Assignment2

Taekang Eom

April 21, 2019

1 Introdution to Code

$$\begin{tikzcd} \text{message source} \arrow[d] & & \text{receiver} \\ \text{source encoder} \arrow[r] & \text{channel} \arrow[r] & \text{source decoder} \arrow[u] \end{tikzcd}$$

coding theory는 변조될 가능성이 있는 매체(보통 channel이라고 한다)를 통해 메세지를 보낼 때, 훼손된 메세지를 복구할 수 있는 방법에 대해 연구하는 학문이다. 메세지 m을 보낼 때, m보다 더 긴 메세지 c로 변환하는 encoding과정을 거쳐 channel을 통해 전송하고, decoding 과정에서 channel을 거치면서 훼손된 메세지 d에서 m을 알아낸다.

Definition Σ를 문자들의 집합이라고 하자. 이 때 $C \subset \Sigma^n$를 길이가 n인 Code라고 한다.

앞으로 $q = |\Sigma|$ 로 사용한다.

Definition $x = x_1x_2 \dots x_n, y = y_1y_2 \dots y_n \in \Sigma^n$일 때 Hamming distance d를

$$d(x,y) = d(x_1,y_1) + \cdots + d(x_n,y_n)$$

$$d(x,y) = \begin{cases} 1 & \text{if } x_i \neq y_i \\ 0 & \text{if } x_i = y_i \end{cases}$$

정의한다.

Properties

1.$d(x,y) \geq 0$

2.$d(x,y) = 0 \iff x = y$

3.$d(x,y) = d(y,x) \quad \forall x,y \in \Sigma^n$

4.$d(x,y) + d(y,z) \geq d(x,z) \quad \forall x,y,z \in \Sigma^n$

Proof 1,2,3은 자명.
4는 임의의 $i = 1 \sim n$에 대해 $d(x_i, y_i) + d(y_i, z_i) \geq d(x_i, z_i)$이므로 모든 $1 \leq i \leq n$에 대해 더해주면 증명된다. ///

Definition
1. $d(C) = \min\{d(x,y)|x, y \in C, x \neq y\}$은 code C의 distance이다.
2. $x \in C, 0 < d(x,y) \leq u \Rightarrow y \notin C$이면 C는 u-error-detecting이다.
3. $x \in C, d(x,y) \leq v \Rightarrow \arg\min_{z \in C} d(y,z) = x$이면 C는 v-error-correcting이다.
4. $B_d(x,c) = \{y \in \Sigma^n | d(x,y) \leq c\}$를 x에서의 c-ball이라 한다.

Observation Code C에 대해 $d = d(C)$일때, C는 $d-1$-error-detecting이고, $\lfloor \frac{d-1}{2} \rfloor$-error-correcting이다.

Example $C = \{000, 111\}$은 길이가 3인 1-error-correcting code이다.

이제 C의 길이와 distance가 정해져 있을 때 가능한 C의 최대 크기에 대해 알아보자. C의 최대 크기 중 가장 많이 알려진 2가지는 Hamming bound와 Singleton bound 이다

Hamming Bound 길이가 n인 v-error-correcting-code C에 대해

$$|C| \leq \frac{q^n}{\sum_{i=0}^{v} \binom{n}{i} (q-1)^i}$$

이 성립한다.

Proof C는 v-error-correcting이므로 $x, y \in C, x \neq y$이면 $B_d(x,v) \cap B_d(y,v) = \emptyset$이다.//
길이가 n인 문자열에서 i개의 문자가 틀린 경우, 틀릴 수 있는 위치가 $\binom{n}{i}$이고 각 위치마다 원래의 문자를 제외한 $q-1$개의 문자를 선택할 수 있으므로 경우의 수는 $\binom{n}{i}(q-1)^i$이다.
따라서, 모든 $x \in C$에 대해 $|B_d(x,v)| = \sum_{i=0}^{v} \binom{n}{i}(q-1)^i$이 성립하고, $\Sigma^n \supset \bigcup_{x \in C} B_d(x,v)$이므로 $q^n \geq \left(\sum_{i=0}^{v} \binom{n}{i}(q-1)^i\right) |C|$ 이다. ///

Singleton Bound 길이가 n, distance가 d인 code C에 대해

$$|C| \leq q^{n-d+1}$$

이 성립한다.

Proof $x, y \in C, x \neq y$이면 x, y는 최소 d개의 문자가 다르기 때문에 맨 뒤의 $d-1$개의 문자를 지워도 서로 다르다. 따라서 C의 원소중 앞 $n-d+1$자리가 같은 원소는 없고, $|C| \leq q^{n-d+1}$임을 알 수 있다. ///

2 Linear Codes and Hamming Codes

2.1 Linear Codes

code의 정의에 따르면 code는 scalar가 finite field인 vector space위의 subspace로 잡을 수 있고, 이러한 code들을 linear code라 한다. code를 vector space로 생각하면 훨씬 더 간단해지기 때문에 실제로 사용되는 code들은 대부분 linear code이다.

Definition

1. C가 $\mathbb{F}_q^n$의 subspace이면 C를 $\mathbb{F}_q$ 위에서 길이가 n인 linear code라 한다.
2. $\dim(C)$(C를 $\mathbb{F}_q$ 위의 vectorspace로 볼 때의 dimension)을 linear code C의 dimension이라 한다.
3. 길이가 $n, \dim(C) = k, d(C) = d$인 $\mathbb{F}_q$위의 linear code C를 $[n, k, d]_q$-code라 한다.

2.2 Hamming Codes

이 section에서는 1-error-correcting-code중 하나인 Hamming Code에 대해 소개할 것이다. Hamming code는 $[2^r - 1, 2^r - r - 1, 3]_2$-code 이다($r \geq 2$). 여기서는 $r = 3$를 예시로 들어 Hamming code를 구성하는 방법을 설명 할 것이다.

길이가 7인 어떤 linear code $C \in \mathbb{F}_2^7$를 생각하자. C는 linear code이므로 $\mathbf{0} \in C$이다. i번째 자리만 1이고 나머지는 모두 0인 벡터를 $\mathbf{e}_i (1 \leq i \leq 7)$ 으로 두고, H를 아래와 같이 정의할 때, $\mathbf{He}_i$는 아래에서 부터 읽을 때, i의 2진수 표현임을 알 수 있다.

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

마찬가지로 임의의 $\mathbf{v} \in C$에 대해 i번째 자리가 틀리면 $\mathbf{v} + \mathbf{e}_i$가 되고, 만약 $\mathbf{H}(\mathbf{v} + \mathbf{e}_i) = \mathbf{He}_i$가 된다면 이 결과를 보고 i번째 자리가 틀렸음을 알 수 있다. 이 결과가 성립하는 최대의 C는 $\mathbf{H}$의 nullspace이다. C를 조금 더 쉽게 구하기 쉽게 하기 위해 $\mathbf{H}$의 3,4번째 column을 바꿔 아래와 같이 만든다.

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} = (\mathbf{I}_r | \mathbf{A})$$

이 때, $\mathbf{G}$를 아래와 같이 두면 $\mathbf{HG}^T = \mathbf{0}$ 이 됨을 알 수 있다.

$$\mathbf{G} = \left(-\mathbf{A}^T | \mathbf{I}_{2^r - r - 1}\right) = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

따라서, C는 $\mathbf{G}$의 row space이다.

위의 내용에 기초해서 Hamming Code의 encoding과 decoding과정에 대해 설명할 것이다. 메세지 $\mathbf{m} \in \mathbb{F}_2^{2^r - r - 1}$을 encoding하는 것은 $\mathbf{c} = \mathbf{mG}$로 변환하면 된다.

channel을 통해 받은 $\mathbf{d} \in \mathbb{F}_2^{2^r-1}$을 decoding 하려면 먼저 어느 자리가 틀렸는지 알아야 한다. 이는 $\mathbf{Hd}^T = \mathbf{He}_i$인 i를 알아내면 알 수 있다. $\mathbf{Hd}^T$을 아래에서 부터 읽었을 때, 이진수로 l가 되면 i는 다음과 같다.

$$i = \begin{cases} j+1 & \text{if } l = 2^j \\ l+r-j & \text{if } 2^{j-1} < l < 2^j \end{cases}$$

그러면 원래의 메세지 $\mathbf{m}$은 $\mathbf{mG} = \mathbf{d} - \mathbf{e}_i$으로 encoding되었고, $\mathbf{m}$은 $\mathbf{d} - \mathbf{e}_i$의 뒷 $2^r - r - 1$자리 이므로 $\mathbf{d}$로부터 $\mathbf{m}$으로 decoding을 했음을 알 수 있다.

3 Reed-Solomon Codes

이 section에서는 모든 finite field는 $|F| = p^m$(p는 소수, $m \in \mathbb{N}$)을 만족해야 하고, 그런 p, m이 주어질 때, $|F| = p^m$인 field F도 항상 존재한다는 증명하지 않고 사용할 것이다. $|F| = p^m$인field는 $x^{p^m} - x \in \mathbb{F}_p[x]$를 p^m개의 해를 가지도록 확장한 field라고 생각하면 된다. $|F| = p^m$인 F를 $\mathbb{F}_{p^m}$로 표기하기로 한다.

Reed-Solomon Code는 $\mathbb{F}_q$위에서의 linear code이다. 그리고 Code 의 길이는 $n \leq q$(일반적으로 $n = q-1$을 가장 많이 사용), message의 길이는 $k \leq n$을 만족해야 한다. 이 때 이 Code의 distance는 $d = n - k + 1$이 된다. 이 코드의 경우는 Singleton bound를 만족하기 때문에 효율이 좋은 Code(보내는 message대비 많은 오류를 교정할 수 있음)중 하나이며, 통신이나 저장매체에 많이 사용되었던 Code이기도 하다(이 때는 $\mathbb{F}_{2^m}$ 위에서의 Code를 사용하게 된다). 이 Code를 Encoding하는 방법과 Decoding하는 방법에 대해 알아보자.

3.1 Encoding

Reed-Solomon Code를 encoding하는 방법에는 크게 2가지가 있다. 첫 번째 방법은 서로 다른 $b_i \in \mathbb{F}_{p^m}(1 \leq i \leq n-k)$를 잡고,

$$g(x) = \prod_{i=1}^{n-k}(x - b_i)$$

로 둘 때, 메세지 $\mathbf{m} = (m_1, m_2, \ldots, m_k) \in \mathbb{F}_q^k$에 대해 $m(x) = \sum_{i=1}^{k} m_i x^{i-1}$로 바꿔 $c(x) = g(x)m(x)$로 encoding하는 것이다. 이 방법은 직관적으로 이해하기는 쉽지만 decoding방법이 어려운 편이다. 따라서 encoding은 조금 비직관적이지만 decoding을 더 쉽게 할 수 있는 방법을 소개할 것이다.

두 번째 방법은 서로 다른 $a_i \in \mathbb{F}_{p^m}(1 \leq i \leq n)$를 잡고, $\mathbf{m} = (m_1, m_2, \ldots, m_k) \in \mathbb{F}_q^k$을 아래와 같이 encoding하는 것이다($m(x)$는 첫 번째 방법에서 사용한 것과 같다).

$$\mathbf{c} = (m(a_1), m(a_2), \ldots, m(a_n)) \in \mathbb{F}_q^n$$

실제로는 이와 같은 연산을 빠르게 하기 위해 FFT(Fast Fourier Transform)가 사용된다(Appnedix 참고).

3.2 Decoding

위에서 쓴 것처럼 여기에서는 두번째 encoding에 대한 decoding방법만 소개할 것이다. 이 알고리즘에서는 Euclidean Algorithm의 다항식 버전인 Extended Euclidean Algorithm(EEA)을 사용한다. $\deg r_0 \geq \deg r_1$ 인 다항식 $r_0, r_1 \in \mathbb{F}_q[x]$ 에대한 EEA는 다음과 같다.
$i \in \mathbb{N}$에 대해 $r_0, r_1, \ldots, r_i$까지 구해졌고, $r_i \neq 0$ 이면 $r_{i+1} = r_{i-1} - q_i r_i, \deg(r_{i+1}) < \deg(r_i)$를 만족하는 q_i, r_{i+1}을 구한다. 이 때, 이 조건을 만족하는 q_i, r_{i+1}는 유일하게 존재한다(증명은 Euclidean Algorithm에서와 거의 비슷하다).
이 과정을 반복해 $r_{m+1} = 0$을 얻으면 알고리즘을 종료한다(decoding할 때는 알고리즘을 완전히 수행하지 않고 중간에 끊을 것이다). 그 결과로 $\gcd(r_0, r_1) = r_m$을 얻는다.
추가적으로, $u_0 = 1, u_1 = 0, v_0 = 0, v_1 = 1, u_{i+1} = u_{i-1} - q_i u_i, v_{i+1} = v_{i-1} - q_i v_i \quad (1 \leq i \leq m)$으로 두면 $r_i = u_i r_0 + v_i r_1 \quad (0 \leq i \leq m+1)$, $\deg u_i + \deg r_{i-1} = \deg r_1, \deg v_i + \deg r_{i-1} = \deg r_0 \quad (2 \leq i \leq m+1)$ 임을 수학적 귀납법을 통해 얻을 수 있다.
EEA를 이용한 decoding은 아래와 같이 이루어 진다.

Algorithm of Decoding Reed-Solomon Code
Input: 변조된 메세지 $\mathbf{d} = (d_1, d_2, \ldots, d_n) \in \mathbb{F}_q^n$
Output: 원래 전송했던 메세지 $\mathbf{m} = (m_1, m_2, \ldots, m_k) \in \mathbb{F}_q^k$ 또는 decoding 실패
Step1 $\deg g_1(x) < n, g_1(a_i) = d_i \quad (1 \leq i \leq n)$ 인 $g_1 \in \mathbb{F}_q[x]$을 찾는다. 이를 만족하는 g_1은 유일하게 존재한다. (Inverse FFT)
Step2 $g_0(x) = \prod_{i=1}^{n}(x - a_i), g_1(x)$에 대해 $\deg(g(x)) < \frac{1}{2}(n+k)$ 인 $g(x)$를 얻을 때까지 EEA를 적용해 아래와 같은 결과를 얻는다.

$$g(x) = u(x)g_0(x) + v(x)g_1(x)$$

Step3 g를 v로 나눈다. 몫과 나머지를 각각 f_1, r이라 할 때, $r = 0, \deg(f_1) < k$이면 f_1이 output이고, 그 이외에는 decoding에 실패한다.

Proof of Algorithm
알고리즘의 정당성을 증명하기 위해 먼저 2개의 Lemma를 증명하겠다.
Lemma1 EEA에서 $u_{m+1} = (-1)^{m+1}\frac{r_1}{r_m}, v_{m+1} = (-1)^m \frac{r_0}{r_m}$
Proof

$$\begin{pmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix} \begin{pmatrix} u_{i-1} & v_{i-1} \\ u_i & v_i \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix} \cdots \begin{pmatrix} 0 & 1 \\ 1 & -q_1 \end{pmatrix} \begin{pmatrix} u_0 & v_0 \\ u_1 & v_1 \end{pmatrix}$$

$$\det \begin{pmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{pmatrix} = \prod_{j=1}^{i} \det \begin{pmatrix} 0 & 1 \\ 1 & -q_j \end{pmatrix} = (-1)^i$$

$\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = \begin{pmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{pmatrix} \begin{pmatrix} r_0 \\ r_1 \end{pmatrix}$ $(0 \le i \le m)$ 이므로

$$\begin{pmatrix} r_0 \\ r_1 \end{pmatrix} = \begin{pmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{pmatrix}^{-1} \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = (-1)^m \begin{pmatrix} v_{i+1} & -v_i \\ -u_{i+1} & u_i \end{pmatrix} \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix}$$

$i = m$ 이면 $r_{m+1} = 0$ 이므로 $r_0 = (-1)^m v_{m+1} r_m, r_1 = (-1)^{m+1} u_{m+1} r_m$ 을 얻는다. ///

Lemma2 $\deg r_0 \le t, \deg \epsilon_i \le l \quad (i = 0, 1), \deg w_0 \ge d_0 > l + t, \gcd(r_0, r_1) = 1$ 을 만족하는 $d_0, l, t \in \mathbb{Z}, w_0, r_i, \epsilon_i \in \mathbb{F}_q[x]$ 가 존재한다고 하자. $g_0 = w_0 r_0 + \epsilon_0, g_1 = w_0 r_1 + \epsilon_1$ 에 대해 EEA를 적용하고, $\deg g < d_0$ 이면 멈춘다. EEA의 결과로 $g = ug_0 + vg_1$ 를 얻을 때, 0이 아닌 $\alpha \in \mathbb{F}_q \backslash \{0\}$ 가 존재해 $u(x) = -\alpha r_1(x), v(x) = \alpha r_0(x)$ 가 된다.
Proof 먼저 r_0, r_1 에 대해 EEA를 적용해 $r_{m+1} = 0$ 이 되었다고 가정하자. 이 때, g_0, g_1 에 대해 EEA를 적용할 때, , r_0, r_1 에 대해 EEA를 적용할 때와 같은 q_i, u_i, v_i 를 얻고, 위의 방법대로 멈추면 정확히 $k = m + 1$ 이 됨을 증명 할 것이다.
$g_i = u_i g_0 + v_i g_1 \quad (2 \le i \le m + 1)$ 로 두면 $g_{i+1} = g_{i-1} - q_i g_i \quad (1 \le i \le m)$ 를 얻는다.

$$g_i = u_i(w_0 r_0 + \epsilon_0) + v_i(w_0 r_1 + \epsilon_1) = w_0(u_i r_0 + v_i r_1) + (u_i \epsilon_1 + v_i \epsilon_1) = w_0 r_i + (u_i \epsilon_1 + v_i \epsilon_1)$$

이고,$\deg(u_i \epsilon_1 + v_i \epsilon_1) \le l + t \le d_0$ $(\deg u_i, \deg v_i \le \deg r_0)$ 이므로,

$$\deg g_i = \deg g_0 + \deg r_i \ge \deg w_0 \ge d_0 (0 \le i \le m),$$

$$\deg g_{m+1} = \deg(u_{m+1} \epsilon_1 + v_{m+1} \epsilon_1) \le l + t < d_0$$

을 얻는다. 또한, EEA의 정의에 따라, $r_1, r_2, \ldots, r_m$ 의 degree는 감소하고, 따라서 $g_1, g_2, \ldots, g_{m+1}$ 의 degree도 감소해야 함을 알 수 있다. 따라서 g_0, g_1 에 대해 EEA를 적용할 때, , r_0, r_1 에 대해 EEA를 적용할 때와 같은 q_i, u_i, v_i 를 얻고, 위의 방법대로 멈추면 정확히 $k = m + 1$ 이 되고, $g = g_{m+1}$ 임을 증명하였다.
또한, $\gcd(r_0, r_1) = 1$ 이므로 $r_m \in \mathbb{F}_q \backslash \{0\}$ 이다. $\alpha = \frac{(-1)^m}{r_m}$ 으로 두면 Lemma1에 의해 $g_{m+1} g_0 + v_{m+1} g_1$ 에서 $g = -\alpha r_1 g_0 + \alpha r_0 g_1$ 을 얻는다. ///

Proof of the Algorithm
변조된 메세지 $\mathbf{d} = (d_1, d_2, \ldots, d_n)$ 과 $f(x)$ 를 통해 encoding했던 $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ 와의 hamming distance가 $t \le \frac{d-1}{2}$ 를 만족한다고 가정하자. w, w_0 을

$$w(x) = \prod_{1 \le i \le n, c_i \ne d_i} (x - a_i), g_0(x) = w_0(x) w(x)$$

로 정의하면 $\deg w = t$ 가 된다. 그리고 inverse FFT를 통해 $\deg \bar{w} < t, c_i \ne d_i, 1 \le i \le n$ 를 만족하는 모든 i 에 대해 $\bar{w}(a_i) = \frac{d_i - c_i}{w_0(a_i)}$ 가 성립하는 유일한 $\bar{w} \in \mathbb{F}_q[x]$ 를 구할 수 있다.
또한 모든 $1 \le i \le n$ 에 대해 $g_1(a_i) = w_0(a_i)\bar{w}(a_i) + f(a_i) = c_i$ 이고, $\deg f < k$ 이므로 $g_1 = w_0 \bar{w} + f$ 가 성립한다.

$d_0 = \frac{n+k}{2}$로 둘 때, $\gcd(w, \bar{w}) = 1$이고, $\deg w_0 = n - t \geq d_0 \geq k - 1 + t \geq \deg f + \deg w$이므로 g_0, g_1에 대해 EEA를 적용하면 Lemma2에 의해 $g(x) = u(x)g_0(x) + v(x)g_1(x) = -\alpha\bar{w}(x)g_0(x) + \alpha w(x)g_1(x)$를 만족하는 $\alpha \in \mathbb{F}_q \backslash \{0\}$가 존재한다.
$g = -\alpha\bar{w}g_0 + \alpha w g_1 = -\alpha\bar{w}g_0 + \alpha w(w_0\bar{w} + f) = \alpha w f = v f$이므로 g를 v로 나눈 나머지는 0이 되어야 하고, 이 알고리즘에서 얻은 결과는 f로 원하는 결과를 얻었다.
반대로, 이 알고리즘을 통해 $f_1(x)(\deg f_1 < k)$를 얻었다고 가정하자. Step2,Step3에서 얻은 식에 의해 $u(x)g_0(x) = v(x)(f_1(x) - g_1(x))$이고, 따라서 $v(a_i)(f_1(a_i) - g_1(a_i)) = 0 (1 \leq i \leq n)$이다. EEA를 적용한 결과가 $g(x) = g_{m+1}(x)$라고 하면 $\deg g_m \geq \frac{n+k}{2}$이므로, $\deg v = \deg g_0 - \deg g_m \leq n - \frac{n+k}{2} = \frac{d-1}{2}$를 얻는다. 따라서 $n - \frac{d-1}{2}$개 이상의 i에 대해 $f_1(a_i) = g_1(a_i)$이 성립하고, f_1으로 encoding된 메세지와 channel을 통해 받은 메세지의 hamming distance가 $\frac{d-1}{2}$이하임을 알 수 있다. 이 사실을 통해 알고리즘이 제대로 동작함을 알 수 있다. ///

4 Application for Group Testing Problem

Group Testing Problem은 2차 세계대전 때, 미국에서 군에 지원하는 사람들을 대상으로 매독검사를 효율적으로 하기 위해 고안된 문제이다. 이 때, 매독검사의 비용이 비쌌고 감염된 사람의 비율도 적었기 때문에 정부에서는 최소한의 검사를 통해 감염된 사람을 추려내고 싶어 했고, 이 문제를 당시 통계학자이자 미국 연방 정부의 연구원이었던 Robert Dorfman이 1943년 부분적인 해법을 제시하면서 수학자들에게 알려지게 되었다. 이 문제를 조금 더 정확하게 서술하면 다음과 같다.

n명의 사람이 있고, 그 중 $k \leq n$명 이하의 사람이 감염되어 있다는 사실을 알고 있다고 가정하자. 검사 T번 한다고 가정할 때, $A_i \subset \{1, 2, \ldots, T\} \quad (1 \leq i \leq n)$이 i번째 사람이 받은 검사들의 집합이라 하고, $\mathbb{A} = \{A_1, \ldots, A_n\}$으로 두자. 만약 어떤 검사가 음성 반응이 나오면 그 검사를 받은 사람들은 모두 감염되지 않았다고 확신할 수 있다. 따라서 $\mathbb{A}$에서 감염된 k명과 대응되는 A_i들의 합집합을 구한 결과가 A라 할 때, $A_i \not\subseteq A$이면 i번 째 사람은 감염된 사람들이 아무도 받지 않은 검사를 받았다는 것이고, i번 째 사람이 감염되지 않았음을 확신할 수 있다. 따라서$\mathbb{A}$에서 임의의 집합 k개의 합집합을 구했을 때, 나머지 집합에서 그 합집합에 속하지 않은 원소가 항상 나오게 되면 그 검사로 감염된 사람들을 정확히 골라낼 수 있다. 이 때 T의 값을 최소화 하는 것이 이 문제의 목표이다.
Reed-Solomon Code를 이용하면 $T = O\left(\left(k\frac{\log n}{\log k}\right)^2\right)$가 되도록 T를 잡을 수 있다. 증명은 아래와 같다.
Proof이 증명에서는 Reed-Solomon Code인 $[q, m, q - m + 1]_q$ Code를 사용할 것이고 이 Code를 C라 하자. $|\mathbb{F}_q| = q$이므로 bijective $f : \mathbb{F}_q \to \{1, \ldots, q\}$가 존재한다. 이 때, $c = (c_1, \ldots, c_q) \in C$에 대해 $A_c = \{q(i-1) + f(c_i) | 1 \leq i \leq q\} \subset \{1, \ldots, q^2\}$로 정의하고, $\mathbb{A} = \{A_c | c \in C\}$라 두자. 이 때, 모든 $c \in C$에 대해 $|A_c| = q$이고, 서로 다른 $c_1, c_2 \in C$에 대해 $|A_{c_1} \cap A_{c_2}| = q - d(c_1, c_2) \leq q - (q - m + 1) = m - 1$이다.

$k = \lfloor \frac{q-1}{m-1} \rfloor$ 로 두면 서로 다른 $c, c_1, \ldots, c_k \in C$ 에 대해

$$|A_c \cap (A_{c_1} \cup \cdots \cup A_{c_k})| = |\bigcup_{i=1}^{k} (A_c \cap A_{c_i})| \leq \sum_{i=1}^{k} |A_c \cup A_{c_i}| \leq k(m-1) \leq q-1$$

이 성립하고,

$$A_c \not\subseteq A_{c_1} \cup \cdots \cup A_{c_k}$$

이 된다. 따라서, 만약, k 명 이하가 감염되었다는 사실을 알면, $n = q^m$ 명의 사람에 대해 $T = q^2$ 번의 test를 통해 감염된 사람들을 모두 골라낼 수 있다는 사실을 알 수 있다.
n, k 가 주어진 경우 $2k\frac{\log n}{\log k} \leq q \leq 4k\frac{\log n}{\log k}$ 인 소수 q 를 잡을 수 있다(by Bertrand's postulate).
$m = \left\lceil \frac{\log n}{\log q} \right\rceil$ 으로 잡으면,

$$\left\lfloor \frac{q-1}{m-1} \right\rfloor \geq \left\lfloor \frac{2k\frac{\log n}{\log k} - 1}{\frac{\log n}{\log q} + 1 - 1} \right\rfloor = \left\lfloor 2k\frac{\log q}{\log k} - \frac{\log q}{\log n} \right\rfloor \geq k$$

이고, $T = O\left(\left(k\frac{\log n}{\log k}\right)^2\right)$ 를 만족함을 알 수 있다. ///

5 Fast Fourier Transform(FFT)

이 section에서는 Reed-Solomon Code의 encoding과 decoding을 빠르게 할 수 있게 하는 Fast Fourier Transform(FFT)에 대해 다룰 것이다. 이를 이해하기 위해서는 finite field의 성질들과 추상적인 vector space에 대한 이해가 필요하므로 현대대수학을 수강하지 않은 경우 이 section을 무시해도 좋다. 이 section에서는 편의상 $n = p^m - 1$ 로 둔다.

$\mathbb{F}_{p^m}$ 위에서 $c(x) \in \mathbb{F}_{p^m}[x]$ Fourier Transform C 는 $C_k = c(\alpha^k) \quad (0 \leq k \leq n-1)$ 로 정의한다(α 는 $\mathbb{F}_{p^m}$ 의 primitive element). $c(x) = \sum_{k=0}^{n-1} c_k x^k$ 로 나타낼 수 있을 때, Fourier Transform을 행렬로 나타내면 다음과 같다.

$$\begin{pmatrix} C_0 & C_1 & \cdots & C_{n-1} \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & \cdots & c_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \alpha & \cdots & \alpha^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{n-1} & \cdots & \alpha^{(n-1)(n-1)} \end{pmatrix}$$

반대로, $C_k \in \mathbb{F}_{p^m} (0 \leq k \leq n-1)$ 이 주어져 있을 때, $C_k = c(\alpha^k)$ 를 만족하는 $c(x) = \sum_{k=0}^{n-1} c_k x^k$ 를 구하는 것을 Inverse Fourier Transform이라 한다.

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \alpha & \cdots & \alpha^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{n-1} & \cdots & \alpha^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \alpha^{-1} & \cdots & \alpha^{-(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{-(n-1)} & \cdots & \alpha^{-(n-1)(n-1)} \end{pmatrix} = n\mathbf{I}_n$$

가 성립하고, $n \equiv -1 \pmod p$ 이기 때문에, Inverse Fourier Transform은 행렬로 표현하면 다음과 같다.

$$\begin{pmatrix} c_0 & c_1 & \cdots & c_{n-1} \end{pmatrix} = -\begin{pmatrix} C_0 & C_1 & \cdots & C_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \alpha^{-1} & \cdots & \alpha^{-(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{-(n-1)} & \cdots & \alpha^{-(n-1)(n-1)} \end{pmatrix}$$

따라서, Inverse Fourier Transform도 Fourier Transform과 비슷한 방법으로 할 수 있다.

Fast Fourier Transform은 $c(x) = \sum_{k=0}^{n-1} c_k x^k \in \mathbb{F}_{p^m}[x]$ 이 주어져 있을 때, $(c(\alpha^0), \ldots, c(\alpha^{n-1}))$ 를 빠르게 계산하는 것이 목적이다. $\alpha^k (0 \leq k \leq n-1)$ 의 값을 모두 알고 있다고 가정할 때, 정의에 따라 계산하면 $O(n^2)$ 번의 덧셈과 곱셈을 통해 계산할 수 있다. FFT를 통해 $O(n(\log n)^2)$ 번의 덧셈과 곱셈으로 $(c(\alpha^0), \ldots, c(\alpha^{n-1}))$ 를 얻을 수 있다. 이 알고리즘을 간단히 설명하면 C_{α^j} 를 구하기 위해 $c(x)$ 를 $x - \alpha^j$ 로 나누는 방법을 사용했고, 빠르게 계산하기 위해 적절한 다항식을 골라서 여러번 나눗셈을 해준 것이다.

먼저 FFT를 수행하는데 필요한 정리를 설명한 후, FFT 알고리즘에 대해 설명할 것이다.

Theorem $\mathbb{F}_{p^m}$ 의 원소들을 $\beta_{0,0}, \beta_{0,1}, \ldots, \beta_{0,p^m-1}$ 로 재배열해서 다음과 같은 성질을 만족하도록 할 수 있다.

$$q_{0,j}(x) = x - \beta_{0,j} \quad (0 \leq j < p^m)$$

$$q_{i,j}(x) = \prod_{k=0}^{p-1} q_{i-1,jp+k}(x), Q_{i,j} = \prod_{k=0}^{p^i-1} \beta_{0,jp^i+k} \quad (1 \leq i \leq m, 0 \leq j < p^{m-i})$$

로 정의하면 $q_{i,0}$ 은

$$q_{i,0} = \sum_{k=0}^{i} c_{i,k} x^{p^k}$$

와 같은 꼴이 되고, $q_{i,j}(x) = q_{i,0}(x) - Q_{i,j}$ 를 만족한다.

Algorithm of FFT

Input: $c(x) \in \mathbb{F}_{p^m}[x] (\deg c < n)$

Output: $(c(\alpha^0), \ldots, c(\alpha^{n-1}))$

Step1 $r_{m,0} = c(x), i = m$

Step2 $0 \leq j < p^{m-i}$ 에 대해 $r_{i-1,jp+k}(x) = r_{i,j}(x) \mod (q_{i-1,jp+k}(x))$

Step3 $i = i - 1$

Step4 $i > 0$ 이면 Step2, $i = 0$ 이면 Step5

Step5 $0 \leq j < p^m - 1$ 에 대해 $\beta_{0,k} = \alpha^j$ 인 j 를 찾고, $C_j = r_{0,k}$

Step6 return $(C_0, \ldots, C_{p^m-2})$

이 알고리즘의 Step2에서 각각의 나눗셈에 대해 최대 $(p^i - p^{i-1})(i+1)$ 번의 덧셈과 곱셈이 각각 이루어지고 p^{m-i+1} 번의 나눗셈을 하므로 각 i 에서 $p^{m-i+1}(p^i - p^{i-1})(i+1) = (p^{m+1} - p^m)(i+1)$ 번의 연산이 이루어진다. 모든 $1 \leq i \leq m$ 에 대해 더해주면 $n = p^m - 1, m = \log_p p^m$ 이므로

이 알고리즘에서 $O(n(\log n)^2)$ 번의 연산이 이루어짐을 알 수 있다.

Theorem을 증명하기 전 증명에 필요한 Lemma를 증명하겠다.

Lemma $\mathbb{F}_{p^m}$ 을 $\mathbb{F}_p$-vector space로 볼 때, V 를 $\mathbb{F}_{p^m}$ 의 subspace라 하자. 이 때,$\dim V = d$이면,

$$f_V(x) := \prod_{v \in V}(x - v) = \sum_{k=0}^{d} c_k x^{p^k}$$

인 $c_0, \dots, c_d \in \mathbb{F}_{p^m}$ 이 존재한다.

Proof $0 \le i \le d$에 대해 $f_i(x) := x^{p^i}$ 는 $\mathbb{F}_p$-linear function이고, $\{f | f : V \to \mathbb{F}_{p^m}$은 $\mathbb{F}_p$-linear$\}$은 dimension이 d인 $\mathbb{F}_{p^m}$-vector space이다($\{\gamma_1, \dots, \gamma_d\}$가 V 의 basis일 때, $f(\gamma_1), \dots, f(\gamma_d)$가 정해지면 f가 유일하게 정해지기 때문에 dimension이 d인 $\mathbb{F}_{p^m}$-vector space로 생각할 수 있다).

따라서, $f_0, \dots, f_d$는 linearly dependent하고, 모든 $x \in V$에 대해 $g_V(x) = 0$을 만족하는 $g_V(x) = \sum_{k=0}^{d} a_k x^{p^k} \in \mathbb{F}_{p^m}[x], g_V \neq 0$가 존재한다.$f_V | g_V, \deg g_V \le \deg f_V = p^d$이므로 $\deg g_V = \deg f_V, a_d \neq 0$이다. 따라서

$$f_V(x) = \sum_{k=0}^{d} \frac{a_k}{a_d} x^{p^k}$$

가 성립한다. ///

Proof of the Theorem

$\mathbb{F}_{p^m}$을 $\mathbb{F}_p$-vector space로 볼 때, $\mathbb{F}_{p^m}$ 의 basis를 $\{\gamma_0, \dots, \gamma_{m-1}\}$라 하자. $k \in \mathbb{Z}, 0 \le k < p^m$이 $k = \sum_{i=0}^{m-1} a_i p^i (0 \le a_i < q)$로 표현될 때, $\beta_{0,k} = a_0\gamma_0 + \dots + a_{m-1}\gamma_{m-1}$로 정의하자.

$$U_j^i = \{\beta_{0,k} | jp^i \le k < (j+1)p^i\} \quad (1 \le i \le m, 0 \le j < p^{m-i})$$

로 정의하면 $U_0^i (1 \le i \le m)$는 $U_0^i \supset U_0^{i-1}$를 만족하는$\mathbb{F}_{p^m}$ 의 subspace 이고,
$U_j^i = \{\beta + \beta_{0,jp^i} | \beta \in U_0^i\}$이다. 따라서

$$U_j^i = \bigcup_{k=0}^{p-1} U_{jp+k}^{i-1} \quad (1 \le i \le m, 0 \le j < p^{m-i})$$

로 분할할 수 있고, 이를 이용하면 수학적 귀납법으로

$$q_{i,j}(x) = \prod_{\beta \in U_j^i} (x - \beta)$$

임을 보일 수 있다.

Lemma에 의해 $q_{i,0}$은 $q_{i,0}(x) = \sum_{k=0}^{i} c_{i,k} x^{p^k}$ 와 같은 꼴로 나타낼 수 있다. 또한,

$$q_{i,j}(x) = \prod_{\beta \in U_j^i} (x - \beta) = \prod_{\beta \in U_0^i} (x - \beta - \beta_{0,jp^i}) = q_{i,0}(x - \beta_{0,jp^i})$$

$$= \sum_{k=0}^{i} c_{i,k}(x - \beta_{0,jp^i})^{p^k} = \sum_{k=0}^{i} c_{i,k}x^{p^k} + \sum_{k=0}^{i} c_{i,k}(-\beta_{0,jp^i})^{p^k}$$

$$= q_{i,0}(x) + q_{i,0}(-\beta_{0,jp^i}) = q_{i,0}(x) + \prod_{\beta \in U_0^i} (-\beta_{0,jp^i} - \beta) = q_{i,0}(x) + \prod_{\beta \in U_j^i} (-\beta)$$

$$= q_{i,0}(x) + \prod_{\beta \in U_j^i} (-\beta) = q_{i,0}(x) + (-1)^{p^i} Q_{i,j} = q_{i,0}(x) - Q_{i,j}$$

이다. ///