# CS 70 SPRING 2008 — DISCUSSION #7 ON EARTH

LUQMAN HODGKINSON, AARON KLEINMAN, MIN XU

## 1. Error-Correcting Code (Earth Version)

**Exercise 1.** Your are on **Earth**. Your mission in this exercise is to successfully transmit an outgoing message to the Moon with corruption, and to repair and reconstruct an incoming message that has been corrupted. As the Berlekamp-Welsh method is complex, we will just encode our message in $\pmod 5$ and our message will only be two digits long. Also, assume that the channel will only corrupt 1 digit in our message. In another words, let $N = 2$ and $k = 1$. How long should our transmission be?

First, find a student in class who is on the **Moon** and pair up. You must transmit the message $[c_0 = 1, c_1 = 3]$ to your partner. Find a polynomial of degree at most 1 $P(x)$ such that $P(0) = c_0, P(1) = c_1$, then construct the full message you would send. Now, corrupt one of the numbers in your message, you may choose which ever, but I recommend sending setting the second packet $c_1$ to 1 to save your partner some computation time.

Now, receive the corrupted message from your partner and perform the Berlekamp-Welsh method to reconstruct the original message. DON'T TELL YOUR PARTNER WHAT YOU RECONSTRUCTED YET! Move on to the next exercise after you have the message.

## 2. Fingerprinting

**Exercise 2.** Take what you think is the message that your partner on the Moon has sent you and calculate its fingerprint. In our fingerprinting scheme, we will encode our message into a polynomial of degree 1 $\pmod{23}$ by the following method: if you received the message $[m_0, m_1]$, then construct the polynomial $T(x) = 3m_2x + 2m_1 + 9 \pmod{23}$. Now, agree with your partner upon a random $r$ between 0 and 22 and evaluate the fingerprint $T(r)$. Give the fingerprint to your partner, did it work?

If your partner handed you a fingerprint, calculate the finger print of the correct message using the above scheme and compare the two. If they are not the same, (politely) scold your partner and ask him/her to re-try. If they are the same, with what probability are you certain that your partner has deciphered the message correctly?

## 3. Erasure codes

**Exercise 3.** Min sends you a message in the alphabet R = 0, F = 1, A = 2, U = 3, and N = 4 using the polynomial scheme discussed in class. The message size is 3. You receive the following packets.

- clearly corrupted
- U
- N
- N

Assuming the three decipherable packets arrive uncorrupted, what is the value in the corrupted packet?

## 4. More Fingerprinting

**Exercise 4.** We sent the message 00001 to the Moon, but the Moon received 01010. We use the polynomial fingerprinting scheme discussed in class with $P(x) = s_{n-1}x^{n-1} + \ldots + s_1x + s_0$ (where $s_0$ is the least significant bit of the message) to verify that the data was received correctly. Our prime $q = 101$. Is $r = 25$ a lucky or unlucky value?

*Date*: March 12, 2008.
The authors gratefully acknowledge the TA's of CS70 Past for the use of their previous notes: Chris Crutchfield, Alex Fabrikant, David Garmire, Assane Gueye, Amir Kamil, Lorenzo Orecchia, Vahab Pournaghshband, Ben Rubinstein. Their notes form the basis for this handout.

## 5. A Few Bits of Big-O

**Exercise 5.** Fill in the following chart:

You may assume that $x, y$ are already reduced mod p when considering operation performed $\pmod{p}$.

| operation | number of bit-ops in big-O | runtime when performed $\mod p$ |
|---|---|---|
| $x + y$ | | |
| $xy$ | | |
| $x/y$ | | |
| $x \mod y$ | | n/a |
| $x^n$ | | |

Suppose we are given a polynomial $P(x) = a_n x^n + a_{n-1}x^{n-1} + ...a_0$ where $a_n, a_{n-1}, ...a_0$ are all natural numbers between 0 and $q - 1$. Let $r$ be a natural number between 0 and $q - 1$, we wish to evaluate $P(r)$ mod $q$.

**Exercise 6.** Suppose we use the following pseudo-code for our program, how many bit operations will we perform? Express your answers in big-O notation.

```
funct poly-eval:
    val_so_far = a_0

    for i from 1 to n:
       val_so_far = val_so_far + a_i * r^i

    return val_so_far mod q
```

**Exercise 7.** Can you think of a better algorithm that will bring the run-time down to $O(n \lg(q)^2)$?