곱셈 알고리즘

IT정보공학과 신명수

- 양자내성암호에서 큰 크기의 polynomial의 곱셈을 효율적으로 설계하는 것이 중요하다

곱셈 알고리즘

- 양자내성암호에서 큰 크기의 polynomial의 곱셈을 효율적으로 설계하는 것이 중요하다.

A

Noise vector e

Mod 7

$$\begin{bmatrix} 0 & 5 & 2 & 3 \\ 1 & 3 & 6 & 5 \\ 3 & 0 & 4 & 5 \\ 4 & 6 & 5 & 3 \\ 1 & 0 & 6 & 5 \\ 4 & 5 & 2 & 4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 6 \\ 1 \\ 1 \\ 0 \\ 6 \end{bmatrix} = \begin{bmatrix} 6 \\ 1 \\ 0 \\ 5 \\ 2 \\ 3 \end{bmatrix} \quad \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

목차

1. 고전적인 곱셈 방법
2. Karatsuba algorithm
3. Toom-Cook algorithm
4. FFT

1. 고전적인 곱셈 방법

- $995831 \times 274985 = ?$

1. 고전적인 곱셈 방법

- $995831 \times 274985 = ?$

						9	9	5	8	3	1
					×	2	7	4	9	8	5
					4	9	7	9	1	5	5
				7	9	6	6	6	4	8	
			8	9	6	2	4	7	9		
		3	9	8	3	3	2	4			
	6	9	7	0	8	1	7				
1	9	9	1	6	6	2					
2	7	3	8	3	8	5	8	7	5	3	5

1. 고전적인 곱셈 방법

- $995831 \times 274985 = ?$

						9	9	5	8	3	1
					×	2	7	4	9	8	5
+						45	45	25	40	15	5
+					72	72	40	64	24	8	
+				81	81	45	72	27	9		
+			36	36	20	32	12	4			
+		63	63	35	56	21	7				
+	18	18	10	16	6	2					
2	7	3	8	3	8	5	8	7	5	3	5

1. 고전적인 곱셈 방법

- $995831 \times 274985 = ?$

```
vector<int> mul1(vector<int>& a, vector<int>& b)
{
    vector<int> ret(a.size()+b.size()+2, 0);
    reverse(a.begin(), a.end());
    reverse(b.begin(), b.end());
    for(size_t i=0;i<b.size();i++)
    {
        for(size_t j=0;j<a.size();j++)
        {
            ret[i+j] += a[j] * b[i];
        }
    }
    int carry = 0;
    for(size_t i=0;i<ret.size();i++)
    {
        ret[i] += carry;
        carry = ret[i]/10;
        ret[i] %= 10;
    }
    while(ret.back() == 0) ret.pop_back();
    reverse(ret.begin(), ret.end());
    return ret;
}
```

1. 고전적인 곱셈 방법

두 n자리수 곱셈

- 시간복잡도 : $O(n^2)$

2. Karatsuba algorithm

두 n자리수 곱셈을 $O\left(n^{log_2 3}\right)$ 회만에 수행.

2. Karatsuba algorithm

- $995831 \times 274985 = ?$

2. Karatsuba algorithm

- $995831 \times 274985 = ?$

$$995831 \times 274985 = (995 \times 10^3 + 831) \times (274 \times 10^3 + 985)$$

$$a_0 = 831, \qquad a_1 = 995$$

$$b_0 = 985, \qquad b_1 = 274$$

$$(a_1 \times 10^3 + a_0) \times (b_1 \times 10^3 + b_0) = a_1 b_1 \times 10^6 + (a_1 b_0 + a_0 b_1) \times 10^3 + a_0 b_0$$

2. Karatsuba algorithm

$$(a_1 \times 10^3 + a_0)(b_1 \times 10^3 + b_0) = a_1 b_1 \times 10^6 + (a_1 b_0 + a_0 b_1) \times 10^3 + a_0 b_0$$

$$c_2 = a_1 b_1, \qquad c_0 = a_0 b_0$$

$$c_1 = (a_1 b_0 + a_0 b_1)$$

$$\begin{aligned}(a_1 + a_0)(b_1 + b_0) &= a_1 b_1 + a_1 b_0 + a_0 b_1 + a_0 b_0 \\ &= c_2 + c_1 + c_0\end{aligned}$$

$$c_1 = (a_1 + a_0)(b_1 + b_0) - c_2 - c_0$$

2. Karatsuba algorithm

$$(a_1 \times 10^3 + a_0)(b_1 \times 10^3 + b_0) = \boxed{a_1 b_1} \times 10^6 + (\boxed{a_1 b_0} + \boxed{a_0 b_1}) \times 10^3 + \boxed{a_0 b_0}$$

$$c_2 = a_1 b_1, \qquad c_0 = a_0 b_0$$

$$c_1 = (a_1 + a_0)(b_1 + b_0) - c_2 - c_0$$

$$(a_1 \times 10^3 + a_0)(b_1 \times 10^3 + b_0)$$

$$= \boxed{a_1 b_1} \times 10^6 + (\boxed{(a_1 + a_0)(b_1 + b_0)} - a_1 b_1 - a_0 b_0) \times 10^3 + \boxed{a_0 b_0}$$

2. Karatsuba algorithm

$$a_1 b_1 \times 10^6 + \big((a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0\big) \times 10^3 + a_0 b_0$$

$$a_1 b_1 = 995 \ \times 274 = 272{,}630$$

$$a_0 b_0 = 831 \ \times 985 = 818{,}535$$

$$(a_1 + a_0)(b_1 + b_0) = (995 + 831)(274 + 985) = 1826 \times 1259 = 2{,}298{,}934$$

$$= 272{,}630{,}000{,}000 + (2{,}298{,}934 - 272{,}630 - 818{,}535) \times 10^3 + 818{,}535$$

$$= 272{,}630{,}000{,}000 + 1{,}207{,}769{,}000 + 818{,}535$$

$$= 273{,}838{,}587{,}535$$

2. Karatsuba algorithm

$$(a_1 \times K^m + a_0)(b_1 \times K^m + b_0) = a_1 b_1 \times K^{2m} + (a_1 b_0 + a_0 b_1) \times K^m + a_0 b_0$$

$$c_2 = a_1 b_1, \qquad c_0 = a_0 b_0$$

$$c_1 = (a_1 + a_0)(b_1 + b_0) - c_2 - c_0$$

$$(a_1 \times K^m + a_0)(b_1 \times K^m + b_0)$$

$$= a_1 b_1 \times K^{2m} + ((a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0) \times K^m + a_0 b_0$$

2. Karatsuba algorithm

$$a_1 b_1 \times K^{2m} + ((a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0) \times K^m + a_0 b_0$$

- 모든 K와 m에 대해 작동하지만 $m = n/2$ 일때 가장 효율적임.
- 양의 정수 K 에 대해 $\mathrm{n} = 2^K$ 이고, 재귀가 $n = 1$ 일때 끝난다면 한 자리 곱셈의 횟수는 3^k이 된다.

2. Karatsuba algorithm

```cpp

vector<int> karatsuba(vector<int>& a, vector<int>& b)
{
    int alen = a.size(), blen = b.size();
    if(alen<blen) return karatsuba(b, a);
    if(alen==0 || blen==0) return vector<int>();
    if(alen<=250) return multiple(a, b);

    int half = alen/2;
    vector<int> a0(a.begin(), a.begin()+half);
    vector<int> a1(a.begin()+half, a.end());
    vector<int> b0(b.begin(), b.begin()+min((int)b.size(), half));
    vector<int> b1(b.begin()+min((int)b.size(), half), b.end());

    vector<int> z2 = karatsuba(a1, b1);
    vector<int> z0 = karatsuba(a0, b0);
    addto(a0, a1, 0);
    addto(b0, b1, 0);
    vector<int> z1 = karatsuba(a0, b0);
    subfrom(z1, z0);
    subfrom(z1, z2);

    vector<int> ret;
    addto(ret, z0, 0);
    addto(ret, z1, half);
    addto(ret, z2, half+half);
    return ret;
}

```

2. Karatsuba algorithm

```
vector<int> karatsuba(vector<int>& a, vector<int>& b)
{
    int alen = a.size(), blen = b.size();
    if(alen<blen) return karatsuba(b, a);
    if(alen==0 || blen==0) return vector<int>();
    if(alen<=250) return multiple(a, b);

    int half = alen/2;
    vector<int> a0(a.begin(), a.begin()+half);
    vector<int> a1(a.begin()+half, a.end());
    vector<int> b0(b.begin(), b.begin()+min((int)b.size(), half));
    vector<int> b1(b.begin()+min((int)b.size(), half), b.end());

    vector<int> z2 = karatsuba(a1, b1);
    vector<int> z0 = karatsuba(a0, b0);
    addto(a0, a1, 0);
    addto(b0, b1, 0);
    vector<int> z1 = karatsuba(a0, b0);
    subfrom(z1, z0);
    subfrom(z1, z2);

    vector<int> ret;
    addto(ret, z0, 0);
    addto(ret, z1, half);
    addto(ret, z2, half+half);
    return ret;
}
```

2. Karatsuba algorithm

```
vector<int> karatsuba(vector<int>& a, vector<int>& b)
{
    int alen = a.size(), blen = b.size();
    if(alen<blen) return karatsuba(b, a);
    if(alen==0 || blen==0) return vector<int>();
    if(alen<=250) return multiple(a, b);

    int half = alen/2;
    vector<int> a0(a.begin(), a.begin()+half);
    vector<int> a1(a.begin()+half, a.end());
    vector<int> b0(b.begin(), b.begin()+min((int)b.size(), half));
    vector<int> b1(b.begin()+min((int)b.size(), half), b.end());

    vector<int> z2 = karatsuba(a1, b1);
    vector<int> z0 = karatsuba(a0, b0);
    addto(a0, a1, 0);
    addto(b0, b1, 0);
    vector<int> z1 = karatsuba(a0, b0);
    subfrom(z1, z0);
    subfrom(z1, z2);

    vector<int> ret;
    addto(ret, z0, 0);
    addto(ret, z1, half);
    addto(ret, z2, half+half);
    return ret;
}
```

2. Karatsuba algorithm

```
vector<int> karatsuba(vector<int>& a, vector<int>& b)
{
    int alen = a.size(), blen = b.size();
    if(alen<blen) return karatsuba(b, a);
    if(alen==0 || blen==0) return vector<int>();
    if(alen<=250) return multiple(a, b);

    int half = alen/2;
    vector<int> a0(a.begin(), a.begin()+half);
    vector<int> a1(a.begin()+half, a.end());
    vector<int> b0(b.begin(), b.begin()+min((int)b.size(), half));
    vector<int> b1(b.begin()+min((int)b.size(), half), b.end());

    vector<int> z2 = karatsuba(a1, b1);
    vector<int> z0 = karatsuba(a0, b0);
    addto(a0, a1, 0);
    addto(b0, b1, 0);
    vector<int> z1 = karatsuba(a0, b0);
    subfrom(z1, z0);
    subfrom(z1, z2);

    vector<int> ret;
    addto(ret, z0, 0);
    addto(ret, z1, half);
    addto(ret, z2, half+half);
    return ret;
}
```

2. Karatsuba algorithm

```
vector<int> karatsuba(vector<int>& a, vector<int>& b)
{
    int alen = a.size(), blen = b.size();
    if(alen<blen) return karatsuba(b, a);
    if(alen==0 || blen==0) return vector<int>();
    if(alen<=250) return multiple(a, b);

    int half = alen/2;
    vector<int> a0(a.begin(), a.begin()+half);
    vector<int> a1(a.begin()+half, a.end());
    vector<int> b0(b.begin(), b.begin()+min((int)b.size(), half));
    vector<int> b1(b.begin()+min((int)b.size(), half), b.end());

    vector<int> z2 = karatsuba(a1, b1);
    vector<int> z0 = karatsuba(a0, b0);
    addto(a0, a1, 0);
    addto(b0, b1, 0);
    vector<int> z1 = karatsuba(a0, b0);
    subfrom(z1, z0);
    subfrom(z1, z2);

    vector<int> ret;
    addto(ret, z0, 0);
    addto(ret, z1, half);
    addto(ret, z2, half+half);
    return ret;
}
```

2. Karatsuba algorithm

```
vector<int> karatsuba(vector<int>& a, vector<int>& b)
{
    int alen = a.size(), blen = b.size();
    if(alen<blen) return karatsuba(b, a);
    if(alen==0 || blen==0) return vector<int>();
    if(alen<=250) return multiple(a, b);

    int half = alen/2;
    vector<int> a0(a.begin(), a.begin()+half);
    vector<int> a1(a.begin()+half, a.end());
    vector<int> b0(b.begin(), b.begin()+min((int)b.size(), half));
    vector<int> b1(b.begin()+min((int)b.size(), half), b.end());

    vector<int> z2 = karatsuba(a1, b1);
    vector<int> z0 = karatsuba(a0, b0);
    addto(a0, a1, 0);
    addto(b0, b1, 0);
    vector<int> z1 = karatsuba(a0, b0);
    subfrom(z1, z0);
    subfrom(z1, z2);

    vector<int> ret;
    addto(ret, z0, 0);
    addto(ret, z1, half);
    addto(ret, z2, half+half);
    return ret;
}
```

```
Typical : 12.165
karatsuba : 1.223
[Finished in 15.1s]
```

3. Toom-Cook algorithm

- Karatsuba algorithm을 일반화한 방법.
  - Karatsuba algorithm은 각 수를 2분할로 나눴다면 Toom-Cook algorithm은 k분할로 나눈다.

3. Toom-Cook Algorithm

- 5단계를 거쳐 완료된다.

1. 분할
2. 평가
3. 점별 곱셈
4. 보간
5. 합성

3. Toom-Cook Algorithm

1. 분할

정수 m, n을 k개의 조각으로 자르기 위한 적절한 단위인 $B = b^i$를 찾는 단계.

$$i = \max\{\lfloor \lfloor \log_b m \rfloor / k \rfloor, \lfloor \lfloor \log_b n \rfloor / k \rfloor\} + 1$$

3. Toom-Cook Algorithm

$$m = 1{,}234{,}567{,}890{,}123{,}456{,}789{,}012$$

$$n = 987{,}654{,}321{,}987{,}654{,}321{,}098$$

해당 예시에서는 편의상 $b = 10^4$를 사용한다.

3개의 조각으로 자를 것이므로 $k = 3$이고 아래 식에 따라 $B = b^2 = 10^8$이다.

$$i = \max\{\lfloor\lfloor \log_b m \rfloor / k \rfloor, \lfloor\lfloor \log_b n \rfloor / k \rfloor\} + 1$$

3. Toom-Cook Algorithm

$B = b^2 = 10^8$에 따라 m, n을 자르면 다음과 같이 된다.

$$m = 1{,}234{,}567{,}890{,}123{,}456{,}789{,}012$$

$$n = 987{,}654{,}321{,}987{,}654{,}321{,}098$$

$$m_2 = 123456$$

$$m_1 = 78901234$$

$$m_0 = 56789012$$

$$n_2 = 98765$$

$$n_1 = 43219876$$

$$n_0 = 54321098$$

3. Toom-Cook Algorithm

이 수를 계수로 하는 $k-1$차 다항식을 만들면 다음과 같다.

$$p(x) = m_2x^2 + m_1x + m_0 = 123456x^2 + 78901234x + 56789012$$

$$\mathrm{q}(x) = n_2x^2 + n_1x + n_0 = 98765x^2 + 43219876x + 54321098$$

이 두 다항식의 곱 $r(x) = p(x)q(x)$를 정의한다.

$x = B$를 대입하면 $p(B) = m, q(B) = n$ 이 되고 $r(B) = mn$이 된다.

3. Toom-Cook Algorithm

2. 평가

$r(x)$의 각 계수를 구하기 위해 적당한 x를 선정하여 $p(x), q(x)$의 값을 구한다.

$$p(0) = m_0 + m_1(0) + m_2(0)^2 = m_0$$

$$p(1) = m_0 + m_1(1) + m_2(1)^2 = m_0 + m_1 + m_2$$

$$p(-1) = m_0 + m_1(-1) + m_2(-1)^2 = m_0 - m_1 + m_2$$

$$p(-2) = m_0 + m_1(-2) + m_2(-2)^2 = m_0 - 2m_1 + 4m_2$$

$$p(\infty) = m_2, (\lim_{x \to \infty} p(x))$$

3. Toom-Cook Algorithm

$$q(0) = n_0 + n_1(0) + n_2(0)^2 = n_0$$

$$q(1) = n_0 + n_1(1) + n_2(1)^2 = n_0 + n_1 + n_2$$

$$q(-1) = n_0 + n_1(-1) + n_2(-1)^2 = n_0 - n_1 + n_2$$

$$q(-2) = n_0 + n_1(-2) + n_2(-2)^2 = n_0 - 2n_1 + 4n_2$$

$$q(\infty) = n_2,$$

3. Toom-Cook Algorithm

$$\begin{pmatrix} p(0) \\ p(1) \\ p(-1) \\ p(-2) \\ p(\infty) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & -2 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} m_0 \\ m_1 \\ m_2 \end{pmatrix}$$

$$\begin{pmatrix} q(0) \\ q(1) \\ q(-1) \\ q(-2) \\ q(\infty) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & -2 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} n_0 \\ n_1 \\ n_2 \end{pmatrix}$$

3. Toom-Cook Algorithm

3. 점별곱셈

두 다항식 $p(x), q(x)$을 통째로 곱하는 대신 앞에서 구한 여러개의 지점에 대해서 $p(a)q(a)$를 구한다. 크기가 작아 고전 곱셈법으로도 충분히 빠르게 구할 수 있다.

$$r(0) = p(0)q(0)$$

$$r(1) = p(1)q(1)$$

$$r(-1) = p(-1)q(-1)$$

$$r(-2) = p(-2)q(-2)$$

$$r(\infty) = p(\infty)q(\infty)$$

3. Toom-Cook Algorithm

4. 보간

점별 곱셈 결과를 이용하여 다항식 $r(x)$의 미지계수를 구하는 과정이다.

$$\begin{pmatrix} r(0) \\ r(1) \\ r(-1) \\ r(-2) \\ r(\infty) \end{pmatrix} = \begin{pmatrix} 0^0 & 0^1 & 0^2 & 0^3 & 1^4 \\ 1^0 & 1^1 & 1^2 & 1^3 & 1^4 \\ -1^0 & -1^1 & -1^2 & -1^3 & -1^4 \\ -2^0 & -2^1 & -2^2 & -2^3 & -2^4 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{pmatrix}$$

3. Toom-Cook Algorithm

가우스 소거법을 통해 역행렬을 구할 수도 있지만 미리 구한 점들이 단순한 점들이기에 복잡한 계산없이 행렬을 풀어낼 수 있다.

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{pmatrix} = \begin{pmatrix} 0^0 & 0^1 & 0^2 & 0^3 & 1^4 \\ 1^0 & 1^1 & 1^2 & 1^3 & 1^4 \\ -1^0 & -1^1 & -1^2 & -1^3 & -1^4 \\ -2^0 & -2^1 & -2^2 & -2^3 & -2^4 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} r(0) \\ r(1) \\ r(-1) \\ r(-2) \\ r(\infty) \end{pmatrix}$$

3. Toom-Cook Algorithm

다음은 Marco Bodraco가 제시한 $k = 3$일때의 효율적인 방법이다.

$$r_0 \leftarrow r(0)$$

$$r_4 \leftarrow r(\infty)$$

$$r_3 \leftarrow \frac{r(-2) - r(1)}{3}$$

$$r_1 \leftarrow \frac{r(1) - r(-1)}{2}$$

$$r3 \leftarrow \frac{r_2 - r_3}{2} + 2r(\infty)$$

$$r_2 \leftarrow r_2 + r_1 - r_4$$

$$r_1 \leftarrow r_1 - r_3$$

3. Toom-Cook Algorithm

이를 계산하면 $r(x)$를 알 수있다.

$$
\begin{aligned}
r(x) = {} & 3084841486175176 \\
& +6740415721237444x \\
& +3422416581971852x^2 \\
& +13128433387466x^3 \\
& +12193131840x^4
\end{aligned}
$$

3. Toom-Cook Algorithm

5. 합성

구한 $r(x)$에 $x = B(= b^2, b = 10^4)$를 대입하여 최종적으로 $m \times n$을 구할 수 있다.

							3084	**8414**	**8617**	**5176**
					6740	**4157**	**2123**	**7444**		
			3422	**4165**	**8197**	**1852**				
	13	**1284**	**3338**	**7466**						
121	**9313**	**1840**								
121	**9326**	**3124**	**6761**	**1632**	**4937**	**6009**	**5208**	**5858**	**8617**	**5176**

3. Toom-Cook algorithm

- 시간복잡도 : $O\left(n^{\log_2 5/\log_2 3}\right)\ if\ k = 3$
- Karatsuba algorithm은 $k = 2$인 Toom-Cook algorithm이다.

- k값을 증가시켜 거의 $O(n)$에 가깝게 낮출 수 있지만, 부가연산의 수가 급속하게 커져 소용이 없다.

4. FFT

- 준비중입니다.

감사합니다