CSCI-762 Assignment #4
Thomas Bottom Due 3 March 2020

Solve parts 1 and 2 by hand, use computer help to solve 3 and 4. In all exercises explain what you did and show the details of your work. Attach source code as applicable.

1. The polynomial x^4+x+1 is irreducible in $Z_2[x]$. Compute $x^{15} \mod x^4+x+1$ in $Z_2[x]$, i.e. in the Galois field GF(2^4). Use two approaches: standard square-and-multiply for exponent 15, and for the exponent written as (16 - 1).

2. Find all irreducible polynomials in $Z_2[x]$ of degree 5. You can assume that the polynomial x^2+x+1 is the only irreducible binary quadratic (you do not need to show that).

3. Solve exercise 6.12 pages 277/278 (7.12 pages 305/306). You can use this representation of the Galois field GF(27)

4. Let $p = 131$. First show that (x^2+1) is irreducible in $Z_p[x]$ - this can be done by hand using the Euler criterion for quadratic residuosity. Next, represent GF(p^2) by polynomials modulo (x^2+1). Use naive algorithm to find the number of elements of each order in GF(p^2), and list 10 smallest monic primitive (generators with coefficient 1 in the highest degree term) elements. Illustrate the computation of discrete logarithm of (x+101) with base equal to the smallest such generator using Shanks' algorithm.

1.

Compute $x^{15} \mod x^4+x+1$ in $Z_2[x]$

operation	power	result	mod x^4+x+1
start	1	x	x
square	2	x^2	x^2
$*x$	3	x^3	x^3
square	6	x^6	x^3+x^2
$*x$	7	x^4+x^3	x^3+x+1
square	14	x^6+x^2+1	x^3+1
$*x$	15	x^4+x	$\mathbf{1}$

Compute $x^{16-1} \mod x^4+x+1$ in $Z_2[x]$

$x^{16-1} \equiv x^{16} * x^{-1} \mod x^4+x+1$
$x^{-1} \equiv x^{2^4-2} \equiv x^{14} \equiv x^3+1$ in GF(2^4)

operation	power	result	mod x^4+x+1
start	1	x	x
square	2	x^2	x^2
square	4	x^4	$x+1$
square	8	x^2+1	x^2+1
square	16	x^4+1	x
$*(x^3+1)$	15	x^4+x	$\mathbf{1}$

2. Find all irreducible polynomials in $Z_2[x]$ of degree 5. You can assume that the polynomial x^2+x+1 is the only irreducible binary quadratic (you do not need to show that).

Start with the following irreducible linear and given quadratic factors:

x	$x+1$	x^2+x+1

Check polynomials for irreducibility. Immediately rule out any polynomial not ending in + 1 because it is divisible by x. Polynomials of degree 3 are irreducible if they do not have linear factors. Polynomials up to degree 5 must have a factor of degree 1 or 2 (since any factor of degree 3 or 4 would only produce a polynomial of degree 5 when multiplied by a factor of degree 2 or 1 respectively), so it suffices to check for linear and quadratic factors for these polynomials.

Irreducible polynomials have been marked in **bold** and begin with an asterisk (*).

x^3+1	$(x+1)(x^2+x+1)$	*$\boldsymbol{x^5+x^2+1}$	no linear or quadratic factors
*$\boldsymbol{x^3+x+1}$	no linear factors, degree 3	x^5+x^2+x+1	has linear factor; f(1) = 0
*$\boldsymbol{x^3+x^2+1}$	no linear factors, degree 3	*$\boldsymbol{x^5+x^3+1}$	no linear or quadratic factors
x^3+x^2+x+1	has linear factor; f(1) = 0	x^5+x^3+x+1	has linear factor; f(1) = 0
x^4+1	$(x+1)(x^3+x^2+x+1)$	$x^5+x^3+x^2+1$	has linear factor; f(1) = 0
*$\boldsymbol{x^4+x+1}$	no linear or quadratic factors	*$\boldsymbol{x^5+x^3+x^2+x+1}$	no linear or quad factors
x^4+x^2+1	$(x^2+x+1)^2$	x^5+x^4+1	$(x^2+x+1)(x^3+x+1)$
x^4+x^2+x+1	has linear factor; f(1) = 0	x^5+x^4+x+1	has linear factor; f(1) = 0
*$\boldsymbol{x^4+x^3+1}$	no linear or quadratic factors	$x^5+x^4+x^2+1$	has linear factor; f(1) = 0
x^4+x^3+x+1	has linear factor; f(1) = 0	*$\boldsymbol{x^5+x^4+x^2+x+1}$	no linear or quad factors
$x^4+x^3+x^2+1$	has linear factor; f(1) = 0	$x^5+x^4+x^3+1$	has linear factor; f(1) = 0
*$\boldsymbol{x^4+x^3+x^2+x+1}$	no linear or quad factors	*$\boldsymbol{x^5+x^4+x^3+x+1}$	no linear or quad factors
x^5+1	has linear factor; f(1) = 0	*$\boldsymbol{x^5+x^4+x^3+x^2+1}$	no linear or quad factors
x^5+x+1	$(x^3+x^2+1)(x^2+x+1)$	$x^5+x^4+x^3+x^2+x+1$	has linear factor; f(1) = 0

3. We give an example of the ElGamal Cryptosystem implemented in F_{3^3}. The polynomial x^3+2x^2+1 is irreducible over $Z_3[x]$ and hence $Z_3[x]/(x^3+2x^2+1)$ is the field F_{3^3}. We can associate the 26 letters of the alphabet with the 26 nonzero field elements, and thus encrypt ordinary text in a convenient way. We will use a lexicographic ordering of the (nonzero) polynomials to set up the correspondence.

Suppose Bob used $\alpha = x$ and $a = 11$ in an ElGamal Cryptosystem; then $\beta = x+2$. Show how Bob will decrypt the following string of ciphertext:
(K,H)(P,X)(N,K)(H,R)(T,F)(V,Y)(E,H)(F,A)(T,W)(J,D)(U,J)

A solution to perform this decryption was implemented in C++ and can be found in Appendix A.

The decryption is performed as $d_k(y_1,y_2) = y_2(y_1^a)^{-1} \mod x^3+2x^2+1$ using the provided lookup table for multiplication in GF(27).

The decrypted text is **GALOISFIELD**.

4. Let $p = 131$. First show that $(x^2 + 1)$ is irreducible in $Z_p[x]$ - this can be done by hand using the Euler criterion for quadratic residuosity. Next, represent GF(p^2) by polynomials modulo $(x^2 + 1)$. Use naive algorithm to find the number of elements of each order in GF(p^2), and list 10 smallest monic primitive (generators with coefficient 1 in the highest degree term) elements. Illustrate the computation of discrete logarithm of (x+101) with base equal to the smallest such generator using Shanks' algorithm.

Proof that $x^2 + 1$ is irreducible in $Z_{131}[x]$

$x^2 + 1 \equiv 0 \mod 131$
$x^2 \equiv 130 \mod 131$
$130^{130/2} \equiv 130 \mod 131$
130 is not a quadratic residue mod 131

For this question I implemented basic polynomial multiplication, subtraction, exponentiation, and modular reduction in Python. I chose Python for this question because of the convenient syntax when working with list representations of polynomials, and the fact that the Python modulus operator conveniently returns the smallest positive congruence (as opposed to C which may return a negative value in some cases).

The list representation of polynomials is such that the each entry in a list represents the coefficient of the power at that index, e.g. [1,0,1] would represent $1 * x^0 + 0 * x^1 + 1 * x^2 \equiv x^2 + 1$. This implementation is not the most efficient possible, but the code is both simple and intuitive.

How many elements of each order in GF(131^2)?

order	numPolynomials	order	numPolynomials	order	numPolynomials
3	2	104	48	572	240
6	2	110	40	660	160
8	4	120	32	715	480
11	10	132	40	780	192
12	4	143	120	858	240
15	8	156	48	1144	480
22	10	165	80	1320	320
24	8	195	96	1430	480
30	8	220	80	1560	384
33	20	264	80	1716	480
39	24	286	120	2145	960
40	16	312	96	2860	960
44	20	330	80	3432	960
55	40	390	96	4290	960
60	16	429	240	5720	1920
66	20	440	160	8580	1920
78	24	520	192	17160	3840
88	40				

List the 10 smallest monic primitive polynomial elements:

$x + 3$ $x + 5$ $x + 7$
$x + 9$ $x + 11$ $x + 12$
$x + 14$ $x + 15$ $x + 23$
$x + 26$

Illustrate the computation of discrete logarithm of (x+101) with base equal to the smallest such generator using Shanks' algorithm.

The smallest generator is $x + 3$.

Compute $\text{Log}_{x+3}(x + 101)$ using Shanks' Algorithm.

$G = Z_{131^2}[x]$, $n = 17160$, $\alpha = x + 3$, $\beta = x + 101$
$m = 131$
$\alpha^{-1} \equiv \alpha^{p^n-2} \equiv \alpha^{131^2-2} \equiv 13x + 92$

I used the code from Appendix B interactively to run Shanks' Algorithm by hand.

```
>>> import q4
>>> L1 = []
>>> L2 = []
>>> alpha    = [3,1]       # x + 3
>>> beta     = [101, 1]    # x + 101
>>> modulus = [1,0,1]      # x^2 + 1
>>> alpha_inverse = q4.polyPow(alpha, (131**2)-2, modulus)
>>> for j in range(1,m):
...     L1.append((j,q4.polyPow(alpha, m*j, modulus)))
...
>>> L1.sort(key = lambda tuple : tuple[1])
>>> for i in range(1,m):
...     acc = q4.polyPow(alpha_inverse, i, modulus)
...     acc = q4.polyMod(q4.polyMult(acc, beta), modulus)
...     L2.append((i,acc))
...
>>> L2.sort(key = lambda tuple : tuple[1])
>>> for j,y1 in L1:
...     for i,y2 in L2:
...             if q4.polyEquals(y1,y2):
...                     print ((m*j) + i) % 17160
...                     break
...
3517
```

Verify the result

```
>>> q4.polyPow(alpha, 3517, modulus)
[101, 1]
```

$$\boldsymbol{Log_{x+3}(x + 101) = 3517}$$

A Question 3 Source Code

```
/// @author Thomas Bottom <tdb4197@cs.rit.edu>
/// CSCI-762 Assignment 4

#include <cassert> // for sanity checks
#include <cctype>  // for tolower
#include <cstring> // for strlen
#include <cstdio>  // for printf

const int gf27MultTable [] = {
//           0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26
/*  0: */    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
/*  1: */    0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
/*  2: */    0,  2,  1,  6,  8,  7,  3,  5,  4, 18, 20, 19, 24, 26, 25, 21, 23, 22,  9, 11, 10, 15, 17, 16, 12, 14, 13,
/*  3: */    0,  3,  6,  9, 12, 15, 18, 21, 24, 11, 14, 17, 20, 23, 26,  2,  5,  8, 19, 22, 25,  1,  4,  7, 10, 13, 16,
/*  4: */    0,  4,  8, 12, 16, 11, 24, 19, 23, 20, 21, 25,  5,  6,  1, 17,  9, 13, 10, 14, 15, 22, 26, 18,  7,  2,  3,
/*  5: */    0,  5,  7, 15, 11, 13, 21, 26, 19,  2,  4,  6, 17, 10, 12, 23, 25, 18,  1,  3,  8, 16,  9, 14, 22, 24, 20,
/*  6: */    0,  6,  3, 18, 24, 21,  9, 15, 12, 19, 25, 22, 10, 16, 13,  1,  7,  4, 11, 17, 14,  2,  8,  5, 20, 26, 23,
/*  7: */    0,  7,  5, 21, 19, 26, 15, 13, 11,  1,  8,  3, 22, 20, 24, 16, 14,  9,  2,  6,  4, 23, 18, 25, 17, 12, 10,
/*  8: */    0,  8,  4, 24, 23, 19, 12, 11, 16, 10, 15, 14,  7,  3,  2, 22, 18, 26, 20, 25, 21, 17, 13,  9,  5,  1,  6,
/*  9: */    0,  9, 18, 11, 20,  2, 19,  1, 10, 17, 26,  8, 25,  7, 16,  6, 15, 24, 22,  4, 13,  3, 12, 21, 14, 23,  5,
/* 10: */    0, 10, 20, 14, 21,  4, 25,  8, 15, 26,  6, 16,  1, 11, 18, 12, 22,  5, 13, 23,  3, 24,  7, 17,  2,  9, 19,
/* 11: */    0, 11, 19, 17, 25,  6, 22,  3, 14,  8, 16, 24, 13, 21,  5, 18,  2, 10,  4, 12, 23,  9, 20,  1, 26,  7, 15,
/* 12: */    0, 12, 24, 20,  5, 17, 10, 22,  7, 25,  1, 13, 15, 18,  3,  8, 11, 23, 14, 26,  2,  4, 16, 19, 21,  6,  9,
/* 13: */    0, 13, 26, 23,  6, 10, 16, 20,  3,  7, 11, 21, 18,  4, 17, 14, 24,  1,  5, 15, 19, 25,  2, 12,  9, 22,  8,
/* 14: */    0, 14, 25, 26,  1, 12, 13, 24,  2, 16, 18,  5,  3, 17, 19, 20,  4, 15, 23,  7,  9, 10, 21,  8,  6, 11, 22,
/* 15: */    0, 15, 21,  2, 17, 23,  1, 16, 22,  6, 12, 18,  8, 14, 20,  7, 13, 19,  3,  9, 24,  5, 11, 26,  4, 10, 25,
/* 16: */    0, 16, 23,  5,  9, 25,  7, 14, 18, 15, 22,  2, 11, 24,  4, 13, 20,  6, 21,  1, 17, 26,  3, 10, 19,  8, 12,
/* 17: */    0, 17, 22,  8, 13, 18,  4,  9, 26, 24,  5, 10, 23,  1, 15, 19,  6, 14, 12, 20,  7, 11, 25,  3, 16, 21,  2,
/* 18: */    0, 18,  9, 19, 10,  1, 11,  2, 20, 22, 13,  4, 14,  5, 23,  3, 21, 12, 17,  8, 26,  6, 24, 15, 25, 16,  7,
/* 19: */    0, 19, 11, 22, 14,  3, 17,  6, 25,  4, 23, 12, 26, 15,  7,  9,  1, 20,  8, 24, 16, 18, 10,  2, 13,  5, 21,
/* 20: */    0, 20, 10, 25, 15,  8, 14,  4, 21, 13,  3, 23,  2, 19,  9, 24, 17,  7, 26, 16,  6, 12,  5, 22,  1, 18, 11,
/* 21: */    0, 21, 15,  1, 22, 16,  2, 23, 17,  3, 24,  9,  4, 25, 10,  5, 26, 11,  6, 18, 12,  7, 19, 13,  8, 20, 14,
/* 22: */    0, 22, 17,  4, 26,  9,  8, 18, 13, 12,  7, 20, 16,  2, 21, 11,  3, 25, 24, 10,  5, 19, 14,  6, 23, 15,  1,
/* 23: */    0, 23, 16,  7, 18, 14,  5, 25,  9, 21, 17,  1, 19, 12,  8, 26, 10,  3, 15,  2, 22, 13,  6, 20, 11,  4, 24,
/* 24: */    0, 24, 12, 10,  7, 22, 20, 17,  5, 14,  2, 26, 21,  9,  6,  4, 19, 16, 25, 13,  1,  8, 23, 11, 15,  3, 18,
/* 25: */    0, 25, 14, 13,  2, 24, 26, 12,  1, 23,  9,  7,  6, 22, 11, 10,  8, 21, 16,  5, 18, 20, 15,  4,  3, 19, 17,
/* 26: */    0, 26, 13, 16,  3, 20, 23, 10,  6,  5, 19, 15,  9,  8, 22, 25, 12,  2,  7, 21, 11, 14,  1, 24, 18, 17,  4
};

/// @return integer representation of c in GF(27)
inline int gf27CharToInt(const char c) {
    int lc = tolower(c);
    assert(lc >= 'a' && lc <= 'z');
    return lc - 96;
}

/// @return character representation of x in GF(27)
inline char gf27IntToChar(int x) {
    assert(x > 0 && x < 27);
    return (char)(x+96);
}

/// @return index of element at specified row and column in table of given width
inline int rowMajorIndex(int row, int col, int width) {
    assert(row<width); assert(col<width);
    return (width*row)+col;
}

/// @return  x * y in GF(27)
inline int gf27Mult(int x, int y) {
    assert(x<27); assert(y<27);
    // result is at table index in row major order
    return gf27MultTable[rowMajorIndex(x,y,27)];
}
```

```
/// @return x^e in GF(27)
int gf27Pow(int x, int e) {
    assert(x<27); assert(e>=0);
    if(0 == e) return 1;
    int y = 1;
    while(e > 1) {
        if(0 == e%2) {
            x = gf27Mult(x,x);
            e /= 2;
        } else {
            y = gf27Mult(x,y);
            x = gf27Mult(x,x);
            e = (e-1)/2;
        }
    }
    return gf27Mult(x,y);
}

/// @return the multiplicative inverse of x in GF(27)
inline int gf27FindInverse(int x) {
    assert(x<27);
    for(int i = 1; i < 27; ++i) {
        if(1 == gf27Mult(x,i)) return i;
    }
    // control should never reach here
    assert(false);
    return -1;
}

/// @return d_k(y1,y2) = y2(y1^a)^-1
int gf27Decrypt(int y1, int y2, int a) {
    int acc = gf27Pow(y1, a);    // (y1^a)
    acc = gf27FindInverse(acc); // (y1^1)^-1
    return gf27Mult(y2, acc);
}

int main(int argc, char ** argv) {
    // Decrypt the following ciphertexts
    // (K,H)(P,X)(N,K)(H,R)(T,F)(V,Y)(E,H)(F,A)(T,W)(J,D)(U,J)
    const char* ct = "khpxnkhrtfvyehfatwjduj";
    const int bound = strlen(ct);
    const int a = 11;

    for(int i = 0; i < bound; i+=2) {
        int y1  = gf27CharToInt(ct[i]);
        int y2  = gf27CharToInt(ct[i+1]);
        int pti = gf27Decrypt(y1, y2, a);
        char pt = gf27IntToChar(pti);
        printf("%c", pt);
    }
    printf("\n");
    return 0;
}
```

B Question 4 Source Code

```
# Author: Thomas Bottom <tdb4197@cs.rit.edu>
# CSCI-762 Assignment 4

# Polynomials are represented as lists where the index of the coefficient
# is the power of x is represents
# e.g. [1,2,3] represents 1*x^0 + 2*x^1 + 3 *x^2

base_modulus = 131
field_extension = 2
field_irreducible = [1,0,1] # x^2 + 1

# remove trailing zeros from the polynomial representation
def stripPoly(p):
    for i in xrange(len(p) - 1, -1, -1):
        if p[i] != 0:
            break
    del p[i + 1:]
    return p

# get the degree of the polynomial
def polyDegree(p):
    i = len(p) - 1
    while i >= 0:
        if p[i] != 0: break
        i -= 1
    return i

# Perform a polynomial multiplication with coefficients reduced by the base_modulus
def polyMult(p1, p2):
    result = []
    for i in range(len(p1)):
        for j in range(len(p2)):
            power = i+j
            coefficient = p1[i] * p2[j]
            # Ensure list is large enough to contain result
            if len(result) <= power: result.extend([0] * (power - len(result) + 1))
            result[power] = (result[power] + coefficient) % base_modulus
    return stripPoly(result)

# return polynomial p mod m (m must be monic)
def polyMod(p, m):
    dp = polyDegree(p) # degree of p
    dm = polyDegree(m) # degree of m
    while dp >= dm:    # do long division
        i = dp-dm          # degree of quotient q
        q = [0]*(i+1)      # create polynomial q such that
        q[i] = p[dp]       # p-(mq) reduces degree of p by 1
        p = polySub(p, polyMult(m, q))
        dp -= 1
    return stripPoly(p)
```

```
# Perform polynomial subtraction modulo the coefficient base modulus
def polySub(p1, p2):
    if len(p1) < len(p2): p1.extend([0] * (len(p2)-len(p1)))
    result = []
    for i in range(len(p2)):
        result.append((p1[i] - p2[i]) % base_modulus)
    return stripPoly(result)

# returns true iff polynomials a and b are equal
def polyEquals(a, b):
    a = stripPoly(a)
    b = stripPoly(b)
    return a == b

# return p^e mod m
def polyPow(p, e, m):
    y = [1]
    while e > 1: # square and multiply
        if 0 == e % 2:
            p = polyMod(polyMult(p,p), m)
            e = e / 2
        else:
            y = polyMod(polyMult(p, y), m)
            e = e - 1
    return polyMod(polyMult(p,y), m)

if __name__ == "__main__":
    orders = {}
    for a in range(1,base_modulus):
        for b in range(1,base_modulus):
            # test all polynomials of the form (ax + b), 0 < a < 131
            poly = [b, a]
            counter = 1 # counts the order of poly
            accumulator = poly
            while False == polyEquals(accumulator, [1]):
                counter += 1
                # accumulator = accumulator^counter
                accumulator = polyMod(polyMult(accumulator,poly), field_irreducible)
            if counter in orders:
                orders[counter].append(poly)
            else:
                orders[counter] = [poly]

    # print orders in latex table format
    orderlist = []
    for k,v in orders.iteritems(): orderlist.append( (k, len(v)))
    orderlist.sort(key = lambda t: t[0])
    print("order & numPolynomials \\")
    for t in orderlist: print("%d & %d \\\\" % t)

    print "\nSmallest 10 Monic Primitives"
    for i in range(10): print orders[17160][i]
```