15-853: Algorithms in the Real World

Error Correcting Codes II
- Cyclic Codes
- Reed-Solomon Codes

Reed-Solomon: Outline
A \((n, k, n-k+1)\) Reed Solomon Code:
Consider the polynomial
\[ p(x) = a_{k-1}x^{k-1} + \ldots + a_1x + a_0 \]
**Message:** \((a_{k-1}, \ldots, a_1, a_0)\)
**Codeword:** \((p(1), p(2), \ldots, p(n))\)
To keep the \(p(i)\) fixed size, we use \(a_i \in GF(p^r)\)
To make the \(p(i)\) distinct, \(n < p^r\)
Any subset of size \(k\) of \((p(1), p(2), \ldots, p(n))\) is enough to reconstruct \(p(x)\).

And now a word from our founder...
Governor Sandford [sic] made a useless rival as you and I saw when in San Francisco, to the State University. I could be no party to such a thing.

- Andrew Carnegie in a letter to Andrew White, Ambassador to Berlin, on the establishment of Stanford University, 1901.
Reed Solomon: Outline

Correcting $s$ errors:
1. Find $k+s$ symbols that agree on a polynomial $p(x)$. These must exist since originally $k+2s$ symbols agreed and only $s$ are in error
2. There are no $k+s$ symbols that agree on the wrong polynomial $p'(x)$
   - Any subset of $k$ symbols will define $p'(x)$
   - Since at most $s$ out of the $k+s$ symbols are in error, $p'(x) = p(x)$

Reed Solomon: Outline

Systematic version of Reed-Solomon
$p(x) = a_{k-1}x^{k-1} + ... + a_1x + a_0$
Message: $(a_{k-1}, ..., a_1, a_0)$
Codeword: $(a_{k-1}, ..., a_1, a_0, p(1), p(2), ..., p(2s))$ This has the advantage that if we know there are no errors, it is trivial to decode.
Later we will see that version of RS used in practice uses something slightly different than $p(1), p(2), ...$ This will allow us to use the “Parity Check” ideas from linear codes (i.e. $Hc^T = 0$?) to quickly test for errors.

RS in the Real World

$(204,188,17)_{256}$: ITU J.83(A)$^2$
$(128,122,7)_{256}$: ITU J.83(B)
$(255,223,33)_{256}$: Common in Practice
   - Note that they are all byte based (i.e. symbols are from $GF(2^8)$).
Performance on 600MHz Pentium (approx.):
   - $(255,251) = 45$Mbps
   - $(255,223) = 4$Mbps
Dozens of companies sell hardware cores that operate 10x faster (or more)
   - $(204,188) = 320$Mbps (Altera decoder)

Applications of Reed-Solomon Codes

- **Storage**: CDs, DVDs, "hard drives",
- **Wireless**: Cell phones, wireless links
- **Satellite and Space**: TV, Mars rover, ...
- **Digital Television**: DVD, MPEG2 layover
- **High Speed Modems**: ADSL, DSL, ...

Good at handling burst errors.
Other codes are better for random errors.
   - e.g. Gallager codes, Turbo codes
RS and “burst” errors

Let's compare to Hamming Codes (which are “optimal”).

<table>
<thead>
<tr>
<th>Code</th>
<th>Code bits</th>
<th>Check bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS (255, 253, 3)_{ma}</td>
<td>2040</td>
<td>16</td>
</tr>
<tr>
<td>Hamming (2^{11-1}, 2^{11-1-1}, 3)_{2}</td>
<td>2047</td>
<td>11</td>
</tr>
</tbody>
</table>

They can both correct 1 error, but not 2 random errors.
- The Hamming code does this with fewer check bits.
- However, RS can fix 8 contiguous bit errors in one byte.
- Much better than lower bound for 8 arbitrary errors.

\[
\log \left( 1 + \binom{n}{1} + \cdots + \binom{n}{8} \right) > 8 \log(n - 7) \approx 88 \text{ check bits}
\]

Galois Fields

The polynomials
\[ Z_p[x] \mod p(x) \]
where
\[ p(x) \in Z_p[x], \]
\[ p(x) \text{ is irreducible}, \]
and \[ \deg(p(x)) = n \]
form a finite field. Such a field has \( p^n \) elements.
These fields are called Galois Fields or \( GF(p^n) \).
The special case \( n = 1 \) reduces to the fields \( Z_p \).
The multiplicative group of \( GF(p^n)/\{0\} \) is cyclic (this will be important later).

GF(2^n)

Hugely practical!
The coefficients are bits \( \{0,1\} \).
For example, the elements of \( GF(2^8) \) can be represented as a byte, one bit for each term, and \( GF(2^{64}) \) as a 64-bit word.
- e.g., \( x^6 + x^4 + x + 1 = 01010011 \)

How do we do addition?

Addition over \( Z_2 \) corresponds to xor.
- Just take the xor of the bit-strings (bytes or words in practice). This is dirt cheap.

Multiplication over GF(2^n)

If \( n \) is small enough can use a table of all combinations.
The size will be \( 2^n \times 2^n \) (e.g., 64K for \( GF(2^8) \)).
Otherwise, use standard shift and add (xor)

Note: dividing through by the irreducible polynomial on an overflow by 1 term is simply a test and an xor.

- e.g. \( 0111 / 1001 = 0111 \)
  \( 1011 / 1001 = 1011 \text{ xor } 1001 = 0010 \)
  \( ^\wedge \text{ just look at this bit for } GF(2^3) \)
Multiplication over $GF(2^5)$

typedef unsigned char uc;

uc mult(uc a, uc b) {
    int p = a;
    uc r = 0;
    while (b) {
        if (b & 1) r = r ^ p;
        b = b >> 1;
        p = p << 1;
        if (p & 0x10) p = p ^ 0x11B;
    }
    return r;
}

Finding inverses over $GF(2^n)$

Again, if $n$ is small just store in a table.
- Table size is just $2^n$.
For larger $n$, use long division algorithm.
- This is again easy to do with shift and xors.

Galois Field

$GF(2^3)$ with irreducible polynomial: $x^3 + x + 1$

$\alpha = x$ is a generator

| $\alpha^0$ | $x^3$ | $010$ | 2 |
| $\alpha^1$ | $x^2$ | 100  | 3 |
| $\alpha^2$ | $x^1 + 1$ | 011  | 4 |
| $\alpha^3$ | $x^2 \cdot x$ | 110  | 5 |
| $\alpha^4$ | $x^2 + x + 1$ | 111  | 6 |
| $\alpha^5$ | $x^0 + 1$ | 101  | 7 |
| $\alpha^7$ | $1$ | 001  | 1 |

Will use this as an example.

Discrete Fourier Transform

Another View of Reed-Solomon Codes

$\alpha$ is a primitive $n$th root of unity ($\alpha^n = 1$) - a generator

$$T = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-4} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & \alpha^{n-4} & \alpha^{2(n-4)} & \cdots & \alpha^{(n-4)(n-4)} \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ c_{k-1} \end{pmatrix} = \begin{pmatrix} m_0 \\ \vdots \\ m_{k-1} \end{pmatrix}$$

The Discrete Fourier Transform (DFT)

$$m = T^{-1}c$$

Inverse DFT
**DFT Example**

$\alpha = x$ is the 7th root of unity in $GF(2^3) / x^3 + x + 1$

(ie, multiplicative group, which excludes additive inverse)

Recall $\alpha = 2^2$, $\alpha^2 = 3^2$, ..., $\alpha^7 = 1 = 1$'

$$
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha & \alpha^2 & \alpha^4 \\
1 & \alpha^3 & \alpha^3 & \alpha^3 & \alpha^3 & \alpha^3 & \alpha^3 \\
1 & \alpha^4 & \alpha^4 & \alpha^4 & \alpha^4 & \alpha^4 & \alpha^4 \\
1 & \alpha^5 & \alpha^5 & \alpha^5 & \alpha^5 & \alpha^5 & \alpha^5 \\
1 & \alpha^6 & \alpha^6 & \alpha^6 & \alpha^6 & \alpha^6 & \alpha^6 \\
\end{bmatrix}
= \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 2^2 & 2^4 & 2^6 & 1 & 3^2 & 3^3 \\
1 & 4^2 & 4^4 & 4^5 & 1 & 5^3 & 5^3 \\
1 & 6 & 6 & 6 & 1 & 7 & 7^3 \\
\end{bmatrix}

$$

Should be clear that $c = T \cdot (m_0, m_1, ..., m_{k-1}, O, ...)^T$

is the same as evaluating $p(x) = m_0 + m_1 x + ... + m_{k-1} x^{k-1}$

at $n$ points.

---

**Decoding**

Why is it hard?

Brute Force: try $k$ choices $k + 2s$ possibilities and solve for each.

---

**Cyclic Codes**

A linear code is cyclic if:

$(c_0, c_1, ..., c_{n-1}) \in C \Rightarrow (c_{n-1}, c_0, ..., c_{n-2}) \in C$

Both Hamming and Reed-Solomon codes are cyclic.

Note: we might have to reorder the columns to make the code "cyclic".

**Motivation:** They are more efficient to decode than general codes.

---

**Generator and Parity Check Matrices**

**Generator Matrix:**

A $k \times n$ matrix $G$ such that:

$C = \{ m \cdot G \mid m \in \Sigma^k \}$

Made from stacking the basis vectors

**Parity Check Matrix:**

A $(n-k) \times n$ matrix $H$ such that:

$C = \{ v \in \Sigma^n \mid H \cdot v^T = 0 \}$

Codewords are the nullspace of $H$

These always exist for linear codes $H \cdot G^T = 0$
Generator and Parity Check Polynomials

**Generator Polynomial:**
A degree \((n-k)\) polynomial \(g\) such that:
\[
C = \{ m \cdot g \mid m \in \Sigma[x] \}
\]
such that \(g \mid x^n - 1\)

**Parity Check Polynomial:**
A degree \(k\) polynomial \(h\) such that:
\[
C = \{ v \in \Sigma^n[x] \mid h \cdot v \equiv 0 \pmod{x^n-1} \}
\]
such that \(h \mid x^n - 1\)

These always exist for linear cyclic codes
\(h \cdot g = x^n - 1\)

---

**Viewing \(g\) as a matrix**
If \(g = g_0 + g_1 x + \ldots + g_{n-k} x^{n-k}\)
We can put this generator in matrix form:
\[
G = \begin{pmatrix}
g_0 & g_1 & \cdots & g_{n-k} & 0 & \cdots & 0 \\
0 & g_0 & \cdots & g_{n-k-1} & g_{n-k} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots \\
0 & 0 & \cdots & g_0 & g_1 & \cdots & g_{n-k}
g \end{pmatrix}
\]
Write \(m = m_0 + m_1 x + \ldots m_{n-k} x^{n-k}\) as \((m_0, m_1, \ldots, m_{n-k})\)
Then \(c = mG\)

---

**Viewing \(h\) as a matrix**
If \(h = h_0 + h_1 x + \ldots + h_k x^k\)
we can put this parity check poly. in matrix form:
\[
H = \begin{pmatrix}
0 & \cdots & 0 & h_k & \cdots & h_1 & h_0 \\
0 & h_k & \cdots & h_1 & \cdots & h_0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\
h_k & \cdots & h_1 & h_0 & 0 & \cdots & 0
\end{pmatrix}
\]
\(Hc^T = 0\)

---

\(g\) generates cyclic codes
\[
G = \begin{pmatrix}
g_0 & g_1 & \cdots & g_{n-k} & 0 & \cdots & 0 \\
0 & g_0 & \cdots & g_{n-k-1} & g_{n-k} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots \\
0 & 0 & \cdots & g_0 & g_1 & \cdots & g_{n-k} \\
g \end{pmatrix}
\]
Codes are linear combinations of the rows.
All but last row is clearly cyclic (based on next row)
Shift of last row is \(x^k g\) mod \((x^n - 1)\)
Consider \(h = h_0 + h_1 x + \ldots + h_k x^k\) \((gh = x^n - 1)\)
\[
h_0 g + (h_1 x)g + \ldots + (h_{n-k} x^{n-k})g + (h_k x^k)g = x^n - 1
\]
\(x^k g = -h_k (h_0 g + h_1 (xg) + \ldots + h_{n-k-1} (x^{n-k-1} g)) \bmod (x^n - 1)\)
This is a linear combination of the rows.
Hamming Codes Revisited

The Hamming $(7,4,3)_2$ code.

\[ g = 1 + x + x^3 \quad h = x^4 + x^2 + x + 1 \]

\[
G = \begin{pmatrix}
1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1
\end{pmatrix}
\quad H = \begin{pmatrix}
0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0
\end{pmatrix}
\]

\[ gh = x^7 - 1, \quad GH^T = 0 \]

The columns are not identical to the previous example Hamming code.

Another way to write \( g \)

Let \( g \) be a generator of \( GF(p^n) \).

Let \( n = p^n - 1 \) (the size of the multiplicative group)

Then we can write a generator polynomial as

\[ g(x) = (x - \alpha)(x - \alpha^2) \ldots (x - \alpha^{n-1}) \]

**Lemma:** \( g \mid x^n - 1 \) (\( a \mid b \), means \( a \) divides \( b \))

**Proof:**

- \( \alpha^n = 1 \) (because of the size of the group)
  \[ \Rightarrow \alpha^n - 1 = 0 \]
  \[ \Rightarrow \alpha \) root of \( x^n - 1 \]
  \[ \Rightarrow (x - \alpha) \mid x^n - 1 \]
- similarly for \( \alpha^2, \alpha^3, \ldots, \alpha^{n-1} \)
- therefore \( x^n - 1 \) is divisible by \( (x - \alpha)(x - \alpha^2) \ldots \)

Factors of \( x^n - 1 \)

Intentionally left blank

Back to Reed-Solomon

Consider a generator polynomial \( g \in GF(p^n)[x], \) s.t. \( g \mid (x^n - 1) \)

Recall that \( n - k = 2s \) (the degree of \( g \))

**Encode:**

- \( m' = m x^{2s} \) (basically shift by \( 2s \))
- \( b = m' \) (mod \( g \))
- \( c = m' - b = (m_{k+1}, \ldots, m_b, -b_{2s+1}, \ldots, -b_0) \)
- Note that \( c \) is a cyclic code based on \( g \)
  - \( m' = qg + b \)
  - \( c = m' - b = qg \)

**Parity check:**

- \( h c = 0 \)?
Example

Let's consider the $(7,3,5)_3$ Reed-Solomon code. We use $GF(2^3)/x^3 + x + 1$

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$x$</th>
<th>$\alpha^2$</th>
<th>$\alpha^3$</th>
<th>$\alpha^4$</th>
<th>$\alpha^5$</th>
<th>$\alpha^6$</th>
<th>$\alpha^7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$001$</td>
<td>2</td>
<td>$010$</td>
<td>$100$</td>
<td>$110$</td>
<td>$111$</td>
<td>$101$</td>
<td>$001$</td>
</tr>
</tbody>
</table>

Example RS $(7,3,5)_3$

Let $g = (x - \alpha)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4)$

\[ g = x^4 + \alpha^3 x + x^2 + \alpha x + \alpha^3 \]

\[ h = (x - \alpha^5)(x - \alpha^6)(x - \alpha^7) \]

\[ h = x^3 + \alpha^3 x^2 + \alpha^2 x + \alpha^4 \]

\[ gh = x^7 - 1 \]

Consider the message: 110 000 110

\[ m = (\alpha^6, 0, \alpha^4) = \alpha^4 x^2 + \alpha^4 \]

\[ m' = x^4 m = \alpha^3 x^5 + \alpha^6 x^4 \]

\[ g + (\alpha^4 x^2 + x + \alpha^3)g + (\alpha^3 x^2 + \alpha^6 x + \alpha^6) \]

\[ c = (\alpha^6, 0, \alpha^4, 0, \alpha^6, \alpha^6) \]

\[ = 110 000 110 011 000 101 101 \]

\[ ch = 0 \, (\text{mod} \, x^7-1) \]

A useful theorem

**Theorem:** For any $\beta$, if $g(\beta) = 0$ then $\beta^2 m(\beta) = b(\beta)$

**Proof:**

\[ x^2 m(x) = g(x) g(\beta) + b(x) \]

\[ \beta^2 m(\beta) = g(\beta) g(\beta) + b(\beta) = b(\beta) \]

**Corollary:** $\beta^2 m(\beta) = b(\beta)$ for $\beta \in \{\alpha, \alpha^2, ..., \alpha^{2^s}\}$

**Proof:**

$\{\alpha, \alpha^2, ..., \alpha^{2^s}\}$ are the roots of $g$ by definition.

Fixing errors

**Theorem:** Any $k$ symbols from $c$ can reconstruct $c$ and hence $m$

**Proof:**

We can write 2$s$ equations involving $m (c_{i_1}, ..., c_{i_s})$ and $b (c_{2s-i}, ..., c_0)$. These are

\[ \alpha^{2^s} m(\alpha) = b(\alpha) \]

\[ \alpha^{4s} m(\alpha^2) = b(\alpha^2) \]

\[ \alpha^{2s(2^s)} m(\alpha^{2^s}) = b(\alpha^{2^s}) \]

We have at most 2$s$ unknowns, so we can solve for them. (I'm skipping showing that the equations are linearly independent).
Efficient Decoding

I don't plan to go into the Reed-Solomon decoding algorithm, other than to mention the steps.