2016 Fall EE205

**Project #2**

**Huffman coding**

**Due date: 11. 27 (Sun), 11:59 PM**

**TA: Changjun Kim (**[**changjun0605@kaist.ac.kr**](mailto:changjun0605@kaist.ac.kr)**)**

**Make sure understand clearly, what you do in project#2 by carefully reading this assignment. If you have anything unclear, ask your questions on the klms Lecture Q&A Board. Other private questions can be forwarded to TAs with CC to me (**[**rhee.jk@kaist.ac.kr**](mailto:rhee.jk@kaist.ac.kr)**).**

**Ⅰ. Introduction**

In this project, you are expected to compress and uncompress plain texts using Huffman code. This project will guide you to be acquainted with listed tree operations.

**Ⅱ. Goal**

Your program has to take input from a file. There are some sentences to process. You have to count the number of whole letters in the file until you reach the EOF (End of File). With the number of letters, you have to create the tree to generate the Huffman code  
(<https://en.wikipedia.org/wiki/Huffman_coding>) for the sentence. Here is an example:

Data structure aceaceaue

At first you have to count each letter.

' ' 2

D 1

a 5

c 3

e 4

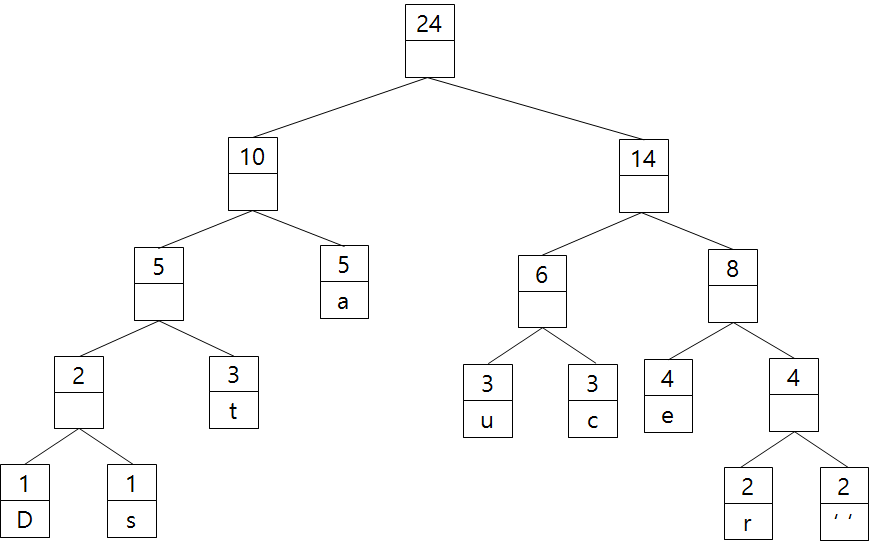
r 2

s 1

t 3

u 3

By using the statistics, you have to create the tree. One of the tree will have following structure. The shape of tree might be different base on your aligning algorithm. However, here’s one rule that summation of left child must be smaller than right child, according to the Huffman coding.



So each letter has the code as follows.

' ' 1111

D 0000

a 01

c 101

e 110

r 1110

s 0001

t 001

u 100

Then the sentence  
  
Data structure aceaceaue  
  
will be 0000010010111110001001111010010100110011101101111011011100110111001100110

Your program should run in the format of

> prj3 inputfilename *command*

The result should appear as text on the command screen via “stdio”.

**Ⅲ. Ground rules**

Your program must be written in C language and runnable on the provided Linux machines. You are not allowed to use any generics of C. In this project, you can use only the following header files**.**

*stdio.h, stdlib.h, string.h stdbool.h, stddef.h, stdint.h, stdmath.h proj2.h*

**IV. Queries on Database**

1. > **proj2** *inputfile* **letter**  
     
   This query shows the <letter in ASCII code>, <the number of letters they show> and <code of the letter> that appear in the input file in the ascending order. For example of the previous input data,   
     
   $ ./proj2 input.txt letter  
   32 2 1111

68 1 0000

97 5 01

99 3 101

101 4 110

114 2 1110

115 1 0001

116 3 001

117 3 100

1. > **proj2** *inputfile* **tree**  
     
   The query reports the tree you created in in-order. You should show the <summation value of the node, left child node and right child node> and <letter> if it exists. For the previous input data,  
     
   $ ./proj2 input.txt tree  
   1 0 0 68

2 1 1

1 0 0 115

4 2 2

3 0 0 116

9 4 5

5 0 0 97

23 9 14

3 0 0 117

6 3 3

3 0 0 99

14 6 8

4 0 0 101

8 4 4

2 0 0 114

4 2 2

2 0 0 32

1. > **proj2** *inputfile* **code**  
   The query print out the <code for each letter>, <whole sentence in code> and the <ratio of code and original sentence>. You consider one letter is 7 bits long (ASCII) in the original source data. For the previous input data for a source data,  
     
   $ ./proj2 input.txt code

32 1111

68 0000

97 01

99 101

101 110

114 1110

115 0001

116 001

117 100  
0000010010111110001001111010010100110011101101111011011100110111001100110

Ratio: 0.440

1. **proj2** *inputfile* **decode**

This query decodes the data into original sentence. You can use the result of the requirement 3, and the result must be same as original sentence. Before decoding, you must construct the code tree.  
  
$ ./proj2 input.txt decode  
Data sructure aceaceaue

For the given input.txt,

32 1111

68 0000

97 01

99 101

101 110

114 1110

115 0001

116 001

117 100  
0000010010111110001001111010010100110011101101111011011100110111001100110

#### V. input file

An *inputfile is a plain text ending with* <EOF>.

**VI.** **output\_file**

An *output* will be should be created by stdout redirection. Output file conditions will be provided by TAs shortly.

**VII. Delay penalty**

- 80% within 24 hours

- 60% within 48 hours

- 0% after 48 hours

**VIII. Extra penalty**

There will be penalty if you don’t

- use heap when you pick the minimum node

- reconstruct the tree when you decode