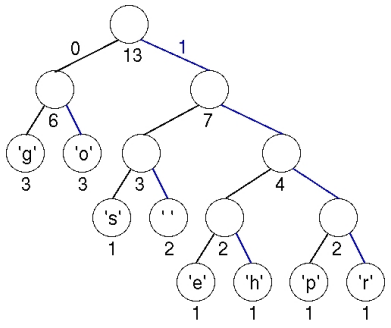
ECE 264 – Spring 2013
Huffman Coding (Part II)

(Derived from an assignment by Professor Vijay Raghunathan)

Huffman coding is a widely used compression algorithm used in JPEG compression as well as in MP3 audio compression. This document explains how to decompress data that has been compressed using the technique. Please see the respective SPEC files for PA10 and PA11 for concrete details on what to do. Also refer to the pa10.pdf file for background material on Huffman coding and the Huffman header we used in the previous assignment

1 Recap: Binary Coding Trees

Using a binary tree for coding, all characters are stored at the leaves of a tree. A left-edge is numbered 0 and a right-edge is numbered 1. The code for any character/leaf node is obtained by following the root-to-leaf path and concatenating the 0's and 1's. The specific structure of the tree determines the coding of any leaf node using the 0/1 edge convention described. As an example, the tree below yields the coding table following it.



Character	Binary code
' '	101
'e'	1100
'g'	00
'h'	1101
'o'	01
'p'	1110
'r'	1111
's'	100

Using this coding, "go go gophers" is encoded (again, spaces would not appear in the bit-stream) as: `00 01 101 00 01 101 00 01 1110 1101 1100 1111 100`. This is a total of 37 bits, two bits fewer than the improved encoding in which each of the 8 characters has a 3-bit encoding! The bits are saved by coding frequently occurring characters like 'g' and 'o' with fewer bits (here two bits) than characters that occur less frequently like 'p', 'h', 'e', and 'r'.

Recall that to decode a given stream that has been coded by the given tree, start at the root of the tree, and follow a left-branch if the next bit in the stream is a 0, and a right branch if the next bit in the stream is a 1. When you reach a leaf, write the character stored at the leaf, and start again at the top of the tree. The bit stream `100111011011100111111100` yields right-left-left to the letter 's', followed (starting again at the root) with right-right-right-left to the letter 'p', followed by right-right-left-right to the letter 'h'. Continuing thus yields a decoded string "sphere."

2 Huffman Decompression

Given a compressed file, which contains the header information, followed by a bit stream corresponding to the encoding of the original file, the decompression program should perform the following tasks:

1. Build a Huffman coding tree based on the header information. Use the same approach as in PA11. (Note that unlike previous years, we will let the header continue to be in ASCII format and not bit-packed.)

2. Read the number of characters stored in the compressed file from the header information. Remember to consume the newline as well.

3. Read bit-by-bit from the compressed file (starting at the location after the header information). Use the read bit to traverse the Huffman coding tree

(0 for left and 1 for right), starting from the root node. When the program reaches a leaf node, print the corresponding ASCII character to the output file. Start anew from the root node of the Huffman coding tree when the next bit is read. The program terminates when the number of characters decoded matches the number of characters stored in the header information.

A challenge is in the reading of individual bits from the compressed file. Again, reading from a file always occurs at the byte-level. In the decompression program, you have to read Huffman codes that potentially straddles two bytes in the compressed file when you are decoding the compressed file. This will probably require your decoding function to have access to the file pointer so that it can read another byte when necessary. When would that be necessary? When you have consumed the current byte.

Also note that our convention is to store our Huffman codes starting from bit position 7 and then in descending bit order (bit 7, 6, 5, 4, 3, 2, 1, 0 and then start over from 7 on the next byte read).