Decoding Reed-Solomon

CS6025 Data Encoding

Yizong Cheng

4-14-15

The Block of Codewords as a polynomial

- QR version 1 with H error-correction level is a block code (26, 9, 8).
- Among the 26 bytes, as elements in $GF(2^8)$, nine bytes are data codewords, 17 bytes are error correction codewords computed from the data codewords, and correction can be made if the errors occur only in up to eight codewords (both data and error correction ones).
- The 26 codewords form a polynomial $C(x)$ over $GF(2^8)$ that is a multiple of the generator polynomial $G(x)$, which has degree 17 and have $1, \alpha, \alpha^2, \ldots, \alpha^{16}$ as its roots. (α is a primitive element of $GF(2^8)$ and is chosen to be 2 in our case.)
- The 17 syndromes are $S_i = C(\alpha^i)$ for $i = 0, \ldots, 16$. They are all zero because $C(x)$ is a multiple of $G(x)$ and $G(\alpha^i) = 0$ for all these i's.

Up to Eight Erroneous Codewords

- The error polynomial $E(x)$ is added (XORed) to $C(x)$ after transmission/ strage and the result is $C'(x) = C(x) + E(x)$.
- If $E(x)$ is not zero, the syndromes are no longer zero, too.
- $S_i = C'(\alpha^i) = C(\alpha^i) + E(\alpha^i) = E(\alpha^i)$ because $C(\alpha^i) = 0$ for $i = 0,...,16$
- Let eight codewords at locations $p(j)$ for $j = 0,...,7$ be the ones with errors $E(j)$.
- $E(x) = E(0)x^{p(0)} + E(1)x^{p(1)} + ... + E(7)x^{p(7)}$.
- $S_i = E(\alpha^i) = E(0)\,\alpha^{i\,p(0)} + E(1)\,\alpha^{i\,p(1)} + ... + E(7)\,\alpha^{i\,p(7)}$
- $= E(0)\,(\alpha^{p(0)})^i + E(1)\,(\alpha^{p(1)})^i + ... + E(7)\,(\alpha^{p(7)})^i$.

17 Equations for 16 Unknowns

- Let $\beta_i = \alpha^{p(i)}$. Then $p(i) = \log2[\beta_i]$.
- $S_0 = E(0) + E(1) + ... + E(7)$
- $S_1 = E(0)\,\beta_0 + E(1)\,\beta_1 + ... + E(7)\,\beta_7$
- $S_2 = E(0)\,(\beta_0)^2 + E(1)\,(\beta_1)^2 + ... + E(7)\,(\beta_7)^2$
- …
- $S_{16} = E(0)\,(\beta_0)^{16} + E(1)\,(\beta_1)^{16} + ... + E(7)\,(\beta_7)^{16}$
- Unknowns: $E(0), ..., E(7), \beta_0, ..., \beta_7$

The Error Locator Polynomial

- $\sigma(x) = (1-x\beta_0)\,(1-x\beta_1)\,(1-x\beta_2)\ \ldots\ (1-x\beta_7)$
- $\quad = 1 + \sigma_1 x + \sigma_2 x^2 + \ldots + \sigma_8 x^8$
- has eight roots β_j^{-1} for j = 0,...,7 and that is
- $\sigma(\beta_j^{-1}) = 1 + \sigma_1\,\beta_j^{-1} + \sigma_2\,\beta_j^{-2} + \ldots + \sigma_8\,\beta_j^{-8} = 0$, for j = 0,... 7.
- Need to find σ_1 , σ_2 , ... , σ_8 and then β_j^{-1} for j = 0,...,7.
- $E(j)\,\beta_j^{k+7}\,\sigma(\beta_j^{-1})$ // for j = 0,...,7
- $\quad = E(j)\,\beta_j^{k+7} + \sigma_1\,E(j)\,\beta_j^{k+6} + \sigma_2\,E(j)\,\beta_j^{k+5} + \ldots + \sigma_8\,E(j)\,\beta_j^{k-1} = 0$
- Add these from j = 0 to 7, we have
- $S_{k+7} + \sigma_1\,S_{k+6} + \sigma_2\,S_{k+5} + \ldots + \sigma_8\,S_{k-1} = 0$, for k = 1 to 9

Learning a Recurrence Relation

- $S_8 = \sigma_1 S_7 + \sigma_2 S_6 + \sigma_3 S_5 + \sigma_4 S_4 + \sigma_5 S_3 + \sigma_6 S_2 + \sigma_7 S_1 + \sigma_8 S_0$
- $S_9 = \sigma_1 S_8 + \sigma_2 S_7 + \sigma_3 S_6 + \sigma_4 S_5 + \sigma_5 S_4 + \sigma_6 S_3 + \sigma_7 S_2 + \sigma_8 S_1$
- $S_{10} = \sigma_1 S_9 + \sigma_2 S_8 + \sigma_3 S_7 + \sigma_4 S_6 + \sigma_5 S_5 + \sigma_6 S_4 + \sigma_7 S_3 + \sigma_8 S_2$
- $S_{11} = \sigma_1 S_{10} + \sigma_2 S_9 + \sigma_3 S_8 + \sigma_4 S_7 + \sigma_5 S_6 + \sigma_6 S_5 + \sigma_7 S_4 + \sigma_8 S_3$
- $S_{12} = \sigma_1 S_{11} + \sigma_2 S_{10} + \sigma_3 S_9 + \sigma_4 S_8 + \sigma_5 S_7 + \sigma_6 S_6 + \sigma_7 S_5 + \sigma_8 S_4$
- $S_{13} = \sigma_1 S_{12} + \sigma_2 S_{11} + \sigma_3 S_{10} + \sigma_4 S_9 + \sigma_5 S_8 + \sigma_6 S_7 + \sigma_7 S_6 + \sigma_8 S_5$
- $S_{14} = \sigma_1 S_{13} + \sigma_2 S_{12} + \sigma_3 S_{11} + \sigma_4 S_{10} + \sigma_5 S_9 + \sigma_6 S_8 + \sigma_7 S_7 + \sigma_8 S_6$
- $S_{15} = \sigma_1 S_{14} + \sigma_2 S_{13} + \sigma_3 S_{12} + \sigma_4 S_{11} + \sigma_5 S_{10} + \sigma_6 S_9 + \sigma_7 S_8 + \sigma_8 S_7$
- $S_{16} = \sigma_1 S_{15} + \sigma_2 S_{14} + \sigma_3 S_{13} + \sigma_4 S_{12} + \sigma_5 S_{11} + \sigma_6 S_{10} + \sigma_7 S_9 + \sigma_8 S_8$

Berlekamp-Massey Algorithm (1965)

- A fast algorithm solving the linear system of equations that have the recurrence property.
- Syndromes based on evaluating the error polynomial with consecutive powers of σ are not completely independent.
-

The Case of One Byte Error

- Suppose we have only one byte error and thus the error locator polynomial is $1 + \sigma_1 x$ and all other coefficients are zero.
- In this case, $\sigma'_1 = S_1 S_0^{-1}$ from $S_1 = \sigma_1 S_0$
  - $S_0 = E(0)$, $S_1 = E(0)\, \beta_0$, $\beta_0 = S_1 S_0^{-1}$
- $E(0)$, the magnitude of the error plays no role in the determination of $\sigma_1 = \beta_0 = \alpha^{p(0)}$ or the location of the error, $p(0)$.
- Suppose there are two errors, $1 + \sigma_1 x + \sigma_2 x^2$ and thus there is a discrepancy from the solution before.
- Discrepancy $d_1 = S_2 + \sigma'_1 S_1$.
  - $S_1 = E(0)\, \beta_0$, $S_2 = E(0)\, (\beta_0)^2$, $\sigma'_1 = \beta_0$, $d = 0$ if indeed there is only one error.

The Case of Two Byte Errors

- $S_0 = E(0) + E(1)$
- $S_1 = E(0)\ \beta_0 + E(1)\ \beta_1$
- $S_2 = E(0)\ (\beta_0\ )^2 + E(1)\ (\beta_1\ )^2 = \sigma_1\ S_1 + \sigma_2\ S_0$
- $\sigma'_1 = S_1\ S_0^{-1}$ but $\sigma_1 = \beta_0 + \beta_1$, $\sigma_2 = \beta_0\ \beta_1$
- Discrepancy $d_1 = S_2 + \sigma'_1\ S_1$.

Homework 23: due 4-20-15

- The given H23.java can be applied to test21.txt
  - java H23 < test21
- It randomly generate 8 error positions with 8 no-zero error magnitudes to alter the codewords in test21.txt.
- After non-zero symdromes are computed, the Berlekamp-Massey Algorithm is applied to find the error-location polynomial.
- The roots of this polynomials are β_j^{-1} .
- From the relation $\beta_i = \alpha^{p(i)}$ you should be able to find p(i)'s to fill the errorPositions array, which should match the error positions randomly generated by H23.
- Change the constant maxError to smaller and larger values and run the program and comment on the results.

```
void addErrors(){ // randomly generate 8 error locations and error values
   Random random = new Random();
   for (int i = 0; i < capacity; i++)
      codewords[i] = nextSymbol(i * 8, 8); // data from true symbol
   displayPolynomial("Original Message", codewords); // display the 26 codewords

   for (int i = 0; i < maxErrors; i++){
     errorPositions[i] = random.nextInt(capacity);
     boolean unique = false;
     while (!unique){
      int j = 0; for (; j < i; j++)
        if (errorPositions[i] == errorPositions[j]) break;
      if (j == i) unique = true;
      else errorPositions[i] = random.nextInt(capacity);
     }
     errorMagnitudes[i] = random.nextInt(oneLessFieldSize) + 1;
     codewords[errorPositions[i]] ^= errorMagnitudes[i];  // adding errors
   }

   displayPolynomial("Random Error Positions", errorPositions);
   displayPolynomial("Random Error Magnitudes", errorMagnitudes);
   displayPolynomial("Message with Errors", codewords);
 }
```

```
int inverse(int a){  // multiplicative inverse of (non-zero) a in GF(2^8)
    return alog[oneLessFieldSize - log2[a]];
  }

  int mul(int a, int b){ // multiplication in GF(2^8)
    if (a == 0 || b == 0) return 0;
    return alog[(log2[a] + log2[b]) % oneLessFieldSize];
  }

  int evaluatePolynomial(int[] p, int x){ // what is p(x)
    int len = p.length;
    int sum = p[0];
    for (int i = 1; i < len; i++)
      sum = mul(sum, x) ^ p[i];
    return sum;
  }

  void computeSyndromes(){ // syndromes are codewords(2^i)
    for (int i = 0; i < correctionCapacity; i++)
      syndromes[i] = evaluatePolynomial(codewords, alog[i]);
    displayPolynomial("Syndromes", syndromes);
  }
```

```java
void displayPolynomial(String title, int[] p) { // display array with title
  System.out.print(title + "  ");
  for (int i = 0; i < p.length - 1; i++)
     System.out.print(p[i] + " ");
  System.out.println(p[p.length - 1] + " ");
}

int[] shiftPolynomial(int[] p){ // xp(x)
  int[] shifted = new int[p.length + 1];
  for (int i = 0; i < p.length; i++) shifted[i] = p[i];
  shifted[p.length] = 0;
  return shifted;
}

int[] scalePolynomial(int[] p, int a){ // ap(x)
   int[] scaled = new int[p.length];
   for (int i = 0; i < p.length; i++) scaled[i] = mul(p[i], a);
   return scaled;
}
```

```
int[] addPolynomials(int[] p, int[] q) { // p + q
  int[] tmp = new int[Math.max(p.length, q.length)];
  for (int i = 0; i < p.length; i++)
    tmp[i + tmp.length - p.length] = p[i];
  for (int i = 0; i < q.length; i++)
    tmp[i + tmp.length - q.length] ^= q[i];
  return tmp;
}
```

```
void solveLocatorsBerlekampMassey() {
   System.out.println(“\nBerlekamp-Massey Decoder");
   int[] ep = new int[1]; ep[0] = 1;  // approximation for error locators poly
   int[] op = new int[1]; op[0] = 1;
   int[] np = null;
   for (int i = 0; i < syndromes.length; i++) { // Iterate over syndromes
     op = shiftPolynomial(op);
     int delta = syndromes[i]; // discrepancy from
     for (int j = 1; j < ep.length; j++) // recurrence for syndromes
        delta ^= mul(ep[ep.length - 1 - j], syndromes[i - j]);
     if (delta != 0) { // has discrepancy
       if (op.length > ep.length){
        np = scalePolynomial(op, delta);
         op = scalePolynomial(ep, inverse(delta));
        ep = np;
      }
       ep = addPolynomials(ep, scalePolynomial(op, delta));
       displayPolynomial("Iteration " + i, ep); // display Berlekamp-Massey steps
     }
   }
   locators = ep;
   displayPolynomial("Error-Location Polynomial", locators);
 }
```

You need to complete this function.

```
void solveErrorPositions() {
  // find roots of error-location polynomial then error positions
   errorPositions = new int[maxErrors];

   for (int i = 0, j = 0; i < capacity; i++)
      if (evaluatePolynomial(locators, //i or a function of i here//) == 0)
        errorPositions[j++] = //another function of i here//;

   displayPolynomial("Decoded Error Positions", errorPositions);
 }
```

H23.main()

```
public static void main(String[] args){
   H23 h23 = new H23();
   h23.makeLog2();
   h23.readRawBitmap();
   h23.getMatrix();
   h23.demask();
   h23.getDataBitStream();
   h23.addErrors();
   h23.computeSyndromes();
   h23.solveLocatorsBerlekampMassey();
   h23.solveErrorPositions();
 }
```