Documentation: Reed-Solomon Error-Correcting Codes

Preliminaries. It is assumed that the reader of this document understands operating within a Galois Field. Specifically, we will be using Galois Field 256. Knowledge on logarithms, antilogarithms and polynomial operations are also required. Python 3 will be used in writing the algorithm to generate Reed-Solomon error-correcting codes.

Error Correction Programming. We first determine the number of codewords (c) in the error-correcting code using two parameters, the version and error correction level. For this exercise, only two of the forty versions are considered. Using this value, the generator polynomial $g_c(x)$ is computed using the formula below.

$$g_c(x) = \prod_{i=0}^{c-1}(x - \alpha^i)$$

Any given message can be represented as a sequence of integers, which will be the coefficients of the message polynomial $m(x)$. Note that in this project, it is assumed that the message is already correctly encoded. We divide $m(x)$ by the generator polynomial and obtain the remainder $r(x)$. The coefficients of the remainder are used to represent the error-correcting code.

Logarithms and Antilogarithms. For any element e in $GF(256)$, we may represent it as 2^n where n is any integer from 0 to 255, inclusive. Instead of computing for the value of α^n or e every time we need it, we generate the values once and store it in two lookup tables (`ALOG_TABLE` and `LOG_TABLE`).

As seen on the code snippet below, `ALOG_TABLE` will initially have a value of 1 at index 0. Each iteration, we multiply the value at the end of the table (`temp`) by 2. If this value exceeds 255, we XOR it with 285 and set that as the current value. Meanwhile, `LOG_TABLE` is generated in a similar manner except that the index-value pairs are swapped.

```
temp = 1
for i in range(0x100-1):
    ALOG_TABLE.append(temp)
    LOG_TABLE[temp] = i
    temp <<= 1
    if (temp & 0x100):
        temp ^= 0x11d
ALOG_TABLE.append(1)
```

Generator Polynomial. We obtain the generator polynomial as explained earlier. To simplify calculations, we express the coefficient of the polynomial in the form 2^n where each element of the polynomial array is a valid exponent n. Upon initialization, $g_c(x) = x - 1$. Each iteration, we multiply the generator polynomial to $x - 2^i$, where i is the current number of iterations. Note that for any $g_c(x)$ the highest order term will always have a coefficient of 2^0 so we exclude that in our calculations.

```
gen_poly = [0]
for i in range(1,c):
    gen_poly.append(0)
    gen_temp = list(map(lambda g: (g+i)%255, gen_poly))
    for j in range(len(gen_temp)-1, -1, -1):
        gen_poly[j] = LOG_TABLE[ALOG_TABLE[gen_temp[j]]^\
            ALOG_TABLE[gen_poly[j-1]]]
    gen_poly[0] = gen_temp[0]
gen_poly.append(0)
```

Reed-Solomon Codeword. Now that we have a generator polynomial, we simply divide the message polynomial and get the remainder $r(x)$ that will represent the error-correcting code (ECC). The algorithm used to solve for this is extended synthetic division. The flow of the algorithm and the Python code relevant to each step is discussed below. Note that unlike $m(x)$, the generator polynomial is not preprocessed since it does not really matter if we pad the $g(x)$ array with 0s or not.

1. Multiply the message $m(x)$ by x^c
2. Repeat the steps below d times, where d is the message length.
3. Get the coefficient of the highest term of $r(x)$.
4. Multiply each term of the generator polynomial by `h_term`.
5. XOR the result in 3 and the current value of remainder.
6. Remove the highest term of the new $r(x)$.

```
data.extend([0 for i in range(num_cw)])
for i in range(d_size):
    …
h_term = LOG_TABLE[data[i]]

ALOG_TABLE[(gen_poly[j]+h_term)%255]

data[i+j] ^= result_3
```

None of the terms in $r(x)$ are removed. Instead, we move to the next highest term

```
def get_rs_code(data, num_cw, d_size):
    gen_poly = init_gen_poly(num_cw)
    gen_poly.reverse()
    data.extend([0 for i in range(num_cw)])
    for i in range(d_size):
        h_term = LOG_TABLE[data[i]]
        if (h_term != 0 or data[i] == 1):
            for j in range(1,len(gen_poly)):
                data[i+j] ^= ALOG_TABLE[(gen_poly[j]+h_term)%255]
    return data[d_size:]
```