

The UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Comp 120 Computer Organization
Spring 2005

Problem Set #1 Solutions
Issued Thursday, 1/20/05

Problem 1 (20 points)

- (A) (4 points) Since revealing the first card reduces the possibilities of the card from 52 choices to 1 choice, there are $\log_2 52$ bits of information. Similarly, revealing the fifth card conveys $\log_2 48$ bits of info. When the last card is revealed, we don't get any new information, because there was only one possible choice left for that card. The formula confirms this, giving the result $\log_2 1 = 0$ bits.
- (B) (4 points) Finding out that a card is red reduces the number of potential cards from 52 to 26. Thus, it provides $\log_2(52/26) = \log_2(2) = 1$ bit of information. Recognizing it as a face card, resolves the card down to one of 6 possibilities (a king, queen, or jack of either red suit) and provides $\log_2(26/6)$ additional bits of information. Finding out that it is a king further resolves the card down to one of two and provides $\log_2(6/2)$ bits. Only, the suit of the red kind still remains leaving $\log_2(2/1) = 1$ bit unresolved.

The following relation must also hold:

$$\log_2(52/26) + \log_2(26/6) + \log_2(6/2) + \log_2(2/1) = \log_2(52/1),$$

indicating that the that the card is uniquely identified.

- (C) (4 points)

0	0	10	10	0	11	0	0	0	0	0
A	A	B	B	A	C	A	A	A	A	A

- (D) (4 points)

Expected length of bit string of 1000 choices
= 1000 × Expected length of 1 choice
= 1000 × [1 × p(A) + 2 × p(B) + 2 × p(C)]
= 1000 × [1×0.8 + 2×0.1 + 2×0.1] = 1000 × 1.2
= 1200

In the worst case, each choice would require 2 bits to encode, thus the length of the bit string would be 2000.

$1000 \times \log_2(3/1) \approx 1585$, which is the number of bits to encode 1000 choices if all choices had the same probability. This is greater than the expected length using the code above and less than the worst case. This is consistent with the fact that the entropy of an equiprobable random process is always greater than a non-equiprobable random process with the same number of possibilities.

(E) (2 points)

$$\begin{aligned}\text{Bits of information} &= -[p_1 \log_2 p_1 + p_2 \log_2 p_2] \\ &= -[0.3(\log_2 0.3) + 0.7(\log_2 0.7)] \\ &\approx 0.881\end{aligned}$$

This is less than the 1 bit of information received from a flip of a fair coin.

(F) (1 points) 27358 bytes = 218864 bits

The file has at most 218864 bits of information (an upper bound). This would be the case if, for example, every bit represented the result of a different fair coin toss.

We would expect a departmental phone book to be very redundant; generally, the names would be alphabetized, the phone number themselves would share common prefixes, and the records would be most likely broken into fields containing a series of only letters followed by a field of only numbers. Therefore, we would expect the information content to be significantly less than the 218864 bits.

(G) (1 point) Yes, this is consistent with the answer to (E), since $3412 * 8 = 27296 \leq 218864$ (upper bound). Compression programs strip out redundant information, and encode the data without the redundant information, thus using fewer bits, thereby achieving compression.

Problem 2 (20 points)

(A) (1 point)

At most $2^{32} = 4,294,967,296$ values

(B) (6 points)

$$0_{10} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2$$

The most positive integer is $0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2$.

The most negative integer is $1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2$.

The decimal value of the most positive integer is $2^{31} - 1 = 2,147,483,647$.

The decimal value of the most negative integer is $-2^{31} = -2,147,483,648$.

Negating the most negative integer gives:

$$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = -2,147,483,648, \text{ which is itself, and this answer is}$$

wrong.

(C) (5 points)

$$37_{10} = 00000025_{16}$$

$$-32768_{10} = \text{FFFF}8000_{16}$$

$$1101\ 1110\ 1010\ 1101\ 1011\ 1110\ 1110\ 1111_2 = \text{DEADBEEF}_{16}$$

$$1010\ 1011\ 1010\ 1101\ 1100\ 1010\ 1111\ 1110_2 = \text{ABADCAFE}_{16}$$

$$-1_{10} = \text{FFFFFFF}_{16}$$

(D) (7 points)

$$13 = 001101_2$$

$$+ 10 = 001010_2$$

$$\hline 23 = 010111_2$$

$$18 = 010010_2$$

$$- 15 = 110001_2$$

$$\hline 3 = 000011_2$$

$$\begin{array}{r} 15 = 001111_2 \\ - 18 = 101110_2 \\ \hline -3 = 111101_2 \end{array}$$

$$\begin{array}{r} 27 = 011011_2 \\ - 6 = 111010_2 \\ \hline 21 = 010101_2 \end{array}$$

$$\begin{array}{r} -6 = 111010_2 \\ + -15 = 110001_2 \\ \hline -21 = 101011_2 \end{array}$$

$$\begin{array}{r} 21 = 010101_2 \\ + -21 = 101011_2 \\ \hline 0 = 000000_2 \end{array}$$

$$\begin{array}{r} 31 = 011111_2 \\ + 12 = 001100_2 \\ \hline -21 = 101011_2 \quad (43) \end{array}$$

In the last addition, $31+12 (=43)$ exceeds the maximum representable positive integer in 6-bit two's complement arithmetic, which is 31. The addition caused the most significant bit to become 1, resulting in a negative number. This, -21 is the "correct" answer since we were asked to use 6-bit two's complement. In practice, computers detect a situation like this, called an "overflow".

(E) (1 point)

Negating a number by complementing it and adding one can be explained as follows:

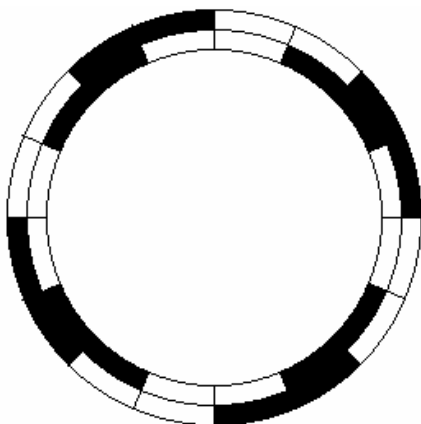
$x + \bar{x} = -1$, which states that a number added to its complement results in all one's, the 2's complement representation of -1. Adding 1 to each side gives:

$x + \bar{x} + 1 = 0$. Subtracting x from both sides gives:

$\bar{x} + 1 = -x$. Thus, complementing a number and adding one is equivalent to negation.

Problem 3 (20 points)

(A) (6 points)



The outer ring represents the 2^0 digit, and the inner ring represents the 2^1 digit. A coding pattern in which only one bit changes between successive encodings, like this one, is called a "Gray" code.

Pulse streams:

...00, 01, 11, 10... clockwise

...00, 10, 11, 01... counterclockwise

■ Not a slot
□ Slot

It is acceptable for this part to use a normal binary counting scheme, rather than a Gray code.

(B) (6 points)

We can use a 3-bit Gray code: 000, 001, 011, 010, 110, 111, 101, 100

This counting scheme avoids the problem of glitches by changing only 1 bit at a time.

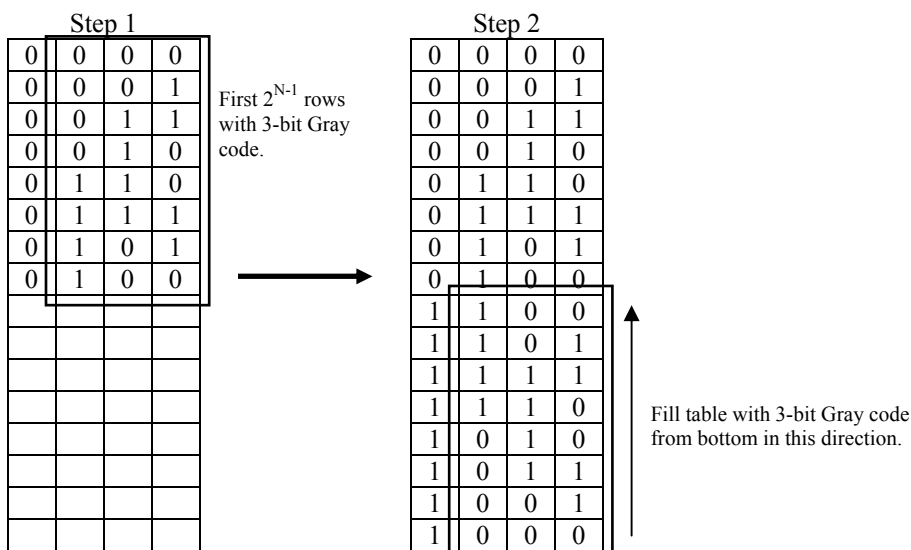
(C) (8 points)

To create an N-bit Gray code, take a 2^{N-1} bit Gray code and perform the following steps:

1. Fill out the first 2^{N-1} rows with the 2^{N-1} bit code, using 0's for the most significant bit (MSB).
2. Fill the second 2^{N-1} rows with the 2^{N-1} bit code in reverse, using 1's in the MSB position.

This definition works recursively with a terminating condition that the 2^1 bit code is {0, 1}.

For example, for a 4-bit Gray code:



Problem 4 (20 points)

(A) (4 points)

1 and 3 have an odd number of bits, indicating an error.

(B) (4 points)

- | | | | |
|-----------------|----------|----------------|-----------------|
| (1) 0011 | (2) 1100 | (3) 000 | (4) 0110 |
| 0110 | 0000 | 101 | 1001 |
| 0011 | 0101 | 10 | 0110 |
| 011 | 100 | | 100 |

(1) and (3) had single bit errors. (2) had no detectable errors, and (4) had an error in a parity bit. The bold digit represents the corrected value.

(C) (3 points)

If parity bits for exactly one column and exactly one row occur, this will be interpreted as a single bit error in the corresponding data position. (This occurred in part (B.1)). The fact that two errors occurred is not detected, and even worse, the decoded data is incorrect.

(D) (3 points)

If p_0 , p_2 , and p_3 fail, the error lies in data bit $0b01101=13$.

The index of an error is the binary number formed by the concatenated digits: $e_4e_3e_2e_1e_0$. If this number is zero, then no error occurred. If the number is a power of two, an error occurred in a parity bit.

(E) (3 points)

Each column in the binary expansion of an index indicates whether the index should be included in a parity sum. I.e., if the binary expansion of an index is $0b1010$, then the data digit would be used in the parity calculations for parity bits p_1 and p_3 . Likewise, the digit with index $0b0011$ would be used in parity calculations for bits p_0 , and p_1 . Note that since no data bit is stored in an index that is a power of two, we guarantee that each bit is represented in at least two different parity bits.

(F) (3 points)

If two errors occur in the parity bits, this will be misinterpreted as a single data bit error.

Another class of error is when certain combinations of two data bits are corrupted. For example, if index 19 and 21 are corrupted, it affects a number of parity bits which makes it look as though a different [single] bit was corrupted.

We can detect these situations by adding an additional parity bit, p_5 , which is the parity of every parity and data bit (excluding itself, of course!). If a single data bit failed, this bit would indicate an error. But if two exactly two bits have failed instead (which was previously indistinguishable from a single data bit error), p_5 would indicate no error but p_0, p_1, \dots, p_4 would indicate the presence of some error (but the wrong error). So, if by looking at p_0, p_1, \dots, p_4 it seems as though an error occurred, we should check p_5 , to make sure that only one error occurred (i.e., p_5 indicates an error.) If p_5 does not indicate an error, than two errors occurred.

Problem 5 (20 points)

(A) (1 points)

There are 10 equiprobable possibilities, thus $\log_2 10=3.329$.

(B) (3 points)

A transmitted value of 0 means that $f(d)$ is 1. $f(d)=1$ has a probability of occurring of $1/10$, thus, the amount of information transmitted is $\log_2 10=3.329$ bits. A transmitted value of 1 means that $f(d)=2$, which has a probability of $3/10$, which means that $\log_2 10/3 = 1.737$ bits were transmitted.

(C) (3 points)

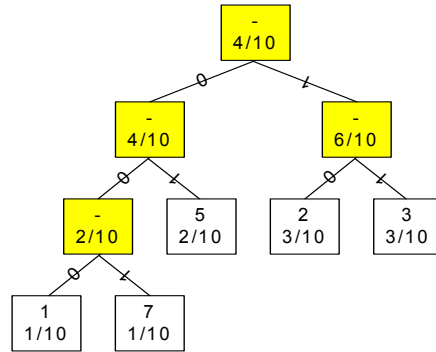
Applying the entropy formula, we find:

$$1/10*\log_2(10)+3/10*\log_2(10/3)+3/10*\log_2(10/3)+2/10*\log_2(10/2) +1/10*\log_2(10) = 2.171 \text{ bits}$$

(D) (3 points)

Each iteration consumes two nodes and generates one new one. Thus, it will take $(n-1)$ iterations to reduce n nodes to a single node.

(E) (10 points)



This code has an expected length of

Value	Prob	Encoding
1	1/10	000
2	3/10	10
3	3/10	11
5	2/10	01
7	1/10	001

2.2 bits.