CSci 242: Algorithms and Data Structures **Fall 2012**

Instructor: Dr. E. Kim **Date: November 19th (Mon.)**

Due: December 10th (Mon.) 3:00pm

Project: Huffman Coding

Huffman coding is a widely used and very effective technique for compressing data; savings of 20% to 90% are typical, depending on the characteristics of the data being compressed. Huffman code is a variable length code whose length depends on the frequencies of characters in a message. It is constructed by building a Huffman Tree based on the frequencies of characters. A binary bit code for each character is determined from the Huffman tree and used to encode a message. The Huffman tree is also used to decode an encoded message as it provides a way to determine which bit sequences translate back to a character.

Write a Java program which compresses the data of the given message using *Huffman code*, and then decompresses a compressed file in order to retrieve the original message.

In this project, your program has to do following tasks:

A. Construction of a Frequency Table of characters/symbols
B. Construction of Huffman Tree
C. Encoding of a message to binary codes.
D. Decoding of the encoded message.
E. Analysis

A. Construction of Frequency Table of characters/symbols:

(1) Create an input file **'input'** with the poem '*Desiderata*' written by Max Ehrmann in 1920's .

(2). Parse the text of input file, count the frequency of each character/symbol, generating the table of frequencies, and print this *Frequency_Table* into the output file named '**output'**.

NOTE: Every character is case-sensitive, and a space character(' ') and carriage-return character (i.e. linefeed) should be also distinguished.

B. Construction of Huffman Tree

(1) Using the *Frequency_Table* and a *priority queue* implemented by a *minimum heap*, construct your Huffman Tree.

(2) Store the binary codeword of each character generated from the above Huffman tree in the 'Huffman_Table', and print this *Huffman_Table* in the same output file '**output**'.

(3) Draw the above Huffman Tree using a word process or using a graphic software and prepare its image file, named **'HTree.docx'** or '**HTree.jpg**' , etc., depending a graphic tool.

C. Encoding of the message

(1) Encode the given message in the input file and store the encoded message in the output file '**encoded**'.
e.g.) 11100011010100011 00……….

(2) What is the size of the encoded message? i.e. the length of encoded message.

(3) Print all the results in the output file **'output'**:

a) Frequency Table of characters,
e.g.) Freq[A] = 30;
Freq[B] = 15; …. etc.
b) Huffman Table of characters with their codewords,
e.g.) HT [A] = 1110;
HT [B] = 010001; … etc.
c) The *size* of the above encoded message: i.e. the total number of bits as 0 and 1.
e.g.) 3157 bits.

D. Decoding:

Suppose that you've received the compressed message file '**encoded**'.

Now, you should decode the binary message in '**encoded**', restoring the original text message.

1. Read the encoded message from a file '**encoded**' ,
2. Parse it and decode the message, using your Huffman_Table.

Your decoded message will be put in the 3rd output file '**decoded**'.

If the decoded message in '**decoded**' is equal to the original message in '**input**', both of your encoding and decoding of the message are successful.

E. Printing the Result and Analysis

(1) Printing in **output** file:

Print all the results in the output file '**output**', including

a) Frequency_Table,
b) Huffman_Table,
c) the *size* of the encoded message by your Huffman code, and

- The Huffman_Table is printed in the following format in **output** file.

  character = its encoded codeword

  e.g.) A = 010011, i.e. the encoded codeword of 'a' is 010011.

- The 1st line of the file will contain the *carriage-return character*, followed by '=' and its encoding.

  Since the carriage-return character, when viewed, forces a line-feed.
  the 1st line of '**output**' file is blank, and the 2nd line starts with a '=', followed by the encoded codeword of the carriage-return character.

  Thus, it'll look like:

  e.g.) blank line

  -1011101: the encoding of 'carriage-return' is 1011101 .

- Every character in the message, including space, comma(,), period(.), semicolon(;) will be encoded.
- The upper case letters *should be distinguished* from their lower case letters in their encodings; 'G', 'A', 'S', 'I', 'E', 'K', 'B', 'N', 'T', 'M', 'Y', 'W'.
- Before printing each table, give a caption of the table to print: e.g.) Frequency_Table, or Huffman_Table.

(2) Comparison of the size of original file in ASCII code and the encoded file:

Suppose that the input message is encoded in ASCII code which is a 7 bit fixed binary code: refer to http://en.wikipedia.org/wiki/ASCII

What is the size of ASCII encoding of the message in **input**?

Compare the size of ASCII encoding of input file with the size of your Huffman encoding of input.

d) Write this comparison result as well as the above a) – d) in ‘**output’** file.

Submission:

1. Create a directory called ‘Huffman-YourLastName’: e.g.) Huffman-Kim
2. Put all of the files of Java source codes, compiled class, input file(input) and output files(output, encoded, decoded, HTree) under the directory of Huffman.
3. Create a README file in the same directory, which contains:

- the instruction of compilation/execution of your program
- the description of each class of your java program: e.g.) class Huffman { ….} -- NOT a compiled ‘*.class’ file.
- The description of method in each class; i.e. what each method performs.

4. Compress the directory ‘Huffman’ to its .zip file.
5. Upload your compressed file to the ‘Submission’ section in ez_LMS.