Algorithms in the Real World (15-853), Fall 05
Assignment #1

Complete 80 points.
Due October 3.
You are not permitted to look at solutions of previous year assignments. Unfortunately even if I remove them from the web page, Google has them cached.

Problem 1: Conditional Probabilities (10pt)
Given the following conditional probabilities for a two state Markov Chain what factor would one save by using the conditional entropy instead of the unconditional entropy?

$$\begin{array}{cc} p(w|w) = .95 & p(b|w) = .05 \\ p(w|b) = .2 & p(b|b) = .8 \end{array}$$

Problem 2: Uniquely Decipherable Codes (10pt)
Devise a uniquely decipherable code that is not a prefix code.

Problem 3: Arithmetic Codes (10pt)
Given the following probability model:

Letter	$p(a_i)$	$f(a_i)$
a	.1	0
b	.2	.1
c	.7	.3

Decode the 4 letter message given by `00011011010` assuming it was coded using arithmetic coding. Why is this message longer than if we simply had used a fixed-length code of 2 bits per letter, even though the entropy of the set $\{.1, .2, .7\}$ is just a little more than 1 bit per letter. Note: once you figure out how to do the decoding, it should not take more than five minutes on a calculator or scripting language.

Problem 4: Decoding Prefix Codes (20pt)
Being able to quickly decode prefix codes is extremely important in many applications.
Assume you have a machine with word length w (e.g. 32 bits). Assume you are give some prefix code for a message set, and the longest codeword is $w/2$ bits (or less). Assume that a sequence of codes is stored in memory broken into words (i.e. the first w bits are in the first word, etc.).
The naive way to decode is to use a binary tree and take constant time per bit by traversing the tree. Describe how to decode each codeword in constant time (independently of w). Hint, you can use $O(2^{w/2})$ preprocessing time, and the same amount of memory.
Please don't use more than half a page to describe the method

Problem 5: Bounds on Integer Arithmetic Codes (20pt)

Assume you are given a set of n possible messages with fixed integer counts $c_1, c_2, \ldots, c_n$ such that the probability p_i of the i^{th} message is c_i/C, where $C = \sum_{i=1}^{n} c_i$. Assume this probability distribution is static when sending a sequence of messages from this set.
As discussed in class for real Arithmetic Codes, the number of bits required to send a sequence of m messages is bounded by $\sum_{j=1}^{m} s_j + 2$. Now assume we are using integer arithmetic codes using k bits of precisions (*i.e.*, integers in the range $0, \ldots, 2^k - 1$). Specify a bound on the number of bits required to send a sequence of message as a function of C, k, and the s_i. This should be as tight as you can make it. You can assume $4C < 2^k$.

Problem 6: Bounds on Prefix Codes (20pt)

A. Prove the first part of the Kraft-McMillan inequality for Prefix Codes. In particular show that for any prefix code C,

$$\sum_{(s,w) \in C} 2^{-l(w)} \leq 1.$$

B. Prove that if you have $n = 2^k$ codewords in a prefix code and that if one of them is shorter than k bits, then at least two must be longer than k bits.

Problem 7: PPM, LPZ and BW (20pt)

The string `bcabccabc` is encoded using PPM. The (partial) dictionary constructed during encoding is given below. For the following questions, assume that escape count is given by the number of different characters for each context, and exclusion is *not* used, unless specified. Assume that the alphabet has 26 characters, and use $k = 2$.

(a) Fill in the empty spaces in the dictionary below.

Context	Counts	Context	Counts	Context	Counts
empty	$a = 2$	a	$b = 2$	ab	$c = 2$
	$b = 3$				
	$c = 4$	b			
		c			

Figure 1: Dictionary

(b) Suppose the next letter in the string is `b`. Compute the number of bits required to encode `b`, and also list the changes made to the dictionary. (You do not have to compute the exact number of bits, simply write it as an expression containing logs). The answer need not be an integer.

(c) Now assume that exclusion is used. Recompute the number of bits required to encode the character `b`.

(d) Encode the above string `bcabccabc` using LZ77, with an unbounded lookahead buffer, and a window of size 4.

(e) Encode the above string using Burrows-Wheeler. Just show the sequence of characters after the BW transform (don't bother compressing using move to front).