decoding.
- *freqFile*: The name of the frequency file created while encoding. Use this file to construct the Huffman Tree and perform decoding.

You can test your code with the provided sample files, `alice30.txt` and `ur.jpg`.
For this project, you just need to submit the `HuffmanSubmit.java` file on Blackboard.

Grading Details

- Frequency file correctly generated – 30 pts
- `encode()` method performs correctly – 30 pts
- `decode()` method performs correctly – 40 pts
  (For getting any credit for decoding, your encoding must work correctly)

  You should thoroughly check your code for encoding and decoding before submission.

Acknowledgement

Thanks to Robert Sedgewick and Kevin Wayne for providing `BinaryIn.java` and `BinaryOut.java` files.

Tips and Suggestions

Note: We will update this section based on questions from the students.

- When writing the bit patterns to the encoded file, you do not write the ASCII characters '0' and '1' (That would rather increase the file size), instead the bits are written as true/false (0/1) using the write(boolean) function given by BinaryOut.java.
- Frequency file is a regualr text file. So, you really do not need to use BinaryOut for that
- If you prefer zip file over .tar file, you can download it from: http://www.cs.rochester.edu/courses/172/fall2017/zip/proj3_upload.zip
- You are allowed to submit only one file — HuffmanSubmit.java. So, you can create a Node class **inside** HuffmanSubmit class. (Refer: piazza post:cid=238)