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

Comp 411 Computer Organization
Spring 2007

Problem Set #1
Issued Tuesday, 1/16/2007; Due Tuesday, 1/23/2007

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. "Miss Information?"

In lecture we learned that information resolves uncertainty, and that information is measured in units of *bits*. In order to uniquely identify one of **N** equally likely alternatives, $\log_2 \mathbf{N}$ bits of information must be communicated. This is equivalent to saying that $\log_2 \mathbf{N}$ bits are needed to encode any particular choice. If we learn a fact that only narrows down a list of possible alternatives to a choice of **M** candidates ($\mathbf{M} < \mathbf{N}$), we say that we've received $\log_2(\mathbf{N}/\mathbf{M})$ bits of information. For example, if we start with 4 equally likely alternatives, we'll need 2 bits to provide a unique encoding for each of the 4 alternatives. If we learn a fact that narrows the number of alternatives down to 2, that fact has conveyed $\log_2(4/2) = 1$ bit of information.

(A) How many bits are required to uniquely encode all playing cards in a standard deck of 52? If you are told a card's suit, how many bits of information are conveyed? If you are told that the same card is a face card, how many *additional* bits are conveyed?

(B) How many total bits of information are in a single deck of cards? If all cards with values 2 to 6 inclusive are removed from the deck (all suits), how many total bits are required to encode the resulting smaller deck? In this smaller deck, how many bits are required to encode the card's suit? It's face value (7, 8, 9, 10, J, Q, K, or A)?

Consider the situation when the **N** alternatives are *not* equally likely. It still takes $\log_2 \mathbf{N}$ bits to encode any particular choice, but with clever encoding the *average* number of bits needed to encode a choice might be smaller. For example, suppose there were three alternatives
("A", "B", and "C") with the following probabilities of being chosen:

p("A") = 0.5
p("B") = 0.3
p("C") = 0.1
p("D") = 0.1

We might encode "A" with the string "0", "B" with "10", "C" with "110", and "D" with "111".

(C) If we record the results of making a sequence of choices by concatenating from left-to-right the bit strings that encode each choice, what sequence of choices is represented by the bit string "0100111110010"?

(D) 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 do these numbers compare with $1000*\log_2(4/1)$?

Intuitively it seems that we get less information when we learn of a likely choice and more information when we learn of an unlikely choice. This intuition is reflected in the following formula for the average number of bits of information (Entropy) received about a choice made from **N** alternatives when each alternative has a (possibly different) probability p_i:

$$\text{Bits of information} = -\sum_{i=1}^{N} p_i \log_2(p_i)$$

(E) How much information do you receive, on average, from a single flip of a crooked coin where $p_{tail} = 0.375$ and $p_{head} = 0.625$? Is this more or less than the amount of information you receive from the flip of a fair coin?

(F) A data stream is composed of a sequence of three characters, {*u*, *d*, *s*}. What probabilities would you assign to these to maximize the entropy? What probabilities would you assign to minimize the entropy? Assume that the probabilities of u, d, and s are p(*u*) = 0.7, p(*d*) = 0.2, and p(*s*) = 0.1. What is the entropy of this data stream? How close can you come to achieving this number of bits in practice? Suppose, that you assign a unique code to every 2-character combination, {*uu*, *ud*, *us*, *du*, *dd*, *ds*, *su*, *sd*, *ss*}. Devise a variable length code to for this data stream, and compute its entropy (you can assume all data streams are composed of an even number of characters). What is the entropy of this data stream? How close can you come to achieving this number of bits in practice?

Problem 2. Modular Arithmetic and 2's Complement Representation

Most computers choose a particular *word length* (measured in bits) for representing integers and provide hardware that performs operations on word-size operands. Many current generation processors have word lengths of 32 bits. Restricting the size of the operands and the result to a single word means that the arithmetic operations are actually performing arithmetic modulo 2^{32}.

(A) How many different values can be encoded in a 32-bit word?

Almost all modern computers use a 2's complement representation for integers since the 2's complement addition operation is the same for both positive and negative numbers. In 2's complement notation, one negates a number by complementing each bit in its representation (i.e., changing 0's to 1's and vice versa) and adding 1. By convention, we write 2's complement integers with the most-significant bit (MSB) on the left and the least-significant bit (LSB) on the right. Also by convention, if the MSB is 1, the number is negative; otherwise it's non-negative.

(B) Please use a 32-bit 2's complement representation to answer the following questions. What's the representation for 0? For the most positive integer that can be represented? For the most negative integer that can be represented? What are the decimal values for the most positive and most negative integers? What do you get if you negate the largest negative integer (given both the binary and decimal values)?

(C) Since writing a string of 32 bits gets tedious, it's often convenient to use hexadecimal notation where a single digit in the range 0—9 or A—F is used to represent adjacent groups of 4 bits (starting from the left). Give the corresponding 8-digit hexadecimal encoding for each of the following numbers:

(C.1) 411_{10}
(C.2) -131072_{10}
(C.3) $000110001000001011010001111010011_2$
(C.4) $11111111111111111111111110001000_2$
(C.5) -1_{10}

(D) Calculate the following using 8-bit 2's complement arithmetic (which is just a fancy way of saying to do ordinary addition in base 2 keeping only 8 bits of your answer). Remember that subtraction can be performed by negating the second operand and then adding it to the first operand.

(D.1) 42 + 36
(D.2) 96 - 69
(D.3) 69 - 96
(D.4) 120 - 60
(D.5) -60 + 120
(D.6) 120 + (-120)
(D.7) 120 + 120

Explain what happened in the last addition and in what sense your answer is "right".