Huffman Coding

Date assigned: Friday, November 13, 2015
Date due: Friday, December 4, 2015
Points: 40

Conventionally, a character set is defined by associating a different, fixed bit pattern with each character. For example, ASCII characters are encoded as bit strings of length 7; however, one can compress the number of bits required for a sequence of characters by associating bit strings of shorter length with frequently occurring characters and longer strings with rarely occurring characters. This can reduce the space required for character strings and shorten the time required to transmit character strings serially. Such a coding scheme is called a "Huffman code".

This method depends on advanced knowledge of the relative frequencies of occurrence of the characters, which may appear. As an example, suppose the text consists of only decimal digits, and the probabilities of occurrence are:

digit	0	1	2	3	4	5	6	7	8	9
prob	.20	.25	.15	.08	.07	.06	.05	.05	.05	.04

The first step in constructing the code is to write down each character with its associated probability. Think of this as a list of single-node binary trees (roots only). Next, find the two roots with the smallest frequencies, delete them from the list, and add to the list a 3-node tree with the original nodes as leaves and with a root containing the sum of the probabilities of the left and right child nodes.

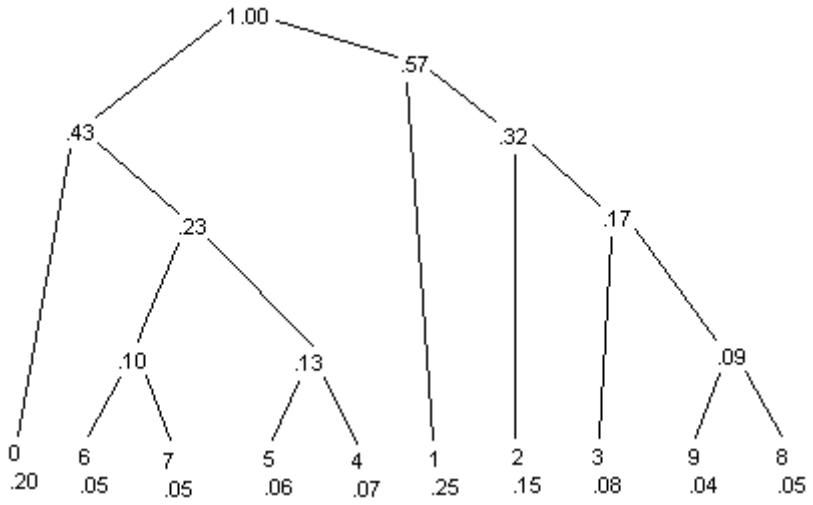
Continue this process until there is only one tree on the list containing the interior nodes which reflect the sum of the probabilities in the left and right subtrees, and leaf nodes containing the original character set.

The Huffman tree is used for both encoding and decoding text. You are to write a C program that will:

1. Build the Huffman tree out of the probability table on page 3.
2. Print out the Huffman tree on its side showing both the letters and weights. Interior nodes will have only weights. Make sure your spacing is consistent.
3. Decode the message in the file 'huffman.txt' which will be placed in the CS300Public folder on zeus next week. Print the decoded text.

4. Encode your winter break plans in a file myplans.txt. Where are you going and what are you planning on doing? That is, a good day would be what? ☺☺

To encode text, one needs a table of characters and associated bit strings. To derive the bit string for a character, follow the path from the root of the tree to the character, writing down 0 when a left branch is taken and 1 when a right branch is taken. Continuing the example, the Huffman code for digits is:



Digit	Huffman Code
0	00
1	10
2	110
3	1110
4	0111
5	0110
6	0100
7	0101
8	11111
9	11110

The shortest codes, 00 and 10, are assigned to the most frequently occurring characters, and the rarest characters have the longest bit patterns 11110 and 11111. Observe that characters 0, 1, and 2 account for 60% of the characters that occur, and all are 2 or 3 bits. 45% will require 2 bits, 15% need 3 bits, 31% need 4 bits, and 9% require 5 bits. This averages 3.04 bits per character, whereas 4 bits would be required for a fixed length code.

The decoding process proceeds this way. Examine the first bit to see if a character is defined by it (go left (a zero) or right (a one) one step in the Huffman tree and check if a leaf node is found). If not, use the next bit to see if these 2 bits define a character. If not, consider the third bit, etc. Eventually a character will be recognized. Decode the next character beginning with the next bit.

For a particular body of text consisting only of letters and blanks, the relative probabilities of occurrence are:

A	.0622	J	.0007	S	.0581
B	.0132	K	.0031	T	.0768
C	.0311	L	.0307	U	.0227
D	.0297	M	.0248	V	.0070
E	.1053	N	.0573	W	.0113
F	.0168	O	.0606	X	.0025
G	.0165	P	.0189	Y	.0107
H	.0363	Q	.0010	Z	.0005
I	.0614	R	.0587	space	.1821

Note1: There will be one unique Huffman tree if you enforce the following convention: The root with the smaller probability becomes the left subtree, and the root with the larger probability becomes the right subtree.

Note2: If a newline character is read from the file, begin printing on the next line.

You will need to create at least two additional projects (modules) for this assignment: BinaryTree and Huffman. You are to have a Makefile at the same level as these two projects which creates a driver called huffman that I will use to run your program in the following three ways:

1. **`huffman -p`** simply prints the huffman tree as described above
2. **`huffman -d`** `datafile` decodes the datafile displaying the decoded message
3. **`huffman -e`** `datafile` encodes the datafile displaying the encoded message

IMPORTANT

1. Submit a file called **cs300_7_PUNetID.tar.gz** by 9:15am on the due date. This file must include the modules **Huffman, BinaryTree** as well as any other modules your program needs to build and run successfully. Each module's Makefile must have a target called **valgrind** such that typing in **make valgrind** executes the valgrind command.
2. Data file input is to be from the command line.

3. Turn in a color, double sided, stapled packet of your huffman and binary tree code by the same deadline in 1. Also, turn in a printout of the huffman tree on its side with edges drawn in. The packet must be in the following order:
    - huffman (.h then .c if you have both, otherwise just .c)
    - binary tree (.h then .c if you have both)
    - Any extra .h/.c pairs you have. (do not include any code from previous projects)
    - huffman Makefile
    - binary tree Makefile
    - huffman tree on its side

You can make changes to any of your existing code including any of the header files from me.