CS 70 Discrete Mathematics and Probability Theory
Fall 2019 Alistair Sinclair and Yun S. Song

HW 5

Note: *This homework consists of two parts. The first part (questions 1-4) will be graded and will determine your score for this homework. The second part (questions 5-6) will be graded if you submit them, but will not affect your homework score in any way. You are strongly advised to attempt all the questions in the first part. You should attempt the problems in the second part only if you are interested and have time to spare.*

For each problem, justify all your answers unless otherwise specified.

Part 1: Required Problems

1 Recursively Defined Polynomials

Let's define two polynomials $f_1(x) = x - 2$, $f_2(x) = x^2 + 3$, and for each natural number $n \geq 2$, recursively define $f_n(x) = xf_{n-1}(x) - f_{n-2}(x)$.

(a) Compute $f_3(x)$ and $f_4(x)$.

(b) What is the largest number of roots that $f_n(x)$ can have over $\mathbb{R}$? What is the smallest number of roots it can have? Both answers should depend only on the degree of $f_n(x)$, and one will depend on whether n is even or odd.

(c) Prove that $f_n(2) = 0 \mod 7$ for every n.

Solution:

(a) Directly from the definition of $f_n(x)$ in the problem,

$$\begin{aligned} f_3(x) &= xf_2(x) - f_1(x) \\ &= x(x^2+3) - (x-2) \\ &= x^3 + 2x + 2 \end{aligned}$$

and

$$\begin{aligned} f_4(x) &= xf_3(x) - f_2(x) \\ &= x(x^3+2x+2) - (x^2+3) \\ &= x^4 + x^2 + 2x - 3. \end{aligned}$$

(b) Recall that the *degree* of a polynomial is the highest exponent that appears in it, and that a polynomial of degree d has at most d roots over any field. Over $\mathbb{R}$, a polynomial of even degree need not have any roots (consider $x^{2d}+a^2$), but a polynomial of odd degree must have at least one: if $f(x) = x^{2d+1} + f_{2d}x^{2d} + \cdots + f_0 x_0$ then $f(x)$ goes to positive infinity as x gets very large and positive, to negative infinity as x gets very large and negative, and we can use the intermediate value theorem to prove that there is a root somewhere in between. So all we need to do is compute the degree of $f_n(x)$.

In our case, the degree of $f_1(x)$ is one, so it has exactly one root (a degree one polynomial over any field has exactly one root). The degree of $f_2(x)$ is two, so it could in principle have up to two roots, but because $f_2(x) = x^2+3$ and $3 > 0$, it has none. For $n > 1$, let's prove by (strong) induction that $f_n(x)$ has degree n, and that the coefficient of the x^n term is 1. We've already done the base case, so assume that $f_k(x)$ and $f_{k-1}(x)$ have degree k and $k-1$ respectively. Then by definition

$$\begin{aligned} f_{k+1}(x) &= xf_k(x) - f_{k-1}(x) \\ &= x\left(x^k + a_{k-1}x^{k-1} + \cdots + a_1x + a_0\right) - \left(x^{k-1} + b_{k-2}x^{k-2} + \cdots + b_1x + b_0\right) \qquad \text{Ind. Assump.} \\ &= x^{k+1} + a_{k-1}x^k + (a_{k-2}-1)x^{k-1} + (a_{k-3}-b_{k-2})x^{k-2} + \cdots + (a_0 - b_1)x - b_0, \end{aligned}$$

and the last line clearly has degree $k+1$.

Since we've proved that $f_n(x)$ is a degree-n polynomial, we know it can have at most n roots, and has at least zero if n is even, and at least one if n is odd.

(c) Let's prove this by strong induction as well. We'll need two base cases, since each $f_k(x)$ is defined in terms of the two previous polynomials in the sequence: these base cases are

$$\begin{aligned} f_0(2) &= 2-2 = 0 \\ f_1(2) &= 2^2+3 = 7 = 0 \mod 7. \end{aligned}$$

For our inductive assumption, assume that $f_k(2) = 0 \mod 7$ and $f_{k-1}(2) = 0 \mod 7$. In other words, there exist $p, q \in \mathbb{Z}$ so that $f_k(2) = 7p$ and $f_{k+1}(2) = 7q$. Using the definition of f_{k+1}, we can write

$$f_{k+1}(2) = 2f_k(2) - f_{k-1}(2) = 2(7p) - (7q) = 7(2p-q) = 0 \mod 7.$$

2 Equivalent Polynomials

This problem is about polynomials with coefficients in GF(q) for some prime $q \in \mathbb{N}$. We say that two such polynomials f and g are *equivalent* if $f(x) = g(x)$ for every $x \in \text{GF}(q)$.

(a) Use Fermat's Little Theorem to find a polynomial equivalent to $f(x) = x^5$ over GF(5); then find one equivalent to $g(x) = 1 + 3x^{11} + 7x^{13}$ over GF(11).

(b) Prove that whenever $f(x)$ has degree $\geq q$, it is equivalent to some polynomial $\tilde{f}(x)$ with degree $< q$.

Solution:

(a) Fermat's Little Theorem says that for any nonzero integer a and any prime number q, $a^{q-1} \equiv 1 \mod q$. We're allowed to multiply through by a, so the theorem is equivalent to saying that $a^q \equiv a \mod q$; note that this is true even when $a = 0$, since in that case we just have $0^q \equiv 0 \mod q$. The problem asks for a polynomial $\tilde{f}(x)$, different from $f(x)$, with the property that $\tilde{f}(a) \equiv a^5 \mod 5$ for any integer a. Directly using the theorem, $\tilde{f}(x) = x$ will work. We can do something similar with $g(x) = 1 + 3x^1 1 + 7x^{13}$ modulo 11: set $\tilde{g}(x) = 1 + 3x + 7x^3$.

(b) One proof uses Fermat's Little Theorem. As a warm-up, let $d \geq q$; we'll find a polynomial equivalent to x^d. For any integer, we know

$$\begin{aligned} a^d &= a^{d-q}a^q \\ &\equiv a^{d-q}a \mod q \\ &\equiv a^{d-q+1} \mod q. \end{aligned}$$

In other words x^d is equivalent to the polynomial $x^{d-(q-1)}$. If $d-(q-1) \geq q$, we can show in the same way that x^d is equivalent to $x^{d-2(q-1)}$. Since we subtract $q-1$ every time, the sequene $d, d-(q-1), d-2(q-1), \ldots$ must eventually be smaller than q. Now if $f(x)$ is any polynomial with degree $\geq q$, we can apply this same trick to every x^k that appears for which $k \geq q$.

Another proof uses Lagrange interpolation. Let $f(x)$ have degree $\geq q$. By Lagrange interpolation, there is a unique polynomial $\tilde{f}(x)$ of degree at most $q-1$ passing through the points $(0, f(0)), (1, f(1)), (2, f(2)), \ldots, (q-1, f(q-1))$, and we designed it exactly so that it would be equivalent to $f(x)$.

3 Secret Veto

In the usual secret-sharing scenario we consider (for instance) a secret vault at the United Nations, which we want to design with the property that any k representatives can pool their information and open it, but any smaller number has no hope of doing so. Assume that the solution in the notes has been implemented, so that the key is some number s, and each member has been assigned a number $f(i) \mod q$ for some degree $k-1$ polynomial f with coefficients in GF(q) and satisfying $f(0) = s$.

(a) A group of $k+\ell$ representatives get together to discuss opening the vault. What will happen if ℓ representatives are opposed to opening the vault and, instead of revealing their true numbers, secretly reveal some *different* numbers from GF(q)? Will the group be able to open the vault? If so, how long will it take?

(b) Repeat part (a) in the event that only $\ell/2$ of the ℓ representatives in opposition reveal different numbers than they were assigned—assume that ℓ is even.

Solution:

(a) In this situation, the polynomial interpolating the $k+\ell$ points will be some polynomial $\tilde{f}$, differant then f, but with the property that $\tilde{f}(i) = f(i)$, for any representative i who gave the correct number. In fact, we claim that $\tilde{f}$ must have degree striclty larger than that of f. This is because Lagrange interpolation produces the *unique* polynomial of lowest degree passing through a given set of points: if f and $\tilde{f}$ had the same degree, they would have to be the same, since both pass through the k points $(i, f(i))$ of the representatives who answered truthfully.

In other words, when group sees that the polynomial has degree strictly larger than $k-1$, they will know that someone has given faulty information, but not who did so. By trying all possible subsets of k of the $k+\ell$ representatives, it would be possible to both recover the key and discover which members were engaging in the subterfuge. However, there are exponentially many such subsets in ℓ.

(b) If only $\ell/2$ representatives give incorrect information, the resulting polynomial will be of degree at most $k+\ell/2-1$. This time, however, the group can use the Berlekamp-Welch algorithm to efficiently figure out who was responsible and what the true polynomial is! Recall that in the Berlekam-Welch setting you receive a list $f(1), f(2), ..., f(k+\ell)$, of which up to $\ell/2$ entries may have been corrupted; the algorithm allows you to recover the locations of the errors and the true polynomial f. This setting is functionally the same: the group has access to $k+\ell$ evaluations of a polynomial, and they know that $k+\ell/2$ of them are correct. The only difference is that the polynomial is not evaluated at the points $1, 2, ..., k+\ell$, but rather at $k+\ell$ arbitrary points among the numbers $1, ..., n$.

4 Berlekamp-Welch Algorithm

In this question we will send the message $(m_0, m_1, m_2) = (1, 1, 4)$ of length $n = 3$. We will use an error-correcting code for $k = 1$ general error, doing arithmetic over GF(5).

(a) Construct a polynomial $P(x) \pmod 5$ of degree at most 2, so that

$$P(0) = 1, \qquad P(1) = 1, \qquad P(2) = 4.$$

What is the message $(c_0, c_1, c_2, c_3, c_4)$ that is sent?

(b) Suppose the message is corrupted by changing c_0 to 0. Set up the system of linear equations in the Berlekamp-Welch algorithm to find $Q(x)$ and $E(x)$.

(c) Assume that after solving the equations in part (b) we get $Q(x) = 4x^3 + x^2 + x$ and $E(x) = x$. Show how to recover the original message from Q and E.

Solution:

(a) We use Lagrange interpolation to construct the unique quadratic polynomial $P(x)$ such that $P(0) = m_0 = 1, P(1) = m_1 = 1, P(2) = m_2 = 4$. Doing all arithmetic over GF(5), so that i.e. $2^{-1} = 3 \pmod 5$,

$$\begin{aligned}
\Delta_0(x) &= \frac{(x-1)(x-2)}{(0-1)(0-2)} = \frac{x^2-3x+2}{2} \equiv 3(x^2-3x+2) \pmod 5 \\
\Delta_1(x) &= \frac{(x-0)(x-2)}{(1-0)(1-2)} = \frac{x^2-2x}{-1} \equiv 3(x^2-x) \pmod 5 \\
\Delta_2(x) &= \frac{(x-0)(x-1)}{(2-0)(2-1)} = \frac{x^2-x}{2} \equiv 3(x^2-3x+2) \pmod 5 \\
P(x) &= m_0\Delta_0(x) + m_1\Delta_1(x) + m_2\Delta_2(x) \\
&= 1\Delta_0(x) + 1\Delta_1(x) + 4\Delta_2(x) \\
&\equiv 4x^2 + x + 1 \pmod 5
\end{aligned}$$

For the final message we need to add 2 redundant points of P. Since 3 and 4 are the only points in GF(5) that we have not used yet, we compute $P(3) = 0, P(4) = 4$, and so our message is $(1,1,4,0,4)$.

(b) The message received is $(c'_0, c'_1, c'_2, c'_3, c'_4) = (0,1,4,0,4)$. Let $R(x)$ be the function such $R(i) = c'_i$ for $0 \le i < 5$. Let $E(x) = x + b_0$ be the error-locater polynomial, and $Q(x) = P(x)E(x) = a_3x^3 + a_2x^2 + a_1x + a_0$. Since $Q(i) = P(i)E(i) = R(i)E(i)$ for $1 \le i < 5$, we have the following equalities $\pmod 5$

$$\begin{aligned}
Q(0) &= 0E(0) \\
Q(1) &= 1E(1) \\
Q(2) &= 4E(2) \\
Q(3) &= 0E(3) \\
Q(4) &= 4E(4)
\end{aligned}$$

which can be rewritten as a system of linear equations

$$\begin{array}{rcrcrcrcrcl}
 & & & & & & a_0 & & & = & 0 \\
a_3 & + & a_2 & + & a_1 & + & a_0 & - & b_0 & = & 1 \\
8a_3 & + & 4a_2 & + & 2a_1 & + & a_0 & - & 4b_0 & = & 8 \\
27a_3 & + & 9a_2 & + & 3a_1 & + & a_0 & & & = & 0 \\
64a_3 & + & 16a_2 & + & 4a_1 & + & a_0 & - & 4b_0 & = & 1
\end{array}.$$

(c) From the solution, we know

$$\begin{aligned}
Q(x) &= 4x^3 + x^2 + x, \\
E(x) &= x + b_0 = x.
\end{aligned}$$

Since $Q(x) = P(x)E(x)$, the recipient can compute $P(x) = Q(x)/E(x) = 4x^2 + x + 1$, the polynomial $P(x)$ from part (a) used by the sender. The error locating polynomial $E(x)$ is degree

one, so there is only one error, and as $E(x) = x = x - 0$, the corrupted bit was the first one. To correct this error we evaluate $P(0) = 1$ and combine this with the two uncorrupted bits m_1, m_2, to get the original message

$$(m_0, m_1, m_2) = (1, 1, 4).$$

Note: initially there was a typo in this part of the problem, asking that we assume $Q(x) = 2x^3 + 2x^2$. From this assumption, we deduce that $P(x) = Q(x)/E(x) = 2x^2 + 2x$. The intended message would then be

$$(m_0, m_1, m_2) = (P(1), P(1), P(2)) = (0, 4, 2)$$

Note: *This concludes the first part of the homework. The problems below are optional, will not affect your score, and should be attempted only if you have time to spare.*

Part 2: Optional Problems

5 Repeated Roots

Let $p(x) = a_k x^k + \cdots + a_0$ be a polynomial in the variable x, where k is a positive integer and the coefficients $a_0, \ldots, a_k$ are from some field F (here, F can be $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, or GF(p) for some prime p). Let's define this polynomial's **derivative** to be the polynomial $p'(x) := ka_k x^{k-1} + \cdots + a_1 = \sum_{j=1}^{k} j a_j x^{j-1}$. We say that α is a **repeated root of** p if $p(x)$ can be factored as $(x - \alpha)^2 q(x)$ for some polynomial q. Show that α is a repeated root of p if and only if $p(\alpha) = p'(\alpha) = 0$.

[Note: You may be familiar with the derivatives of polynomials from studying calculus, but we are not using any calculus here, because it does not really make sense to perform calculus on finite fields! Think of the polynomial's derivative as a formal definition, i.e., in this context, it has nothing to do with rate of change, etc. In particular, you should not use any calculus rules such as the product rule without proof.]

Solution:

Recall Claim 1 from Note 8, Page 4, which says that α is a root of p if and only if $p(x) = (x - \alpha)r(x)$ for some polynomial r.

Assume first that α is a repeated root of p; we'll prove that it is a root of p' as well. Applying

Claim 1 twice gives us $p(x) = (x-\alpha)^2 q(x)$. Note that if we write $q(x) = \sum_{j=0}^{m} b_j x^j$,

$$p(x) = (x^2 - 2\alpha x + \alpha^2) \sum_{j=0}^{m} b_j x^j = \sum_{j=0}^{m} b_j (x^{j+2} - 2\alpha x^{j+1} + \alpha^2 x^j),$$

$$p'(x) = \sum_{j=1}^{m} b_j \big((j+2)x^{j+1} - 2\alpha(j+1)x^j + \alpha^2 j x^{j-1}\big) + b_0(2x - 2\alpha),$$

$$p'(\alpha) = \sum_{j=1}^{m} b_j \big((j+2)\alpha^{j+1} - 2(j+1)\alpha^{j+1} + j\alpha^{j+1}\big) = 0,$$

as we had hoped. Conversely, assume that $p(\alpha) = p'(\alpha) = 0$. We'll prove that α is a repeated root. Using Claim 1 again, we can write $p(x) = (x-\alpha)r(x)$, and it suffices to prove that α is a root of r, since (applying Claim 1 a final time) this will mean $r(x) = (x-\alpha)\tilde{q}(x)$, and thus $p(x) = (x-\alpha)^2\tilde{q}(x)$. Writing $r(x) = \sum_{j=0}^{\ell} c_j x^j$, we have

$$p(x) = (x-\alpha) \sum_{j=0}^{\ell} c_j x^j = \sum_{j=0}^{\ell} c_j (x^{j+1} - \alpha x^j),$$

$$p'(x) = \sum_{j=1}^{\ell} c_j \big((j+1)x^j - \alpha j x^{j-1}\big) + c_0,$$

$$p'(\alpha) = \sum_{j=1}^{\ell} c_j \big((j+1)\alpha^j - j\alpha^j\big) + c_0 = \sum_{j=1}^{\ell} c_j \alpha^j + c_0 = r(\alpha) = 0,$$

and so α is a root of r.

6 Green Eggs and Hamming

The *Hamming distance* between two length-n bit strings b_1 and b_2 is defined as the minimum number of bits in b_1 you need to flip in order to get b_2. For example, the Hamming distance between 101 and 001 is 1 (since you can just flip the first bit), while the Hamming distance between 111 and 000 is 3 (since you need to flip all three bits).

(a) Sam-I-Am has given you a list of n situations, and wants to know in which of them you would like green eggs and ham. You are planning on sending him your responses encoded in a length n bit string (where a 1 in position i says you would like green eggs and ham in situation i, while a 0 says you would not), but the channel you're sending your answers over is noisy and sometimes corrupts a bit. Sam-I-Am proposes the following solution: you send a length $n+1$ bit string, where the $(n+1)$st bit is the XOR of all the previous n bits (this extra bit is called the parity bit). If you use this strategy, what is the minimum Hamming distance between any two valid bit strings you might send? Why does this allow Sam-I-Am to detect an error? Can he correct the error as well?

(b) If the channel you are sending over becomes more noisy and corrupts two of your bits, can Sam-I-Am still detect the error? Why or why not?

(c) If you know your channel might corrupt up to k bits, what Hamming distance do you need between valid bit strings in order to be sure that Sam-I-Am can detect when there has been a corruption? Prove as well that that your answer is tight—that is, show that if you used a smaller Hamming distance, Sam-I-Am might not be able to detect when there was an error.

(d) Finally, if you want to *correct* up to k corrupted bits, what Hamming distance do you need between valid bit strings? Prove that your condition is sufficient.

Solution:

(a) The minimum Hamming distance is 2. In order to prove this, we need to show both that there exits two valid strings we could send that have a Hamming distance of 2 from eachother, and that there are no valid strings we could send that are at Hamming distance 1. The former is easy: if we do not like green eggs and ham in any situation, our answer to Sam-I-Am would just be $00\ldots0$. Since the XOR of n zeros is 0, our $(n+1)$st digit would also be a zero, so we would send a message containing $n+1$ zeros. In addition, if we only like green eggs and ham in the nth situation, but not any of the previous ones, our answer to Sam-I-Am would be $00\ldots01$, meaning that our message would be $n-1$ zeros followed by two ones (since the XOR of $0\cdots010\cdots0$ is one). These two bit strings are at Hamming distance 2 from one another, which is what we wanted to show.

To show that there is no pair of valid strings at Hamming distance 1, there are two cases to consider: either the bit that is different between them is in the first n, or it is the last (parity) bit. In the former case, flipping one of the n bits in our answer also flips the XOR of all our bits, meaning that the parity bit would have to change. However, both strings have to have the same parity bit, so one of the strings must have an incorrect parity bit and thus be invalid. In the second case, we know that the two strings share the same first n bits, but have different parity bits. However, the parity bit is uniquely determined by the first n bits, so one of the two strings must have an incorrect parity bit. Thus, in either case, one of the two strings must be invalid, so we can conclude that now two valid strings can have Hamming distance 2.

Finally, why is this relevant to error detecting? Let's call the list of possible strings you could send to Sam-I-Am your *codebook*. If no two strings in your codebook are within Hamming distance 2 of one another, then upon receiving a string with at most one bit flipped, Sam-I-Am can detect the error: if a bit was flipped, the new string is at Hamming distance 1 from the intended string, and thus is no longer in the codebook.

(b) Sam-I-Am cannot necessarily still detect the error. Because two valid strings can be a Hamming distance of two from eachother, it is possible that when the channel flips two bits, we end up at another valid string. As an example, say $n = 3$, Sam-I-Am received the string 1100. This message could have either resulted from us sending 1100 and not having any bits flipped or from sending 0000 and having the first two bits flipped. Thus, Sam-I-Am has no idea whether or not an error occurred.

(c) You need a Hamming distance of $k+1$ between valid strings. If no valid strings are closer than $k+1$, then there is no way to get from one valid code word to another by corrupting at most

k bits. Thus, if Sam-I-Am gets a valid string, he can conclude that your message got through without error; otherwise, he has to ask you to send it again.

We also have to prove that this is tight, meaning that if our minimum Hamming distance was k or smaller, Sam-I-Am might not be able to detect an error. If there existed two valid strings b_1 and b_2 at a Hamming distance of k or less from each other and Sam-I-Am received b_1, he wouldn't know if we had sent b_1 and it had gotten through uncorrupted, or if we had tried to send b_2 and the proper bits had gotten flipped to make it look like b_1. Thus, if the minimum Hamming distance was k or less, Sam-I-Am wouldn't necessarily be able to detect whether or not error(s) occurred.

(d) You need a minimum Hamming distance of $2k+1$. The idea is that, upon receiving your possibly corrupted message, Sam-I-Am finds the string in the codebook closest in Hamming distance to the one he received. To prove that this works, we need to know that if we start at a bit string b and flip $\ell \leq k$ different bits, the Hamming-nearest string in our codebook is still b. Call the corrupted string b' and assume (looking for contradiction) that some string c in our codebook is closer to b' than b is to b'. This means c can be obtained from b' by flipping $m \leq \ell \leq k$ bits, and b' can be obtained from b by flipping $\ell \leq k$ bits. Concatenating these sequences of bit flips, we can get c from b by flipping no more than $m+\ell \leq 2k$ bits. But this means the Hamming distance between b and c is less than $2k+1$, and we assumed that the minimum such distance was $2k+1$.