The UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

Comp 411 Computer Organization
Spring 2012

Problem Set #1
Issued Wednesday, 1/18/12; Due Wednesday, 2/1/12

Homework Information: Some of the problems are probably too long to be done the night before the due date, so plan accordingly. Late homework will not be accepted. Feel free to get help from others, but the work you hand in should be your own.

Problem 1. "This is a Computer Science class?"

Computers are not the only systems that manipulate and store information. During the past 50 years the field of biology has experienced a revolution that was prompted by the discovery of the structure DNA molecule and its role in genetics. The information content of every biological system is encoded in its DNA sequence as a series of four nucleic acids— cytosine, guanine, adenine, and thymine, which I will abbreviate as C, G, A, and T. Genes are merely subsequences of DNA that encode assembly instructions for a particular protein. Proteins are the primary molecular machines that perform the tasks of life. Within genes, trios of nucleic acids, called *codons*, are used to encode one of 20 amino acids, which are the building blocks of proteins. This coding is described in the following table:

		2nd base			
		T	**C**	**A**	**G**
1st base	**T**	TTT Phenylalanine TTC Phenylalanine TTA Leucine TTG Leucine	TCT Serine TCC Serine TCA Serine TCG Serine	TAT Tyrosine TAC Tyrosine TAA *Stop* TAG *Stop*	TGT Cysteine TGC Cysteine TGA *Stop* TGG Tryptophan
	C	CTT Leucine CTC Leucine CTA Leucine CTG Leucine	CCT Proline CCC Proline CCA Proline CCG Proline	CAT Histidine CAC Histidine CAA Glutamine CAG Glutamine	CGT Arginine CGC Arginine CGA Arginine CGG Arginine
	A	ATT Isoleucine ATC Isoleucine ATA Isoleucine ATG Methionine, *Start*	ACT Threonine ACC Threonine ACA Threonine ACG Threonine	AAT Asparagine AAC Asparagine AAA Lysine AAG Lysine	AGT Serine AGC Serine AGA Arginine AGG Arginine
	G	GTT Valine GTC Valine GTA Valine GTG Valine	GCT Alanine GCC Alanine GCA Alanine GCG Alanine	GAT Aspartic acid GAC Aspartic acid GAA Glutamic acid GAG Glutamic acid	GGT Glycine GGC Glycine GGA Glycine GGG Glycine

Notice that every protein is initiated with the same amino acid (Methionine) and that there are three codes reserved for terminating the protein sequence.

(A) Assuming that all nucleic acids are equally likely, how many bits of information are represented in a 3-nucleic acid sequence? What are the minimal number of bits necessary to encode one of 20 possible amino acids and a stop code? Do these two numbers agree? If not, speculate on a reason why they might differ.

(B) Assume that all codons occur with equal frequency. If you are told that a given codon represents some amino acid, how many bits of information are conveyed? Likewise, how many bits are conveyed if you find out that some codon encodes the stop message? How many bits are conveyed by discovering that a codon encodes Serine? (Look over the table entries carefully!)

(C) Suppose you are told that a given unknown codon with at least one T nucleotide, how many bits of information were you given? Suppose you find out that the same unknown codon encodes a stop code. How many additional bits does this new information add?

All 20 amino acids share a common chemical substructure, but are distinguished by a unique side chain. Amino acids are usually classified by the properties of their side chains. For example the side chains of Lysine, Histidine, and Arginine are chemical bases.

(D) How many bits are conveyed by finding out that a given unknown *amino acid* is a base? How many bits are conveyed by finding out that a given *codon* represents an amino acid that has a basic side chain? In this second case, how many *additional* bits are conveyed by resolving that the given codon represents Lysine?

Mutations occur in DNA when one nucleic acid is randomly substituted for another. DNA substitution mutations are of two types. Transitions are interchanges of *purines* (A and G) or of *pyrimdines* (C and T), which involve nucleotides of similar shape. Transversions are interchanges between purines and pyrimdines, which involve exchanging dissimilar chemical structures. Although there are twice as many possible transversions, transition mutations occur at a significantly higher frequency than transversions. It is often the case that a transition mutation in the *third* position of a codon does not change the amino acid that is coded for.

(E) When told that a given transition mutation causes a modification in the resulting protein but does not change its overall chain length how many bits are conveyed by this information? (This one is a little tricky)

Next we consider case where codon alternatives are *not* equally likely. In human genes we observe that Glycine coding codons appear with the following frequencies:

p(“GGG”) = 0.24
p(“GGA”) = 0.14
p(“GGT”) = 0.12
p(“GGC”) = 0.50

(F) Clearly in nature’s encoding scheme the prefix “GG” is used to encode for Glycine. How many bits of information are conveyed by the codon’s last nucleic acid?

(G) Suppose you are collecting information about every occurrence of a Glycine codon within the human genome, and you plan to keep track of each coding variation. There are four possibilities, which could be trivially encoded using 2 bits. However, what is the

entropy of this set of codes? On average, how many bits would be wasted by using the fixed length 2-bit coding scheme?

You might instead encode the choice of "GGC" with the bit string "0", the choice of "GGG" with the bit string "10", the choice of "GGA" with the bit string "110" and the choice of "GGT" with the bit string "111".

(H) If we record Glycine encoding codon sequences by concatenating these bit strings in left-to-right order, what sequence of choices is represented by the bit string "0011011101010"?

(I) What is the expected length of the bit string that encodes the results of making 1000 choices? What is the length in the worst case? How does this number compare with $1000*\log_2(4/1)$?

Problem 2. Huffman Coding

On an architectural dig in somewhere between Raleigh and Chapel Hill, researchers have discovered a deck of fossilized punch cards containing the source for a circa 1960 program called Seunti. The program was invented to promote sharing of the digitized Elvis tunes enjoyed by the genetic ancestors of modern programmers. Seunti was designed to run on the then-current *decimal* computers, whose unit of storage is the decimal digit rather than the bit. Each tune is encoded as a stream of ***equally-probable*** single digits (0 through 9) and transmitted to the receiver, where the stream is decoded to reproduce the scratchy audio signal.

(A) How much information, in bits, does each transmitted digit convey?

Looking closely at the source code, you notice that each digit *d* read by the receiver is processed by first computing the value *f(d),* as follows:

f(d) = the largest prime factor of *d+1, i.e.*
If d+1 = 1 then 1
Else if d+1 = 2, 4, or 8 then 2
Else if d+1 = 3, 6, or 9 then 3
Else if d+1 = 5 or 10 then 5
Else 7.

All further processing depends only upon the value *f(d)*; thus information is lost during the conversion from d to f(d) is not used to reproduce the tune. Having attended the first two weeks of Comp 411 lectures, you recognize that some of the information transmitted between Seunti users is unnecessary. For example, the digits 4 and 8 are transmitted as distinct symbols, despite the fact that their distinction is immediately lost by the above conversion, and hence not needed.

You begin thinking about writing an improved version of Seunti that works by transmitting only *f(d)* rather than each digit *d*. It occurs to you that this approach both reduces the alphabet of transmitted symbols from 10 to 5, and introduces asymmetries in the probabilities of each symbol transmitted (i.e., they are not equally likely).

(B) What is the amount of information conveyed an f(d) output of "1"? Of "3"? Of "5"?

(C) What is the *average* amount of information conveyed by an f(d) output?

You recall the brief mention of Huffman coding in lecture, and suspect that this coding technique could improve the efficiency of your improved Seunti system by eliminating some of the redundancy. Intrigued by your suspicion that this technique could be applied to Seunti transmissions, you search the web for information on Huffman coding. Huffman's insight was that a variable-length binary decoding tree can be constructed by a simple greedy algorithm as follows:

1. Begin with the set S of symbols to be encoded as binary strings, together with the probability P(x) for each symbol x. The probabilities sum to 1, and measure the frequencies with which each symbol appears in the input stream. In the example from lecture, the initial set S contains the four symbols and associated probabilities in the above table.

2. Repeat the following steps until there is only 1 symbol left in S:
    a. Choose the two members of S having *lowest* probabilities. Choose arbitrarily to resolve ties. In the example from lecture, 2 and 12 are the first nodes chosen.
    b. Remove the selected symbols from S, and create a new node of the decoding tree whose children (sub-nodes) are the symbols you've removed. Label the left branch with a "0", and the right branch with a "1". In the first iteration of the lecture example, the bottom-most internal node (leading to 2 and 12) would be created.
    c. Add to S a new symbol (e.g., "2-or-12" in our example) that represents this new node. Assign this new symbol a probability equal to the sum of the probabilities of the two nodes it replaces.

When S contains a single symbol, the decoding tree is complete. It contains the essential information necessary to specify how each of the original symbols is to be represented as a binary string.

(D) How many iterations of step 2 will it take to generate a decoding tree for a set of *n* symbols? [*HINT*: This is easy!].

(E) Create a Huffman decoding tree for the efficient coding of *f(d)* as variable-length binary strings. Present your results as in the example above, i.e. as a table listing probabilities and binary encodings of the five transmitted *f(d)* symbols (1, 2, 3, 5, and 7) as well as a binary decoding tree.

(F) How close does the coding you derived come to approximating the average information content of a digit as derived in part (C)?

Problem 3. "Some Assembly Required"

The conversion of a mnemonic instruction to its binary representation is called assembly. This tedious process is generally delegated to a computer program for a variety of reasons. The first is that it alleviates the need to keep track of all the various bit encodings for each type of instruction. A second reason is that frequently the precise encoding of an instruction cannot be determined in a single pass. This is particularly true when referencing labels. In the following exercises, you will get a taste of what the task of translating from assembly to machine language is like.

Give binary and hexadecimal encodings for the following instructions:

```
(A)         addi $21,$0,$5
(B)         ori   $t0,$t0,0xff
(C)         lui   $s0,0xdad
(D)         lw    $v0,53($gp)
(E)         sw    $sp,-4($sp)
(F)         sll   $t9,$a0,3
(G)   loop: bne   $s0,$s0,loop
```

(H) Suppose that you made a mistake encoding of the immediate field in part (G). Would this mistake effect the execution of your program? Explain how or why not.

Problem 4. "A True Memory Dump"

Bud Levile has hacked into an online map server. Of course, the server is a MIPS ISA compliant processor. He has managed to grab the contents of the memory locations that he believes holds the MIPS code responsible for computing the traveling distance between points, and would like you to help discover how the code works. The memory contents are shown in the table below:

Address	*Contents (in hexadecimal)*
0x100	0x3c1c1000
0x104	0x8f880100
0x108	0x20090001
0x10C	0x00005024
0x110	0x214a0001
0x114	0x01094022
0x118	0x21290002
0x11C	0x0109582a
0x120	0x1160fffc
0x124	0x00000000
0x128	0xaf8a0100
0x12C	0xaf880104

(A) Reconstruct the MIPS assembly code that corresponds to the memory dump given above. If the code sequence contains branches, be sure to indicate the destination of each branch.

(B) Describe the function that this code computes. Specifically, describe the relationship between and values read from memory to those stored. (Hint: Start by assuming a small value is read from memory, e.g. 8, 9, 10, etc.)

(C) Which instruction could be removed from the sequence without affecting the code's operation?

(D) Does the operation of this code fragment depend upon where in memory it is loaded? Explain why or why not.