2024/07/18, 2024 보안 기초 세미나

암호학과 네트워크 보안

-9장 암호 수학 (1)-

이 은 진(eunjin@pel.sejong.ac.kr)

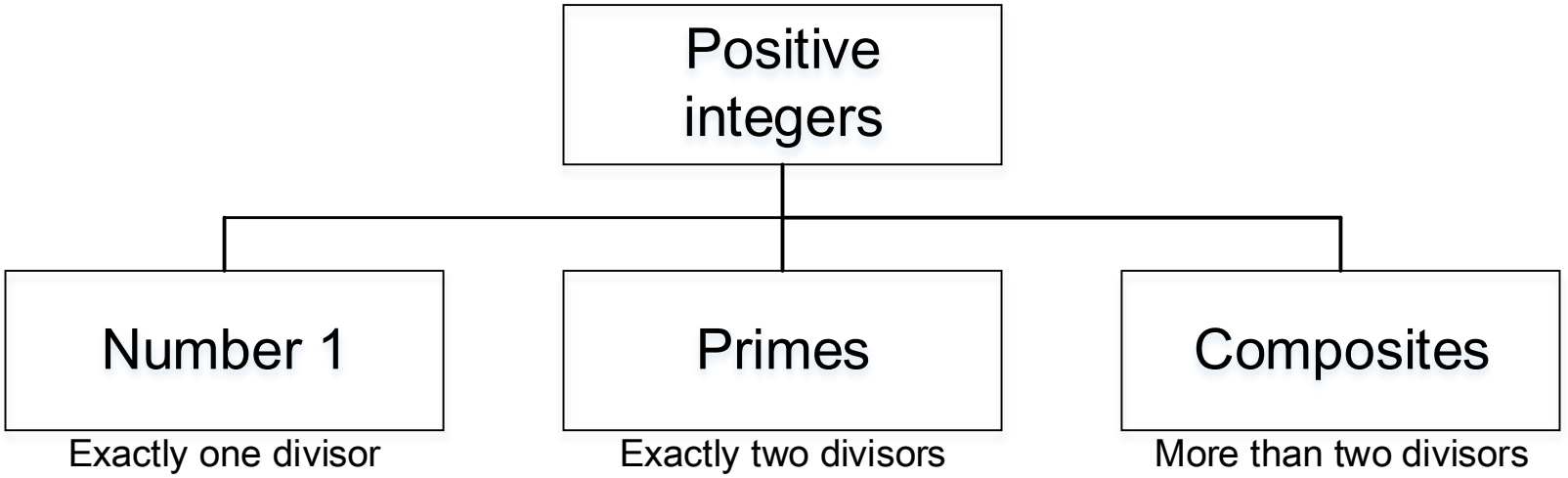
세종대학교 프로토콜공학연구실

목 차

목 차

소수 (1/28)

- 정의
  - 오직 1과 자기 자신으로만 나누어 떨어지는 수



- 서로소(Relatively Prime, Coprime)
  - 양의 정수 a와 b가 $\gcd(a, b) = 1$을 만족할 경우, a와 b는 서로소임
  - p가 소수라면 1부터 $p - 1$까지의 모든 정수는 p와 서로소가 됨

소수 (2/28)

- 소수 집합의 크기 (1/4)
  - 명제
    - 무한히 많은 소수

표기법	설명
p	가장 큰 소수
P	유한개의 소수를 모두 곱한 수
q	p보다 작거나 같은 소수

증명

- 모순증명을 이용하여 다음과 같이 증명할 수 있음
  - 소수의 개수가 유한개라고 가정함
  - 유한개의 소수를 모두 곱한 수는 $P = 2 \times 3 \times \cdots \times p$임
  - $(P+1)$은 p보다 작거나 같은 소수 q를 약수로 갖지 못함
    - 만약 $q|(P+1)$이고 $q|P$인 경우, $q|(P+1) - P = 1$이므로 모순이 발생함
  - $\therefore q > p$임

소수 (3/28)

- 소수 집합의 크기 (2/4)
  - 예제 9.3

    소수 전체 집합이 $\{2, 3, 5, 7, 11, 13, 17\}$일 경우, 17보다 큰 소수가 존재함을 증명하시오.

    - 풀이
      - 소수 전체 집합 $\{2, 3, 5, 7, 11, 13, 17\}$의 곱인 $P = 510510$임
      - $P + 1 = 510511$로 $19 \times 97 \times 277$로 표현됨
      - $19, 97, 277$은 소수 전체 집합에 포함되지 않음
      - 17보다 큰 소수가 3개 존재함

소수 (4/28)

- 소수 집합의 크기 (3/4)
  - 소수의 개수 (1/2)
    - 함수 $\pi(n)$은 n보다 작거나 같은 소수의 개수를 나타냄
      - e.g., $\pi(1) = 0$, $\pi(2) = 1$, $\pi(3) = 2$, $\pi(10) = 4$
      - n이 매우 큰 경우, $\pi(n)$을 근사 값으로 구할 수 있음
        - $\left[\frac{n}{(\ln n)}\right] < \pi(n) < \left[\frac{n}{(\ln n - 1.08366)}\right]$

◈[](가우스 기호): 실수 n에 대하여 n을 넘지 않는 최대 정수를 $[n]$이라고 함

소수 (5/28)

- 소수 집합의 크기 (4/4)
  - 소수의 개수 (2/2)
    - 예제 9.4

$1{,}000{,}000$보다 작은 소수의 개수를 구하시오.

      - 풀이
        - $\left\lceil \frac{n}{(\ln n)} \right\rceil < \pi(n) < \left\lceil \frac{n}{(\ln n - 1.08366)} \right\rceil$ 이 범위에 들어오는 소수의 개수는 $72{,}383$에서 $78{,}543$개 사이의 수로 실제 $1{,}000{,}000$보다 작은 소수의 개수는 $78{,}498$개임

소수 (6/28)

- 소수 판정 (1/4)
  - 방법
    - 주어진 수 n이 $\sqrt{n}$보다 작은 모든 소수로 나누어지는지 확인
      - 나누어지면 합성수, 나누어지지 않으면 소수임
  - 필요성
    - 공개키 암호를 만들 때 소수를 사용하여 만듦
    - 매우 큰 정수의 소인수분해는 현실적으로 어려운 문제로 암호 시스템의 안전성을 보장해줌

소수 (7/28)

- 소수 판정 (2/4)
  - 예제 9.5

    수 97은 소수인가?

    - 풀이
      - $9 < \sqrt{97} < 10$이므로 $\sqrt{97}$보다 작은 소수는 $2, 3, 5, 7$임
      - $2, 3, 5, 7$은 97을 나누지 못하므로 97은 소수임
  - 예제 9.6

    수 301은 소수인가?

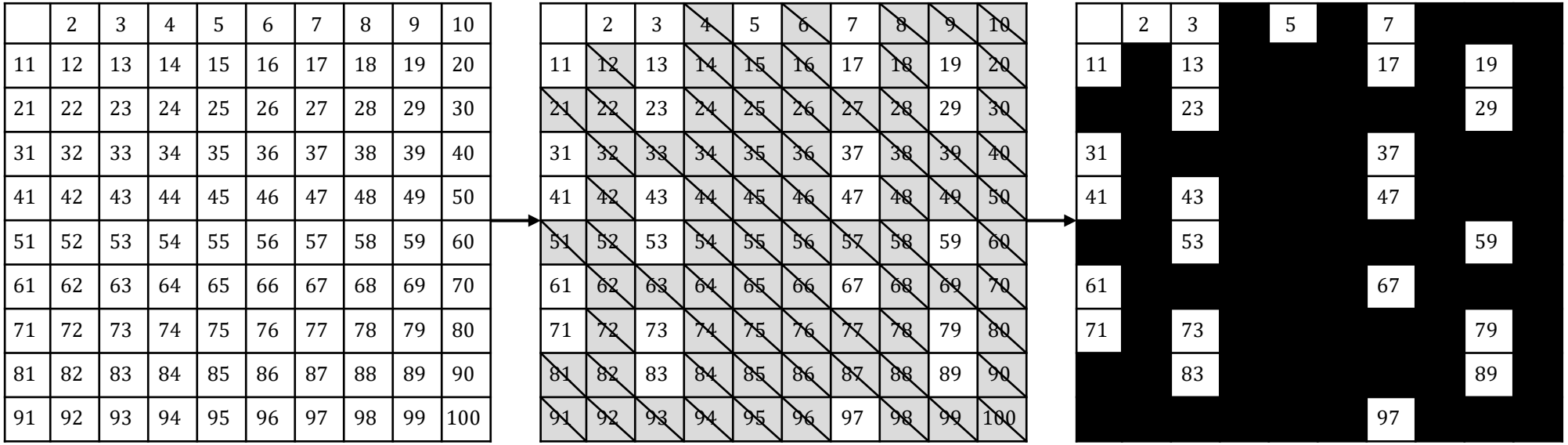
    - 풀이
      - $17 < \sqrt{301} < 18$이므로 $\sqrt{301}$보다 작은 소수는 $2, 3, 5, 7, 11, 13, 17$임
      - $2, 3, 5, 11, 13, 17$은 301을 나누지 못하지만 7은 301을 나누므로 301은 소수가 아님

소수 (8/28)

- 소수 판정 (3/4)
  - 에라토스테네스의 체(Sieve of Eratosthenes) (1/2)
    - 개요
      - 에라토스테네스가 고대에 발견한 것으로, 주어진 수 n보다 작은 모든 소수를 찾는 방법임
    - 방법
      - $\sqrt{n}$보다 작거나 같은 소수들로 나누어떨어지는 수는 모두 지우고 나누는 수들 자신은 남김
        - 지워지지 않은 수가 소수임
    - 특징
      - 작은 수 n이 주어진다면 쉽게 n보다 작은 모든 소수를 구할 수 있음
      - 큰 수 n이 주어진다면 2부터 n까지 모든 수를 쓰지 못하기 때문에 큰 수에는 적합하지 않은 한계점이 있음

소수 (9/28)

- 소수 판정 (4/4)
  - 에라토스테네스의 체(Sieve of Eratosthenes) (2/2)
    - e.g., 100보다 작은 모든 소수를 구하여보자.
      1. 2부터 100까지 모든 수를 씀
      2. $\sqrt{100} = 10$으로 10보다 작은 소수 $2, 3, 5, 7$로 나누어떨어지는 수는 모두 지우고 $2, 3, 5, 7$ 자신은 남김
      3. 지워지지 않고 남아있는 수가 소수임



소수 (10/28)

- 오일러의 φ 함수(Euler's Phi-function) (1/5)
  - 개요
    - 1763년, 오일러가 이 함수에 대해 발표하였지만 그 당시에는 특정 기호를 사용하지 않았음
    - 표기법 φ은 1801년 가우스의 논문에서 유래한 것임
  - 정의
    - $\varphi(n)$은 n과 서로소인 양의 정수의 개수
  - 특징
    - $\varphi(n)$을 구하는 어려움의 정도는 n을 소인수분해 하는 어려움의 정도에 종속됨
    - $\varphi(n)$에서 $n > 2$인 경우, $\varphi(n)$ 값은 항상 짝수임

소수 (11/28)

- 오일러의 φ 함수(Euler's Phi-function) (2/5)
  - 성질
    - $\varphi(1) = 1$
    - 소수 p는 1부터 $p-1$까지의 자연수가 모두 p와 서로소이므로 p가 소수일 경우, $\varphi(p) = p - 1$
      - e.g., $\varphi(13) = 13 - 1 = 12$
    - m과 n이 서로소인 경우, $\varphi(m \times n) = \varphi(m) \times \varphi(n)$
      - e.g., $\varphi(10) = \varphi(2) \times \varphi(5) = (2-1) \times (5-1) = 1 \times 4 = 4$
    - p가 소수인 경우, $\varphi(p^e) = p^e - p^{e-1}$
      - e.g., $\varphi(240) = \varphi(2^4) \times \varphi(3) \times \varphi(5) = (2^4 - 2^3) \times (3-1) \times (5-1) = 64$

소수 (12/28)

- 오일러의 φ 함수(Euler's Phi-function) (3/5)
  - 가정
    - 만약 n이 소인수분해되어 $p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_k^{e_k}$가 됨

$\varphi(n) = \left(p_1^{e_1} - p_1^{e_1-1}\right) \times \left(p_2^{e_2} - p_2^{e_2-1}\right) \times \cdots \times \left(p_k^{e_k} - p_k^{e_k-1}\right)$로 $\varphi(n)$값을 구할 수 있음

소수 (13/28)

- 오일러의 φ 함수(Euler's Phi-function) (4/5)
  - $\varphi(n)$에서 $n > 2$인 경우, $\varphi(n)$ 값은 항상 짝수임
    - 증명 (1/2)
      - 소수 $p_1, p_2, \cdots, p_k$와 중복 가능한 $m_1, m_2, \cdots, m_k$가 존재할 때 n은 다음과 같음

$$n = p_1^{m_1} \times p_2^{m_2} \times \cdots \times p_k^{m_k}$$

      - φ의 성질을 사용하여 식을 정리하면 다음과 같음

$$\varphi(n) = \varphi(p_1^{m_1}) \times \varphi(p_2^{m_2}) \times \cdots \times \varphi(p_k^{m_k}) = (p_1^{m_1} - p_1^{m_1-1}) \times \varphi(p_2^{m_2}) \times \cdots \times \varphi(p_k^{m_k})$$

      - $p_1 = 2$일 경우, 다음과 같음

$$\varphi(n) = (2^{m_1} - 2^{m_1-1}) \times \varphi(p_2^{m_2}) \times \cdots \times \varphi(p_k^{m_k}) =$$
$$2^{m_1} \times (1 - \frac{1}{2}) \times \varphi(p_2^{m_2}) \times \cdots \times \varphi(p_k^{m_k}) = 2^{m_1-1} \times \varphi(p_2^{m_2}) \times \cdots \times \varphi(p_k^{m_k})$$

소수 (14/28)

- 오일러의 φ 함수(Euler's Phi-function) (5/5)
  - $\varphi(n)$에서 $n > 2$인 경우, $\varphi(n)$ 값은 항상 짝수임
    - 증명 (2/2)
      - $m_1 \geq 2$일 때 다음과 같으므로 $\varphi(n)$은 짝수임

$$\varphi(n) = 2 \times 2^{m_1-2} \times \varphi(p_2^{m_2}) \times \cdots \times \varphi(p_k^{m_k})$$

      - $m_1 = 1$인 경우, $n \geq 3$이므로 $p_1 = 2 < p_2$가 존재함
      - p_2 는 홀수이므로 $p_2 - 1$은 짝수이기에 다음과 같음

$$\begin{aligned}\varphi(n) &= 2^{m_1-1} \times (p_2^{m_2} - p_2^{m_2-1}) \times \varphi(p_3^{m_3}) \times \cdots \times \varphi(p_k^{m_k}) \\ &= p_2^{m_2-1} \times (p_2 - 1) \times \varphi(p_3^{m_3}) \times \cdots \times \varphi(p_k^{m_k})\end{aligned}$$

      - $p_1 \neq 2$인 경우, $2 < p_1$이므로 다음과 같으므로 $\varphi(n)$은 짝수임

$$\varphi(n) = p_1^{m_1-1} \times (p_1 - 1) \times \varphi(p_2^{m_2}) \times \cdots \times \varphi(p_k^{m_k})$$

소수 (15/28)

- 페르마의 리틀 정리(Fermat's Little Theorem) (1/4)
  - 개요
    - 페르마가 1640년 친구와의 편지에서 이 정리를 처음 언급함
    - 이 정리의 증명은 페르마가 아닌 오일러가 함
  - 정의
    - 버전 1
      - 가정
        - 만약 p가 소수이고 a가 하나의 어떤 정수로서 p는 a를 나누지 못함

$$a^{p-1} \equiv 1 \, mod \, p$$

    - 버전 2
      - 가정
        - 만약 p가 소수이고 a가 한 정수임

$$a^{p} \equiv a \, mod \, p$$

소수 (16/28)

- 페르마의 리틀 정리(Fermat's Little Theorem) (2/4)
  - 버전 1 증명 (1/2)
    - 1부터 $p-1$까지의 집합 Z_p에 p와 서로소인 a를 곱하면 다음과 같음

$$Z_p = \{1, 2, 3, \cdots, p-1\},\ aZ_p = \{a1, a2, a3, \cdots, a(p-1)\}$$

    - 집합 aZ_p를 p로 나누면 나오는 나머지가 모두 다름

- 모순증명으로, 두 수 ia, $ja(0 < i < j < p)$가 존재해서 그 나머지가 같다고 할 경우, 그 두 수의 차 $(j-i)a$는 p로 나누어짐
- $0 < j - i < p$이므로 $j-i$는 p의 배수가 아니며, 문제의 가정에 따라 a는 p와 서로소임
- 같은 나머지를 가지는 수가 없으므로, $p-1$개의 수는 모두 나머지가 다름

소수 (17/28)

- 페르마의 리틀 정리(Fermat's Little Theorem) (3/4)
  - 버전 1 증명 (2/2)
    - 집합 aZ_p를 p로 나눈 나머지는 Z_p의 원소와 대응됨

$\gcd(a, p) = 1$일 경우, a^{-1}가 존재하므로 aZ_p와 Z_p은 일대일대응임

    - 집합 Z_p와 집합 aZ_p는 원소의 값과 수가 같으므로 다음과 같음

$$a \times 2a \times 3a \times \cdots \times (p-1)a \equiv 1 \times 2 \times \cdots \times (p-1) (mod\ p)$$

    - 양변을 $(p-1)!$로 나눔

$$a^{p-1} \equiv 1\ mod\ p$$

소수 (18/28)

- 페르마의 리틀 정리(Fermat's Little Theorem) (4/4)
  - 곱에 대한 역원
    - 페르마의 리틀 정리를 사용하면 모듈로 값이 소수일 경우, 곱에 관한 역원을 구할 수 있음
      - 가정
        - 만약 p가 소수이고 a가 정수로서 p로 나누어지지 않는 수임

$$a^{-1} \, mod \, p = a^{p-2} mod \, p$$

증명

$$a \times a^{-1} \, mod \, p = a \times a^{p-2} \, mod \, p = a^{p-1} \, mod \, p = 1 \, mod \, p$$

소수 (19/28)

- 오일러 정리(Euler's Theorem) (1/6)
  - 개요
    - 1763년, 오일러가 페르마 리틀 정리에 대한 다른 증명을 제시하여 논문에서 n이 소수가 아닌 경우에 대한 일반화를 증명함
  - 정의
    - 버전 1

$$a^{\varphi(n)} \equiv 1 \; mod \; n \; (\gcd(a,n) = 1)$$

    - 버전 2

$$a^{k \times \varphi(n)+1} \equiv a \; (mod \; n) \; (n = p \times q, \; a < n, \; k \in Z)$$

소수 (20/28)

- 오일러 정리(Euler's Theorem) (2/6)
  - 버전 1 증명 (1/2)
    - n 이하 자연수에서 n과 서로소인 수들의 집합에 a를 곱한 집합은 다음과 같음

$$S = \{s_1, s_2, \cdots, s_{\varphi(n)}\},\ AS = \{as_1,\ as_2,\ \cdots,\ as_{\varphi(n)}\}$$

    - 집합 AS를 n으로 나누면 나오는 나머지가 모두 다름

- 귀류법으로, 두 수 $ia, ja(0 < i < j < n)$가 존재해서 그 나머지가 같다고 할 경우, 그 두 수의 차 $(j-i)a$는 n로 나누어짐
- $0 < j - i < n$이므로 $j - i$는 n의 배수가 아니며, 문제의 가정에 따라 a는 n과 서로소임
- 같은 나머지를 가지는 수가 없으므로, $n - 1$개의 수는 모두 나머지가 다름

소수 (21/28)

- 오일러 정리(Euler's Theorem) (3/6)
  - 버전 1 증명 (2/2)
    - AS의 원소들을 n으로 나눈 나머지는 S의 원소들과 대응됨

$\gcd(a, n) = 1$일 경우, a^{-1}가 존재하므로 AS와 S는 일대일대응임

    - 집합 S와 집합 AS는 원소의 값과 수가 같으므로 다음과 같음

$$as_1 \times as_2 \times \cdots \times as_{\varphi(n)} \equiv s_1 \times s_1 \times \cdots \times s_{\varphi(n)} \ (mod\ n)$$

    - 양변을 S의 원소들을 나눔

$$a^{\varphi(n)} \equiv 1\ mod\ n$$

소수 (22/28)

- 오일러 정리(Euler's Theorem) (4/6)
  - 버전 2 증명 (1/2)
    - $a < n$이기 때문에 세 가지 경우가 생김
      - 가정 1
        - 만약 a가 p의 배수도 아니고 q의 배수도 아님

$$\gcd(a, n) = 1$$

증명

$$a^{k \times \varphi(n)+1} (mod\ n) = (a^{\varphi(n)})^k \times a\ mod\ n = 1^k \times a\ mod\ n = a\ mod\ n$$

소수 (23/28)

- 오일러 정리(Euler's Theorem) (5/6)
  - 두 번째 버전 증명 (2/2)
    - 가정 2
      - 만약 a가 p의 배수가 되어 $a = i \times p$이지만 q의 배수는 아니라고 함

증명

$$a^{\varphi(n)} \bmod q = (a^{\varphi(q)} \bmod q)^{\varphi(p)} \bmod q = 1 \rightarrow a^{\varphi(n)} \bmod q = 1$$
$$a^{k \times \varphi(n)} \bmod q = (a^{\varphi(n)} \bmod q)^{k} \bmod q = 1 \rightarrow a^{k \times \varphi(n)} \bmod q = 1$$
$$a^{k \times \varphi(n)} \bmod q = 1 \rightarrow a^{k \times \varphi(n)} = 1 + j \times q$$
$$a^{k \times \varphi(n)+1} = a \times (1 + j \times q) = a + j \times q \times a = a + (i \times j) \times q \times p = a + (i \times j) \times n$$
$$a^{k \times \varphi(n)+1} = a + (i \times j) \times n \rightarrow a^{k \times \varphi(n)+1} = a \bmod n$$

    - 가정 3
      - 만약 a가 q의 배수로서 $a = i \times q$이지만 p의 배수는 아니라고 함
        - 증명은 2의 경우와 동일함 (위 증명에서 p와 q의 역할만 바꿈)

소수 (24/28)

- 오일러 정리(Euler's Theorem) (6/6)
  - 곱에 대한 역원
    - 오일러 정리를 사용하면 소수 모듈로를 갖는 경우에 곱에 대한 역원을 구할 수 있음
    - 오일러 정리를 사용하면 합성수를 모듈로로 갖는 경우에도 곱에 대한 역원을 구할 수 있음
      - 가정
        - 만약 n과 a가 서로소임

$$a^{-1} \bmod n = a^{\varphi(n)-1} \bmod n$$

증명

$$a \times a^{-1} \bmod n = a \times a^{\varphi(n)-1} \bmod n = a^{\varphi(n)} \bmod n = 1 \bmod n$$

소수 (25/28)

- 소수 생성 (1/4)
  - 메르센 소수(Mersenne Prime) (1/2)
    - 개요
      - 1644년, 메르센이 $2^n - 1$ 형태가 소수가 되는 것은 $n \leq 257$일 때 $n = 2, 3, 5, 7, 13, 17, 19, 31, 67, 127, 257$뿐이라고 발표함
      - M_{61}, M_{89}, M_{107}는 소수이며, 목록에 포함되어 있는 M_{67}, M_{257}은 합성수인 게 밝혀져 이 주장의 일부가 잘못됨이 알려짐
      - 지속적으로 메르센 소수의 발견과 검증을 진행하고 있음
    - 정의
      - 메르센 수 중에서 소수인 수
        - 메르센 수(Mersenne number): $M_n = 2^n - 1$
    - 특징
      - $M(n)$이 메르센 소수이면, n도 소수임

소수 (26/28)

- 소수 생성 (2/4)
  - 메르센 소수(Mersenne Prime) (2/2)
    - e.g.,

$$M_2 = 2^2 - 1 = 3$$
$$M_3 = 2^3 - 1 = 7$$
$$M_5 = 2^5 - 1 = 31$$
$$M_7 = 2^7 - 1 = 127$$
$$M_{11} = 2^{11} - 1 = 2047 \text{ (이 수는 소수가 아님. } 2047 = 23 \times 89)$$
$$M_{13} = 2^{13} - 1 = 8191$$
$$M_{17} = 2^{17} - 1 = 131071$$

소수 (27/28)

- 소수 생성 (3/4)
  - 페르마 소수(Fermat Prime) (1/2)
    - 개요
      - 1637년, 페르마는 $F_n = 2^{2n} + 1 (0 \leq n \leq 4)$ 형식인 수들을 소수라고 추측함
      - 오일러가 $n = 5$일 때 소인수분해 결과를 내놓으면서 반증함
      - F_0, F_1, F_2, F_3, F_4 외에는 더 이상 페르마 소수가 없을 것이라고 예측하는 수학자가 증가하고 있음
    - 정의
      - 페르마 수 중 소수인 수
        - 페르마 수(Fermat Number): $F_n = 2^{2n} + 1$
    - 특징
      - 페르마 소수의 존재에 대해서 아직 완전한 해답이 나오지 않음

소수 (28/28)

- 소수 생성 (4/4)
  - 페르마 소수(Fermat Prime) (2/2)
    - e.g.,

$$F_0 = 2^{2\times 0} + 1 = 3$$
$$F_1 = 2^{2\times 1} + 1 = 5$$
$$F_2 = 2^{2\times 2} + 1 = 17$$
$$F_3 = 2^{2\times 3} + 1 = 257$$
$$F_4 = 2^{2\times 4} + 1 = 65537$$
$$F_5 = 2^{2\times 5} + 1 = 4294967297 = 641 \times 6700417$$ (이 수는 소수가 아님)

목 차

- 소수
- 소수 판정
- 지수와 로그

소수 판정 (1/25)

- 개요
  - 매우 큰 정수가 주어졌을 때 이 수가 소수인지 합성수인지 정확하고 효율적인 판단 방법
    - 알고리즘 종류
      - 결정적 알고리즘과 확률적 알고리즘
    - 결정적 알고리즘이 이상적이긴 하지만 일반적으로 확률적 알고리즘에 비해서 실행 시간이 오래 걸려 효율성이 떨어짐

소수 판정 (2/25)

- 결정적 알고리즘(Deterministic Algorithm) (1/6)
  - 정의
    - 정확한 답을 제시하는 알고리즘
      - 하나의 정수를 입력하면 그 수가 소수인지 합성수인지 분명하게 알려줌
  - 나눔 알고리즘 (1/4)
    - 정의
      - $\sqrt{n}$보다 작은 모든 수를 나누는 수로 사용하는 알고리즘
        - 이 수들 중에서 한 개의 수라도 n을 나누는 경우, n은 합성수
    - 특징
      - 결정적 알고리즘의 검증 방법 중 가장 기초적인 나눔 검증을 사용함

소수 판정 (3/25)

- 결정적 알고리즘(Deterministic Algorithm) (2/6)
  - 나눔 알고리즘 (2/4)
    - 주요 동작 과정

```
Divisibility_Test(n)   // n에 대해서 소수 검증을 함
{
   r ← 2   // r은 나누는 수
   while(r < √n)
   {
   if(r|n) return "a composite"
   r ← r + 1
   }
   return "a prime"
}
```

    - 한계점
      - 매우 초보적이고 큰 수를 입력하면 시간이 오래 걸리기에 비효율적인 형태임

소수 판정 (4/25)

- 결정적 알고리즘(Deterministic Algorithm) (3/6)
  - 나눔 알고리즘 (3/4)
    - 비트-연산 복잡도는 $f(n_b) = \sqrt{2^{n_b}} = 2^{n_b/2}$
      - 여기서 n_b는 n을 2진수로 변경했을 때의 비트 수임
      - Big-O 표기법을 사용하여 표현하면 시간 복잡도는 $O(2^{n_b})$임
        - 나눔 알고리즘은 n_b가 매우 커지는 경우, 별로 실용적이지 못함

◈ 비트-연산 복잡도: 알고리즘이 실행되는 동안 수행되는 비트 연산의 총 횟수를 나타내는 척도

나눔 검증법의 비트-연산 복잡도는 지수적임

소수 판정 (5/25)

- 결정적 알고리즘(Deterministic Algorithm) (4/6)
  - 나눔 알고리즘 (4/4)
    - 예제 9.18

n의 비트 수가 200이라고 가정하자. 나눔 알고리즘을 수행한다고 한다면 비트 연산 회수는 얼마나 되는가?

- 풀이
  - 이 알고리즘의 비트-연산 복잡도는 $2^{n_b/2}$으로 2^{100}번의 비트 연산을 수행하는 경우, 이 수가 소수인지 아닌지를 알아낼 수 있음
  - 초당 2^{30}번의 비트 연산을 수행할 수 있는 컴퓨터를 사용하는 경우, 이 알고리즘을 이용하여 2^{70}초 만에 소수 검증이 가능함
  - $\therefore 2^{70}$초는 영원한 시간으로 나눔 알고리즘은 비효율적임

소수 판정 (6/25)

- 결정적 알고리즘(Deterministic Algorithm) (5/6)
  - AKS 알고리즘(Agrawal-Kayal-Saxena algorithm) (1/2)
    - 정의
      - Agrawal, Kayal, Saxena가 발표한 것으로 다항 시간 내에 소수 판별을 수행할 수 있는 최초의 알고리즘

◈ 다항 시간: 어떠한 문제를 계산하는 데에 걸리는 시간 $m(n)$이 문제의 크기 n의 다항식 함수보다 크지 않은 것, 다룰 수 있는 시간 내에 해결 가능

    - 특징
      - 원래 버전은 복잡도 $O((\log_2 n_b)^{12})$인 다항식 비트-연산 시간을 가짐
      - 후에 복잡도 $O((\log_2 n_b)^6)$을 가지는 알고리즘으로 개선됨
    - 동작 과정
      - 정수 $n(\geq 2)$은 n과 서로소인 모든 정수 a에 대해 $(x-a)^n \equiv (x^n - a) \bmod n$이 성립할 때만 소수이며, 그렇지 않으면 합성수라는 결과값을 나타냄
    - 한계점
      - 이론적으로는 다항식 시간 복잡도를 가지지만, 다른 알고리즘들보다 느리기 때문에 큰 수를 판정할 때는 많이 사용되지 않음

소수 판정 (7/25)

- 결정적 알고리즘(Deterministic Algorithm) (6/6)
  - AKS 알고리즘(Agrawal-Kayal-Saxena algorithm) (2/2)
    - 예제 9.19

n의 비트 수가 200이라고 가정하자. AKS 알고리즘을 수행한다고 한다면 비트 연산 회수는 얼마나 되는가?

      - 풀이
        - 이 알고리즘의 비트-연산 복잡도는 $(\log_2 n_b)^{12}$으로 $(\log_2 200)^{12} = 39{,}547{,}615{,}483$번의 비트 연산을 수행하는 경우, 이 수가 소수인지 아닌지를 알아낼 수 있음
        - 초당 10억 번의 비트 연산을 수행할 수 있는 컴퓨터를 사용하는 경우, 이 알고리즘을 이용하여 40초 만에 소수 검증이 가능함

소수 판정 (8/25)

- 확률적 알고리즘(Probabilistic Algorithm) (1/18)
  - 정의
    - 거의 정확하다는 판단을 내리지만 항상 맞는 것은 아닌 알고리즘
      - 확률적 알고리즘으로 만들어진 소수는 반드시 소수라는 보장이 없음
  - 알고리즘 규칙
    - 만약 검증하려는 정수가 정말로 소수인 경우, 이 알고리즘은 반드시 소수라고 판정함
    - 만약 검증하려는 정수가 정말로 합성수인 경우, 이 알고리즘은 이 수를 합성수라고 판정할 확률이 $1-\varepsilon$이고, 소수라고 판정할 확률이 ε이 됨

소수 판정 (9/25)

- 확률적 알고리즘(Probabilistic Algorithm) (2/18)
  - 특징
    - 이 알고리즘을 m번 수행할 경우, 정확하지 않은 판단을 할 확률은 ε^m으로 줄어듦
      - 이 알고리즘을 다른 매개변수를 가지고 한 번 이상 수행하거나 다른 방법을 사용하는 경우, 합성수를 소수라고 판정할 확률을 매우 작게 줄일 수 있음

소수 판정 (10/25)

- 확률적 알고리즘(Probabilistic Algorithm) (3/18)
  - 페르마 검증 (1/2)

만약 n이 소수라면, $a^{n-1} \equiv 1 \, mod \, n$임
만약 n이 합성수라면, $a^{n-1} \equiv 1 \, mod \, n$일 수도 있음

    - 소수는 페르마 검증을 해보면 소수로 판정되지만 합성수가 소수라고 판정 날 확률이 ε임
    - 여러 가지의 밑수(a_1, a_2, a_3, etc.)를 사용해서 검증할 경우, 이 확률 값을 더 크게 개선할 수 있음

소수 판정 (11/25)

- 확률적 알고리즘(Probabilistic Algorithm) (4/18)
  - 페르마 검증 (2/2)
    - 예제 9.20

수 561을 페르마 검증하면 어떻게 되는가?

- 풀이
  - 2을 밑수로 사용할 경우, $2^{561-1} \equiv 1 \; mod \; 561$을 만족하므로 페르마 검증을 통과하지만 소수가 아님 $(561 = 33 \times 17)$

소수 판정 (12/25)

- 확률적 알고리즘(Probabilistic Algorithm) (5/18)
  - 제곱근 검증 (1/4)

만약 n이 소수라면, $\sqrt{1}\ mod\ n = \pm 1$임
만약 n이 합성수라면, $\sqrt{1}\ mod\ n = \pm 1$이며 다른 값들을 가질 수도 있음

    - 모듈로 연산에서 -1이 의미하는 것은 $n-1$이라는 것에 유의
      - e.g., $-1 \equiv 6\ mod\ 7$

소수 판정 (13/25)

- 확률적 알고리즘(Probabilistic Algorithm) (6/18)
  - 제곱근 검증 (2/4)
    - 예제 9.21

n이 7(소수)이라면 모듈로 n으로 1의 제곱근은 무엇인가?

- 풀이

$$1^2 = 1 \bmod 7 \qquad (-1)^2 = 1 \bmod 7$$
$$2^2 = 4 \bmod 7 \qquad (-2)^2 = 4 \bmod 7$$
$$3^2 = 2 \bmod 7 \qquad (-3)^2 = 2 \bmod 7$$

- 제곱근은 1과 -1(이것은 6)임
- 여기서 $4^2 = 2 \bmod 7$, $5^2 = 4 \bmod 7$, $6^2 = 1 \bmod 7$이므로 $4, 5, 6$에 대해서는 점검할 필요가 없음

소수 판정 (14/25)

- 확률적 알고리즘(Probabilistic Algorithm) (7/18)
  - 제곱근 검증 (3/4)
    - 예제 9.22

n이 8이라면 모듈로 n으로 1의 제곱근은 무엇인가?

- 풀이

$$1^2 = 1 \bmod 8 \qquad (-1)^2 = 1 \bmod 8$$
$$3^2 = 1 \bmod 8 \qquad 5^2 = 1 \bmod 8$$

- 근을 4개 가짐 그 근은 $1, 3, 5, 7$(이것은 -1임)

소수 판정 (15/25)

- 확률적 알고리즘(Probabilistic Algorithm) (8/18)
  - 제곱근 검증 (4/4)
    - 예제 9.23

n이 17이라면 모듈로 n으로 1의 제곱근은 무엇인가?

- 풀이

$$
\begin{aligned}
1^2 &= 1 \bmod 17 & (-1)^2 &= 1 \bmod 17 \\
2^2 &= 4 \bmod 17 & (-2)^2 &= 4 \bmod 17 \\
3^2 &= 9 \bmod 17 & (-3)^2 &= 9 \bmod 17 \\
4^2 &= 16 \bmod 17 & (-4)^2 &= 16 \bmod 17 \\
5^2 &= 8 \bmod 17 & (-5)^2 &= 8 \bmod 17 \\
6^2 &= 2 \bmod 17 & (-6)^2 &= 2 \bmod 17 \\
7^2 &= 15 \bmod 17 & (-7)^2 &= 15 \bmod 17 \\
8^2 &= 13 \bmod 17 & (-8)^2 &= 13 \bmod 17
\end{aligned}
$$

- 오직 2개의 근을 가짐 그 근은 1과 -1(이것은 16임)임
- $9^2 = 13 \bmod 17, \cdots$ 등이기에 8보다 큰 정수에 대해서는 점검할 필요가 없음

소수 판정 (16/25)

- 확률적 알고리즘(Probabilistic Algorithm) (9/18)
  - 밀러-라빈 검증(Miller-Rabin Primality Test) (1/8)
    - 정의

◈ 강 의사소수: 소수일 확률이 매우 큰 수

      - 페르마 검증과 제곱근 검증 방법을 혼합해서 강 의사소수(strong pseudoprime)를 구하는 알고리즘
    - 특징
      - $n-1$을 $m \times 2^k$ (m은 홀수)로 표현함
      - 밑수 a를 사용한 페르마 검증은 다음과 같이 나타냄

$$a^{n-1} = a^{m \times 2^k} = (a^m)^{2^k} = (a^m)^{2^{2^{\cdot^{\cdot^{2}}}}} \quad (k \text{ times})$$

        - $a^{n-1} \bmod n$을 계산할 때 1단계로 하는 대신 $k+1$ 단계로 할 수 있음
          - 각 단계에서 제곱근 검증을 할 수 있어서 유리함
      - 만약 어떤 수가 m번의 검증(m개의 서로 다른 밑수로)을 통과하였다면, 그 수를 소수로 잘못 판별할 확률은 $(¼)^m$임

소수 판정 (17/25)

- 확률적 알고리즘(Probabilistic Algorithm) (10/18)
  - 밀러-라빈 검증(Miller-Rabin Primality Test) (2/8)
    - 초기화 단계
      - 밑수 a를 선택하고 $T = a^m \ mod \ n$을 구함 (여기서 $m = (n-1)/2^k$)
        - 만약 T가 $+1$이거나 -1이라면 n은 강 의사소수라고 선언하고 이곳에서 검증을 끝냄
          - 페르마 검증과 제곱근 검증 모두 통과하였기 때문임
        - 만약 T가 그 이외의 값을 가지는 경우, n이 소수인지 합성수인지 판단할 수 없기에 다음 단계로 넘어가게 됨

소수 판정 (18/25)

- 확률적 알고리즘(Probabilistic Algorithm) (11/18)
  - 밀러-라빈 검증(Miller-Rabin Primality Test) (3/8)
    - T를 제곱하는 단계 (1단계)
      - 만약 제곱한 결과가 $+1$이라면, n을 합성수로 판단함
        - 페르마 검증은 통과했으나 제곱근 검증은 통과하지 못했기 때문임
      - 만약 제곱한 결과가 -1이 된다면, n을 강 의사소수로 판단함
        - 페르마 검증과 제곱근 검증 모두 통과하였기 때문임
      - 만약 제곱한 결과가 ± 1이 아닌 다른 값을 갖는 경우, n이 소수인지 아닌지 알 수 없으므로 다음 단계를 진행함

소수 판정 (19/25)

- 확률적 알고리즘(Probabilistic Algorithm) (12/18)
  - 밀러-라빈 검증(Miller-Rabin Primality Test) (4/8)
    - 2단계에서 $k-1$단계
      - 해당 단계와 $k-1$단계까지의 모든 단계는 단계 1과 동일함
    - k단계
      - 만약 해당 단계까지 왔는데도 결론에 이르지 못한 경우, 해당 단계는 더 이상 의미가 없음
      - 만약 해당 단계의 결과가 1인 경우, 페르마 검증은 통과하나 앞 단계의 결과가 ± 1이 아니었기 때문에 제곱근 검증은 통과하지 못함
        - $k-1$번째 단계 후 절차가 끝나지 않았다면 n이 합성수라고 선언함

소수 판정 (20/25)

- 확률적 알고리즘(Probabilistic Algorithm) (13/18)
  - 밀러-라빈 검증(Miller-Rabin Primality Test) (5/8)
    - 의사코드

```
Miller_Rabin_Test(n, a)   // n은 수이고, a는 밑수임
{
    Find m and k such that n − 1 ≡ m × 2^k
    T ← a^m mod n
    if(T = ±1) return "a prime"
      for(i ← 1 to k − 1)   // k − 1은 최대 단계 수임
    {
       T ← T^2 mod n
       if(T = +1) return "a composite"
       if(T = −1) return "a prime"
    }
    return "a composite"
}
```

소수 판정 (21/25)

- 확률적 알고리즘(Probabilistic Algorithm) (14/18)
  - 밀러-라빈 검증(Miller-Rabin Primality Test) (7/8)
    - 예제 9.25

수 561은 밀러-라빈 검증을 통과하는가?

- 풀이
  - 밑수 a를 2로 사용할 경우, $561 - 1 = 35 \times 2^4$로 표현되어 $m = 35, k = 4, a = 2$

초기화: $T = 2^{35} \; mod \; 561 = 263 \; mod \; 561$

$k = 1$: $T = 263^2 \; mod \; 561 = 166 \; mod \; 561$

$k = 2$: $T = 166^2 \; mod \; 561 = 67 \; mod \; 561$

$k = 3$: $T = 67^2 \; mod \; 561 = +1 \; mod \; 561 \rightarrow$ 합성수

소수 판정 (22/25)

- 확률적 알고리즘(Probabilistic Algorithm) (15/18)
  - 밀러-라빈 검증(Miller-Rabin Primality Test) (8/8)
    - 예제 9.26

> 27은 소수가 아닌 것을 알고 있다. 밀러-라빈 검증을 하시오.

      - 풀이
        - 밑수가 2이면 $27 - 1 = 13 \times 2^1$이므로 $m = 13, k = 1, a = 2$
        - $T = 2^{13} \ mod \ 27 = 11 \ mod \ 27$

소수 판정 (23/25)

- 확률적 알고리즘(Probabilistic Algorithm) (16/18)
  - 추천하는 소수 검증 (1/3)
    - 현재 가장 많이 사용되는 소수 검증 방법은 나눔 검증과 밀러-라빈 검증을 조합한 것임
      1. 홀수 정수를 하나 선택함
      2. 알려진 소수에 대해서 나눔 검증을 하여 합성수를 가지고 검증을 하려고 하는 것은 아닌지 확인함
      3. 검증을 하기 위해서 밑수로 사용할 수들의 집합을 결정함
      4. 각 밑수로 밀러-라빈 검증을 실시함
      5. 한 단계라도 통과하지 못했다면 1단계로 다시 돌아감

소수 판정 (24/25)

- 확률적 알고리즘(Probabilistic Algorithm) (17/18)
  - 추천하는 소수 검증 (2/3)
    - 예제 9.28 (1/2)

4033은 37×109이므로 합성수이다.
이 수는 위에 언급한 추천하는 소수 검증을 통과하는가?

- 풀이
  1. 나눔 검증을 실시함
     - $2, 3, 5, 7, 11, 17, 23$은 4033의 약수가 아님
  2. 밑수를 2로 사용하여 밀러-라빈 검증을 실시함
     - $4033 - 1 = 63 \times 2^6$이므로 $m = 63,\ k = 6$임

초기화: $T = 2^{63}\ mod\ 4033 = 3521\ mod\ 4033$
$k = 1$: $T = 3521^2\ mod\ 4033 = -1\ mod\ 4033 \rightarrow$ 통과

소수 판정 (25/25)

- 확률적 알고리즘(Probabilistic Algorithm) (18/18)
  - 추천하는 소수 검증 (3/3)
    - 예제 9.28 (1/2)
      - 풀이

3. 만족스럽지 못하므로 밑수로 3을 사용함

초기화: $T = 3^{63} \ mod \ 4033 = 3551 \ mod \ 4033$

$k = 1$: $T = 3551^2 \ mod \ 4033 = 2443 \ mod \ 4033$

$k = 2$: $T = 2443^2 \ mod \ 4033 \equiv 3442 \ mod \ 4033$

$k = 3$: $T = 3442^2 \ mod \ 4033 \equiv 2443 \ mod \ 4033$

$k = 4$: $T = 2443^2 \ (mod \ 4033) \equiv 3442 \ mod \ 4033$

$k = 5$: $T = 3442^2 \ mod \ 4033 \equiv 2443 \ mod \ 4033 \rightarrow$ 실패(합성수)

목 차

- 소수
- 소수 판정
- 지수와 로그

지수와 로그 (1/31)

- 지수(Exponentiation) (1/8)
  - 정의

$$y = a^x$$

    - x를 지수라고 부르고 a를 밑이라고 부름
    - 밑수를 몇 번 곱해야 하는지 나타냄
  - 특징
    - 지수와 로그는 역관계의 연산임

지수와 로그 (2/31)

- 지수(Exponentiation) (2/8)
  - 고속 지수계산 (1/7)
    - 제곱-곱 방법(Square-and-Multiply Method)을 사용
      - 제곱과 곱셈을 모두 사용함
    - 이 방법의 주요한 아이디어는 지수를 n_b 비트(x_0에서 x_{n_b-1})를 갖는 2진수로 간주하는 것임

$$x = x_{n_b-1} \times 2^{n_b-1} + x_{n_b-2} \times 2^{n_b-2} + \cdots + x_2 \times 2^2 + x_1 \times 2^1 + x_0 \times 2^0$$

      - e.g., $x = 22 = (10110)_2$일 때 $22 = 2^4 \times 1 + 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 0$로 표현 가능함

지수와 로그 (3/31)

- 지수(Exponentiation) (3/8)
  - 고속 지수계산 (2/7)

$$x_{n_b-1} \times 2^{n_b-1} + x_{n_b-2} \times 2^{n_b-2} + \cdots + x_2 \times 2^2 + x_1 \times 2^1 + x_0 \times 2^0$$

$y = a$

in which x_i is 0 or 1

$$y = \left(a^{2n_b-1} \text{ or } 1\right) \times \left(a^{2n_b-2} \text{ or } 1\right) \times \cdots \times \left(a^{2} \text{ or } 1\right) \times \left(a^{1} \text{ or } 1\right)$$

Example:

$$y = a^9 = a^{1001_2} = a^8 \times 1 \times 1 \times a$$

지수와 로그 (4/31)

- 지수(Exponentiation) (4/8)
  - 고속 지수계산 (3/7)
    - 그림에서 나타낸 것처럼 $y = a^x$로 표현할 수 있음
      - y가 n_b개 항의 곱으로 표시되는 점에 유의
    - 비트가 1인 경우에 항 a^{2^i}는 곱에 포함되고 비트가 0인 경우, 곱에 포함되지 않음
    - 밑수를 연속해서 제곱하여 $a, a^2, a^4, \cdots, a^{2n_b-1}$을 구할 수 있음

지수와 로그 (5/31)

- 지수(Exponentiation) (5/8)
  - 고속 지수계산 (4/7)
    - 의사코드

```
Square_and_Multiply(a, x, n)
{
   y ← 1
   for(i ← 0 to n_b − 1) // n_b는 x의 비트 수임
   {
      if(x_i = 1)   y ← a × y mod n  // 비트가 1인 경우만 곱함
      a ← a^2 mod n  // 마지막 반복에서는 제곱이 필요 없음
   }
   return y
}
```

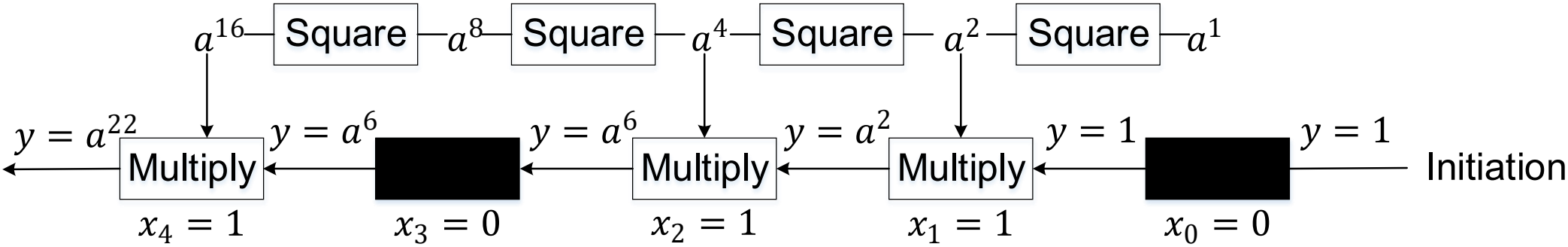
지수와 로그 (6/31)

- 지수(Exponentiation) (6/8)
  - 고속 지수계산 (5/7)
    - 동작 방식 (n_b번 반복됨)
      1. 각각의 반복에서 해당 비트 값을 점검하여 그 값이 1이면 결과의 이전 값을 현재의 밑수에 곱함
      2. 이어지는 반복에 사용하기 위해 밑수를 제곱함
      3. n_b번 반복함
      4. 마지막 반복에서는 제곱이 필요 없음

지수와 로그 (7/31)

- 지수(Exponentiation) (7/8)
  - 고속 지수계산 (6/7)
    - 예제 9.45

그림은 알고리즘을 이용하여 $y = a^x$를 구하는 절차를 보여주고 있다. (그림을 간단히 하기 위해서 모듈로 값은 나타내지 않았다.) 이 경우에 2진수로 표시하면 $x = 22 = (10110)_2$이다. 지수는 5비트이다.



- $y = a^{22}$까지 진행되는 과정을 보여줌
- 검은 색으로 된 사각형이 의미하는 것은 곱셈이 생략된 곳이고 이전 단계의 y값이 다음 단계로 그대로 넘겨지는 것을 알 수 있음

지수와 로그 (8/31)

- 지수(Exponentiation) (8/8)
  - 고속 지수계산 (7/7)
    - 복잡도
      - 알고리즘에서는 n_b가 모듈로 값의 비트 수($n_b = \log_2 n$)일 경우, 최대 $2n_b$번의 연산을 사용함
      - 이 알고리즘의 비트연산 복잡도는 다항식 수준인 $O(n_b)$임
        - n_b번의 반복이 있기 때문

지수와 로그 (9/31)

- 로그(Logarithm) (1/23)
  - 정의

$$x = \log_a y$$

    - 지수를 다른 방법으로 표현한 것
    - $a > 0$, $a \neq 1$, $y > 0$

  - 성질
    - $\log_a 1 = 0,\ \log_a a = 1$
    - $\log_a MN = \log_a M + \log_a N$
    - $\log_a \frac{M}{N} = \log_a M - \log_a N$
    - $\log_a L^k = k \log_a L$

지수와 로그 (10/31)

- 로그(Logarithm) (2/23)
  - 특징
    - 지수와 로그는 역관계의 연산임
  - 필요성
    - 암호화 및 복호화 과정에서 지수계산을 한 경우, 지수계산의 역을 계산하는 것이 어려운 문제라는 것을 이해하는데 필요한 정보를 제공함
      - 이를 통해 암호 시스템의 안전성을 확보할 수 있음

지수와 로그 (11/31)

- 로그(Logarithm) (3/23)
  - 전부 찾기
    - $y = a^x \ mod \ n$의 역을 계산하기 위한 방법은 $x = \log_a y \ mod \ n$을 푸는 것임
      - $y = a^x \ mod \ n$을 연속적으로 계산하여 주어진 y 값을 찾을 때까지 수행

```
Modular_Logarithm (a, y, n)
{
    for (x = 1 to n − 1)
    {
      if(y ≡ a^x mod n) return x
    }
    return failure
}
```

      - 이 알고리즘은 비효율적이고 비트연산 복잡도는 지수적인 $O(2^{n_b})$임

지수와 로그 (12/31)

- 로그(Logarithm) (4/23)
  - 이산 대수를 이해하기 위한 성질 (1/15)
    - 유한 곱셈군(Multiplicative Finite Group)
      - 군 $G =< Z_n^*, \times >$에서 Z_n^*는 n과 서로소가 되는 1부터 $n-1$ 안의 정수들을 원소로 갖는 집합이고 이 집합의 항등원 e는 1임
      - 모듈로 값 n이 소수 p일 때는 $G =< Z_p^*, \times >$가 됨
    - 군의 크기
      - 유한군 G의 크기는 그 군에 속하는 원소의 개수이고 $|G|$로 나타냄
      - $G =< Z_n^*, \times >$의 크기는 $\varphi(n)$임
      - 예제 9.46

        군 $G =< Z_{21}^*, \times >$의 크기는 얼마인가?

        - 풀이
          - $|G| = \varphi(21) = \varphi(3) \times \varphi(7) = 2 \times 6 = 12$임
          - 이 군은 12개의 원소인 $1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20$으로 이루어져 있음

지수와 로그 (13/31)

- 로그(Logarithm) (5/23)
  - 이산 대수를 이해하기 위한 성질 (2/15)
    - 원소의 위수 (1/2)
      - $G =< Z_n^*, \times >$에서 원소 a의 위수는 $a^i \equiv e \ (mod\ n)$이 되는 정수 중에서 가장 작은 정수 i로 정의됨
        - 여기서 항등원 e는 1임
      - $ord(a)$라고 표기함

지수와 로그 (14/31)

- 로그(Logarithm) (6/23)
  - 이산 대수를 이해하기 위한 성질 (3/15)
    - 원소의 위수 (2/2)
      - 예제 9.47

군 $G =< Z_{10}^{*}, \times >$의 모든 원소에 대한 위수를 계산하시오.

- 풀이
  - 이 군은 $\varphi(10) = 4$개의 원소 $1, 3, 7, 9$를 가짐
  - 각 원소의 위수를 구하기 위해 시행착오 방법을 사용하지만 라그랑지 정리를 이용할 경우, 군에 속하는 원소의 위수는 군의 크기를 나누어야 함
  - 4의 약수는 $1, 2, 4$로 각 원소의 위수를 계산하기 위해 이 3가지 수에 대해서만 거듭제곱수를 계산함

a. $1^1 \equiv 1 \, mod \, (10) \; ; \rightarrow ord(1) = 1$

b. $3^1 \equiv 3 \, mod \, (10) \; ; 3^2 \equiv 9 \, mod \, (10) \; ; 3^4 \equiv 1 \, mod \, (10) \rightarrow ord(3) = 4$

c. $7^1 \equiv 7 \, mod \, (10) \; ; \; 7^2 \equiv 9 \, mod \, (10) \; ; 7^4 \equiv 1 \, mod \, (10) \rightarrow ord(7) = 4$

d. $9^1 \equiv 9 \, mod \, (10) \; ; 9^2 \equiv 1 \, mod \, (10) \; ; \rightarrow ord(9) = 2$

지수와 로그 (15/31)

- 로그(Logarithm) (7/23)
  - 이산 대수를 이해하기 위한 성질 (4/15)
    - 오일러 정리 (1/3)
      - a가 군 $G =< Z_n^*, \times >$의 원소일 경우, $a^{\varphi(n)} \equiv 1 \, mod \, n$임
        - $i = \varphi(n)$일 경우, $a^i \equiv 1 \, (mod \, n)$이 성립됨
      - 예제 9.48

표에 군 $G =< Z_8^*, \times >$에 대한 $a^i \equiv 1 \, mod \, 8$의 값을 나타내었다.
$\varphi(8) = 4$이고 원소는 $1, 3, 5, 7$이라는 점에 유의하기 바란다.

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$
$a=1$	$\boldsymbol{x:1}$	$x:1$	$x:1$	$x:1$	$x:1$	$x:1$	$x:1$
$a=3$	$x:3$	$\boldsymbol{x:1}$	$x:3$	$x:1$	$x:3$	$x:1$	$x:3$
$a=5$	$x:5$	$\boldsymbol{x:1}$	$x:5$	$x:1$	$x:5$	$x:1$	$x:5$
$a=7$	$x:7$	$\boldsymbol{x:1}$	$x:7$	$x:1$	$x:7$	$x:1$	$x:7$

지수와 로그 (16/31)

- 로그(Logarithm) (8/23)
  - 이산 대수를 이해하기 위한 성질 (5/15)
    - 오일러 정리 (2/3)
      - 예제 9.48 (2/2)

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$
$a=1$	$\boldsymbol{x:1}$	$x:1$	$x:1$	$x:1$	$x:1$	$x:1$	$x:1$
$a=3$	$x:3$	$\boldsymbol{x:1}$	$x:3$	$x:1$	$x:3$	$x:1$	$x:3$
$a=5$	$x:5$	$\boldsymbol{x:1}$	$x:5$	$x:1$	$x:5$	$x:1$	$x:5$
$a=7$	$x:7$	$\boldsymbol{x:1}$	$x:7$	$x:1$	$x:7$	$x:1$	$x:7$

        - 풀이
          - 음영으로 처리된 부분은 오일러 정리를 적용한 결과임
            - $i=\varphi(8)=4$이고 모든 a에 대해서 그 결과는 $x=1$임
          - 최초로 x값이 1이 되었을 때 i의 값이 그 원소의 위수임
          - 각 원소의 위수는 $ord(1)=1$, $ord(3)=2$, $ord(5)=2$, $ord(7)=2$임

지수와 로그 (17/31)

- 로그(Logarithm) (9/23)
  - 이산 대수를 이해하기 위한 성질 (6/15)
    - 오일러 정리 (3/3)
      - 예제 9.48 (2/2)

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$
$a=1$	$\boldsymbol{x:1}$	$x:1$	$x:1$	$x:1$	$x:1$	$x:1$	$x:1$
$a=3$	$x:3$	$\boldsymbol{x:1}$	$x:3$	$x:1$	$x:3$	$x:1$	$x:3$
$a=5$	$x:5$	$\boldsymbol{x:1}$	$x:5$	$x:1$	$x:5$	$x:1$	$x:5$
$a=7$	$x:7$	$\boldsymbol{x:1}$	$x:7$	$x:1$	$x:7$	$x:1$	$x:7$

- 최초로 x값이 1이 되었을 때 i의 값이 그 원소의 위수임
- 각 원소의 위수는 $ord(1)=1, ord(3)=2, ord(5)=2, ord(7)=2$임

지수와 로그 (18/31)

- 로그(Logarithm) (10/23)
  - 이산 대수를 이해하기 위한 성질 (7/15)
    - 원시근(Primitive Root) (1/7)
      - 정의
        - $G =< Z_n^*, \times >$에서 한 원소의 위수가 $\varphi(n)$과 동일한 경우, 그 원소를 해당 군의 원시근이라고 함
      - 특징
        - 군 $G =< Z_n^*, \times >$는 $n = 2, 4, p^t, 2p^t$일 경우, 원시근을 가짐 (p는 2가 아닌 소수, t는 양의 정수)
        - 한 군이 한 개의 원시근을 가지는 경우, 일반적으로 그 군에는 또 다른 여러 개의 원시근이 존재함
        - $G =< Z_n^*, \times >$이 원시근을 갖는 경우, 원시근의 개수는 $\varphi(\varphi(n))$개임

지수와 로그 (19/31)

- 로그(Logarithm) (11/23)
  - 이산 대수를 이해하기 위한 성질 (8/15)
    - 원시근(Primitive Root) (2/7)
      - 예제 9.49

이 군 $G =< Z_8^*, \times >$ 에 원시근을 구하시오.

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$
$a=1$	$\boldsymbol{x:1}$	$x:1$	$x:1$	$x:1$	$x:1$	$x:1$	$x:1$
$a=3$	$x:3$	$\boldsymbol{x:1}$	$x:3$	$x:1$	$x:3$	$x:1$	$x:3$
$a=5$	$x:5$	$\boldsymbol{x:1}$	$x:5$	$x:1$	$x:5$	$x:1$	$x:5$
$a=7$	$x:7$	$\boldsymbol{x:1}$	$x:7$	$x:1$	$x:7$	$x:1$	$x:7$

- 어떤 원소도 위수가 $\varphi(8)=4$가 되지 않고 위수가 모두 4보다 작으므로 $G =< Z_8^*, \times >$ 는 원시근을 가지지 않음

지수와 로그 (20/31)

- 로그(Logarithm) (12/23)
  - 이산 대수를 이해하기 위한 성질 (9/15)
    - 원시근(Primitive Root) (3/7)
      - 예제 9.50 (1/2)

표를 살펴보면 군 $G =< Z_7^*, \times >$에 대한 $a^i \equiv x \; mod \; 7$의 값을 계산해 놓았다.
이 군에 원시근을 구하시오.

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
$a=1$	$\boldsymbol{x:1}$	$x:1$	$x:1$	$x:1$	$x:1$	$x:1$
$a=2$	$x:2$	$x:4$	$\boldsymbol{x:1}$	$x:2$	$x:4$	$x:1$
원시근 → $a=3$	$x:3$	$x:2$	$x:6$	$x:4$	$x:5$	$\boldsymbol{x:1}$
$a=4$	$x:4$	$x:2$	$\boldsymbol{x:1}$	$x:4$	$x:2$	$x:1$
원시근 → $a=5$	$x:5$	$x:4$	$x:6$	$x:2$	$x:3$	$\boldsymbol{x:1}$
$a=6$	$x:6$	$\boldsymbol{x:1}$	$x:6$	$x:1$	$x:6$	$x:1$

지수와 로그 (21/31)

- 로그(Logarithm) (13/23)
  - 이산 대수를 이해하기 위한 성질 (10/15)
    - 원시근(Primitive Root) (4/7)
      - 예제 9.50 (2/2)
        - 풀이
          - 이 군의 크기는 $\varphi(7) = 6$임
          - $ord(1) = 1, ord(2) = 3, ord(3) = 6, ord(4) = 3, ord(5) = 6, ord(6) = 1$
          - 두 원소 3과 5만이 $i = \varphi(n) = 6$에서 위수 6을 가짐
            - 이 군은 2개의 원시근 3과 5를 가짐

지수와 로그 (22/31)

- 로그(Logarithm) (14/23)
  - 이산 대수를 이해하기 위한 성질 (11/15)
    - 원시근(Primitive Root) (5/7)
      - 예제 9.51

$n = 17, 20, 38, 50$ 중 어떤 n에 대해서 군 $G =< Z_n^*, \times >$가 원시근을 갖는가?

- $G =< Z_{17}^*, \times >$은 원시근을 가짐
  - $17 = 17^1$이고 17은 소수이기 때문임 $(p = 17, t = 1)$
- $G =< Z_{20}^*, \times >$은 원시근을 가지지 않음
  - $20 = 2^2 \times 5^1$이기 때문임
- $G =< Z_{38}^*, \times >$은 원시근을 가짐
  - $38 = 2 \times 19^1$이고 19는 소수이기 때문임 $(p = 19, t = 1)$
- $G =< Z_{50}^*, \times >$은 원시근을 가짐
  - $50 = 2 \times 5^2$이고 5는 소수이기 때문임 $(p = 5, t = 2)$

지수와 로그 (23/31)

- 로그(Logarithm) (15/23)
  - 이산 대수를 이해하기 위한 성질 (12/15)
    - 원시근(Primitive Root) (6/7)
      - 원시근 판별 문제

        Q1) 주어진 원소 a와 $G =< Z_n^*, \times>$에 대해서 a가 G의 원시근인지 아닌지를 어떻게 알아낼 수 있는가?
        - $\varphi(n)$값을 알아내야 함 (n을 소인수분해 하는 것만큼 힘듦)
        - $ord(a) = \varphi(n)$인지 아닌지를 점검해야 함
      - 원시근 탐색 문제

        Q2) 주어진 군 $G =< Z_n^*, \times>$에 대해서 G의 모든 원시근을 알아낼 수 있을까?
        - 군에 속하는 각 원소에 대해서 $ord(a) = \varphi(n)$인지 아닌지 점검을 반복해야 하기에 앞의 질문보다 더 어려움

지수와 로그 (24/31)

- 로그(Logarithm) (16/23)
  - 이산 대수를 이해하기 위한 성질 (13/15)
    - 원시근(Primitive Root) (7/7)
      - 원시근 선택 문제

        Q3) 주어진 군 $G =< Z_n^*, \times >$에 대해서 G의 한 원시근을 어떻게 선택할 것인가?
        - 암호론에 있어서 응용을 하려면 주어진 군에서 적어도 한 개의 원시근을 찾아내야 함
          - 사용자는 여러 개의 원소들에 대해서 시도를 해 보아서 첫 번째 원소를 찾을 때까지 수행함

지수와 로그 (25/31)

- 로그(Logarithm) (17/23)
  - 이산 대수를 이해하기 위한 성질 (14/15)
    - 순환 군(Cyclic Group) (1/2)
      - 정의
        - 군 $G =< Z_n^*, \times>$가 원시근을 갖는 군
      - 특징
        - 순환 군의 생성자는 각 원시근이므로 군 전체 집합 Z_n^*을 $Z_n^* = \{q^1, q^2, q^3, \cdots, q^{\varphi(n)}\}$과 같이 생성해낼 수 있음
        - p가 소수이면 군 $G =< Z_p^*, \times>$ 항상 순환 군임

지수와 로그 (26/31)

- 로그(Logarithm) (18/23)
  - 이산 대수를 이해하기 위한 성질 (15/15)
    - 순환 군(Cyclic Group) (2/2)
      - 예제 9.52

군 $G =< Z_{10}^{*}, \times >$는 $\varphi(10) = 4$이고 $\varphi(\varphi(10)) = 2$이므로 두 개의 원시근을 갖는다. 이 두 개의 원시근은 3과 7이라는 것을 구해낼 수 있다. 이 두 개의 원시근이 각각 어떻게 전체 군의 집합을 생성해내는지 알아보자.

      - 풀이

$$q = 3 \rightarrow q^1 \, mod \, 10 = 3, \;\; q^2 \, mod \, 10 = 9, q^3 \, mod \, 10 = 7, q^4 \, mod \, 10 = 1$$
$$q = 7 \rightarrow q^1 \, mod \, 10 = 7, \;\; q^2 \, mod \, 10 = 9, q^3 \, mod \, 10 = 3, q^4 \, mod \, 10 = 1$$

지수와 로그 (27/31)

- 로그(Logarithm) (19/23)
  - 이산 대수의 아이디어
    - 군 $G =< Z_p^*, \times >$의 성질
      1. 이 군의 원소는 1부터 $p-1$까지의 모든 수임
      2. 이 군은 항상 원시근을 가짐
      3. 이 군은 순환 군이고 원소들은 q^x를 사용하여 생성해낼 수 있음
         - 여기서 x는 1부터 $\varphi(n) = p-1$ 안의 정수임
      4. 원시근은 로그의 밑수임
         - 이 군이 k개의 원시근을 가질 경우, k개의 서로 다른 밑수로 계산이 가능함
         - 집합의 한 원소 y에 대해 $x = \log_q y$ 일 경우, q를 밑수로 갖는 y의 로그인 새로운 원소 x가 존재함
           - 이와 같은 로그를 이산 대수라고 함
           - 이산 대수는 밑수 q를 나타내기 위해서 기호 L_q라고 표현함

지수와 로그 (28/31)

- 로그(Logarithm) (20/23)
  - 이산 대수를 이용한 모듈로 로그의 해 구하기 (1/2)
    - 이산 대수의 도표화
      - y가 주어진 경우, $y = a^x \ (mod\ n)$을 풀기 위해 서로 다른 밑수별로 Z_p^*에 대한 표를 사용함
        - e.g., Z_7^*에 대한 이산 대수 표

y	1	2	3	4	5	6
$x = L_3 y$	6	2	1	4	5	3
$x = L_5 y$	6	4	5	2	1	3

      - 모든 군과 모든 가능한 밑수별로 다른 이산 대수표가 주어지는 경우, 어떠한 이산 대수 문제라도 풀 수 있음
        - 전통적인 로그를 사용하는 것과 동일한 방법임
        - 계산기나 컴퓨터가 발명되기 이전에는 밑수가 10인 상용 대수표를 가지고 로그 값을 계산해옴

지수와 로그 (29/31)

- 로그(Logarithm) (21/23)
  - 이산 대수를 이용한 모듈로 로그의 해 구하기 (2/2)
    - 예제 9.53

다음 경우 각각에 대해 x를 구하시오.
a. $4 \equiv 3^x \ (mod\ 7)$. b. $6 \equiv 5^x \ (mod\ 7)$.

- 풀이

y	1	2	3	4	5	6
$x = L_3 y$	6	2	1	4	5	3
$x = L_5 y$	6	4	5	2	1	3

a. $4 \equiv 3^x \ (mod\ 7) \rightarrow x = L_3 4 \ mod\ 7 = 4 \ mod\ 7$
b. $6 \equiv 5^x \ (mod\ 7) \rightarrow x = L_5 6 \ mod\ 7 = 3 \ mod\ 7$

지수와 로그 (30/31)

- 로그(Logarithm) (22/23)
  - 이산 대수 성질
    - 이산 대수가 전통적인 로그와 같은 성질을 가짐
      - 모듈로 값으로 n 대신에 $\varphi(n)$을 사용함

$Traditional\ Logarithm$	$Discrete\ Logarithms$
$\log_a 1 = 0$	$L_q 1 \equiv 0\ (mod\ \varphi(n))$
$\log_a(x \times y) = \log_a x + \log_a y$	$L_q(x \times y) \equiv (L_q x + L_q y)(mod\ \varphi(n))$
$\log_a x^k = k \times \log_a x$	$L_q x^k \equiv k \times L_q x\ (mod\ \varphi(n))$

지수와 로그 (31/31)

- 로그(Logarithm) (23/23)
  - 이산 대수 기반의 알고리즘
    - n이 매우 커질 경우, 이산 대수 성질과 이산 대수표를 가지고 $y = a^x \ (mod\ n)$을 푸는 것은 불가능함
      - 이 경우, 이산 대수 문제를 풀기 위해서 기본적인 성질을 활용한 여러 가지 알고리즘들이 개발됨
        - 전부 찾기 알고리즘보다는 효율성이 높으나 어느 하나도 계산의 복잡도가 빠르지 않음
        - 대부분의 알고리즘이 소인수분해의 복잡도 정도 수준을 유지함

이산 대수 문제의 복잡도는 소인수분해 문제의 복잡도와 같음

Thanks!

이 은 진(eunjin@pel.sejong.ac.kr)