2024/07/18, 2024 보안 기초 세미나

암호학과 네트워크 보안

-9장 암호 수학 (2)-

이 정 민(jeongmin@pel.sejong.ac.kr)

세종대학교 프로토콜공학연구실

목 차

- 소인수분해
- 중국인의 나머지 정리
- 2차 합동

목 차

- 소인수분해
- 중국인의 나머지 정리
- 2차 합동

소인수분해 (1/20)

- 정의
  - 1보다 큰 자연수를 소인수들의 곱으로 나타내는 방법
    - e.g., $60 = 2^2 \times 3 \times 5$
- 특징
  - 산술의 기본 정리 (Fundamental Theorem of Arithmetic)에 의해 소인수분해 형태로 유일하게 표현될 수 있음
    - 소수 $p_1, p_2, \cdots, p_k$와 양의 정수 $e_1, e_2, \cdots, e_k$에 대해, $n = p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_k^{e_k}$ 형태를 유일하게 가짐
  - 문제의 크기 n에 대한 시간 복잡도가 다항식으로 표현되는 소인수분해 알고리즘이 발견되지 않음
  - 공개 키 암호 시스템의 보안성에 핵심적인 역할임
    - 공개 키 암호는 주로 DLP (Discrete Logarithm Problem)와 IF (Integer Factorization) 문제에 기반함

소인수분해 (2/20)

- 응용 (1/2)

$\text{gcd}(a, b)$: a와 b의 최대 공약수
$\text{lcm}(a, b)$: a와 b의 최소 공배수

  - 최대 공약수 계산
    1. a, b를 소인수분해하여 $a = p_1^{a_1} \times p_2^{a_2} \times \cdots \times p_k^{a_k}$와 $b = p_1^{b_1} \times p_2^{b_2} \times \cdots \times p_k^{b_k}$를 계산함
    2. $\text{gcd}(a, b) = p_1^{\min(a_1, b_1)} \times p_2^{\min(a_2, b_2)} \times \cdots \times p_k^{\min(a_k, b_k)}$로 정의됨
  - 최소 공배수 계산
    1. a, b를 소인수분해하여 $a = p_1^{a_1} \times p_2^{a_2} \times \cdots \times p_k^{a_k}$와 $b = p_1^{b_1} \times p_2^{b_2} \times \cdots \times p_k^{b_k}$를 계산함
    2. $\text{lcm}(a, b) = p_1^{\max(a_1, b_1)} \times p_2^{\max(a_2, b_2)} \times \cdots \times p_k^{\max(a_k, b_k)}$로 정의됨

소인수분해 (3/20)

- 응용 (2/2)
  - 최대 공약수와 최소 공배수의 관계
    - $\gcd(a, b)$와 $\mathrm{lcm}(a, b)$에 대해, $\mathrm{lcm}(a, b) \times \gcd(a, b) = a \times b$가 성립함
    - 증명
      1. $\gcd(a, b) = p_1^{\min(a_1, b_1)} \times p_2^{\min(a_2, b_2)} \times \cdots \times p_k^{\min(a_k, b_k)}$과 $\mathrm{lcm}(a, b) = p_1^{\max(a_1, b_1)} \times p_2^{\max(a_2, b_2)} \times \cdots \times p_k^{\max(a_k, b_k)}$을 곱함
      2. $\gcd(a, b) \times \mathrm{lcm}(a, b)$의 각 항에 대해 $\min(a_i, b_i) + \max(a_i, b_i) = a + b$이고, 결국 a와 b를 소인수분해한 결과를 서로 곱한 것과 동일함
      3. $\therefore \mathrm{lcm}(a, b) \times \gcd(a, b) = a \times b$임

소인수분해 (4/20)

- 방법
  - 구분
    - 특수 목적
      - 특징
        - 알고리즘의 실행 시간이 인수분해할 수 (n)의 속성 (e.g., 두 소인수가 가까움, n이 작은 소인수를 가짐), 크기 등에 따라 달라짐
      - 종류
        - 전수 나눔 소인수분해 방법 (Trial Division Factorization Method)
        - 페르마 소인수분해 방법 (Fermat's Factorization Method)
        - Pollard $p-1$ 소인수분해 방법 (Pollard $p-1$ Factorization Method)
        - Pollard rho 소인수분해 방법 (Pollard $p-1$ Factorization Method)
    - 일반 목적
      - 특징
        - 알고리즘의 실행 시간이 인수분해할 수의 크기에 따라서만 달라짐
      - 종류
        - 2차 체 방법 (Quadratic Sieve Method)
        - 정수체 체 방법 (Number Field Sieve Method)

소인수분해 (5/20)

- 방법: 특수 목적 (1/14)
  - 전수 나눔 소인수분해 방법
    - 원리
      - n이 합성수라면 $\sqrt{n}$ 보다 작은 하나의 소수 p가 존재함
    - 동작
      1. 2부터 $\sqrt{n}$까지, a를 증가시키며 n을 나누는 수를 찾음
      2. a를 찾으면 나누어 떨어지지 않을 때까지 a로 나눔
      3. 위 과정을 반복 수행함
    - 성능
      - $n < 2^{10}$인 경우에 효과가 있음
      - $O(\sqrt{n})$의 시간 복잡도를 가짐

```
Trial_Divison_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 (6/20)

- 방법: 특수 목적 (2/14)
  - 예제 9.29, 예제 9.30

1233과 1523357784를 소인수분해하기 위해서 전수 나눔 알고리즘을 사용하시오.

- 풀이
  1. a를 2로 초기화
  2. a가 $\sqrt{n}$까지 순회하며 n을 나누는 지 판단
  3. n을 나누는 경우 반복적으로 나눠 인수와 차수를 리스트에 저장함
  4. 저장된 리스트 값을 출력함

```
def trial_division(n):
    factor = []
    a = 2

    while a*a <= n:
        e = 0

        while n%a == 0:
            e += 1
            n //= a

        if e > 0:
            factor.append(f'{a}^{e}')
        a += 1
    if n > 1:
        factor.append(f'{n}^1')
    return ' * '.join(factor)

print(trial_division(1233))
print(trial_division(1523357784))
```

```
division.py"
3^2 * 137^1
2^3 * 3^2 * 13^1 * 37^1 * 43987^1
```

소인수분해 (7/20)

- 방법: 특수 목적 (3/14)
  - 페르마 소인수분해 방법 (1/2)
    - 원리
      - $n = x^2 - y^2$인 x와 y를 찾을 수 있으면, $a = (x + y)$이고 $b = (x - y)$인 a, b를 찾을 수 있음
    - 동작
      1. $y^2 = x^2 - n$에서 $x = \lceil\sqrt{n}\rceil$으로 둠
      2. x를 1씩 증가하며 $x^2 - n$이 제곱수가 되는 지 검사함
      3. $x^2 - n$이 제곱수이면 x, y를 찾을 수 있고 $(x + y), (x - y)$를 인수로 가짐
    - 성능
      - 두 소인수 $(x + y), (x - y)$가 서로 가까울 수록 빠르게 작동함
      - $O(\sqrt{n})$의 시간 복잡도를 가짐

```
Fermat_Factoriation (n)
{
    x ← √n
    while(x < n)
    {
        w ← x^2 − n
        if(w is perfect square)
        {
            y ← √w
            a ← x + y
            b ← x − y
            return a and b
        }
        x ← x + 1
    }
}
```

소인수분해 (8/20)

- 방법: 특수 목적 (4/14)
  - 페르마 소인수분해 방법 (2/2)
    - 구현
      1. $x = \sqrt{n},\ w = x^2 - n$으로 초기화
      2. x를 1씩 증가하며 w를 새로 계산
      3. w가 제곱수가 아닌 동안 수행함
      4. 계산한 x와 y를 통해 인수 쌍 a, b를 구함

```
from gmpy2 import *

def fermat_factorization(n):
    assert n % 2 != 0

    x = isqrt(n)
    w = square(x) - n

    while not is_square(w):
        x += 1
        w = square(x) - n
    y = isqrt(w)
    a = x + y
    b = x - y

    return int(a), int(b)

print(fermat_factorization(89 * 97))
```

```
$ python fermat_factorization.py
(97, 89)
```

소인수분해 (9/20)

- 방법: 특수 목적 (5/14)
  - Pollard $p-1$ 방법 (1/4)
    - 원리
      - n이 p라는 소인수를 가진다면, p와 서로소인 a에 대해서 FLT (Fermat's Little Theorem) $a^{p-1} \equiv 1(\mathrm{mod}\; p)$가 성립함
      - $a^{k(p-1)} \equiv 1\ (\mathrm{mod}\; p)$ 또한 성립하며, 이때 $k(p-1) = E$라 지칭
      - $a^E - 1 \equiv 0\ (\mathrm{mod}\; p)$이므로 $p|a^E - 1$이고 $p|n$임
      - 적절한 E를 통해 $\gcd(a^E - 1,\ n) = p$를 구할 수 있음

소인수분해 (10/20)

- 방법: 특수 목적 (6/14)
  - Pollard $p-1$ 방법 (2/4)
    - 동작
      1. 적절한 B를 선택하고 $a^{B!}$를 계산함 (일반적으로, $a=2, E=B!$ 를 사용함)
      2. $p=\gcd(a^E-1, n)$이 $1<p<n$인지 판단함
      3. $1<p<n$이면 소인수 p를 반환, 그렇지 않다면 알고리즘은 실패함

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

소인수분해 (11/20)

- 방법: 특수 목적 (7/14)
  - Pollard $p-1$ 방법 (3/4)
    - E 선택의 중요성
      - $(p-1) \nmid E$인 경우, $p \nmid (a^E - 1)$이고 n과 $a^E - 1$이 서로소임에 따라 $\gcd(a^E - 1, n) = 1$이 됨
        - FLT가 적용되지 않으므로, $(p-1) \nmid E \rightarrow p \nmid (a^E - 1)$임
      - $n = pq$에 대해(p, q는 소수), $(p-1)|E$이고 $(q-1)|E$이면 $pq|a^E - 1$임에 따라 $\gcd(a^E - 1, n) = n$임
        - $(p-1)|E, (q-1)|E$일때 $a^E \equiv 0 \pmod{p}$와 $a^E \equiv 0 \pmod{q}$가 성립하기 때문
      - E가 너무 작으면 $\gcd(a^E - 1, n)$이 1이 나오고, 너무 크면 n이 나옴
      - 일반적으로 $E = 2 \times 3 \times \cdots \times B$나 $E = \mathrm{lcm}(2, 3, \dots, B)$를 두어 계산함

소인수분해 (12/20)

- 방법: 특수 목적 (8/14)
  - Pollard $p-1$ 방법 (4/4)
    - 성능
      - $p-1$이 작은 소인수들을 가질 때에 효과적임
        - 공개 키 암호에서 $p-1$이 큰 소인수를 갖는 소수를 사용해야 안전함
          - e.g., 어떤 소수 q에 대해 $p=2q+1$인 안전 소수 (Safe Prime) p를 사용
      - $\mathrm{O}(B \cdot \log B \cdot \log^2 n)$의 시간 복잡도를 가짐
        - $a^{B!}$을 Repeated Squaring 알고리즘에서 $\mathrm{O}(\log B \cdot \log^2 n)$의 시간 복잡도를 가짐
        - 전체 반복문이 $\mathrm{O}(B)$의 시간 복잡도를 가짐

소인수분해 (13/20)

- 방법: 특수 목적 (9/14)
  - 예제 9.31

57247159의 소인수를 구하기 위해 Pollard $p-1$ 소인수분해 방법을 사용하시오. $(B = 8)$

- 풀이
  1. $p = 421$을 구할 수 있음
  2. $57247159 = 421 \times 135979$임
  3. $421 - 1 = 2^2 \times 3 \times 5 \times 7$로 B보다 작은 인수를 가짐
  4. 135979는 $B = 8$에 의해 None 값을 출력할 것임
     - $p - 1 = 1359759 - 1$은 8보다 큰 인수를 가지기 때문

```python
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def pollard_p_1(n, B=8):
    a = 2
    for j in range(2, B+1):
        a = pow(a, j, n)
    p = gcd(a - 1, n)
    if 1 < p < n:
        return p
    return None

print(pollard_p_1(57247159))
```

```
$ python pollard_p-1.py
421
```

소인수분해 (14/20)

- 방법: 특수 목적 (10/14)
  - Pollard rho 방법 (1/3)
    - 원리
      - $n = pq$에 대해, $x_1 \neq x_2$인 x_1, x_2가 존재하여 $p | (x_1 - x_2)$이고 $n \nmid (x_1 - x_2)$이면 $p = \gcd(x_1 - x_2, n)$임
    - 동작
      1. 랜덤한 시드 (Seed) x_1과 $\mathrm{mod}\ n$에 대한 다항식을 선택함
         - 일반적으로 $f(x) = x^2 + 1$을 다항식으로 사용함
      2. $x_2 = f(x_1) = x_1^2 + 1$인 x_2에 대해, $\gcd(x_1 - x_2, n)$을 계산함
      3. $\gcd(x_1 - x_2, n)$이 1이 아니라면 인수를 찾은 것이고, 1이라면 $x_{i+1} = f(x_i)\ (i \geq 2)$에 대해 위 과정을 반복

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

소인수분해 (15/20)

- 방법: 특수 목적 (11/14)
  - Pollard rho 방법 (2/3)
    - 성능
      - n이 작은 인수를 가질 때 효과적임
      - 순환 탐지 (Cycle Detection) 알고리즘에 따라 성능이 향상될 수 있음
        - Richard Brent는 Floyd’s 알고리즘을 Brent’s 알고리즘으로 변경하여 계산 속도를 약 24% 향상시킴 (Brent, Richard P. "An improved Monte Carlo factorization algorithm." In proc of: *BIT Numerical Mathematics* 20.2, pp.176-184, 1980)
      - $O(\sqrt{p})$의 시간 복잡도를 가짐 (p는 n의 가장 작은 소인수)
        - Birthday Problem* 과 순환 탐지 알고리즘을 통해 확률적으로 추측된 값이며, 엄밀한 증명은 이루어지지 않음
        - $p \le \sqrt{n}$이므로, 약 $n^{\frac{1}{4}}$ 정도의 연산량이 필요하며 비트수 n_b에 대해 비트 연산 복잡도 $O(2^{\frac{n_b}{4}})$를 가짐

*Birthday Problem: n개의 가능한 값들 중에서 k개의 값을 무작위로 선택할 때, 두 값이 동일할 확률을 계산하는 문제로, $k \approx \sqrt{n}$일 때 약 50%의 확률을 가짐

소인수분해 (16/20)

- 방법: 특수 목적 (12/14)
  - 예제 9.32

어떤 컴퓨터가 초당 2^{30} (약 10억)번 정도의 비트 연산을 할 수 있을 때, 다음과 같은 크기의 정수를 Pollard rho 방법으로 소인수분해 할 때 걸리는 시간이 어느 정도 되는가?

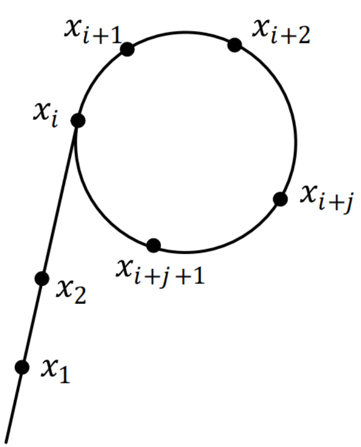
a. 60자리 10진수
b. 100자리 10진수

- 풀이
  a. 2진수로 바꾸면 약 200자리 비트 수를 가짐에 따라 복잡도는 약 2^{50}임 따라서, $\frac{2^{50}}{2^{30}} = 2^{20}$ 초인 약 12일이 걸림

  b. 2진수로 바꾸면 약 300자리 비트 수를 가짐에 따라 복잡도는 약 2^{75}임 따라서, $\frac{2^{75}}{2^{30}} = 2^{45}$초가 걸리며 이는 백만 년 단위로 환산됨

소인수분해 (17/20)

- 방법: 특수 목적 (13/14)
  - Pollard rho 방법 (3/3)
    - 구현
      1. 시드는 2로 설정
      2. Floyd's 알고리즘을 사용하여 구현
         - x_1은 한 단계, x_2는 두 단계씩 함수 f를 수행
      3. $1 < p < n$인 p를 반환함
         - p값이 n이라면 None을 반환하며 실패




```
from math import *

def f(x, n):
    return (x*x + 1) % n

def pollard_rho(n):
    x1 = 2
    x2 = 2
    p = 1

    while p == 1:
        x1 = f(x1, n)
        x2 = f(f(x2, n), n)
        p = gcd(x1 - x2, n)

    if 1 < p < n:
        return p
    return None

print(pollard_rho(89 * 97))
```

```
$ python pollard_rho.py
97
```

소인수분해 (18/20)

- 방법: 특수 목적 (14/14)
  - 예제 9.33

Pollard rho 방법을 사용하는 소인수분해 프로그램에서, 434617의 결과로 709를 얻었다. 그 과정을 순서쌍 x_1, x_2와 p값을 통해 나타내시오.

- 풀이
  - x_1, x_2가 다음과 같이 변화하며 p를 도출함

x1	x2	p
2	2	1
5	26	1
26	23713	1
677	142292	1
23713	157099	1
346589	52128	1
142292	41831	1
380320	68775	1
157099	427553	1
369457	72634	1
52128	63593	1
102901	161353	1
41831	64890	1
64520	21979	1
68775	16309	709

```python
from math import *

def f(x, n):
    return (x*x + 1) % n

def pollard_rho(n):
    x1 = 2
    x2 = 2
    p = 1
    print('|   x1    |   x2    |   p    |')
    print('----------------------------')
    print(f'{x1:7}|{x2:7}|{p:7}')

    while p == 1:
        x1 = f(x1, n)
        x2 = f(f(x2, n), n)
        p = gcd(x1 - x2, n)
        print(f'{x1:7}|{x2:7}|{p:7}')

pollard_rho(434617)
```

소인수분해 (19/20)

- 방법: 일반 목적 (1/2)
  - 2차 체 방법
    - 동작
      - n을 제곱수들의 차로 나타내고 연립하여 소인수를 찾음
    - 성능
      - 100자리 이하 숫자를 소인수분해하는 가장 빠른 방법
      - $\mathrm{O}(e^{C})$의 시간 복잡도를 가짐 ($C \approx \{\ln n \times \ln(\ln n)\}^{\frac{1}{2}}$)
  - 정수체 체 방법
    - 동작
      - 체 (Field)에서 n에 관한 다항식을 찾고 이를 이용하여 소인수를 찾음
    - 성능
      - 100자리보다 큰 정수를 소인수분해하는 가장 빠른 방법
      - $\mathrm{O}(e^{C})$의 시간 복잡도를 가짐 ($C \approx \{(\ln n)^{\frac{1}{3}} \times (\ln(\ln n))^{\frac{2}{3}}\}$)

소인수분해 (20/20)

- 방법: 일반 목적 (2/2)
  - 예제 9.34

어떤 컴퓨터가 초당 2^{30} (약 10억)번 정도의 비트 연산을 할 수 있을 때, 다음과 같은 방법을 사용할 시, 100자리 10진 정수를 소인수분해 하는데 대략 얼마나 걸리는가?

a. 2차 체 방법
b. 정수체 체 방법

- 풀이
  - 자리수가 100인 10진 정수는 2진수로 대략 300비트 정도임
    $\therefore n = 2^{300}, \ln(2^{300}) \approx 207, \ln(\ln 2^{300})) \approx 5$임

a. 2차 체 방법에서는 $(207)^{\frac{1}{2}} \times (5)^{\frac{1}{2}} \approx 32$를 통해 e^{32}번의 비트 연산이 필요함
따라서, $\frac{e^{32}}{2^{30}} \approx 20$시간 정도가 소요됨

b. 정수체 체 방법에서는 $(207)^{\frac{1}{3}} \times (5)^{\frac{2}{3}} \approx 34$를 통해 e^{34}번의 비트 연산이 필요함
따라서, $\frac{e^{34}}{2^{30}} \approx 140$시간 정도가 소요됨

목 차

- 소인수분해
- 중국인의 나머지 정리
- 2차 합동

중국인의 나머지 정리 (1/6)

- 정의
  - 모듈로 값들이 서로소일 때, 다음과 같은 꼴의 합동 방정식이 유일한 해를 갖는다는 것을 보인 정리
    - 합동 방정식은 $\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \qquad \cdots \\ x \equiv a_k \pmod{m_k} \end{cases}$ 꼴로 표현됨

중국인의 나머지 정리 (2/6)

- 과정
  1. 공통 모듈로로 사용할 $M = m_1 \times m_2 \times \cdots \times m_k$를 구함
  2. $M_1 = \frac{M}{m_1}, M_2 = \frac{M}{m_2}, \ldots, M_k = \frac{M}{m_k}$를 구함
  3. $m_1, m_2, \cdots, m_k$ 각각에 대해, $M_1, M_2, \cdots, M_k$의 역원인 $M_1^{-1}, M_2^{-1}, \cdots, M_k^{-1}$ 를 구함
  4. 연립 방정식의 해 $x = \left(a_1 \times M_1 \times M_1^{-1} + a_2 \times M_2 \times M_2^{-1} + \cdots + a_k \times M_k \times M_k^{-1}\right) \bmod M$을 구함

중국인의 나머지 정리 (3/6)

- 예제 9.36

다음 연립 방정식의 해를 구하시오.

$$\begin{cases} x \equiv 2 \bmod 3 \\ x \equiv 3 \bmod 5 \\ x \equiv 2 \bmod 7 \end{cases}$$

- 풀이
  1. $M = 3 \times 5 \times 7 = 105$임
  2. $M_1 = \frac{105}{3} = 35, \quad M_2 = \frac{105}{5} = 21, \quad M_3 = \frac{105}{7} = 15$임
  3. $M_1^{-1} = 2, \quad M_2^{-1} = 1, \quad M_3^{-1} = 1$임
  4. $\therefore x = (2 \times 35 \times 2 + 3 \times 21 \times 1 + 2 \times 15 \times 1) \bmod 105 = 23 \bmod 105$

중국인의 나머지 정리 (4/6)

- 예제 9.37

7과 13으로 나누었을 때 나머지가 3이고 12로 나누어 떨어지는 정수를 구하시오.

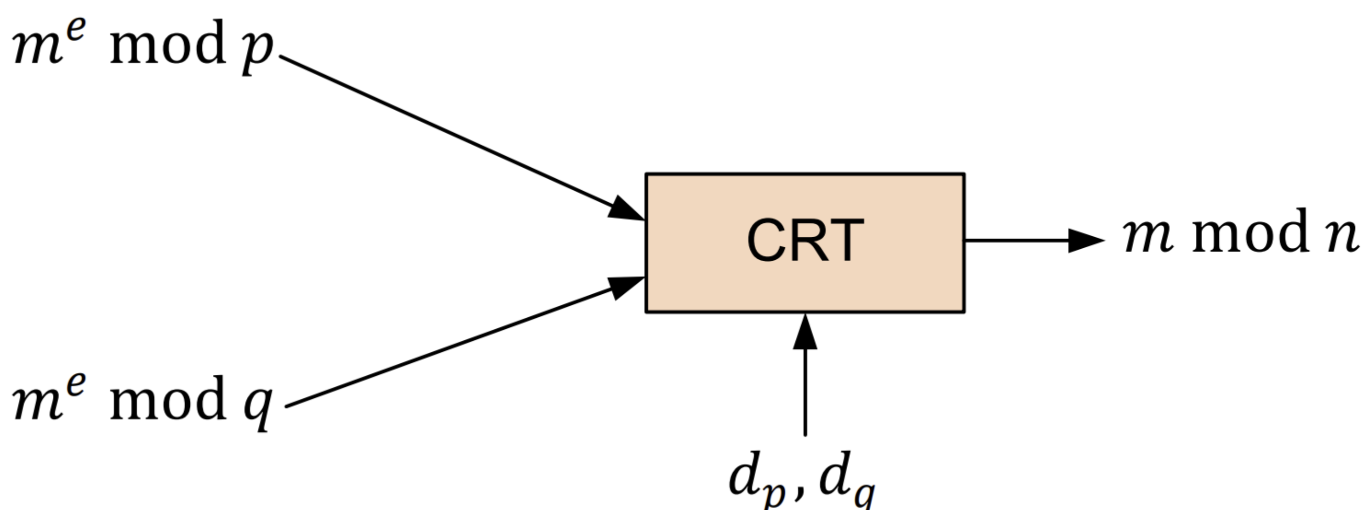
- 풀이
  1. 다음과 같은 방정식 형태로 표현 가능함

$$\begin{cases} x \equiv 3 \bmod 7 \\ x \equiv 3 \bmod 13 \\ x \equiv 0 \bmod 12 \end{cases}$$

  2. 예제 9.36의 4단계 절차에 따라 $x \equiv 276 \bmod 1092$를 구할 수 있음

중국인의 나머지 정리 (5/6)

- 응용
  - $\mathrm{mod}\ n$에서의 연산 효율 상승에 주로 응용
    - e.g., RSA 복호화에 대표적으로 사용됨



$$d_p = e^{-1} \bmod p$$
$$d_q = e^{-1} \bmod q$$

중국인의 나머지 정리 (6/6)

- 예제 9.38

시스템의 능력상 100보다 작은 수만 입력 가능할 때, $x = 123, y = 334$인 $z = x + y$를 계산하시오.

- 풀이
  1. x와 y를 각각 $\text{mod } 99, \text{mod } 98, \text{mod } 97$ 상에서 계산해 둠

$$\begin{cases} x \equiv 24 \pmod{99} \\ x \equiv 25 \pmod{98} \\ x \equiv 26 \pmod{97} \end{cases}, \quad \begin{cases} y \equiv 37 \pmod{99} \\ y \equiv 40 \pmod{98} \\ y \equiv 43 \pmod{97} \end{cases}$$

  2. $\text{mod } 99, \text{mod } 98, \text{mod } 97$ 상에서 각각 $x + y$를 계산하면 다음과 같음

$$\begin{cases} z \equiv x + y \equiv 61 \pmod{99} \\ z \equiv x + y \equiv 65 \pmod{98} \\ z \equiv x + y \equiv 69 \pmod{97} \end{cases}$$

  3. 중국인의 나머지 정리를 이용하여 해 $z \equiv 457 \text{ mod } 941094$를 구할 수 있음

목 차

- 소인수분해
- 중국인의 나머지 정리
- 2차 합동

2차 합동 (1/9)

- 정의
  - $c_2x^2 + c_1x + c_0 \equiv 0 \pmod{n}$과 같은 형태를 갖는 방정식 ($c_i$는 임의의 정수, $c_2 \neq 0$)
    - $x^2 \equiv a \pmod{n}$형태의 해를 논할 예정
- 경우 (Case)
  - 모듈로가 소수인 2차 합동 (p는 홀수인 소수)
  - 모듈로가 합성수인 2차 합동

2차 합동 (2/9)

- 모듈로가 소수인 2차 합동 (1/5)
  - 해의 존재성
    - 해가 존재하지 않거나, 서로 합동이 아닌 두 개의 해를 가짐
    - 2차 잉여 (QR, Quadratic Residue)
      - $x^2 \equiv a \pmod{p}$에서 두 개의 해를 가진다면 a를 2차 잉여라고 부름
    - 2차 비잉여 (QNR, Quadratic NonResidue)
      - $x^2 \equiv a \pmod{n}$에서 해를 가지지 않으면 a를 2차 비잉여라고 부름
    - 오일러의 판정 기준 (Euler's Criterion)
      - $a^{\frac{p-1}{2}} \equiv 1 \pmod{p}$이면 모듈로 p에 대해 a는 2차 잉여임
      - $a^{\frac{p-1}{2}} \equiv -1 \pmod{p}$이면 모듈로 p에 대해 a는 2차 비잉여임

2차 합동 (3/9)

- 모듈로가 소수인 2차 합동 (2/5)
  - 예제 9.39

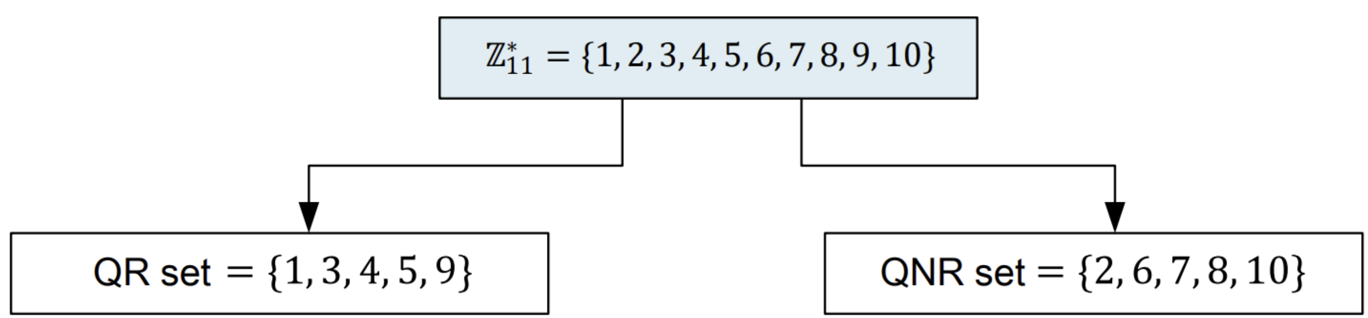
2차 합동 방정식 $x^2 \equiv 3 \pmod{11}$의 두 개의 해 $x \equiv 5 \pmod{11}$과 $x \equiv -5 \pmod{11}$을 비교하여라.

- 풀이
  - $x \equiv -5 \pmod{11}$에서 $-5 \equiv 6 \pmod{11}$이므로 두 개의 해는 서로 합동이 아님

- 모듈로가 소수인 2차 합동 (3/5)
  - 예제 9.40, 예제 9.41

$\mathbb{Z}_{11}^{*}$에서 2, 4, 6이 QR인지 아닌지 판정하라.

- 풀이
  1. $2^{\frac{11-1}{2}} \bmod 11 \rightarrow 32 \bmod 11 \rightarrow -1 \bmod 11$이므로 QNR
  2. $4^{\frac{11-1}{2}} \bmod 11 \rightarrow 1024 \bmod 11 \rightarrow 1 \bmod 11$이므로 QR
  3. $6^{\frac{11-1}{2}} \bmod 11 \rightarrow 7776 \bmod 1 \rightarrow 10 \bmod 11 \rightarrow -1 \bmod 11$이므로 QNR임



2차 합동 (5/9)

- 모듈로가 소수인 2차 합동 (4/5)
  - 풀이
    1. $x^2 \equiv a \pmod p$에서, 오일러 판정을 통해 $a^{\frac{p-1}{2}} \equiv 1 \pmod p$임
    2. $a^{\frac{p-1}{2}} \equiv 1 \pmod p$양변에 a를 곱함
    3. $a^{\frac{p+1}{2}} \equiv a \pmod p$에 루트를 취함
    4. $\pm a^{\frac{p+1}{4}} \equiv x \pmod p$이므로, $p \equiv 3 \bmod 4$에 대해 바로 x를 도출할 수 있음
       - $p \equiv 1 \bmod 4$이면 Tonelli-Shanks 알고리즘을 사용하여 도출함

2차 합동 (6/9)

- 모듈로가 소수인 2차 합동 (5/5)
  - 예제 9.43

다음 2차 합동 방정식의 해를 구하시오.
a. $x^2 \equiv 3 \pmod{23}$
b. $x^2 \equiv 2 \pmod{11}$
c. $x^2 \equiv 7 \pmod{19}$

- 풀이
  - 3은 $\mathbb{Z}_{23}^*$에서 QR이고, $3^{\frac{23+1}{4}} \equiv 729 \equiv 16 \bmod 23$이므로 해는 $x = \pm 16 \bmod 23$
  - 2는 $\mathbb{Z}_{11}^*$에서 QNR임
  - 7은 $\mathbb{Z}_{19}^*$에서 QR이고, $7^{\frac{19+1}{4}} \equiv 16807 \equiv 11 \bmod 19$이므로 해는 $x = \pm 11 \bmod 19$

2차 합동 (7/9)

- 모듈로가 합성수인 2차 합동 (1/3)
  - 해의 존재성
    - $x^2 \equiv a \pmod{n}$을 모듈로가 소수인 여러 개의 방정식으로 분해할 때, 각각의 합동 방정식이 해를 가진다면 해가 존재함
  - 풀이
    - $x^2 \equiv a \pmod{n}$을 모듈로가 소수인 k개의 방정식으로 분해될 때, 각각의 방정식의 해를 구함으로써 x에 대한 k쌍의 해를 찾을 수 있음

$$\begin{array}{c} x^2 \equiv a \bmod n \\ n = p_1 \times p_2 \times \cdots \times p_k \end{array} \longrightarrow \begin{array}{c} x^2 \equiv a_1 \bmod p_1 \\ x^2 \equiv a_2 \bmod p_2 \\ \cdots \\ x^2 \equiv a_k \bmod p_k \end{array} \longrightarrow \begin{array}{c} x_1 \equiv \pm b_1 \bmod p_1 \\ x_2 \equiv \pm b_2 \bmod p_2 \\ \cdots \\ x_k \equiv \pm b_k \bmod p_k \end{array}$$

2차 합동 (8/9)

- 모듈로가 합성수인 2차 합동 (2/3)
  - 예제 9.44

$x^2 = 36 \pmod{77}$의 해를 구하시오.

- 풀이
  1. 77을 소인수분해하여, $x^2 \equiv 36 \pmod{7}$과 $x^2 \equiv 36 \pmod{11}$을 구함
  2. 각각의 해를 풀어내면, $x^2 \equiv 36 \pmod{7}$로부터 $x \equiv \pm 1^{\frac{7+1}{4}} \equiv \pm 1 \pmod{7}$ 이고 $x^2 \equiv 36 \pmod{11}$에서 $x \equiv \pm 3^{\frac{11+1}{4}} \equiv \pm 5 \pmod{11}$임
  3. 각 방정식에서 구한 해로부터, 4개의 연립 방정식을 만들어 해를 구함
  4. $\begin{cases} x \equiv +1 \pmod{7}, & x \equiv +5 \pmod{11} \\ x \equiv +1 \pmod{7}, & x \equiv -5 \pmod{11} \\ x \equiv -1 \pmod{7}, & x \equiv +5 \pmod{11} \\ x \equiv -1 \pmod{7}, & x \equiv -5 \pmod{11} \end{cases}$ 에서 중국인의 나머지 정리를 수행함
  5. $\therefore x \equiv \pm 6, \pm 27 \pmod{77}$

2차 합동 (9/9)

- 모듈로가 합성수인 2차 합동 (3/3)
  - 응용
    - 모듈로가 합성수인 2차 합동 방정식의 해를 구하는 것은 합성수를 소인수분해 하는 어려움과 동등함을 이용
      - e.g., Rabin 암호 시스템에서 사용됨
        - $n = pq$를 사용하여 $c = m^2 \pmod{n}$ 방식으로 암호화 수행

Thanks!

이 정 민(jeongmin@pel.sejong.ac.kr)