

암호학과 네트워크 보안

-9장 암호수학 (2)-

이 성 현(dltjdgus2166@naver.com)

세종대학교 프로토콜공학연구실

목 차

- 소인수분해(Factorization)
- 중국인의 나머지 정리(Chinese Remainder Theorem)
- 2차 합동(Quadratic Congruence)

목 차

- 소인수분해(Factorization)
- 중국인의 나머지 정리(Chinese Remainder Theorem)
- 2차 합동(Quadratic Congruence)

소인수분해

- 정의

- 1보다 큰 자연수를 소인수의 곱으로 나타내는 방법

- 산술 기본 정리(Fundamental Theorem of Arithmetic)
(1/3)

- 1보다 큰 모든 자연수는 하나 이상의 소수의 곱으로 표현됨
 - $n = p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_k^{e_k}$ 로 표현됨
 - e.g., $72 = 2^3 \times 3^2$
- 하나의 정수는 소수의 순서를 제외하면 유일하게 표현됨
 - e.g., $360 = 2^3 \times 3^2 \times 5^1$ 로만 표현됨

소인수분해

• 산술 기본 정리(Fundamental Theorem of Arithmetic) (2/3)

$\gcd(a, b)$: a 와 b 의 최대공약수	$\text{lcm}(a, b)$: a 와 b 의 최소공배수
$\min(a, b)$: a 와 b 중 작은 수	$\max(a, b)$: a 와 b 중 큰 수

• 해당 정리를 이용하여 최대공약수를 구할 수 있음

- $a = p_1^{a_1} \times p_2^{a_2} \times \dots \times p_k^{a_k}, b = p_1^{b_1} \times p_2^{b_2} \times \dots \times p_k^{b_k}$ 일 때
 $\gcd(a, b) = p_1^{\min(a_1, b_1)} \times p_2^{\min(a_2, b_2)} \times \dots \times p_k^{\min(a_k, b_k)}$ 로
표현함

- e.g., $72 = 2^3 \times 3^2, 108 = 2^2 \times 3^3 \rightarrow \gcd(72, 108) = 2^{\min(3, 2)} \times 3^{\min(2, 3)}$
 $= 2^2 \times 3^2 = 36$

• 해당 정리를 이용하여 최소공배수를 구할 수 있음

- $a = p_1^{a_1} \times p_2^{a_2} \times \dots \times p_k^{a_k}, b = p_1^{b_1} \times p_2^{b_2} \times \dots \times p_k^{b_k}$ 일 때
 $\text{lcm}(a, b) = p_1^{\max(a_1, b_1)} \times p_2^{\max(a_2, b_2)} \times \dots \times p_k^{\max(a_k, b_k)}$ 로
표현함

- e.g., $72 = 2^3 \times 3^2, 108 = 2^2 \times 3^3 \rightarrow \text{lcm}(72, 108) = 2^{\max(3, 2)} \times 3^{\max(2, 3)}$
 $= 2^3 \times 3^3 = 216$

소인수분해

- 산술 기본 정리(Fundamental Theorem of Arithmetic)
(3/3)

- 최대공약수와 최소공배수는 다음과 같은 관계를 가짐

- $\gcd(a, b) \times \text{lcm}(a, b) = a \times b$
 $\because \gcd(a, b) \times \text{lcm}(a, b)$
 $= \left(p_1^{\min(a_1, b_1)} \times p_2^{\min(a_2, b_2)} \times \dots \times p_k^{\min(a_k, b_k)} \right)$
 $\quad \times \left(p_1^{\max(a_1, b_1)} \times p_2^{\max(a_2, b_2)} \times \dots \times p_k^{\max(a_k, b_k)} \right)$
 $= p_1^{a_1+b_1} \times p_2^{a_2+b_2} \times \dots \times p_k^{a_k+b_k} = a \times b$

소인수분해

- 방법 (1/18)

제수(Divisor)

나눗셈에서 다른 수를 나누려는 수

피제수(Dividend)

나눗셈에서 다른 수에게 나뉘지는 수

- 전수 나눔 소인수분해(Trial Division Factorization) (1/3)

- 정의

- 피제수 n 에 제수 a 를 하나씩 늘려가며 소인수분해 하는 방법

- 특징

- 소인수분해가 끝날 때까지 인수분해를 반복함
 - $O(2^{n_b/2})$ 의 복잡도를 가짐(n_b 는 n 의 비트 수)
 - 지수적인(Exponential) 복잡도를 가짐
 - $n > 2^{10}$ 인 경우 비효율적임

소인수분해

- 방법 (2/18)

- 전수 나눴을 소인수분해(Trial Division Factorization) (2/3)

- 원리

- n 이 합성수라면 $\sqrt{n}$ 이하의 소수 p 가 하나 이상 존재함

- 과정

1. n 이 a 로 나누어 떨어지지 않을 때까지 나눴
2. 더 이상 a 로 나누어 떨어지지 않으면 n 이 a 로 나누어 떨어질 때까지 a 를 늘려가며 $a \leq \sqrt{n}$ 일 동안 반복함
3. 마지막까지 남은 n 이 1보다 크면 그 n 또한 소수이므로 소인수에 포함함

```
Trial_Division_Factorization ( $n$ )  
{  
     $a \leftarrow 2$   
    while ( $a \leq \sqrt{n}$ )  
    {  
        while ( $n \bmod a = 0$ )  
        {  
            output  $a$   
             $n = n/a$   
        }  
         $a \leftarrow a + 1$   
    }  
    if ( $n > 1$ ) output  $n$   
}
```

소인수분해

- 방법 (3/18)

- 전수 나눴을 소인수분해(Trial Division Factorization) (3/3)
 - 예제 9.1

전수 나눴을 소인수분해 알고리즘을 이용하여 1233과 1523357784를 소인수분해 하라

- 결과

- $n = 1233 = 3^2 \times 137^1$

```
n : 1233
3 is prime factor
3 is prime factor
137 is prime factor
```

- $n = 1523357784 = 2^3 \times 3^2 \times 13^1 \times 37^1 \times 43987^1$

```
n : 1523357784
2 is prime factor
2 is prime factor
2 is prime factor
3 is prime factor
3 is prime factor
13 is prime factor
37 is prime factor
43987 is prime factor
```

```
def Trial_Division(n):
    a = 2 ## 제수
    while(a*a <= n):
        while (n%a == 0):
            print(f'{a} is prime factor')
            n //= a
        a += 1
    if(n>1):
        print(f'{n} is prime factor')
    Trial_Division(int(input('n : ')))
```

소인수분해

- 방법 (4/18)
 - 페르마 소인수분해(Fermat Factorization) (1/3)
 - 정의
 - 피제수를 두 제곱수의 차 형태로 나타내어 소인수분해 하는 방법
 - 특징
 - 구해진 두 수가 항상 소인수인 것은 아님
 - 소인수가 아니라면 반복 적용해야 함
 - $O(2^{n_b/2})$ 의 복잡도를 가짐
 - 지수적인(Exponential) 복잡도를 가짐
 - 두 소인수 간의 차가 커질수록 비효율적임

소인수분해

- 방법 (5/18)

- 페르마 소인수분해(Fermat Factorization) (2/3)

- 원리

- $n = x^2 - y^2$ 인 x 와 y 가 존재한다면 $n = (x + y)(x - y)$ 로 나타낼 수 있음

- 과정

1. 초기값 x 를 $\sqrt{n}$ 보다 큰 수 중 가장 작은 정수로 초기화함
2. $x^2 - n$ 을 통해 $n = x^2 - y^2$ 을 만족하는 y^2 을 찾음
3. 만족하는 x 와 y 를 찾았다면 n 의 인수는 $x + y$ 와 $x - y$ 임
4. 찾지 못했다면 x 를 하나씩 늘려가며 반복함

```
Fermat_Factorization (n)
{
    x ← ⌊√n⌋
    while (x < n)
    {
        w ← x2 - n
        if (w is perfect square)
        {
            y ← √w
            a ← x + y
            b ← x - y
            return a and b
        }
        x ← x + 1
    }
}
```

소인수분해

- 방법 (6/18)

- 페르마 소인수분해(Fermat Factorization) (3/3)
 - 예제 9.2

페르마 소인수분해 알고리즘을 이용하여 2431를 소인수분해 하라

- 결과

- $n = 2431 = 143 \times 17$

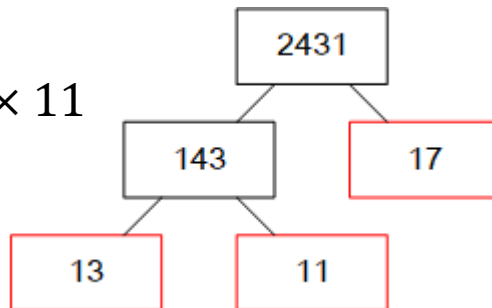
$n : 2431$
 $n = 143 \times 17$

- 143은 소수가 아니므로 143에 대해 한 번 더 페르마 소인수분해 알고리즘 적용

- 추가 실행 결과

- $n = 143 = 13 \times 11$

$n : 143$
 $n = 13 \times 11$



```
import math

def Fermat(n):
    x = math.isqrt(n)
    if(x*x < n):
        x += 1 ## root(n)보다 큰 수 중 가장 작은 정수
    while(x < n):
        w = x*x-n
        if(is_perfect_square(w)):
            y = math.isqrt(w)
            a = x+y
            b = x-y
            return int(a), int(b)
        x += 1

def is_perfect_square(w):
    s = math.isqrt(w)
    return s*s == w

result = Fermat(int(input('n : ')))
print(f'n = {result[0]}*{result[1]}')
```

소인수분해

- 방법 (7/18)

$B - smooth$ n 의 소인수가 전부 B 이하

- Pollard $p - 1$ 소인수분해(Pollard $p - 1$ Factorization) (1/3)

- 정의

- 임의로 선택한 정수 B 로 n 의 소인수 p 를 하나 찾아내어 소인수분해하는 방법

- 특징

- 하나의 소수만을 찾아내므로 소인수분해를 위해서는 반복 적용해야 함
 - $O(e^{\sqrt{\log(p) \times \log(\log(p))}})$ 의 복잡도를 가짐
 - 부지수적인(Sub-Exponential) 복잡도를 가짐
 - 확률적 알고리즘의 성격을 띠며
 - $p - 1 \nmid B - smooth$ 하지 않으면 실패할 수 있음

소인수분해

- 방법 (8/18)

- Pollard $p - 1$ 소인수분해(Pollard $p - 1$ Factorization) (2/3)
 - 원리

- $n = kp$ 에 대해 $p - 1 | B!$ 일 경우, $B! = x(p - 1)$ 임(x 는 임의의 정수)
- 페르마 소정리(Fermat's Little Theorem)에 의해 $a^{p-1} \equiv 1 \pmod p$ 이고 양변에 x 를 제곱하면 $(a^{p-1})^x \equiv 1^x \pmod p \equiv 1 \pmod p$ 이므로 $a^{x(p-1)} \equiv a^{B!} \equiv 1 \pmod p \rightarrow a^{B!} - 1 \equiv 0 \pmod p \rightarrow p | a^{B!} - 1$ 임
- 이때 $n \nmid a^{B!} - 1$ 이라면 $p | n$ 이고 $p | a^{B!} - 1$ 이므로 $p = \gcd(a^{B!} - 1, n)$ 임

- 과정

1. 밑수 a 를 2로 초기화하고 임의의 정수 B 에 대해 $2^{B!}$ 를 계산함
2. 계산한 $2^{B!}$ 를 이용하여 $\gcd(2^{B!} - 1, n)$ 을 계산함
3. $1 < \gcd(2^{B!} - 1, n) < n$ 이 아닐 경우, B 또는 밑수 a 를 새로 정의하여 다시 시도함

```
Pollard_(p - 1)_Factorization (n, B)
{
    a ← 2
    e ← 2
    while (e ≤ B)
    {
        a ← ae mod n
        e ← e + 1
    }
    p ← gcd(a - 1, n)
    if (1 < p < n) return p
    return failure
}
```

소인수분해

- 방법 (9/18)

- Pollard $p - 1$ 소인수분해(Pollard $p - 1$ Factorization) (3/3)
 - 예제 9.3

Pollard $p - 1$ 소인수분해 알고리즘을 이용하여 $B = 8$ 일 때의 57247189의 소인수를 구하라

- 결과

- $n = 57247189, B = 8$
 $\rightarrow p = 421$

```
n : 57247159
B : 8
Prime factor is 421
```

```
def Pollard_p1(n, B):
    a = 2
    e = 2
    while(e <= B):
        a = pow(a, e, n)
        e += 1
    p = gcd(a-1, n) ## 계산에서 사용되는 a는 a^B!
    if(1 < p < n):
        return p
    return 1

def gcd(a, b): ## 유클리드 알고리즘으로 최대공약수 계산
    while(b != 0):
        a, b = b, a % b
    return a

n = int(input('n : '))
B = int(input('B : '))
print(f'Prime factor is {Pollard_p1(n,B)}')
```

소인수분해

- 방법 (10/18)

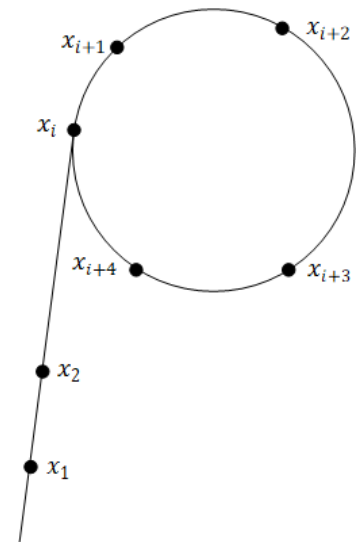
- Pollard Rho 소인수분해(Pollard Rho Factorization) (1/3)

- 정의

- 임의로 선택된 숫자와 반복되는 함수를 사용하여 소인수분해 하는 방법

- 특징

- 하나의 소수만을 찾아내므로 소인수분해를 위해서는 반복 적용해야 함
 - $O(2^{n_b/4})$ 의 복잡도를 가짐
 - 지수적인(Exponential) 복잡도를 가짐
 - 확률적 알고리즘의 성격을 띠며
 - $x \equiv y \pmod n$ 이지만 $\gcd(x - y, n) = n$ 이 되는 경우, 인수를 찾지 못 하고 실패할 수 있음



소인수분해

- 방법 (11/18)

- Pollard Rho 소인수분해(Pollard Rho Factorization) (2/3)

- 원리

- $n = kp$ 와 두 정수 x, y 에 대해 $p|(x - y)$ 일 경우, $p|n$ 이므로 $p = \gcd(x - y, n)$ 임
 - $n|(x - y)$ 일 경우, $\gcd(x - y, n) = n$ 이므로 p 가 의미있는 인수가 되기 위해서는 $n \nmid (x - y)$ 이어야 함

- 과정

1. 순환이 존재하도록 하는 함수 $f(x)$ 정의
 - 주로 $f(x) = x^2 + 1 \bmod n$
2. $x_{i+1} = f(x_i), y_{i+1} = f(f(y_i))$ 를 적용하여 순환을 진행시키며 $\gcd(x - y, n)$ 을 계산함
3. $\gcd(x - y, n) \neq 1$ 일 때까지 순환을 진행하고, $\gcd(x - y, n) = n$ 일 경우, 실패한 것이므로 함수나 초기값을 새로 정의하여 다시 시도함

```
Pollard_Rho_Factorization (n)
{
    x ← 2
    y ← 2
    p ← 2
    while (p = 1)
    {
        x ← f(x) mod n
        y ← f(f(y) mod n) mod n
        p ← gcd(x - y, n)
    }
    return p
}
```

소인수분해

- 방법 (12/18)

- Pollard Rho 소인수분해(Pollard Rho Factorization) (3/3)
 - 예제 9.4

Pollard Rho 소인수분해 알고리즘을 이용하여 434617의 소인수를 구하라

- 결과

- $n = 434617, f(x) = x^2 + 1 \bmod n$
 $\rightarrow p = 709$

n : 434617
Prime factor is 709

단계	x	y	p	단계	x	y	p
0	2	2	1	8	157099	427553	1
1	5	26	1	9	369457	2634	1
2	26	23713	1	10	52128	63593	1
3	677	142292	1	11	102901	161353	1
4	23713	157099	1	12	41831	64890	1
5	346589	52128	1	13	64520	21979	1
6	142292	41831	1	14	68775	16309	709
7	380320	68775	1				

```
def Pollard_Rho(n):  
    x = 2  
    y = 2  
    p = 1  
    while(p == 1):  
        x = f(x,n)  
        y = f(f(y,n),n)  
        p = gcd(x-y,n)  
    return p  
  
def f(x, n): ## 다항식 f(x)  
    return (x*x+1)%n  
  
def gcd(a, b): ## 유클리드 알고리즘으로 최대공약수 계산  
    while(b != 0):  
        a, b = b, a % b  
    return a  
  
print('Prime factor is',Pollard_Rho(int(input("n : "))))
```

소인수분해

- 방법 (13/18)

선형 종속 관계
0이 아닌 벡터들의 합이 0인 관계

- 2차 체(Quadratic Sieve) (1/4)

- 정의

- 선형 종속 관계를 통해 $x^2 \equiv y^2 \pmod{n}$ 을 만족하는 x 와 y 를 구하여 n 을 소인수분해 하는 방법

- 특징

- 구해진 수가 항상 소인수인 것은 아님
 - 소인수가 아니라면 반복 적용해야 함
 - $O(e^{\sqrt{n_b \times \log(n_b)}})$ 의 복잡도를 가짐
 - 부지수적인(Sub-Exponential) 복잡도를 가짐
 - 현재 알려진 소인수분해 알고리즘 중 100 자리 이하(대략 2^{330})에서 가장 효율적인 소인수분해 알고리즘임

소인수분해

- 방법 (14/18)

완전 제곱수 제곱근이 정수인 수

- 2차 체(Quadratic Sieve) (2/4)

- 원리

- $x^2 \equiv y^2 \pmod n$ 을 만족하는 x 와 y 가 존재할 경우, $\gcd(x - y, n)$ 또는 $\gcd(x + y, n)$ 을 이용하여 n 의 인수를 구할 수 있음
 - $x^2 \equiv y^2 \pmod n = x^2 - y^2 \equiv 0 \pmod n = (x - y)(x + y) \equiv 0 \pmod n$ 이므로 $n \mid (x - y)(x + y)$ 이고 $p \mid n$ 이기 때문에 $\gcd(x - y, n)$ 또는 $\gcd(x + y, n)$ 이 n 의 소인수일 수 있음
 - 특정 수의 모든 소인수의 지수가 짝수일 경우, 그 수는 완전 제곱수임
 - e.g., $n = p^2 \times q^4 \rightarrow \sqrt{n} = \sqrt{p^2 \times q^4} = p \times q^2$

소인수분해

- 방법 (15/18)

2차 잉여(Quadratic Residue)

$a^{(p-1)/2} \equiv 1 \pmod{p}$ 를 만족시키는 a

- 2차 체(Quadratic Sieve) (3/4)

- 과정

1. 소수 중 n 을 2차 잉여로 만드는 B 이하의 소수들을 구함
 - Factor Base라고 부르며 특수한 경우를 처리하기 위해 -1 과 2 를 포함함
 - e.g., $n = 1081, B = 30$ 일 때, Factor Base = $\{-1, 2, 3, 5, 7, 11\}$
2. 다항식 $Q(x_i) = x^2 - n$ 이 B -smooth 하게 되는 x_i 들을 구함
 - e.g., $n = 1219, B = 30$ 일 때, $x_1 = 35$ ($\because 35^2 - 1219 = 2^1 \times 3^1$), $x_2 = 36$ ($\because 36^2 - 1219 = 7^1 \times 11^1$) 등
3. 다항식 $Q(x_i)$ 를 소인수분해한 결과를 지수 벡터로 변환하고 $\pmod{2}$ 연산한 후 해당 지수 벡터들 중 더해서 0이 되는 $Q(x_i)$ 들을 구함
 - e.g., $Q(35) = 6 = 2^1 \times 3^1 = (0, 1, 1, 0, 0, 0)$, $Q(37) = 150 = 2^1 \times 3^1 \times 5^2$
 $= (0, 1, 1, 2, 0, 0) = (0, 1, 1, 0, 0, 0) + (0, 0, 0, 2, 0, 0)$
 $Q(33) + Q(37) = (0, 1, 1, 0, 0, 0) + (0, 1, 1, 0, 0, 0)$
 $= (0, 2, 2, 0, 0, 0) = (0, 0, 0, 0, 0, 0)$ 등

소인수분해

- 방법 (16/18)

- 2차 체(Quadratic Sieve) (4/4)

- 과정

4. 구한 x_i 와 $Q(x_i)$ 를 이용하여 $x \equiv \prod x_i \pmod n, y = \sqrt{\prod Q(x_i)}$ 를 구함

- e.g., $x \equiv 35 \times 37 \pmod{1219} \equiv 76 \pmod{1219}, y \equiv \sqrt{6 \times 150} \pmod{1219} \equiv 30 \pmod{1219}$

5. 구한 x, y 가 $x \not\equiv \pm y \pmod n$ 인 경우, $\gcd(x - y, n)$ 또는 $\gcd(x + y, n)$ 는 n 의 인수임

- e.g., $\gcd(76 - 30, 1219) = 23, \gcd(76 + 30, 1219) = 53 \rightarrow n = 23 \times 53$

소인수분해

- 방법 (17/18)

- 정수체 체(Number Field Sieve) (1/2)

- 정의

- 다항식 $f(x)$ 를 통해 생성된 체(Field)를 이용하여 $x^2 \equiv y^2 \pmod{n}$ 을 얻어낸 후 $\gcd(x - y, n)$ 또는 $\gcd(x + y, n)$ 을 계산하여 소인수분해하는 방법

- 특징

- 구해진 수가 항상 소인수인 것은 아님
 - 소인수가 아니라면 반복 적용해야 함
 - $O(e^{n_b^{1/3} \times \log^{2/3}(n_b)})$ 의 복잡도를 가짐
 - 부지수적인(Sub-Exponential) 복잡도를 가짐
 - 현재 알려진 소인수분해 알고리즘 중 100자리 이상(대략 2^{330})에서 가장 효율적인 알고리즘임

소인수분해

- 방법 (18/18)

- 정수체 체(Number Field Sieve) (2/2)

- 원리

- $x^2 \equiv y^2 \pmod n$ 을 만족하는 x 와 y 가 존재할 경우, $\gcd(x - y, n)$ 또는 $\gcd(x + y, n)$ 를 이용하여 n 의 인수를 구할 수 있음

- 과정

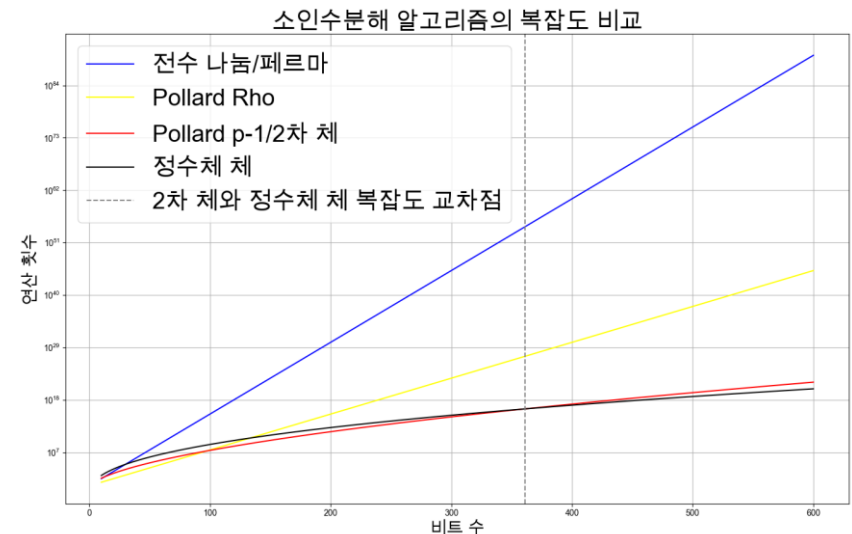
1. 다항식 $f(x)$ 와 $f(m) \equiv 0 \pmod n$ 을 만족하는 정수 m 을 선택함
 - 주로 $m = \lfloor n^{1/d} \rfloor$ (d 는 $f(x)$ 의 차수로, 주로 4~6을 사용함)을 먼저 선택하고 $f(m) \equiv 0 \pmod n$ 을 만족하는 다항식 $f(m)$ 을 선택함
2. $R = a - b \times m, A = b^d \times f(\frac{a}{b})$ 에 대해 R, A 가 모두 B -smooth한 (a, b) 를 여러 개 구함
3. R 과 A 를 소인수분해 하여 지수 벡터를 구하고 하나의 행렬 M 으로 만들어서 선형 종속 관계를 만족하는 R_i 와 A_i 를 여러 개 구함
4. 구한 R_i 와 A_i 로 $x^2 = \prod R_i, y^2 = \prod A_i$ 를 설정하고 $x^2 \equiv y^2 \pmod n$ 을 이용하여 $\gcd(x - y, n)$ 과 $\gcd(x + y, n)$ 으로 n 의 인수를 구함

소인수분해

• 복잡도 (1/3)

- Pollard $p - 1$ 알고리즘과 2차 체 알고리즘의 복잡도는 거의 유사하지만 Pollard $p - 1$ 알고리즘은 특정 조건 하에서만 성공하므로 실제로 가장 많이 쓰이는 알고리즘은 2차 체 알고리즘과 정수체 체 알고리즘임
- 현재는 100자리 이하의 소인수분해는 2차 체 알고리즘을 주로 사용하고 100자리 이상의 소인수분해는 정수체 체 알고리즘을 주로 사용함

방법	복잡도
전수 나눗(Trial Division)	$O(2^{n_b/2})$
페르마(Fermat)	$O(2^{n_b/2})$
Pollard $p - 1$	$O(e^{\sqrt{\log(p) \times \log(\log(p))}})$
Pollard Rho	$O(2^{n_b/4})$
2차 체(Quadratic Sieve)	$O(e^{\sqrt{n_b} \times \log(n_b)})$
정수체 체(Number Field Sieve)	$O(e^{n_b^{1/3} \times \log^{2/3}(n_b)})$

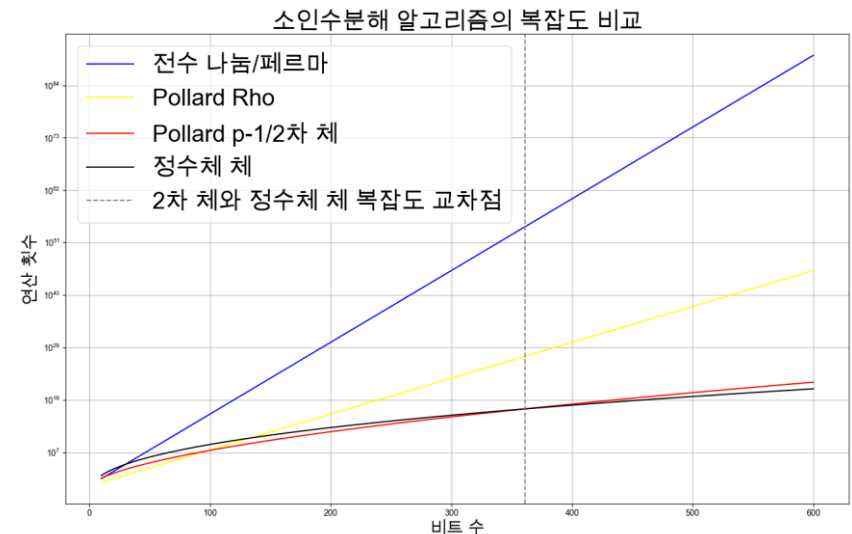


소인수분해

• 복잡도 (2/3)

- 현재까지 알려진 일반적인 소인수분해 알고리즘의 비트-연산 복잡도는 대부분 지수적이거나 부지수적인 형태를 가짐
- 소인수분해 하려는 수가 커질수록 결과를 얻기 위해 걸리는 시간이 급격히 증가함
 - e.g., 초당 2^{30} 번의 연산을 하는 컴퓨터로 1024 비트의 수를 Pollard rho, 2차 체, 정수체 체 알고리즘으로 소인수분해 시도 시 대략 4.1×10^{60} , 6.3×10^{19} , 4.4×10^{19} 년의 시간이 걸림

방법	복잡도
전수 나눗(Trial Division)	$O(2^{n_b/2})$
페르마(Fermat)	$O(2^{n_b/2})$
Pollard $p - 1$	$O(e^{\sqrt{\log(p) \times \log(\log(p))}})$
Pollard Rho	$O(2^{n_b/4})$
2차 체(Quadratic Sieve)	$O(e^{\sqrt{n_b \times \log(n_b)}})$
정수체 체(Number Field Sieve)	$O(e^{n_b^{1/3} \times \log^{2/3}(n_b)})$

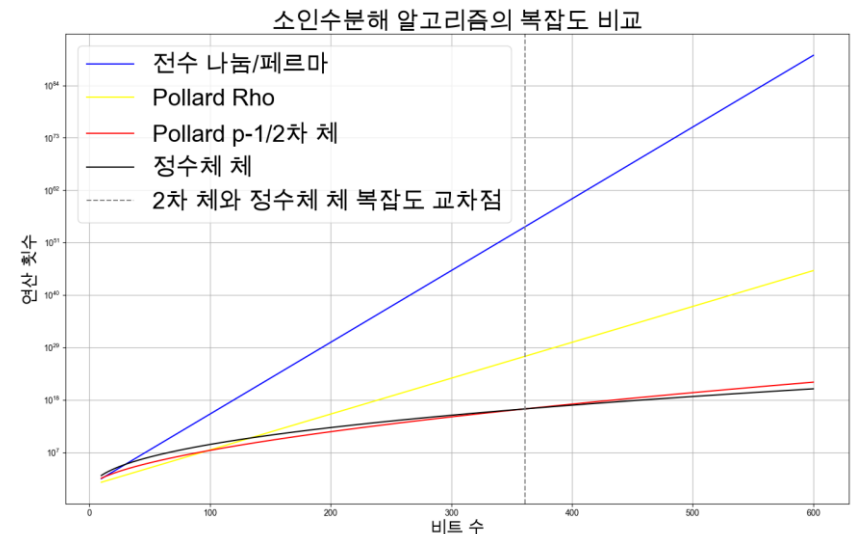


소인수분해

• 복잡도 (3/3)

- RSA 등의 암호는 소인수분해가 어려운 합성수를 키의 일부로 사용하며 해당 키의 보안성은 소인수분해 알고리즘의 복잡도에 의존함
- 쇼어 알고리즘(Shor's Algorithm)은 $O(\log(n)^3)$ 의 다항 시간 복잡도로 소인수분해를 빠르게 할 수 있으므로 RSA 암호의 보안성을 위협할 수 있음
 - 쇼어 알고리즘은 양자 컴퓨터에서 사용됨

방법	복잡도
전수 나눗(Trial Division)	$O(2^{n_b/2})$
페르마(Fermat)	$O(2^{n_b/2})$
Pollard $p-1$	$O(e^{\sqrt{\log(p) \times \log(\log(p))}})$
Pollard Rho	$O(2^{n_b/4})$
2차 체(Quadratic Sieve)	$O(e^{\sqrt{n_b} \times \log(n_b)})$
정수체 체(Number Field Sieve)	$O(e^{n_b^{1/3} \times \log^{2/3}(n_b)})$



목 차

- 소인수분해(Factorization)
- 중국인의 나머지 정리(Chinese Remainder Theorem)
- 2차 합동(Quadratic Congruence)

중국인의 나머지 정리

• 정리

- 서로소인 모듈로를 갖는 연립 합동 방정식은 $0 \leq x < M$ 의 범위에서 유일한 해를 가짐

- 연립 합동 방정식은
$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \vdots \\ x \equiv a_n \pmod{m_n} \end{cases} \text{로 표현됨}$$

- a_n, M 등을 이용하여 연립 합동 방정식의 해 x 를 구함

- $x \equiv \sum_{i=1}^n a_n \times M_n \times M_n^{-1} \pmod{M}$

표기	설명
a_n	임의의 정수
m_n	서로소인 모듈로
M	모든 m_n 의 곱 $m_1 \times m_2 \times \cdots \times m_n$
M_n	m_n 을 제외한 나머지 모듈로의 곱 M/m_n
M_n^{-1}	M_n 의 모듈로 m_n 에서의 곱셈 역원

중국인의 나머지 정리

• 예제 9.5

다음 합동방정식의 해를 구하라

$$x \equiv 2 \pmod{3}$$

$$x \equiv 3 \pmod{5}$$

$$x \equiv 2 \pmod{7}$$

$$M = m_1 \times m_2 \times m_3 = 3 \times 5 \times 7 = 105$$

$$M_1 = M/m_1 = 105/3 = 35$$

$$M_2 = M/m_2 = 105/5 = 21$$

$$M_3 = M/m_3 = 105/7 = 15$$

$$M_1 \times M_1^{-1} \equiv 1 \pmod{3} = 35 \times M_1^{-1} \equiv 1 \pmod{3} \rightarrow M_1^{-1} = 2$$

$$M_2 \times M_2^{-1} \equiv 1 \pmod{5} = 21 \times M_2^{-1} \equiv 1 \pmod{5} \rightarrow M_2^{-1} = 1$$

$$M_3 \times M_3^{-1} \equiv 1 \pmod{7} = 15 \times M_3^{-1} \equiv 1 \pmod{7} \rightarrow M_3^{-1} = 1$$

$$\begin{aligned} x &\equiv (a_1 \times M_1 \times M_1^{-1}) + (a_2 \times M_2 \times M_2^{-1}) + (a_3 \times M_3 \times M_3^{-1}) \pmod{M} \\ &\equiv (2 \times 35 \times 2) + (3 \times 21 \times 1) + (2 \times 15 \times 1) \pmod{105} \equiv 23 \pmod{105} \end{aligned}$$

$$\therefore x \equiv 23 \pmod{105}$$

중국인의 나머지 정리

- 응용 (1/2)

- 큰 정수를 작은 정수로 나누어 빠르게 계산할 수 있음
 - e.g., 큰 수 x 를 x 보다 작은 여러 개의 $\text{mod } m_k$ 의 연립 합동 방정식으로 나누어 계산할 수 있음
- $x^2 \equiv a \text{ mod } n$ 꼴의 2차 합동 문제를 해결하는 데 사용함
 - e.g., $n = p \times q$ 를 소인수분해 하여 모듈로 p 와 모듈로 q 에 대한 해를 구하고, 이를 연립하여 모듈로 n 에서의 해를 구할 수 있음

중국인의 나머지 정리

- 응용 (2/2)

- 예제 9.6

사용하는 시스템의 능력이 100보다 작은 수만 입력받을 수 있다고 가정하고 $x = 123$, $y = 334$ 일 때, $z = x \times y$ 를 중국인의 나머지 정리를 이용하여 구하라

- 임의의 m_1, m_2, m_3 인 99, 98, 97을 선택하여 x, y, z 를 표현함

$x \equiv 24 \pmod{99}$	$y \equiv 37 \pmod{99}$	$x \times y \equiv 96 \pmod{99} \rightarrow z \equiv \mathbf{96 \pmod{99}}$
$x \equiv 25 \pmod{98}$	$y \equiv 40 \pmod{98}$	$x \times y \equiv 20 \pmod{98} \rightarrow z \equiv \mathbf{20 \pmod{98}}$
$x \equiv 26 \pmod{97}$	$y \equiv 43 \pmod{97}$	$x \times y \equiv 51 \pmod{97} \rightarrow z \equiv \mathbf{51 \pmod{97}}$

- 중국인의 나머지 정리로 계산

$$M = 99 \times 98 \times 97 = 941094$$

$$M_1 = 941094/99 = 9506$$

$$M_2 = 941094/98 = 9603$$

$$M_3 = 941094/97 = 9702$$

$$M_1 \times M_1^{-1} \equiv 1 \pmod{3} = 9506 \times M_1^{-1} \equiv 1 \pmod{3} \rightarrow M_1^{-1} = 50$$

$$M_2 \times M_2^{-1} \equiv 1 \pmod{5} = 9603 \times M_2^{-1} \equiv 1 \pmod{5} \rightarrow M_2^{-1} = 97$$

$$M_3 \times M_3^{-1} \equiv 1 \pmod{7} = 9702 \times M_3^{-1} \equiv 1 \pmod{7} \rightarrow M_3^{-1} = 49$$

$$\begin{aligned} z &\equiv (96 \times 9506 \times 50) + (20 \times 9603 \times 97) + (51 \times 9702 \times 49) \pmod{941094} \equiv 88503918 \pmod{941094} \\ &\equiv 41082 \pmod{941094} \end{aligned}$$

$$\therefore z \equiv 41082 \pmod{941094}$$

목 차

- 소인수분해(Factorization)
- 중국인의 나머지 정리(Chinese Remainder Theorem)
- 2차 합동(Quadratic Congruence)

2차 합동

- 정의

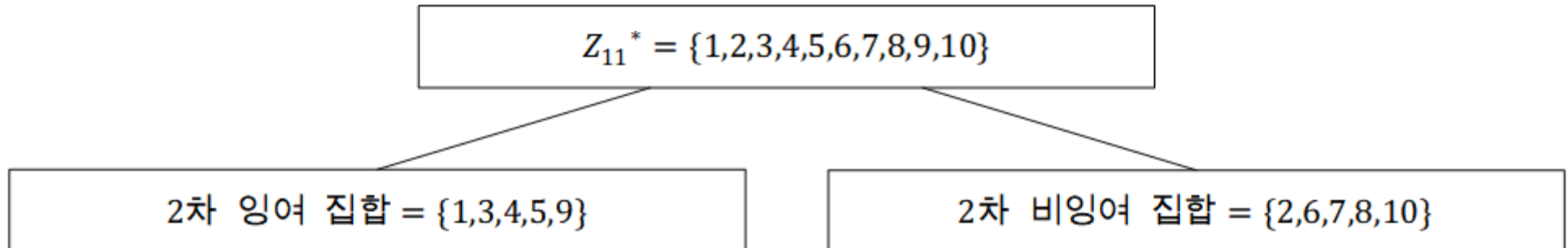
- $Ax^2 + Bx + C \equiv 0 \pmod n$ 의 형태를 갖는 합동 방정식

- 특징

- 모듈로 연산으로 인해 정수 범위 내에서만 해가 정의되므로 일반적인 2차 방정식과 다르게 해가 존재하지 않을 수도 있음
 - e.g., $x^2 \equiv 2 \pmod 3$ 의 해는 존재하지 않음
- 합동식이 $x^2 \equiv a \pmod n$ 의 형태를 가지고 n 이 소인수분해가 어려운 합성수일 경우 단방향 함수의 성격을 띠음
 - x 를 통해 $x^2 \pmod n \equiv a$ 를 구하기는 쉬우나 a 를 통해 x 를 구하는 것은 상대적으로 더 오랜 시간이 걸림

2차 합동

- 모듈로가 소수인 2차 합동 (1/6)
- 2차 잉여(QR, Quadratic Residue)
 - $x^2 \equiv a \pmod{p}$ 에서 x 가 해를 가지게 되는 a
 - e.g., $x^2 \equiv 3 \pmod{11} \rightarrow x \equiv 5 \pmod{11}, x \equiv 6 \pmod{11}$
 $\therefore 3$ 은 $\pmod{11}$ 에서 2차 잉여
- 2차 비잉여(QNR, Quadratic Nonresidue)
 - $x^2 \equiv a \pmod{p}$ 에서 x 가 해를 가지지 않는 a
 - e.g., $x^2 \equiv 2 \pmod{11} \rightarrow$ 해가 존재하지 않음
 $\therefore 2$ 는 $\pmod{11}$ 에서 2차 비잉여



2차 합동

- 모듈로가 소수인 2차 합동 (2/6)
- 오일러 판정법(Euler's Criterion)
 - a 와 p 가 서로소일 때 $a^{(p-1)/2} \bmod p$ 의 결과로 a 의 2차 잉여 여부를 판단함
 - $a^{(p-1)/2} \equiv 1 \bmod p$ 라면, a 는 2차 잉여
 - e.g., $p = 11, a = 3$ 일 때, $3^{(11-1)/2} \bmod 11 \equiv 3^5 \bmod 11 \equiv 243 \bmod 11 \equiv 1 \bmod 11 \therefore 3$ 은 $\bmod 11$ 에서 2차 잉여
 - $a^{(p-1)/2} \equiv -1 \bmod p$ 라면, a 는 2차 비잉여
 - e.g., $p = 11, a = 2$ 일 때, $2^{(11-1)/2} \bmod 11 \equiv 2^5 \bmod 11 \equiv 32 \bmod 11 \equiv 10 \bmod 11 \equiv -1 \bmod 11 \therefore 2$ 은 $\bmod 11$ 에서 2차 비잉여

2차 합동

- 모듈로가 소수인 2차 합동 (3/6)

- 방정식 풀이

- 오일러 판정법을 이용하여 해가 존재하는지 확인함

- e.g., $x^2 \equiv 3 \pmod{11}$, $3^5 \equiv 1 \pmod{11} \rightarrow 3$ 은 $\pmod{11}$ 에서 2차 잉여

- $x \equiv \pm a^{(p+1)/4} \pmod{p}$ 를 이용하여 해를 구함

- e.g., $x \equiv \pm 3^{11+1/4} \pmod{11} \equiv \pm 3^3 \pmod{11} \equiv \pm 27 \pmod{11} \equiv \pm 5 \pmod{11}$

- $p \equiv 3 \pmod{4}$ ($p = 4k + 3$) 형태가 아닌 경우 올바른 해가 나오지 않으므로 Tonelli-Shanks 알고리즘을 사용하여 해를 구함

- e.g., $x^2 \equiv 10 \pmod{13}$ 은 $x \equiv \pm 10^{7/2} \pmod{13}$ 으로 올바른 해가 나오지 않으므로 Tonelli-Shanks 알고리즘을 사용하여 해를 구해야 함

2차 합동

- 모듈로가 소수인 2차 합동 (4/6)
 - 방정식 풀이
 - Tonelli-Shanks 알고리즘 (1/3)
 - 정의
 - 3 이상의 소수 p 에 대해 2차 합동 방정식 $x^2 \equiv a \pmod{p}$ 의 해 x 를 구하는 알고리즘
 - 특징
 - $p = 4k + 1$ 과 $p = 4k + 3$ 형태 모두 사용 가능함
 - a 는 p 에 대해 2차 잉여이어야 함
 - $O(\log^2(p))$ 의 복잡도를 가짐

2차 합동

• 모듈로가 소수인 2차 합동 (5/6)

• 방정식 풀이

• Tonelli-Shanks 알고리즘 (2/3)

• 과정

1. 양의 정수 중 소수 p 에 대해 $p - 1 = Q \times 2^S$ 를 만족하는 Q 와 S 를 구함(Q 는 홀수)
2. 모듈로 p 에 대해 2차 비잉여인 임의의 z 를 구함
3. 변수를 $M = S, c = z^Q, t = a^Q, R = a^{Q+1/2}$ 로 초기화함
4. 변수 t 가 1이 될 때까지 $b = c^{2^{M-i-1}}, M = i, c = b^2, t = tb^2, R = Rb$ 를 반복

```
def E_Criterion(a, p): ## 오일러 판정법
    return pow(a, (p - 1) // 2, p)

def Tonelli-Shanks(a, p):
    if E_Criterion(a, p) != 1:
        return -1
    if a == 0:
        return 0
    if p == 2:
        return a

    # 1단계 : p - 1 = Q * 2^S 꼴로 분해
    Q = p - 1
    S = 0
    while Q % 2 == 0:
        Q //= 2
        S += 1

    # 2단계 : 2차 비잉여 z 찾기
    for z in range(2, p):
        if E_Criterion(z, p) == p - 1:
            break

    # 3단계 : 초기값 설정
    M = S
    c = pow(z, Q, p)
    t = pow(a, Q, p)
    R = pow(a, (Q + 1) // 2, p)

    # 4단계 : 반복
    while t != 1:
        i = 1
        t2i = pow(t, 2, p)
        while t2i != 1: # t^(2^i) == 1인 i의 최소값 계산
            t2i = pow(t2i, 2, p)
            i += 1
        b = pow(c, 2 ** (M - i - 1), p)
        M = i
        c = pow(b, 2, p)
        t = (t * pow(b, 2, p)) % p
        R = (R * b) % p

    return R

a = int(input('a : '))
p = int(input('p : '))
x = Tonelli-Shanks(a, p)

if(x == -1):
    print('a is not QR')
elif(pow(x, 1, p) == pow(x+1, 1, p)):
    print(f'x = {x}')
else:
    print(f"x ≡ {x} mod {p}")
    print(f"x ≡ {-x % p} mod {p}")
```

2차 합동

- 모듈로가 소수인 2차 합동 (6/6)
 - 방정식 풀이
 - Tonelli-Shanks 알고리즘 (3/3)
 - 예제 9.7

Tonelli-Shanks 알고리즘을 이용하여 다음 2차 합동방정식들의 해를 구하라

1. $x^2 \equiv 2 \pmod{11}$
2. $x^2 \equiv 3 \pmod{23}$
3. $x^2 \equiv 10 \pmod{13}$

- 결과

1. $x^2 \equiv 2 \pmod{11} \rightarrow$ 해가 존재하지 않음

```
a : 2
p : 11
a is not QR
```

2. $x^2 \equiv 3 \pmod{23} \rightarrow x \equiv \pm 16 \pmod{23}$

```
a : 3
p : 23
x ≡ 16 mod 23
x ≡ 7 mod 23
```

3. $x^2 \equiv 10 \pmod{13} \rightarrow x \equiv \pm 7 \pmod{13}$

```
a : 10
p : 13
x ≡ 7 mod 13
x ≡ 6 mod 13
```

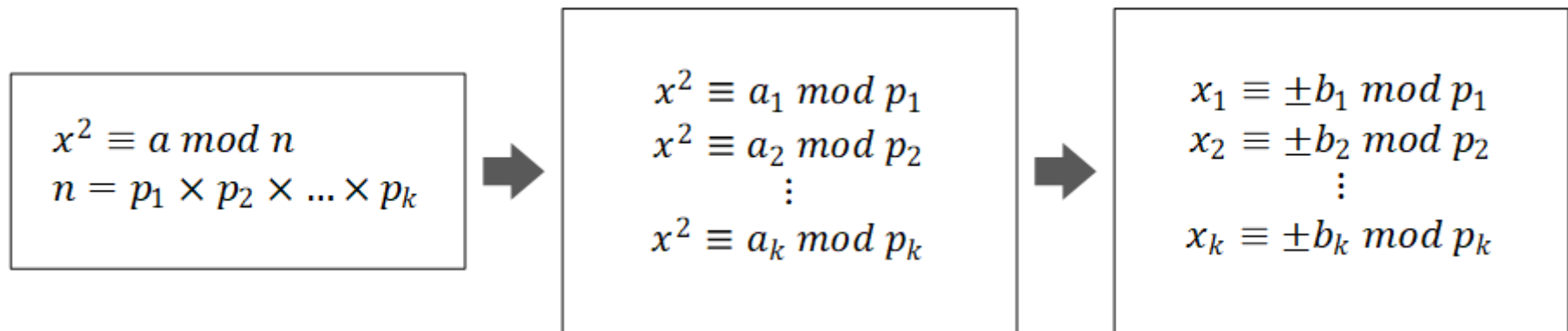
2차 합동

- 모듈로가 합성수인 2차 합동

- 방정식 풀이 (1/2)

- $x^2 \equiv a \pmod n$ 에서 n 을 소인수분해하여 나온 p_k 를 이용하여 여러 개의 $x^2 \equiv a_k \pmod{p_k}$ 로 분해한 후 연립 합동 방정식으로 만들어 해를 구함

- e.g., $x^2 \equiv 36 \pmod{77}$ 에서 $n = 77 = 7 \times 11$ 이므로 $x^2 \equiv 36 \pmod{7} \equiv 1 \pmod{7}$, $x^2 \equiv 36 \pmod{11} \equiv 3 \pmod{11}$ 로 나누어 $\pmod{7}$ 과 $\pmod{11}$ 에서의 해를 구한 후 중국인의 나머지 정리를 이용하여 $\pmod{77}$ 에서의 해를 구함



2차 합동

- 모듈로가 합성수인 2차 합동

- 방정식 풀이 (2/2)

- 예제 9.8

$77 = 7 \times 11$ 임을 이용하여 $x^2 \equiv 36 \pmod{77}$ 의 해를 구하라

- $x^2 \equiv 36 \pmod{77}$ 을 $x^2 \equiv 36 \pmod{7} \equiv 1 \pmod{7}$ 과 $x^2 \equiv 36 \pmod{11} \equiv 3 \pmod{11}$ 로 나눔

- $1^2 \equiv 1 \pmod{7}$ 이고 $3^2 \equiv 9 \equiv 2 \pmod{11}$ 이므로 해가 존재함

- $x \equiv \pm 1 \pmod{7}$ 와 $x \equiv \pm 5 \pmod{11}$ 를 이용하여 연립 합동 방정식을 만들

$$1. \begin{cases} x \equiv 1 \pmod{7} \\ x \equiv 5 \pmod{11} \end{cases} \quad 2. \begin{cases} x \equiv 1 \pmod{7} \\ x \equiv 6 \pmod{11} \end{cases} \quad 3. \begin{cases} x \equiv 6 \pmod{7} \\ x \equiv 5 \pmod{11} \end{cases} \quad 4. \begin{cases} x \equiv 6 \pmod{7} \\ x \equiv 6 \pmod{11} \end{cases}$$

- 각 연립 합동 방정식을 중국인의 나머지 정리를 이용하여 해를 구함

$$1. x \equiv 71 \pmod{77} \quad 2. x \equiv 50 \pmod{77} \quad 3. x \equiv 27 \pmod{77} \quad 4. x \equiv 6 \pmod{77}$$

$$\therefore x^2 \equiv 36 \pmod{77} \text{의 해 } x = \pm 6 \pmod{77}, \pm 27 \pmod{77}$$

Thanks!

이 성 현(dltjdgus2166@naver.com)

부록#1

- 2차 체(Quadratic Sieve) 실제 동작(1/2)
- 파이썬 코드 전문

```
import math

# 소수 판별 함수
def is_prime(p):
    if p < 2:
        return False
    for i in range(2, math.isqrt(p) + 1):
        if p % i == 0:
            return False
    return True

# 팩터 베이스
def prime_range(a, b, n):
    res = []
    res.append(2)
    for i in range(a, b):
        if (is_prime(i)) and ((n+((i-1)/2))%i==1):
            res.append(i)
    return res

# 최대공약수
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

# B-smooth 확인
def is_B_smooth(n, factor_base):
    exponents = []

    neg = 0
    if n < 0:
        n = -n
        neg = 1 # -1에 대한 지수 저장
    exponents.append(neg)

    for p in factor_base:
        count = 0
        while n % p == 0:
            n //= p
            count += 1
        exponents.append(count)
    return (n == 1), exponents if n == 1 else None

# 선형 종속 관계 찾기
def Linear_Dependent(EV):
    n = len(EV)
    m = len(EV[0])
    mat = [row[:] for row in EV]
    depts = [[int(i == j) for j in range(n)] for i in range(n)]
    used = [False] * n
    result = []

    for col in range(m):
        pivot = -1
        for row in range(n):
            if mat[row][col] == 1 and not used[row]:
                pivot = row
                break
        if pivot == -1:
            continue
        used[pivot] = True
        for row in range(n):
            if row != pivot and mat[row][col] == 1:
                for k in range(m):
                    mat[row][k] ^= mat[pivot][k]
                for k in range(n):
                    depts[row][k] ^= depts[pivot][k]

    for i in range(n):
        if all(x == 0 for x in mat[i]):
            result.append(depts[i]) # 선형 종속된 해 저장

    return result
```

```
# Quadratic Sieve
def quadratic_sieve(n, B, max_relations, max_attempts):
    sqrt_n = math.isqrt(n)

    factor_base = prime_range(2, B, n)

    print('Factor Base :', factor_base)

    smooth_relations = [] # 지수 벡터
    x_list = [] # x 값 저장
    attempts = 0
    x = sqrt_n + 1

    while len(smooth_relations) < max_relations and attempts < max_attempts:
        qx = x * x - n # Q(x) 계산
        Bsmooth, expo = is_B_smooth(qx, factor_base) # B-smooth 확인
        if Bsmooth:
            smooth_relations.append(expo)
            x_list.append(x)
            x += 1
            attempts += 1

    # 루프 방지
    if len(smooth_relations) < max_relations:
        print('B-smooth Failure')
        return None

    print('Exponent Vector :')
    for i in range(len(smooth_relations)):
        print(smooth_relations[i])
    print('x_list :')
    for i in range(len(x_list)):
        print(x_list[i])

    # 지수 벡터 mod 2 연산
    Ex_Vect_Mod2 = [[e % 2 for e in row] for row in smooth_relations]

    print('Exponent Vector Mod 2 :')
    for i in range(len(Ex_Vect_Mod2)):
        print(Ex_Vect_Mod2[i])

    # 선형 종속 벡터들로부터 x^2 = y^2 (mod n) 만들기
    LD = Linear_Dependent(Ex_Vect_Mod2)
    print('Selected x_i :')

    for dep in LD:
        X = 1
        Y = 1
        Selected_x = []
        for i in range(len(dep)):
            if dep[i]:
                X *= x_list[i] # x_i들을 곱함
                Selected_x.append(x_list[i])
        print(Selected_x)
        expts = [0] * len(smooth_relations[0])
        for i in range(len(dep)):
            if dep[i]:
                for j in range(len(expts)):
                    expts[j] += smooth_relations[i][j]
        for j in range(len(expts)):
            base = -1 if j == 0 else factor_base[j - 1]
            Y *= pow(base, expts[j] // 2) # 지수의 절반으로 Y 구성

    X %= n
    Y %= n

    print('x :', X)
    print('y :', Y)
```

```
dm = gcd(X - Y, n)

print('gcd(x-y,n) :', dm)

dp = gcd(X + Y, n)

print('gcd(x+y,n) :', dp)

if 1 < dm < n:
    return dm, n // dm

if 1 < dp < n:
    return dp, n // dp

return None

n = 1219
result = quadratic_sieve(n, 30, 20, 500000)
print(f'n = {n}의 인수: {result}')
```

부록#1

• 2차 체(Quadratic Sieve) 실제 동작(2/2)

• 실행 결과

```
Factor Base : [2, 3, 5, 7, 11]
Exponent Vector :
[0, 1, 1, 0, 0, 0]
[0, 0, 0, 0, 1, 1]
[0, 1, 1, 2, 0, 0]
[0, 0, 2, 2, 0, 0]
[0, 1, 1, 0, 1, 1]
[0, 1, 2, 1, 1, 0]
[0, 0, 1, 2, 0, 1]
[0, 0, 3, 1, 0, 1]
[0, 0, 1, 3, 1, 0]
[0, 1, 0, 3, 0, 1]
[0, 1, 4, 1, 1, 0]
[0, 1, 1, 2, 1, 1]
[0, 0, 1, 1, 1, 2]
[0, 1, 1, 0, 4, 0]
[0, 0, 1, 3, 4, 0, 0]
[0, 0, 3, 1, 3, 0]
[0, 1, 2, 1, 2, 1]
[0, 1, 1, 0, 1, 3]
[0, 1, 5, 0, 0, 2]
[0, 0, 3, 1, 2, 1]
x_i :
35
36
37
38
41
43
47
52
62
63
83
113
118
125
187
218
223
239
245
272
```

```
Exponent Vector Mod 2 :
[0, 1, 1, 0, 0, 0]
[0, 0, 0, 0, 1, 1]
[0, 1, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0]
[0, 1, 1, 0, 1, 1]
[0, 1, 0, 1, 1, 0]
[0, 1, 0, 1, 0, 1]
[0, 1, 0, 1, 0, 1]
[0, 0, 1, 1, 0, 1]
[0, 0, 1, 1, 1, 0]
[0, 1, 0, 1, 0, 1]
[0, 1, 0, 1, 1, 0]
[0, 1, 1, 0, 1, 1]
[0, 1, 1, 0, 1, 1]
[0, 0, 1, 1, 1, 0]
[0, 1, 1, 0, 0, 0]
[0, 1, 1, 0, 0, 0]
[0, 0, 1, 1, 1, 0]
[0, 1, 0, 1, 0, 1]
[0, 1, 1, 0, 1, 1]
[0, 1, 1, 0, 0, 0]
[0, 0, 1, 1, 1, 0]
Selected x_i :
[35, 37]
x : 76
y : 30
gcd(x-y,n) : 23
gcd(x+y,n) : 53
n = 1219의 인수 : (23, 53)
```

$n = 1219, B = 30$

Factor Base = {2,3,5,7,11}

$x_i = \{35, 36, 37, 38, 41, 43, 47, 52, 62, 63, 83, 113, 118, 125, 187, 218, 223, 239, 245, 272\}$

선형 종속 관계인 $x_i = \{35, 37\}$

$$\begin{aligned} \because Q(35) + Q(37) &= (2^1 \times 3^1) + (2^1 \times 3^1 \times 5^2) \\ &= (0, 1, 1, 0, 0, 0) + (0, 1, 1, 2, 0, 0) \\ &= (0, 1, 1, 0, 0, 0) + (0, 1, 1, 0, 0, 0) \\ &= (0, 2, 2, 0, 0, 0) = (0, 0, 0, 0, 0, 0) \end{aligned}$$

$$x = \prod x_i \bmod 1219 = 35 \times 37 \bmod 1219 = 76$$

$$y = \sqrt{\prod Q(x_i)} \bmod 1219 = \sqrt{6 \times 150} \bmod 1219 = 30$$

$$1 < \gcd(76 - 30, 1219) < 1219 \rightarrow 23 \text{은 소인수}$$

$$1 < \gcd(76 + 30, 1219) < 1219 \rightarrow 53 \text{은 소인수}$$