# POLYNOMIALS, SECRET SHARING, ERASURE ERRORS, GENERAL ERRORS, SELF REFERENCE 4

COMPUTER SCIENCE MENTORS 70

October 3 to October 7, 2016

## 1 Polynomials

### 1.1 Introduction

1. There is a unique polynomial of degree $n-1$ such that $P(i) = m_i$ for each packet $m_1, \ldots, m_n$
2. To account for errors we send $c_1 = P(1), \ldots, c_{n+j} = P(n+j)$
3. If polynomial $P(x)$ has degree $n-1$ then we can uniquely reconstruct it from any $n$ distinct points.
4. If a polynomial $P(x)$ has degree $n-1$ then it can be uniquely described by its $n$ coefficients

### 1.2 Questions

1. Define the sequence of polynomials by $P_0(x) = x+12$, $P_1(x) = x^2-5x+5$ and $P_n(x) = xP_{n-2}(x) - P_{n-1}(x)$. (For instance, $P_2(x) = 17x - 5$ and $P_3(x) = x^3 - 5x^2 - 12x + 5$.)
   (a) Show that $P_n(7) \equiv 0 \pmod{19}$ for every $n \in N$.

   **Solution:**

   (a) Prove using strong induction.

**Base Case** There are two base cases because each polynomial is defined in terms of the two previous ones except for $P_0$ and $P_1$.

$$P_0(7) \equiv 7 + 12 \equiv 19 \equiv 0 \pmod{19}$$
$$P_1(7) \equiv 7^2 - 5 \cdot 7 + 5 \equiv 49 - 35 + 5 \equiv 19 \equiv 0 \pmod{19}$$

**Inductive Hypothesis** Assume $P_n(7) \equiv 0 \pmod{19}$ for every $n \leq k$.

**Inductive Step** Using the definition of $P_{k+1}$, we have that

$$\begin{aligned} P_{k+1}(7) &\equiv xP_{k-1}(7) - P_k(7) \pmod{19} \\ &\equiv x \cdot 0 - 0 \pmod{19} \\ &\equiv 0 \pmod{19} \end{aligned}$$

Therefore, $P_n(7) \equiv 0 \pmod{19}$ for all natural numbers $n$.

(b) Show that, for every prime $q$, if $P_{2013}(x) \not\equiv 0 \pmod{q}$, then $P_{2013}(x)$ has at most 2013 roots modulo $q$.

**Solution:** This question asks to prove that, for all prime numbers $q$, if $P_{2013}(x)$ is a non-zero polynomial $\pmod{q}$ , then $P_{2013}(x)$ has at most 2013 roots $\pmod{q}$.

The proof of Property 1 of polynomials (a polynomial of degree $d$ can have at most $d$ roots) still works in the finite field $GF(q)$. Therefore we need only show that $P_{2013}$ has degree at most 2013. We prove that $\deg(P_n) \leq n$ for $n > 1$ by strong induction.

**Base cases** There are 4:

$$\begin{aligned} \deg(P_0) &= \deg(x + 12) = 1 \\ \deg(P_1) &= \deg(x^2 - 5x + 5) = 2 \\ \deg(P_2) &= \deg(xP_0(x) - P_1(x)) \leq 2 \\ \deg(P_3) &= \deg(xP_1(x) - P_2(x)) \leq 3 \end{aligned}$$

**Inductive Hypothesis** Assume $\deg(P_n) \leq n$ for all $2 \leq n \leq k$.

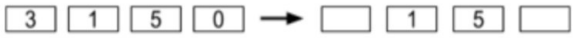
**Inductive Step** Then

$$\begin{aligned} &\deg(P_{k+1}(x)) \\ &\qquad \leq \max\{\deg(xP_{k-1}(x)), \deg(P_k(x))\} \\ &\qquad = \max\{1 + \deg(P_{k-1}(x)), \deg(P_k(x))\} \\ &\qquad \leq \max\{1 + k - 1, k\} \\ &\qquad \leq k \\ &\qquad \leq k + 1 \end{aligned}$$

# 2 Erasure Errors

## 2.1 Introduction

We want to send $n$ packets and we know that $k$ packets could get lost.




How many more points does Alice need to send to account for $k$ possible errors? __

**Solution:** $k$

What degree will the resulting polynomial be? __

**Solution:** $n - 1$

How large should $q$ be if Alice is sending n packets with k erasure errors, where each packet has $b$ bits?

**Solution:** Modulus should be larger than $n + k$ and larger than $2^b$ and be prime

What would happen if Alice instead send $n + k - 1$? Why will Bob be unable to recover the message?

**Solution:** Bob will receive $n - 1$ distinct points and needs to reconstruct a polynomial of degree $n - 1$. By Fact #3 this is impossible. There are $q$ polynomials of at most degree $n - 1$ in $GF(q)$ that go through the $n - 1$ points that Alice sent.

## 2.2 Questions

1. Suppose $A = 1$, $B = 2$, $C = 3$, $D = 4$, and $E = 5$. Assume we want to send a message of length 3. Recover the lost part of the message, or explain why it can not be done.

1. C_AA

   **Solution:** $P(0) = 3, P(2) = 1, P(3) = 1$. Once we interpolate the polynomial over $\mod 7$, as E is 5, we get $3x^2 + 3x + 3$. Now, once we evaluate this at 1, we get 2. So, in the end, its CBAA.

2. CE_ _

   **Solution:** Impossible. In order to get the original degree 2 polynomial, we need at least $3 > 2$ points.

2. Suppose we want to send $n$ packets, and we know $p = 20\%$ of the packets will be erased. How many extra packets should we send? What happens if $p$ increases (say to $90\%$)?

   **Solution:** We want to have $(1-p)*(n+k) = n$, where $k$ is the number of additional packets we send. Solving for $k$, we get $\frac{n}{1-p} - n$. When $p$ is large, we have to send many times the number of original packets. (fraction packets not erased)*(how many packets are sent) = (number packets in original message)

# 3 General Errors

## 3.1 Introduction

Now instead of losing packets, we know that $k$ packets are corrupted. Furthermore, we do not know which $k$ packets are changed. Instead of sending $k$ additional packets, we will send an additional $2k$.


**Solomon-Reed Codes**

1. Identical to erasure errors: Alice creates $n-1$ degree polynomial $P(x)$.

$$P(x) = p_0 + p_1 x + \ldots + p_{n-1}x^n$$

2. Alice sends $P(1), \ldots, P(n+2k)$

3. Bob receives $R(1), \ldots, R(n+2k)$

For how many points does $R(x) = P(x)$?

**Solution:** $n + k$

True or false: $P(x)$ is the unique degree $n-1$ polynomial that goes through at least $n+k$ of the received points.

**Solution:** True

Write the matrix view of encoding the points $P(1), \ldots, P(n+2k)$

**Solution:**

$$\begin{bmatrix} P(1) \\ P(2) \\ P(3) \\ \vdots \\ P(n+2k) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1^2 & \cdot & 1 \\ 1 & 2 & 2^2 & \cdot & 2^{n-1} \\ 1 & 3 & 3^2 & \cdot & 3^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & (n+2k) & (n+2k)^2 & \cdot & (n+2k)^{n-1} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-1} \end{bmatrix}$$

**Berlekamp Welch**

How do we find the original polynomial $P(x)$?

Suppose that $m_1, \ldots, m_k$ are the corrupted packets. Let $E(x) = (x - m_1) \ldots (x - m_k)$ Then $P(i) * E(i) = r_i * E(i)$ for any $i$ greater than 1 and less than $n + 2k$. Why?

**Solution:** Case 1: $i$ is corrupted

$$E(i) = 0 \rightarrow 0 = 0$$

Case 2: $i$ is not corrupted

$$P(i) = ri \rightarrow P(i) * E(i) = r_i E(i)$$

Let $Q(i) = P(i)E(i)$ So we have $Q(i) = P(i)E(i) = r_i * E(i)$ where $1 \leq i \leq 2k + n$ What degree is $Q(i)$?

**Solution:** $n + k - 1$

How many coefficients do we need to describe $Q(i)$?

**Solution:** $n + k$

What degree is $P(i)$?

**Solution:** $k$

How many unknown coefficients do we need to describe $E(i)$?

**Solution:** $k$ (we know that the first coefficient has to be 1)

We can write $Q(i) = r_i E(i)$ for every$i$ that is $1 \leq i \leq 2k + n$.
How many equations do we have? How many unknowns?

**Solution:** $n + 2k$

Once we have the above described equations, how do we determine what $P(i)$ is?

**Solution:** Solve the equations to get the coefficients for $E(i)$ and $Q(i)$. Then divide $\frac{E(i)}{Q(i)}$ to get $P(i)$.

## 3.2 Questions

1. (a) Alice sends Bob a message of length 3 on the Galois Field of 5 (modular space of mod 5). Bob receives the following message: (3, 2, 1, 1, 1). Assuming that Alice is sending messages using the proper general error message sending scheme, set up the linear equations that, when solved, give you the $Q(x)$ and $E(x)$ needed to find the original $P(x)$.

**Solution:** Set up 5 equations for the five values of x that we have, such that

$$Q(x) \mod 5 = r_x * (x - b) \mod 5$$

where

$$Q(x) = a_3 * x_i^3 + a2 * x_i^2 + a_1 * x_i + a_0$$

($r_i$ = ith received number)
In the form, for

$$x = x_i, a3 * x_i^3 + a_2 * x_i^2 + a_1 * x_i + a_0 = r_i * (x - b)$$

$$x = 1, (a_3 + a_2 + a_1 + a_0) \mod 5 = 3(1 - b) \mod 5$$
$$x = 2, (3a_3 + 4a_2 + 2a_1 + a_0) \mod 5 = 2(2 - b) \mod 5$$
$$x = 3, 2a_3 + 4a_2 + 3a_1 + a_0 \mod 5 = 1(3 - b) \mod 5$$
$$x = 4, 4a_3 + 1a_2 + 4a_1 + a_0 \mod 5 = 1(4 - b) \mod 5$$
$$x = 5, 0 + 0 + 0 + a0 \mod 5 = 1(0 - b) \mod 5$$

Solving for these equations, which you should give your students after setting up the above, you get the following:

$$b = 3$$

$$a_3 = 1$$
$$a_2 = 3$$
$$a_1 = 3$$
$$a_0 = 2$$

(b) What is the encoded message that Alice actually sent? What was the original message? Which packet(s) were corrupted?

**Solution:** Now we have $P(x) = \frac{Q(x)}{E(x)}$. Plug in the coefficients for $Q$ and $E$ and divide (using polynomial long division) and you get $P(x) = (x^2)+x+1$. Using the $P(x)$ that we found, we plug in the values 1, 2, 3, 4, 5 to find the encoded 5 packet message. $P(1) = 3, P(2) = 2, P(3) = 3, P(4) = 1, P(5) = 1$. The original message is the first 3 $P(1), P(2), P(3)$. Using the value of b, we know that the 3rd packet was corrupted, which we can confirm in the message that was received.

2. Suppose that the message we want to send consists of 10 numbers. We find a polynomial of degree 9 which goes through all these points and evaluate it on 25 points. How many erasure errors can we recover from? How about general errors?

**Solution: Erasure**: $n + k =$ # points to send
$10 + k = 25$
$k = 15$ erasure errors
**General**: $n + 2k =$ # points to send
$10 + 2k = 25$
$k = 7$ general errors.

3. You want to send a super secret message consisting of 10 packets to the space station through some astronauts. Youre afraid that some malicious spy people are going to tell the wrong message and make the space station go spiraling out of orbit. Assuming that up to 5 of the astronauts are malicious, design a scheme so that the group of astronauts (including the malicious ones) still find the correct message that you want to send. You can send any number of astronauts, but try to make the number that you have to send as small as possible. Astronauts can only carry one packet with them.

**Solution:** We can use the scheme provided in the notes for general errors. If there are up to 5 malicious astronauts, then we have up to 5 general errors in our transmission. To account for those errors and to send the original message we have to send at least the length of the original message plus the twice the possible number of general errors. Thus, we have to send 20 astronauts or more. Create a polynomial of degree 9 (which is the length of the message minus 1), such that $P(0), P(1), \ldots, P(10)$ is the original message, and give each of the 20 astronauts unique points on that polynomial. Then use the methods we used in the previous question to find the message.

# 4 Secret Sharing

## 4.1 Questions

1. Suppose the Oral Exam questions are created by 2 TAs and 3 Readers. The answers are all encrypted and we know that:
   (a) Both TAs should be able to access the answers
   (b) All 3 Readers can also access the answers
   (c) One TA and one Reader should also be able to do the same

   Design a secret sharing scheme to make this work.

   **Solution:** Use a 2 degree polynomial which requires at least 3 shares to recover the polynomial. Generate a total of 7 shares, give each Reader a share, and each TA 2 shares. Then, all possible combinations will have at least 3 shares to recover the answer key. Basically the point of this problem is to assign different weights to different classes of people. If we give one share to everyone, then 2 Readers can also recover the secret and the scheme is broken.

2. An officer stored an important letter in her safe. In case she is killed in battle, she decides to share the password with her troops. Everyone knows there are 3 spies among the troops, but no one knows who they are except for the three spies themselves.The 3 spies can coordinate with each other and they will either lie and make people not able to open the safe, or will open the safe themselves if they can. Therefore, the officer would like a scheme to share the password that satisfies the following conditions:

   1. When $M$ of them get together, they are guaranteed to be able to open the safe even if they have spies among them.
   2. The 3 spies must not be able to open the safe all by themselves.

   Please help the officer to design a scheme to share her password. What is the scheme? What is the smallest $M$? Show your work and argue why your scheme works and any smaller $M$ couldnt work.

   **Solution:** The key insight is to realize that both polynomial-based secret-sharing and polynomial-based error correction work on the basis of evaluating an underlying polynomial at many points and then trying to recover that polynomial. Hence they can be easily combined. Suppose the password is $s$. The officer can construct a polynomial $P(x)$ such that $s = P(0)$ and share $(i, P(i))$ to the $i$-th person in her troops. Then the problem is: what should the degree of $P(x)$ be and what is the smallest $M$? First, the degree of polynomial $d$ should not be less than 3. It is because when $d < 3$, the 3 spies can decide the polynomial $P(x)$ uniquely. Thus, $n$ will be at least 4 symbols. Lets choose a polynomial $P(x)$ of degree 3 such that $s = P(0)$. We now view the 3 spies as 3 general errors. Then the smallest $M = 10$ since n is at least 4 symbols and we have $k = 3$ general errors, leading us to a codeword of $4 + 2 \cdot 3 = 10$ symbols (or people in our case). Even though the 3 spies are among the 10 people and try to lie on their numbers, the 10 people can still be able to correct the $k = 3$ general errors by the Berlekamp-Welch algorithm and find the correct $P(x)$.

   Alternative solution: Another valid approach is making $P(x)$ of degree $M1$ and adding 6 public points to deal with 3 general errors from the spies. In other words, in addition to their own point $(i, P(i))$, everyone also knows the values of 6 more points, $(t+1, P(t+1)), (t+2, P(t+2)), \ldots, (t+6, P(t+6))$, where $t$ is the number of the troops. The spies have access to total of $3 + 6 = 9$ points so the degree $M - 1$ must be at least 9 to prevent the spies from opening the safe by themselves. Therefore, the minimum $M$ is 10.

# 5 Self Reference

## 5.1 Introduction

**The Halting Problem:** Does a given program ever halt when executed on a given input?

$$\texttt{TestHalt(P,x)} = \begin{cases} \text{“yes”}, & \text{if program } P \text{ halts on input } x \\ \text{“no”}, & \text{if program } P \text{ loops on input } x \end{cases}$$

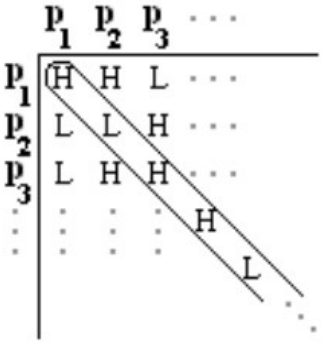
How do we prove that `TestHalt` doesnt exist? Lets assume that it does, and hope we reach a contradiction.
Define another program:

```
Turing(P)
  if TestHalt(P,P) = "yes" then loop forever
  else halt
```

What happens when we call `Turing(Turing)`?

**Solution:**
**Case 1** : It halts If `Turing(Turing)` halts then `TestHalt(Turing, Turing)` must have returned no. But `TestHalt(Turing Turing)` calls `Turing(Turing)` and calling `Turing(Turing)` must loop. But we assumed that `Turing(Turing)` halted. Contradiction.
**Case 2** : It loops This implies that `TestHalt(Turing, Turing)` returned yes, which by the way that `TestHalt` is defined implies that `Turing` halted. But we assumed that `Turing(Turing)` looped. Contradiction.

How is this just a reformulation of proof by diagonalization?




**Solution:** List all possible programs as rows and columns. The rows are the programs and the columns are the inputs. Turing must be one of the rows, say row $n$. If entry $(n, n)$ is $H$ then Turing will loop by definition. If entry $(n, n)$ is $L$ then Turing will halt by definition. Therefore Turing cannot be on the list of all programs and therefore it does not exist.

Therefore the Halting Problem is unsolvable. We can use this to prove that other problems are also unsolvable. Say we are asked if program $M$ is solvable. To prove it is not, we just need to prove the following claim: If we can compute program $M$, then we could also compute the halting problem.
This would then prove that $M$ can not exist, since the halting problem is not computable. This amounts to proof by contradiction.

## 5.2 Questions

1. Say that we have a program $M$ that decides whether any input program halts as long as it prints out the string ABC as the first operation that it carries out. Can such a program exist?

   **Solution:** No. Such a program $M$ can not exist. We proceed as follows: we show that if such a program existed, the halting problem would be computable.

   Consider any program $P$. If we wanted to decide if $P$ halted, we could simply create a new program $P'$ where $P'$ first prints out ABC, then proceed to do exactly what $P$ would. However, if $M$ existed, we could determine whether any program $P$ halted by converting it to a $P'$ and feeding it into $M$. This would solve the halting problem- but this is a contradiction, since by diagonalization we can prove that the halting problem is not solvable.

   Therefore, $M$ can not exist!