Reed-Solomon codes

Reed-Solomon (RS) codes are a type of **error-correcting code** used to detect and correct data transmission or storage errors. They are particularly effective at correcting burst errors, where multiple consecutive bits are corrupted. Reed-Solomon codes are widely used in applications like **CDs, DVDs, QR codes, satellite communications**, and **data storage systems**.

Key Concepts

1. Symbols Instead of Bits

Unlike simpler error-correcting codes (such as Hamming codes, which operate on individual bits), Reed-Solomon codes operate on **symbols**. A symbol is a block of bits (e.g., 8 bits) that represents a larger unit of data.

For example, in an 8-bit RS code, each symbol is an 8-bit value, allowing the code to work on groups of 8 bits (1 byte) at a time. RS codes can handle burst errors, meaning they can correct errors that affect multiple consecutive bits in a single symbol.

2. Block-Based Code

Reed-Solomon codes are **block codes**, meaning they work by dividing data into fixed-size blocks and encoding each block independently.

- **Message Length**: The number of data symbols in a block.
- **Codeword Length**: The total number of symbols (including parity) in a block after encoding.

For example, in a typical RS code, you might have a message length of `k` data symbols and a codeword length of `n` total symbols, where `n > k`. The difference (`n - k`) represents the number of **redundant (parity)** symbols added for error correction.

3. Error Correction Capability

Reed-Solomon codes can detect and correct multiple symbol errors. Specifically, an RS code can correct up to `(n - k) / 2` symbols of errors in a block.

- Error correction capability: The number of correctable symbols depends on how many parity symbols are added. More parity symbols increase the error-correcting strength but also increase the overhead.

We'll use a simplified version with small numbers to make the process easier to understand. Typically, Reed-Solomon codes use larger fields (like 255 symbols for CDs), but in this example, we'll use a smaller field size.

Setup

Let's use the Reed-Solomon code **RS(7, 3)**, which means:

1. **n = 7**: The total number of symbols in the codeword.
2. **k = 3**: The number of original data symbols.
3. **n - k = 4**: The number of parity symbols added for error correction.

The codeword will be composed of 7 symbols: 3 data symbols and 4 parity symbols. We'll assume we're working in **GF(8)** (Galois Field of 8 elements), with the following primitive polynomial:

$$ x^3 + x + 1 $$

The field elements are represented as:

$$ \{ 0, 1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6 \} $$

where α is a primitive element (root of the polynomial) and behaves cyclically.

Step 1: Data Symbols

Let’s assume we want to encode the following **3 data symbols**: $$ \text{Data} = \{ 6, 4, 3 \} $$

Step 2: Create the Generator Polynomial

The generator polynomial $ g(x) $ is a key part of the encoding process. For simplicity, in this example, let’s assume the generator polynomial for **RS(7, 3)** is: $$ g(x) = (x - \alpha^0)(x - \alpha^1)(x - \alpha^2)(x - \alpha^3) $$

Expanding this gives us the generator polynomial: $$ g(x) = x^4 + \alpha^3 x^3 + \alpha^2 x^2 + \alpha x + 1 $$

Step 3: Polynomial Representation of Data

Represent the data as a polynomial: $$ \text{Data Polynomial} = 6x^2 + 4x + 3 $$

Step 4: Multiply Data Polynomial by x^{n-k}

Next, we multiply the data polynomial by $x^{n-k} = x^4$ (to make space for the parity symbols):

$$ x^4(6x^2 + 4x + 3) = 6x^6 + 4x^5 + 3x^4 $$

Step 5: Divide by Generator Polynomial $g(x)$

Now, divide the polynomial $6x^6 + 4x^5 + 3x^4$ by the generator polynomial $g(x)$. The remainder of this division will be the **parity symbols**.

For simplicity, I won't show the detailed polynomial division here (it's tedious and usually done by computer), but let's assume the remainder after division is: $$ \text{Remainder} = 2x^3 + 5x^2 + 4x + 1 $$

Step 6: Codeword

The final codeword is formed by adding the remainder (parity symbols) to the original data polynomial. The full codeword (data + parity) is:

$$ \text{Codeword} = 6x^6 + 4x^5 + 3x^4 + 2x^3 + 5x^2 + 4x + 1 $$

In terms of symbols, the codeword is:

$$ \{ 6, 4, 3, 2, 5, 4, 1 \} $$

This is the transmitted message, containing both the original data and the parity symbols.

Step 7: Error Introduction

Assume there is a transmission error, and the received codeword has errors in one symbol:

$$ \text{Received} = \{ 6, 4, 3, 2, 5, 0, 1 \} $$ In this case, the 6th symbol (originally 4) was corrupted to 0.

Step 8: Error Detection and Correction

Using the Reed-Solomon decoding algorithm (which involves solving the error locator polynomial), the receiver can detect that an error occurred and identify the erroneous symbol(s). Based on the parity and the syndromes, it would determine that the 6th symbol is incorrect and correct it back to the original value (4).

Thus, the corrected codeword would be:

$$ \{ 6, 4, 3, 2, 5, 4, 1 \} $$

Summary of Steps

- 1. **Encode**: Multiply the data polynomial by x^{n-k} and divide by the generator polynomial to find the parity.
- 2. **Transmit**: Send the data and parity as a codeword.
- 3. **Detect and Correct**: Upon receiving the (possibly corrupted) codeword, calculate

syndromes and use the Reed-Solomon algorithm to correct errors.

Applications

This simplified example illustrates the core idea behind Reed-Solomon codes. They are used in real-world systems like **CDs, DVDs, Blu-ray discs, and QR codes** to correct errors that may occur during data storage or transmission.