HOMEWORK 2
Deadline: Thursday November 14th, midnight

Please upload all your material to F: drive under the homeworks folder in the form yourname_lastname_hw2.zip (e.g. emine_yilmaz_hw2.zip).

Problem 1. (40 points) Huffman coding.
Consider the following text fragment:

a man, a plan, a canal, panama

Note: This is a famous palindromic sentence --- the letter sequence is the same forwards or backwards.

This text fragment contains 30 total characters (including the six internal space characters and the three commas), and it includes eight unique characters. Using the frequency counts associated with the unique characters in this text fragment, compute an empirical probability distribution over the eight characters which appear in the text fragment. In other words, for each character, compute the number of occurences of that character divided by the total number of characters in the text fragment.
Using the above distribution, compute a Huffman code for the above characters. Show your Huffman tree and give the codes associated with each character. (You may break ties arbitrarily when computing the Huffman code.)

How many total bits would be required to encode the text fragment with your Huffman code? How many bits per character on average are required?

How many bits per character are required to encode the text message naively, i.e., using the shortest code with an equal number of bits per character? (This is a trivial upper bound on the coding length.)

Compute the entropy of the distribution associated with the frequency counts. (This is the information-theoretic lower bound on coding length.)

How does your Huffman code compare to the upper and lower bounds you computed above? What can you conclude about your Huffman code? Can it be improved? (Hint: Multiply 30 by the entropy; what value do you obtain?.)

Problem 2. (30 points) Lempel-Ziv: Encoding.
Consider your three initials, lower case. (If you do not have a middle name, use the first letter from your first and the first two letters from your last name.) For my name, Emine Yilmaz, the three lower case initials are "eyi".

Consult an ASCII table and convert your three initials to their associated bits. At eights bits per ASCII code, you should obtain 24 total bits. For example, given my initials, you would obtain:
011001010111100101101001

Parse these 24 bits using the Lempel-Ziv parsing algorithm described in class. For each unique substring found, generate a pair corresponding to the prefix back-reference (i.e., how many substrings back is the prefix?) and suffix bit.

Now encode these pairs as described in class. To encode the prefix back-references, use ceil(lg(n)) bits to encode the back-reference corresponding to substring n. In other words, use zero bits to encode the first back reference (i.e., don't encode it---you don't need it), use one bit to encode the second back-reference, use two bits to encode the third and fourth back-references, use three bits to encode the fifth through eigth back references, and so on.

(Note that the encoded length is longer than the unencoded length. Is this a failure of the Lempel-Ziv compression algorithm? No. Lempel-Ziv is guaranteed to compress at nearly the entropy rate for sufficiently long strings; our example is simply too short. Try running gzip on a file containing only three characters, and see what happens.)

Problem 3. (30 points) Lempel-Ziv: Decoding.
Consider the string:

001100100011110110111011100111011100110000110101010001101011

This bit string is the result of encoding a piece of text using the Lempel-Ziv algorithm described above (and in class).
Parse this bit string to obtain blocks corresponding to (back-reference, suffix bit) pairs. Decode the back-references to obtain these pairs. For example, given the string

00101011101001001011101000001101001

we would obtain

0,01,010,111,0100,1001,0111,0100,00011,01001

and

(0,0), (0,1), (1,0), (3,1), (2,0), (4,1), (3,1), (2,0), (1,1), (4,1)

Reconstruct the original bit string using these (back-reference, suffix bit) pairs. For the example given above, we would obtain:

011001100110111101101111

Consult an ASCII table to convert your bit string to a sequence of characters. What string did you obtain? For the above example, we get "foo".