Huffman Exercise

Exercise: Write a Huffman compressor and decompressor.

Huffman is a compression algorithm. This particular algorithm is an entropy encoder that stores a series of values (e.g. bytes from a file), in the least amount of bits, depending on frequency. This is opposed to dictionary compression, which compresses by finding repeating strings of bytes in a file. Both forms of compression complement each other; typical compression programs like WinZip or WinRAR first do dictionary compression followed by entropy encoding. So, an entropy encoding like Huffman (an alternative is arithmetic coding), purely is used for representing data in the least amount of bits once the repetition has already been taken out of the data.

Entropy encoding assigns sequences of bits to values, depending on how frequently that value occurs. So, for English text, it would want to use less bits for "e" than for "q", because that way the overall text can be represented in the least amount of bits. When decompressing, you can find the original values from the bits. To make this possible, the bits assigned must be unique. How can we do this?

For compression, go through these steps:

1. Loading a file into memory (be sure to use binary mode).
2. Loop through all bytes of the file, counting the frequency of occurrence of each byte. You might want to print out the resulting table of frequencies to get an impression of what you're doing.
3. Next, to assign a bit sequence to each character. The easiest way to guarantee they are unique and optimal is by constructing a binary tree. You'll need both leaf nodes (that contain a character) and nodes that have two child nodes (hence, binary).
   - Create a leaf node out of each byte that has a frequency > 0, and put them in a list of nodes.
   - Find in your list of nodes the two nodes that have the lowest frequency, and take them out of the list.
   - Create a new node (and insert it into list) and make these two nodes its children.
   - Make the frequency of this new node the sum of its children.
   - Repeat until the list contains just 1 node, the root of the tree.
4. Once you have the tree, recurse through the tree in depth-first order, and collect the list of bits as a path from the root (where 0 means left branch, and 1 means right). Whenever you arrive at a leaf, the list is the set of bits for that value. Store the bits. Note that the number of bits is variable; it can easily be more than 8 for the less frequent values.
5. Now loop through all the bytes in the file again. For each byte, write out the corresponding bits to a file. To write bits to a file, you'll need to accumulate them in bytes at a time… this requires the use of bit operators.
6. To be able to decompress the file, you'll also need to store the tree in the file. The easiest way to write a tree is to write it depth-first, post order, that way you can reconstruct it using a stack.

Huffman Exercise

Decompression is a lot simpler:

1. Reconstruct the tree from the file. If you read a leaf, put it on a stack, if you read a node, take 2 children from the stack and put the node back on. This should leave you just the root on the stack when you are done.
2. To decompress, read bits from the remainder of the file, and use each bit to walk down the tree (0 means go left, 1 means go right). Whenever you come to a leaf, write the value of that leaf to the output file, and start again at the root of the tree.

The challenge, of course, is to be able to run the two algorithms back to back, and ensure that the final output file is identical to the original input. Try this with a number of different files. If the files ever end up different, there is a bug in your code. The exercise is not done until you have fixed these bugs.

If you need more detailed examples, searching "Huffman coding" on Google will turn up several.