GF(2^n)

CS6025 Data Encoding

Yizong Cheng

2-12-15

Bit Strings as Polynomials over Z_2

- Each non-negative integer has a binary form.
  - in Java, Integer.toBinaryString(n) returns the binary form of integer n
- The binary form shows what powers of 2 can sum up to the integer.
- It can also be viewed as a polynomial of coefficients in Z_2.
- Integers 8-15 are degree-3 polynomials, 16-31 are degree-4 polynomials, 128-255 are degree-7 and 256-511 degree-8.
- Addition of two polynomials is bitwise XOR of the bit strings.
- Multiplication is done with shifts and bitwise XOR.

3 x 7 = 9

- 3 (11 in binary) is the polynomial $x + 1$ (degree 1)
- 7 (111 in binary) is the polynomial x^2+x+1 (degree 2)
- 3 x 7 is 111 left shift by 1 (multiplied by x) and then add 111.
- (7 << 1) ^ 7 is 1110 bitwise XOR 111 and that is 1001 or 9.
- The degree of the product of two polynomials of degrees m and n is m+n.

Polynomials over Z_2

```
int add (int a, int b){
  return a ^ b;
}

int multiply (int a, int b){
  int product = 0;
  for (; b > 0; b >>= 1){
    if ((b & 1) > 0) product ^= a;
    a <<= 1;
  }
  return product;
}
```

Addition of Polynomials over Z_2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Multiplication of Polynomials over Z_2

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
3	6	5	12	15	10	9	24	27	30	29	20	23	18	17
4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
5	10	15	20	17	30	27	40	45	34	39	60	57	54	51
6	12	10	24	30	20	18	48	54	60	58	40	46	36	34
7	14	9	28	27	18	21	56	63	54	49	36	35	42	45
8	16	24	32	40	48	56	64	72	80	88	96	104	112	120
9	18	27	36	45	54	63	72	65	90	83	108	101	126	119
10	20	30	40	34	60	54	80	90	68	78	120	114	108	102
11	22	29	44	39	58	49	88	83	78	69	116	127	98	105
12	24	20	48	60	40	36	96	108	120	116	80	92	72	68
13	26	23	52	57	46	35	104	101	114	127	92	81	70	75
14	28	18	56	54	36	42	112	126	108	98	72	70	84	90
15	30	17	60	51	34	45	120	119	102	105	68	75	90	85

Irreducible Polynomials

- A polynomials is reducible if it is the product of two polynomials of degree at least one.
- It is irreducible if it is not in the “interior” of the multiplication table.

Irreducible Polynomials over Z_2

```
void irreducibles(){  // up to degree 9
    HashSet<Integer> hset = new HashSet<Integer>(100);
    for (int i = 2; i < 512; i++)
      for (int j = i; j < 512; j++) hset.add(multiply(i, j));
    for (int i = 0; i < 1024; i++)
       if (!hset.contains(i)) System.out.print(i + " ");
    System.out.println();
   }
  }
```

Irreducible Polynomials over Z_2

```
0 1 2 3 7 11 13 19 25 31 37 41 47 55 59 61 67 73 87 91 97
103 109 115 117 131 137 143 145 157 167 171 185 191 193
203 211 213 229 239 241 247 253 283 285 299
```

```
0 1 2 3 7 b d 13 19 1f 25 29 2f 37 3b 3d 43 49 57 5b 61
67 6d 73 75 83 89 8f 91 9d a7 ab b9 bf c1
cb d3 d5 e5 ef f1 f7 fd 11b 11d 12b
```

0 1 10 11 111 1011 1101 10011 11001 11111 100101 101001 101111 110111 111011
111101 1000011 1001001 1010111 1011011 1100001 1100111 1101101 1110011 1110101
10000011 10001001 10001111 10010001 10011101 10100111 10101011 10111001 10111111
11000001 11001011 11010011 11010101 11100101 11101111 11110001 11110111 11111101
100011011 100011101 100101011

GF(2^n) via Polynomials over Z_2

- We may represent the 2^n elements in the finite field GF(2^n) as integers 0 to 2^n-1, or all polynomials over Z_2 with degrees less than n.
- Addition of these polynomials/integers can be done with bitwise XOR.
- Modulo any polynomial of degree n after the multiplication will result in an element in 0 to 2^n-1.
- However, only an irreducible polynomial of degree n used as the modulus can generate a multiplication table with multiplicative inverse to all non-zero elements.
- The finite fields generated with different irreducible polynomials are homomorphic in the sense that the multiplication table is the same after the a certain permutation of the symbol representation.

Multiplication of Polynomials Modulo m

```
static final int numberOfBits = 3; // or 4, 7, 8
static final int fieldSize = 1 << numberOfBits;

int modMultiply(int a, int b, int m){
   int product = 0;
   for (; b > 0; b >>= 1){
     if ((b & 1) > 0) product ^= a;
     a <<= 1;
     if ((a & fieldSize) > 0) a ^= m;
   }
   return product;
 }
```

Multiplication mod 11, 12, and 13

1	2	3	4	5	6	7
2	4	6	3	1	7	5
3	6	5	7	4	1	2
4	3	7	6	2	5	1
5	1	4	2	7	3	6
6	7	1	5	3	2	4
7	5	2	1	6	4	3

1	2	3	4	5	6	7
2	4	6	4	6	0	2
3	6	5	0	3	6	5
4	4	0	4	0	0	4
5	6	3	0	5	6	3
6	0	6	0	6	0	6
7	2	5	4	3	6	1

1	2	3	4	5	6	7
2	4	6	5	7	1	3
3	6	5	1	2	7	4
4	5	1	7	3	2	6
5	7	2	3	6	4	1
6	1	7	2	4	3	5
7	3	4	6	1	5	2

Power and Order

α^k is α multiplied with itself k times.

- The smallest k that makes $\alpha^k = 1$ is called the order of α.
- A primitive element of GF(q) is one with order q-1.
- It is also a generator of the cyclic multiplicative group of GF(q).

$\alpha^{q-1} = 1$ in GF(q)

- The multiplication table of GF(q) is a q-1 by q-1 matrix in which each row is a permutation of the first row.
- The product of elements in αth row is the product of all nonzero elements of GF(q) and also the product of the products of α with all nonzero elements.
- $\Pi\beta_i = \Pi(\alpha\beta_i) = \alpha^{q-1}\Pi\beta_i$ and $\alpha^{q-1} = 1$.
- The order of any element in GF(q) must be a factor of q-1.
  - For q=8, q-1 = 7 and the only orders are 1 and 7.

```
for (int i = 0; i < fieldSize; i++){
  for (int j = 0; j < fieldSize; j++) System.out.print(add(i, j) + " ");
  System.out.println();
} // addition table
System.out.println();

for (int i = 1; i < fieldSize; i++){
  for (int j = 1; j < fieldSize; j++)
    System.out.print(modMultiply(i, j, irreducible) + " ");
  System.out.println();
} // multiplication table
System.out.println();

for (int i = 1; i < fieldSize; i++){
  int power = 1;
  for (int j = 1; j < fieldSize; j++){
      power = modMultiply(i, power, irreducible);
      System.out.print(power + " ");
  } // power table
  System.out.println();
}
```

Power Table for GF(8) mod 11

```
1 1 1 1 1 1 1
2 4 3 6 7 5 1  // powers of 2
3 5 4 7 2 6 1  // powers of 3
4 6 5 2 3 7 1
5 7 6 3 4 2 1
6 2 7 4 5 3 1
7 3 2 5 6 4 1  // 2-7 are primitive
```

Discrete Logarithm in GF(q)

- If α is a primitive element of GF(q), then α^1, α^2,…,α^{q-1} are all different.
- Given a nonzero β in GF(q), we can ask which j makes $\alpha^j = \beta$.
- This j is called the discrete logarithm of β base α in GF(q).
- The antilogarithm base α of j is α^j.

Log and alog Tables

- Given the base α, we can build the antilog (alog) table by multiplying α repeatedly.
- By swapping index and content we have the log table. Make alog[0] = 1.
- To multiply β and γ, we use the log table to find log[β] and log[γ], add them up mod q-1, and then apply the alog table.
- $\beta\gamma$ = alog[(log[β] + log[γ]) % (q-1)]

Logarithm for Multiplication

$$\begin{tikzcd}
x, y \arrow[r, "\text{multiply}"] \arrow[d, "\text{log}"'] & xy \\
\log(x), \log(y) \arrow[r, "\text{add}"'] & \log(x)+\log(y)=\log(xy) \arrow[u, "\text{antilog}"']
\end{tikzcd}$$

Multiplication with Discrete Logarithm

```
void makeLog(){
    alog[0] = 1;
    for (int i = 1; i < fieldSize; i++)
      alog[i] = modMultiply(logBase, alog[i - 1], irreducible);
    for (int i = 1; i < fieldSize; i++) log[alog[i]] = i;
  }

  int logMultiply(int a, int b){
    return (a == 0 || b == 0) ? 0 : alog[(log[a] + log[b]) % (fieldSize - 1)];
  }

  int multiplicativeInverse(int a){
    return alog[fieldSize - 1 - log[a]];
  }
```

Global Constants

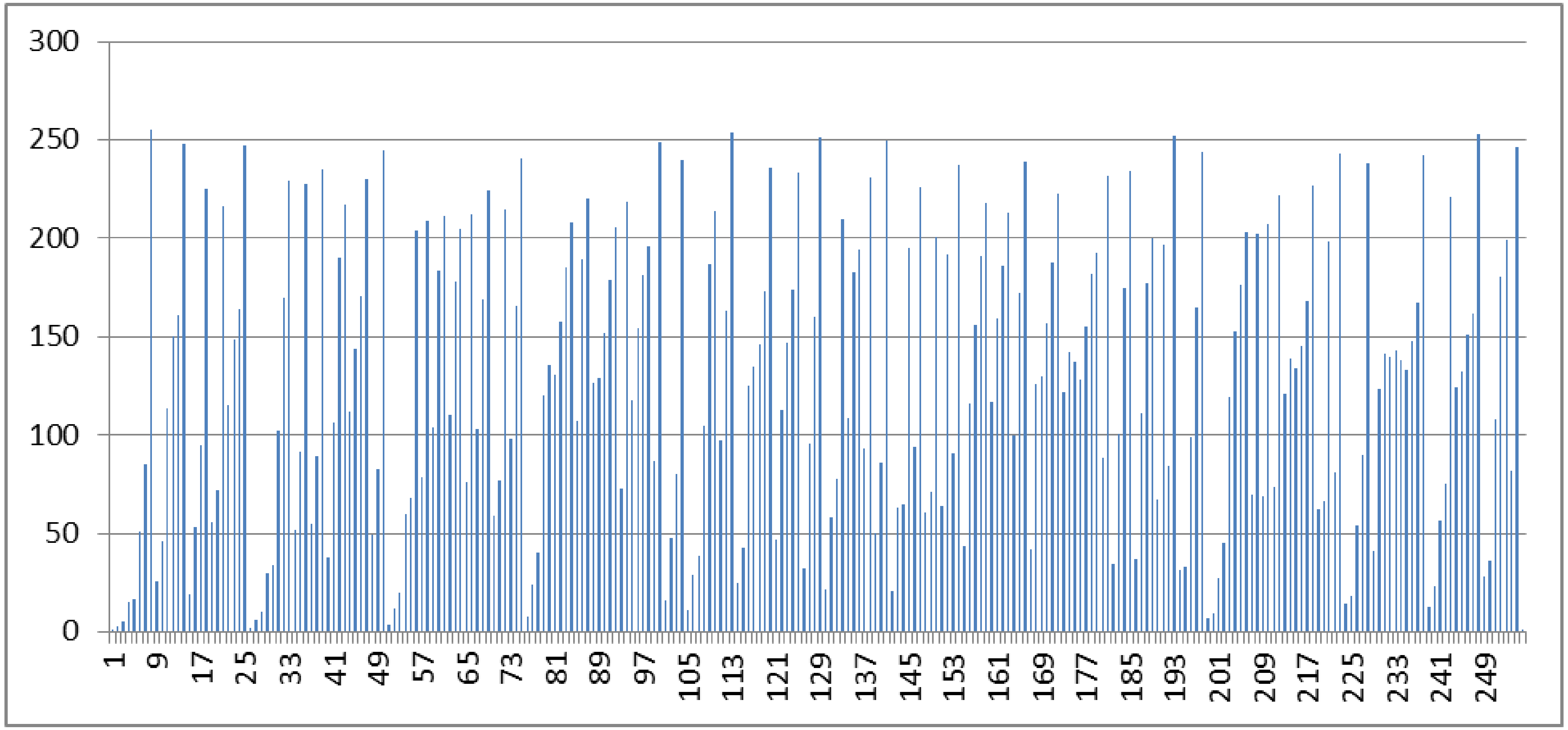
```
public class H10{

  static final int numberOfBits = 3;
  static final int fieldSize = 1 << numberOfBits;
  static final int irreducible = 11;
  static final int logBase = 6;
  int[] alog = new int[fieldSize];
  int[] log = new int[fieldSize];
```

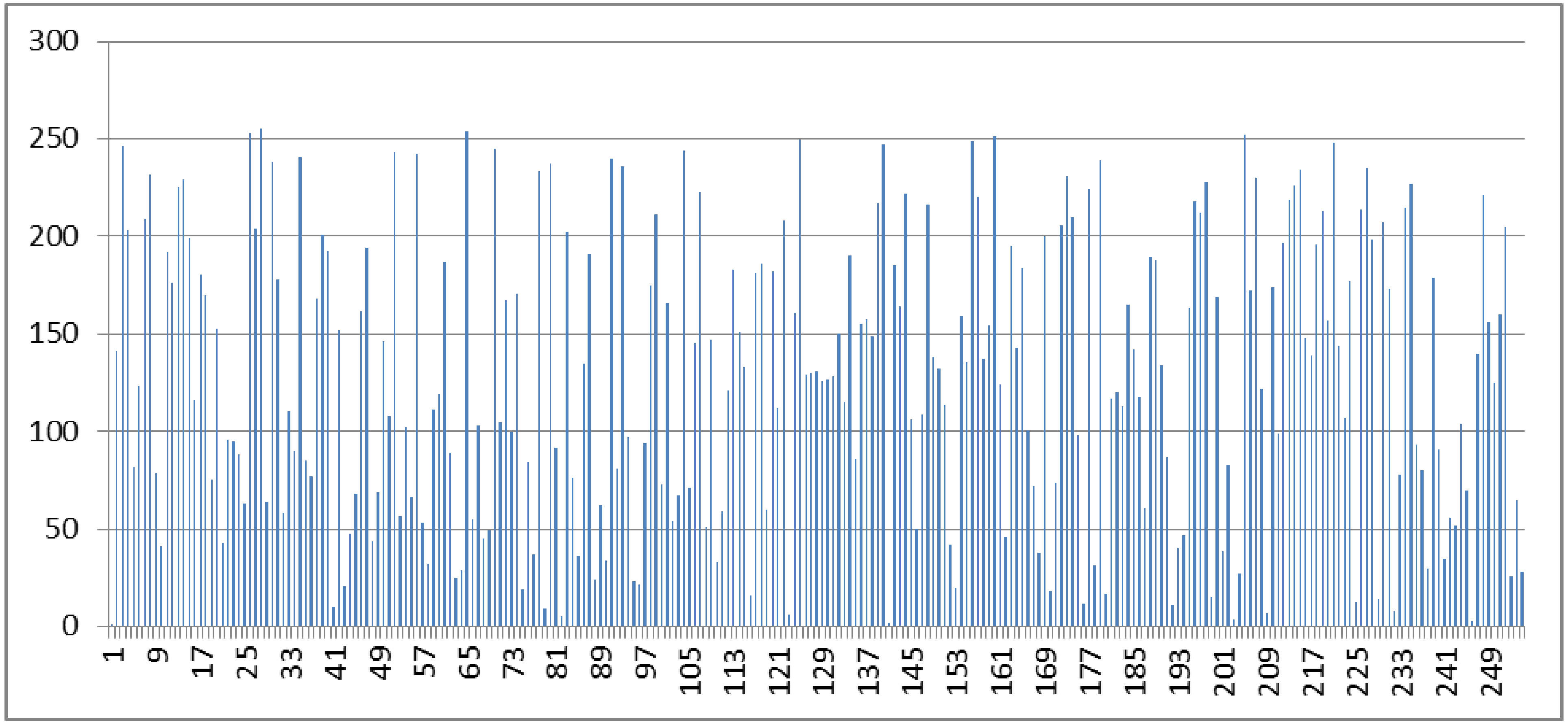
Possible Orders in GF(2^n)

- $n = 3$, $2^n-1 = 7$, possible orders 1, 7
- $n = 4$, $2^n-1 = 15$, possible orders 1, 3, 5, 15
- $n = 5$, $2^n-1 = 31$, possible orders 1, 31
- $n = 6$, $2^n-1 = 63$, possible orders 1, 3, 31, 63
- $n = 7$, $2^n-1 = 127$, possible orders 1, 127
- $n = 8$, $2^n-1 = 255$, possible orders 1, 3, 5, 17, 255
- Need only to check possible non-trivial orders to find primitive elements of the field.

Powers of 3 in GF(2^8) mod 0x11b



Multiplicative Inverses in GF(2^8)



Homework 10: due 2-18-15

- Modify H10.java to generate the multiplication and power tables for GF(16) with a irreducible polynomial of degree 4 (report also the irreducible you choose).
- List all primitive elements in GF(16) with the multiplication table specified above.
- Pick a primitive element as logBase to generate discrete log and alog arrays. Show a few examples using logMultiply and compare those with the multiplication table.
- Choose a irreducible polynomial of degree 8 and list all primitive elements in GF(256).