

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

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

소인수분해

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

- 최대공약수와 최소공배수는 다음과 같은 관계를 가짐

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

소인수분해

- 방법 (1/9)

*제수

나눗셈에서 다른 수를 나누려는 수

- 전수 나눗 소인수분해(Trial Division Factorization) (1/2)

- 정의

- *제수를 늘려가며 소인수분해 하는 방법

- 특징

- 2부터 $\sqrt{n}$ 까지의 범위에서만 계산함
 - $\because n$ 이 합성수라면 $\sqrt{n}$ 이하의 소수 p 가 하나 이상 존재함
 - $O(2^{n_b/2})$ 의 복잡도를 가짐(n_b 는 n 의 비트 수)
 - 지수적인(Exponential) 복잡도를 가짐
 - $n > 2^{10}$ 인 경우 비효율적임

```
Trial_Division_Factorization (n)
{
    a ← 2
    while (a ≤ √n)
    {
        while (n mod a = 0)
        {
            output a
            n = n/a
        }
        a ← a + 1
    }
    if (n > 1) output n
}
```

소인수분해

- 방법 (2/9)

- 전수 나눴을 소인수분해(Trial Division Factorization) (2/2)
 - 예제 9.1

전수 나눴을 소인수분해 알고리즘을 이용하여 1523357784를 소인수분해 하라

```
def Trial_Division(n):  
    p = [] ## 소인수 집합  
    a = 2 ## 제수  
    while(a*a <= n):  
        while (n%a == 0):  
            n //= a  
            p.append(a)  
        a += 1  
    if(n>1):  
        p.append(n)  
    return p  
  
print(Trial_Division(1523357784))
```

- 결과

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

```
n is 1523357784  
2 is prime factor  
n is 761678892  
2 is prime factor  
n is 380839446  
2 is prime factor  
n is 190419723  
3 is prime factor  
n is 63473241  
3 is prime factor  
n is 21157747  
13 is prime factor  
n is 1627519  
37 is prime factor  
n is 43987  
43987 is prime factor  
[2, 2, 2, 3, 3, 13, 37, 43987]
```

소인수분해

- 방법 (3/9)

*피제수
나눗셈에서 다른 수에게 나눠지는 수

- 페르마 소인수분해(Fermat Factorization) (1/2)

- 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

```
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 (4/9)

- 페르마 소인수분해(Fermat Factorization) (2/2)
 - 예제 9.2

페르마 소인수분해 알고리즘을 이용하여 1523357784를 소인수분해 하라

```
import math

def Fermat(n):
    if(n%2 == 0):
        return 2, n//2
    x = math.isqrt(n)
    if(x*x < n):
        x += 1
        # root(n)보다 큰 수 중 가장 작은 정수
    w = x*x-n
    # x^2 - y^2 = n일 때의 y^2
    while not (is_perfect_square(w)):
        x += 1
        w = x*x-n
    y = int(math.isqrt(w))
    a = x+y
    b = x-y

    return int(a), int(b)

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

def repeat(n):
    p, q = Fermat(n)
    print(f'({p}, {q})')
    if(Fermat(p)[1] != 1):
        repeat(p)
    else:
        print(f'{p} is prime factor')
        result.append(p)
    if(Fermat(q)[1] != 1):
        repeat(q)
    else:
        print(f'{q} is prime factor')
        result.append(q)

result = []
repeat(1523357784)
result.sort()
print(result)
```

- 결과

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

```
(2, 761678892)
2 is prime factor
(2, 380839446)
2 is prime factor
(2, 190419723)
2 is prime factor
(43987, 4329)
43987 is prime factor
(111, 39)
(37, 3)
37 is prime factor
3 is prime factor
(13, 3)
13 is prime factor
3 is prime factor
[2, 2, 2, 3, 3, 13, 37, 43987]
```

소인수분해

- 방법 (5/9)

$*B - smooth$
 n 의 소인수가 전부 B 이하

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

- 정의

- 임의로 선택한 B 로 n 의 소인수 p 를 하나 찾아내어 소인수분해 하는 방법

- 특징

- 하나의 소수만을 찾아내므로 소인수분해를 위해서는 반복 적용해야 함
 - $O(B \times \log(B) \times n_b)$ 의 복잡도를 가짐
 - 부지수적인(Sub-Exponential) 복잡도를 가짐
 - 확률적 알고리즘의 성격을 띠م
 - $p - 1 \nmid *B - smooth$ 하지 않으면 실패할 수 있음

```
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 (6/9)

- Pollard $p - 1$ 소인수분해(Pollard $p - 1$ Factorization) (2/2)
 - 예제 9.3

Pollard $p - 1$ 소인수분해 알고리즘을 이용하여 1523357784를 소인수분해 하라

```
def Pollard_p1(n, B, a):
    for e in range(2, B+1):
        a = pow(a, e, n)
        p = gcd(a-1, n)
        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

def repeat(n): ## 소인수를 구할 수 있도록 a, B를 조절
    for i in aset:
        for j in range(1,100):
            pf = Pollard_p1(n,j,i)
            if(pf>1):
                print(f'{pf} is prime factor')
                result.append(pf)
                return pf
    print(f'{n} is prime factor')
    return 1

result = []
composite = 1523357784
aset = [2,3,5,7]
rst = composite

while(rst != 1):
    print(f'composite is {composite}')
    rst = repeat(composite)
    composite //= rst
    result.append(composite)

result.sort()
print(result)
```

- 결과

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

```
n is 1523357784
3 is prime factor
n is 507785928
3 is prime factor
n is 169261976
13 is prime factor
n is 13020152
37 is prime factor
n is 351896
2 is prime factor
n is 175948
2 is prime factor
n is 87974
2 is prime factor
n is 43987
43987 is prime factor
[2, 2, 2, 3, 3, 13, 37, 43987]
```

소인수분해

- 방법 (7/9)

- Pollard Rho 소인수분해(Pollard Rho Factorization) (1/2)

- 정의

- 임의로 선택된 숫자와 반복되는 함수(주로 $f(x) = x^2 + 1$)를 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

```
Pollard_Rho_Factorization (n, B)
{
    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 (8/9)

- Pollard Rho 소인수분해(Pollard Rho Factorization) (2/2)
 - 예제 9.4

Pollard Rho 소인수분해 알고리즘을 이용하여 1523357784를 소인수분해 하라

```
def Pollard_Rho(n):
    if(n%2 == 0):
        return 2
    x = 2
    y = 2
    p = 1
    while(p == 1):
        x = f(x,n)
        y = f(f(y,n),n)
        p = gcd(x-y,n)
    if (1 < p < n):
        return p
    return 1

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

result = []
composite = 1523357784
rst = composite

while(rst != 1):
    print(f'n is {composite}')
    rst = Pollard_Rho(composite)
    if(rst != 1):
        print(f'{rst} is prime factor')
        result.append(rst)
        composite //= rst
    result.append(composite)

result.sort()
print(result)
```

- 결과

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

```
n is 1523357784
2 is prime factor
n is 761678892
2 is prime factor
n is 380839446
2 is prime factor
n is 190419723
3 is prime factor
n is 63473241
3 is prime factor
n is 21157747
37 is prime factor
n is 571831
13 is prime factor
n is 43987
[2, 2, 2, 3, 3, 13, 37, 43987]
```

소인수분해

- 방법 (9/9)

- 체 기반 소인수분해

- 2차 체(Quadratic Sieve)

- $x^2 \equiv y^2 \pmod{n}$ 을 만족하는 x 와 y 를 찾아, $(x+y)(x-y) \equiv 0 \pmod{n}$ 을 이용하여 소인수분해 하는 방법
 - $O(e^{\sqrt{n_b \times \log(n_b)}})$ 의 복잡도를 가짐
 - 부지수적인(Sub-Exponential) 복잡도를 가짐

- 정수체 체(Number Field Sieve)

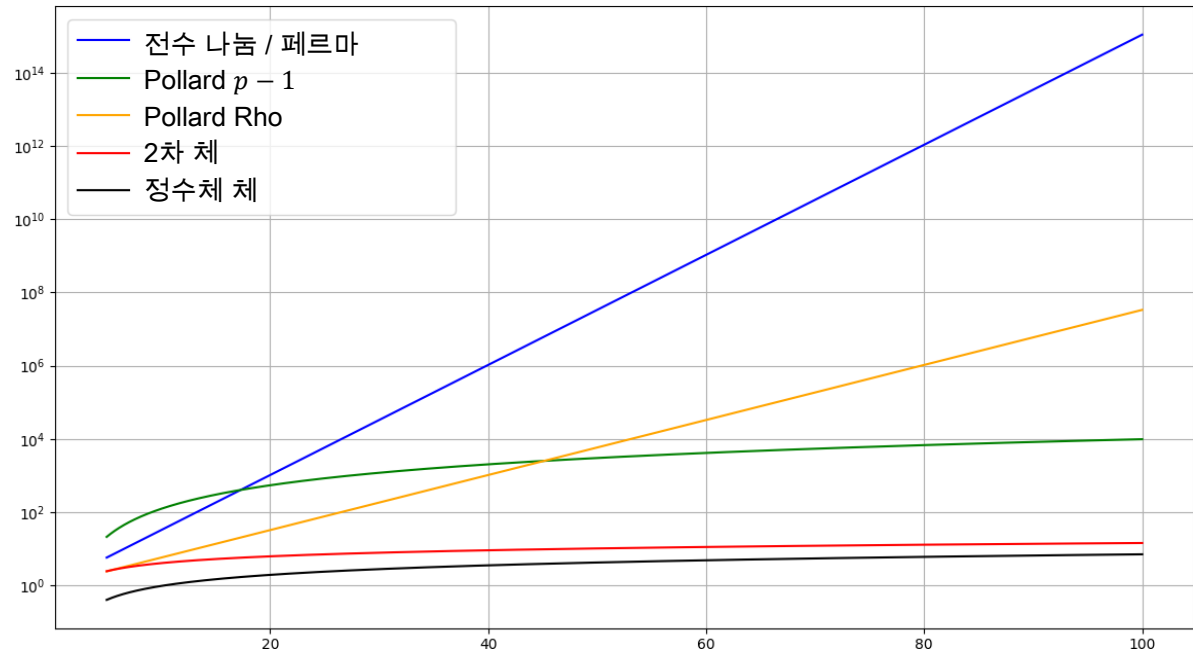
- 다항식 $f(x)$ 를 통해 생성된 체(Field)를 이용하여 $x^2 \equiv y^2 \pmod{n}$ 을 얻어낸 후 $\gcd(x-y, n)$ 또는 $\gcd(x+y, n)$ 을 계산하여 소인수분해 하는 방법
 - $O(e^{n_b^{1/3} \times \log^{2/3}(n_b)})$ 의 복잡도를 가짐
 - 부지수적인(Sub-Exponential) 복잡도를 가짐

소인수분해

• 요약

- 소인수분해의 시간 복잡도는 대부분 지수적이거나 부지수적인 형태를 가짐
- 소인수분해 하려는 수가 커질수록 결과를 얻기 위해 걸리는 시간이 급격히 증가함

방법	복잡도
전수 나눔(Trial Division)	$O(2^{n/2})$
페르마(Fermat)	$O(2^{n/2})$
Pollard $p - 1$	$O(B \times \log B \times n_b)$
Pollard Rho	$O(2^{n/4})$
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 $0 \leq x < M$ 범위에서 유일한 해를 가짐

- 연립 합동 방정식은
$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \vdots \\ x \equiv a_n \pmod{m_n} \end{cases}$$
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 계산함
- $x^2 \equiv a \text{ mod } n$ 꼴의 2차 합동 문제를 해결하는 데 사용함
 - e.g., $p \times q$ 인 합성수 n 를 p 와 q 로 나누어 모듈로 p 와 모듈로 q 에 대한 해를 구하고 이를 연립하여 중국인의 나머지 정리를 통해 모듈로 n 에서의 해를 구함

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

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

$$M_1 = 941094/99 = 9506$$

$$M_2 = 941094/98 = 9603$$

$$M_3 = 941094/97 = 9702$$

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

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

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

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

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

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

목 차

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

2차 합동

- 정의

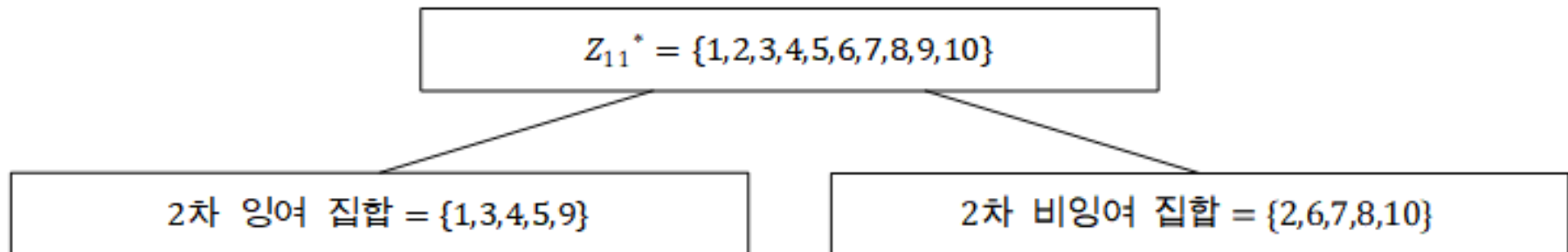
- $Ax^2 + Bx + C \equiv 0 \pmod n$ 의 형태를 갖는 합동 방정식

- 특징

- 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
 - e.g., 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., $3^{(11-1)/2} \bmod 11 \equiv 3^5 \bmod 11 \equiv 243 \bmod 11 \equiv 1 \bmod 11$
 $\therefore 3$ 은 $\bmod 11$ 에서 2차 잉여
 - $a^{(p-1)/2} \equiv -1 \bmod p$ 라면 a 는 2차 비잉여
 - e.g., $2^{(11-1)/2} \bmod 11 \equiv 2^5 \bmod 11 \equiv 32 \bmod 11 \equiv 10 \bmod 11 \equiv -1 \bmod 11$
 $\therefore 2$ 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 9.8

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
```

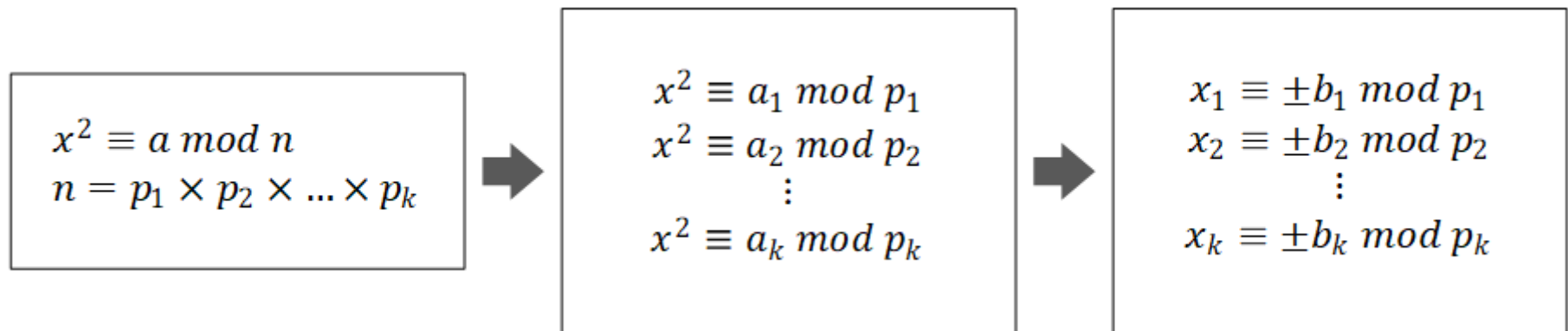
2차 합동

- 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 $\pmod{36}$ 에서의 해를 구함



2차 합동

- 모듈로가 합성수인 2차 합동

- 방정식 풀이 (2/2)

- 예제 9.7

$77 = 7 \times 11$ 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
- $1^3 \equiv 1 \pmod{7}$ 이고 $3^5 \equiv 1 \pmod{11}$ 이므로 해가 존재함
- $x \equiv \pm 1 \pmod{7}$ 와 $x \equiv \pm 5 \pmod{11}$ 를 이용하여 연립 합동 방정식을 만들
 1. $\begin{cases} x \equiv 1 \pmod{7} \\ x \equiv 5 \pmod{11} \end{cases}$
 2. $\begin{cases} x \equiv 1 \pmod{7} \\ x \equiv 6 \pmod{11} \end{cases}$
 3. $\begin{cases} x \equiv 6 \pmod{7} \\ x \equiv 5 \pmod{11} \end{cases}$
 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. $x \equiv 71 \pmod{77}$
 2. $x \equiv 50 \pmod{77}$
 3. $x \equiv 27 \pmod{77}$
 4. $x \equiv 6 \pmod{77}$

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

Thanks!

이 성 현(dltjdgus2166@naver.com)