Project 3

Huffman Coding

Submission Deadline: **6:00 am** on Saturday, 30th July, 2016

Project Description

Huffman encoding is one of the most popular lossless compression algorithm 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. Hopefully, you will impress yourself implementing this excellent algorithm! This particular project can be used to even encrypt and decrypt files those you don't want to share. 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 give 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 in bits (one or many) from a binary input stream. The binary input stream can be from a filename, which is what we will use in our project.

The methods of interest:

```java
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:

```java
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 those you need to implement:

```java
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 of your 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 adn decode methods. We will test your submission on this two methods only.

Source code

You can use the 'wget' command to get the project directory.

```
wget http://www.cse.buffalo.edu/~tamaltan/resources/CSE250/Summer16/Project3.tar
```

Extract the directory.

Run the sample code 'huffmanSample' to check the output expected.

```
./huffmanSample
```

The program should run on Timberlake. If the executable permission is not set, execute the command:

```
chmod u+x huffmanSample
```

to make the program executable.

test.cpp

A sample cpp file to show how bstream works.

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 file to be written after encoding. *freqFile* The name of the frequency file. For encoding, you need to write the frequency to this file. This file stores the freq of each character. 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 file. For decode() method:

- - *inputFile* The name of the already encoded file
  - *outputFile* The name of the decoded file. This is the file to be written after decoding.
  - *freqFile* 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, ur.jpg, etc.

For this project, you just need to submit the HuffmanSubmit.java file on Blackboard.

Grading Details

- Frequency file correctly generated – 30 pts
- File correctly encoded – 30 pts
- File correctly decoded – 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 the 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 writen as true/false (0/1) using the write(boolean) function given by BinaryOut.java.