2025/08/01, 2025 보안 기초 세미나

암호학과 네트워크 보안

-9장 암호수학(1)-

민 윤 홍(muh4316@gmail.com)

세종대학교 프로토콜공학연구실

목 차

- 보충
- 소수
- 소수 판정
- 지수와 로그

보 충

- 결정적 소수 검증 알고리즘 (1/7)
  - 시간 복잡도
    - 계산 방법
      1. 입력 변수 정의(n, m등)
      2. 반복문, *재귀함수 구조 파악
      3. 총 연산 횟수를 수식으로 표현
      4. Big-O 표현으로 단순화

*재귀함수(Recursion Function)
함수 내부에서 자기 자신을 다시 호출하는 함수

```c
#pragma warning(disable:4996)
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main() {
    int r,n;
    scanf("%d", &n);
    int flag = 0;
    r = 2;
        while (r <= sqrt(n))
        {
            if (n % r == 0)
                flag = 1;

            r = r + 1;
    }

        if (flag == 1) {
            printf("composite");
        }
        else if(flag==0)
        {
            printf("prime");
        }

    return 0;
}
```

보 충

- 결정적 소수 검증 알고리즘 (2/7)
  - 시간 복잡도
    - 추가 예제 – 나눔 알고리즘의 시간 복잡도 계산
      1. 입력 변수는 n
      2. 해당 알고리즘의 반복문의개수는 한 개, 종료 조건은 while($r \leq \sqrt{n}$)
      3. 반복문은 약 $\sqrt{n}$번 반복,각 반복에서 $n \,\%\, r$시행, %의 시간 복잡도는 $O(1)$
      4. 총 연산 횟수 $\approx O(\sqrt{n}) \times O(1)$
         $= O(\sqrt{n})$

```c
#pragma warning(disable:4996)
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main() {
    int r,n;
    scanf("%d", &n);
    int flag = 0;
    r = 2;
        while (r <= sqrt(n))
        {
            if (n % r == 0)
                flag = 1;

            r = r + 1;
    }

        if (flag == 1) {
            printf("composite");
        }
        else if(flag==0)
        {
            printf("prime");
        }

    return 0;
}
```

보 충

- 결정적 소수 검증 알고리즘 (3/7)
  - AKS 알고리즘(Agrawal-Kayal-Saxena Algorithm)
    - 알고리즘 의사코드

AKS_Primality_Test(n) { // n이 소수인지 판별
  if (n is a perfect power){
    return "***composite***" // n이 거듭제곱수라면 합성수 판별
  }
  $r \leftarrow 2$
  while ($ord\_r(n) \leq (\log_2 n)^2$){
    $r \leftarrow r + 1$ // r에 대한 n의 위수가 $(\log_2 n)^2$ 보다 작으면 비교 횟수가 너무 적으므로 r값 증가
  }
  for ($a \leftarrow 2$ to $\min(r, n-1)$){
    if($1 < \gcd(a, n) < n$) {//a와 n의 최대공약수가 1보다 크면 합성수 판별
      return "***composite***"
    }
  }

보 충

- 결정적 소수 검증 알고리즘 (4/7)
  - AKS 알고리즘(Agrawal-Kayal-Saxena Algorithm)
    - 알고리즘 의사코드

```
    if(n ≤ r) {
      return "prime"  // 만약 n이 r보다 작다면 이전 과정에서 충분히 걸렀다고 판단하고 소수 판정
    } //아래는  결정적 소수 검증을 위한 항등식 검증
    for (a ←1 to ⌊√φ(r) × log2 n⌋) { // 루프횟수 설정, 내림기호 사용
      if((x + a)^n ≢ x^n + a mod(x^r − 1, n)){ // 페르마의 소정리를 다항식 수준으로 확장하여 검사
        return "composite"
      }
    return "prime"
    }
}
```

보 충

- 결정적 소수 검증 알고리즘 (5/7)
  - AKS 알고리즘(Agrawal-Kayal-Saxena Algorithm)
    - 알고리즘 추가 예제 - n이 67일 때

AKS_Primality_Test(67) { // 67이 소수인지 판별
  if (n is a perfect power){
    return "***composite***" // 67이 거듭제곱수라면 합성수 판별
  }
  $r \leftarrow 2$
  while ($ord\_r(67) \leq (\log_2 67)^2$){
    $r \leftarrow r + 1$ // r에 대한 67의 위수가 $(\log_2 67)^2$보다 커질때 까지 r값 증가 $(\log_2 67)^2 \approx 36, r = 59$
  }
  for ($a \leftarrow 2$ to $\min(r, n-1)$){
    if($1 < \gcd(a, n) < n$) {//a와 67의 최대공약수가 1보다 크면 합성수 판별
      return "***composite***"
    }
  }

보 충

- 결정적 소수 검증 알고리즘 (6/7)
  - AKS 알고리즘(Agrawal-Kayal-Saxena Algorithm)
    - 알고리즘 추가 예제 - n이 67일 때

```
    if(n ≤ r) {//n = 67, r = 59 이므로 위 과정으로 소수검증 실패
      return "prime"  //
    }
    for (a ←1 to ⌊√φ(r) × log2 n⌋) { // a는 1부터 46까지
      if((x + a)^n ≢ x^n + a mod(x^r − 1, n)){  // 항등식 검사
        return "composite"
      }
    return "prime"
    }
}
```

보 충

- 결정적 소수 검증 알고리즘 (7/7)
  - AKS 알고리즘(Agrawal-Kayal-Saxena Algorithm)
    - 알고리즘 구현

```python
#  AKS 소수 판별 메인 함수
def aks(n):
    if n == 1:
        return "composite"  # 1은 소수가 아님

    # 1: 완전 거듭제곱이면 합성수
    if is_perfect_power(n):
        return "composite"

    # 2: ord_r(n) > (log2 n)^2 를 만족하는 최소 r 찾기
    log_n_sq = math.log2(n) ** 2
    r = 2
    while True:
        if gcd(n, r) == 1 and order_mod(n, r) > log_n_sq:
            break
        r += 1

    # 3: 2 ≤ a ≤ min(r, n) 중 gcd(a, n) > 1 이면 합성수
    for a in range(2, min(r, n)):
        if 1 < gcd(a, n) < n:
            return "composite"

    # 4: n ≤ r 이면 소수로 판단 (모든 검사를 통과한 경우)
    if n <= r:
        return "prime"
```

```python
    # 5: 다항식 항등식 검증
    # (x + a)^n ≡ x^n + a (mod x^r - 1, n) 이 모든 a에 대해 성립해야 함
    max_a = int(math.floor(math.sqrt(phi(r)) * math.log2(n)))
    for a in range(1, max_a + 1):
        base = [a, 1]              # (x + a) 다항식
        lhs = poly_pow(base, n, n, r)  # 좌변: (x + a)^n mod (x^r - 1, n)

        # 우변: x^n + a
        rhs = [0] * (n % r) + [1]  # x^n → 차수가 (n mod r)
        if len(rhs) < r:
            rhs += [0] * (r - len(rhs))
        rhs[0] = (rhs[0] + a) % n  # 상수항에 a 더하기

        # 다항식 차수 맞춰 비교
        lhs = lhs + [0] * (r - len(lhs))
        if lhs != rhs:
            return "composite"

    return "prime"

# 사용자 입력 실행
print(aks(int(input('소수 판정 : '))))
```

```
소수 판정 : 67
prime
```

```
소수 판정 : 561
composite
```

```
소수 판정 : 211
prime
```

```
소수 판정 : 1000003
prime
```

목 차

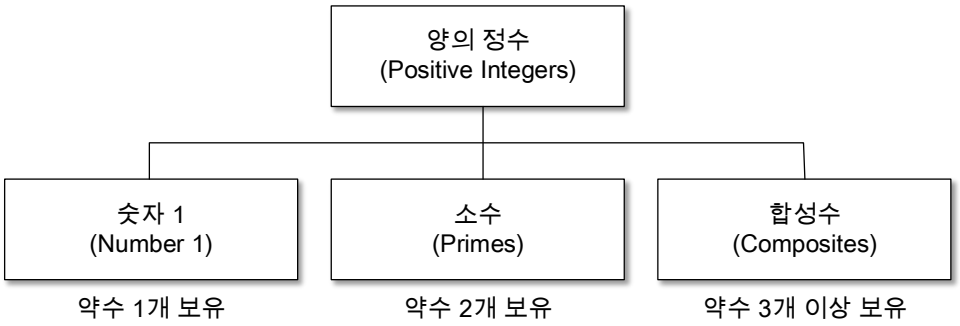
- 소수
- 소수 판정
- 지수와 로그

목 차

- 소수
- 소수 판정
- 지수와 로그

소수

- 개요
  - 양의 정수(Positive Integers)
    - 숫자1
    - 소수
      - 양의 정수에서 오직 1이나 자신으로만 나누어 떨어지는 수
    - 합성수
      - 1과 자기 자신 이외의 양의 약수를 가진 수
  - 서로소(Coprime)
    - 최대공약수가 1인 두 양의 정수의 관계



양의 정수의 3분류

소수

- 소수 집합의 크기 (1/3)
  - 무한히 많은 소수
    - 증명

p	가장 큰 소수
q	$q \leq p$인 소수
r	소수를 모두 곱한 수

- 가정 : 소수의 개수는 유한개
- 모순 증명을 이용하여 다음과 같이 증명
  - 정수 $r+1$은 q를 약수로 갖지 못함
    - $(r+1) mod\ q = 1$
  - 만약 그런 소수 q가 존재한다면 $q \mid r$이고 $q \mid r+1$임
  - 그런 수는 오직 1밖에 존재하지 않음
  - 1은 소수가 아니므로 q는 1이 될 수 없음
  - 처음 가정이 틀렸음
- 따라서 소수의 개수는 무한개임

소수

- 소수 집합의 크기 (2/3)
  - 예제 9.1

소수 전체 집합(q)이 $\{2, 3, 5, 7, 11, 13, 17\}$일 경우, 17보다 큰 소수가 존재함을 증명하시오

- $p = 17,\ q = \{2, 3, 5, 7, 11, 13, 17\}, r = 510{,}510$
- $r + 1 = 510{,}511 = 19 \times 97 \times 277$
- $19, 97, 277$은 소수 전체 집합에 미포함
- 17보다 큰 소수가 3개 있다는 것이 증명

p	가장 큰 소수
q	$q \leq p$인 소수
r	소수를 모두 곱한 수

소수

- 소수 집합의 크기 (3/3)
  - 소수의 개수
    - 공식화
      - 함수 $\pi(n)$은 n보다 작거나 같은 소수의 개수를 나타내는 함수라고 정의
        - e.g., $\pi(1) = 0,\ \pi(2) = 1,\ \pi(3) = 2,\ \pi(10) = 4,\ \pi(100) = 25$
      - 만약 n이 매우 커지게 된다면, $\pi(n)$은 근사로 측정

$$[n/(\ln n)] < \pi(n) < [n/(\ln n - 1.08366)]$$

소수

- 소수 판정 방법 (1/3)
  - 주어진 수 n이 $\sqrt{n}$ 보다 작은 소수로 나누어지는지 확인
    - 나누어지면 합성수, 나누어지지 않으면 소수
    - 예제 9.2

수 97은 소수인지 판별해보자

- $9 < \sqrt{97} < 10$이므로 9이하의 소수는 $2, 3, 5, 7$
- 97이 네 개의 소수로 나누어지는지 확인해보면 어느 소수도 97을 나누는 것은 불가능

$\therefore$ 97은 소수

소수

- 소수 판정 방법 (2/3)
  - 에라토스테네스의 체(Sieve of Eratosthenes)
    - 필요성
      - 주어진 수 n보다 작은 모든 소수를 찾기 위함
    - 방법
      - 표를 그려 $\sqrt{n}$이하의 소수들의 배수로 나누어 떨어지는 수를 지워 소수를 확인하는 방법
    - 특징
      - 표를 그려 가시적으로 확인할 수 있어 직관적임
      - n이 작을 수록 빠름
      - n이 클 경우 모든 수를 표기하지 못하기 때문에 한계가 있음

소수

- 소수 판정 방법 (3/3)
  - 에라토스테네스의 체(Sieve of Eratosthenes)
    - 예제 9.3

100이하의 숫자 중에서 소수를 에라토스테네스의 체를 이용하여 구해보자

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

→

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	~~49~~	~~50~~
~~51~~	~~52~~	53	~~54~~	~~55~~	~~56~~	~~57~~	~~58~~	59	~~60~~
61	~~62~~	~~63~~	~~64~~	~~65~~	~~66~~	67	~~68~~	~~69~~	~~70~~
71	~~72~~	73	~~74~~	~~75~~	~~76~~	~~77~~	~~78~~	79	~~80~~
~~81~~	~~82~~	83	~~84~~	~~85~~	~~86~~	~~87~~	~~88~~	89	~~90~~
~~91~~	~~92~~	~~93~~	~~94~~	~~95~~	~~96~~	97	~~98~~	~~99~~	~~100~~

∴ 100 이하의 소수는 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97이다

소수

- 오일러의 φ함수(Euler's Phi-function) (1/2)
  - 정의
    - 양의 정수 n보다 작으면서 n과 서로소인 정수의 개수
  - 성질
    1. $\varphi(1) = 1$
    2. $\varphi(p) = p - 1,\ p$는 소수

       e.g., $\varphi(13) = 13 - 1 = 12$
    3. $\varphi(m \times n) = \varphi(m) \times \varphi(n),\ (\text{단}, \gcd(m,n) = 1)$

       e.g., $\varphi(15) = \varphi(3) \times \varphi(5) = 2 \times 4 = 8$
    4. $\varphi(p^e) = p^e - p^{e-1},\ p$는 소수

       e.g., $\varphi(16) = \varphi(2^4) = (2^4 - 2^3) = 8$

*p^e
e는 정수 집합이지만 오일러의 φ함수는 양의 정수에서만 정의되므로 $e \geq 1$

소수

- 오일러의 φ함수(Euler's Phi-function) (2/2)
  - 성질 $3, 4$를 이용하여 $\varphi(n)$구하기
    - $\varphi(n) = \left(p_1^{e_1} - p_1^{e_1-1}\right) \times \left(p_2^{e_2} - p_2^{e_2-1}\right) \times \cdots \times \left(p_k^{e_k} - p_k^{e_k-1}\right)$
      $(n = p^{e_1} \times p^{e_2} \times \cdots \times p^{e_k})$
    - 예제 9.4

$\varphi(240)$를 구하시오

- $240 = 2^4 \times 3^1 \times 5^1$
- $\varphi(240) = \varphi(2^4) \times \varphi(3^1) \times \varphi(5^1)$

$= (2^4 - 2^3) \times (3^1 - 3^0) \times (5^1 - 5^0)$

$= 64$

소수

- 페르마의 소 정리(Fermat's Little Theorem) (1/3)
  - 필요성
    - 소수와 정수의 거듭제곱 간 모듈러 연산의 규칙성을 설명
    - 거듭제곱의 계산을 줄일 수 있음
  - 버전1
    - 만약 p가 소수이고, $p \nmid a$ 인 경우
      - $a^{p-1} \equiv 1 \; mod \; p$
        - e.g., $6^{10} mod \; 11 = 1, (p = 11)$
  - 버전2
    - 만약 p가 소수이고 버전1 이외의 경우
      - $a^{p} \equiv a \; mod \; p$
        - e.g., $3^{12} mod \; 11 = (3^{11} \times 3^{1}) mod \; 11 = (3^{11} mod \; 11) \times (3^{1} \; mod \; 11)$
          $= 3 \times 3 = 9$

소수

- 페르마의 소 정리(Fermat's Little Theorem) (2/3)
  - 곱에 관한 역원
    - 필요성
      - 페르마의 소정리를 이용하여 모듈로 값이 소수일 경우 확장 유클리드 알고리즘을 사용하지 않더라도 곱에 관한 역원을 구할 수 있음
    - 만약 p가 소수이고 $p \nmid a$라면 $a^{-1} mod\ p = a^{p-2} mod\ p$
    - 증명

- $a^{-1} mod\ p = a^{p-2} mod\ p$ 양변에 a를 곱함
- $1\ mod\ p = a^{p-1} mod\ p$
- 페르마의 소 정리 중 버전1을 적용

소수

- 페르마의 소 정리(Fermat's Little Theorem) (3/3)
  - 예제 9.5

$8^{-1} mod\ 17$을 구해보자

$$8^{-1} mod\ 17 = 8^{17-2} mod\ 17 = 8^{15} mod\ 17$$

$8^2 mod\ 17 = 64\ mod\ 17 = 13$
$8^3 mod\ 17 = (13 \times 8) mod\ 17 = 2$
$8^4 mod\ 17 = (2 \times 8) mod\ 17 = 16$
$\vdots$
$8^{15} mod\ 17 = (8 \times 4) mod\ 17 = 15$
$\therefore 8^{-1} mod\ 17 = 15$

  - 예제 9.6

$5^{-1} mod\ 23$을 구해보자

$$5^{-1} mod\ 23 = 5^{23-2} mod\ 23 = 5^{21} mod\ 23 = 14\ mod\ 23 = 14$$

소수

- 오일러 정리(Euler's Theorem) (1/6)
  - 필요성
    - 모듈로 값이 소수일 경우 뿐만 아니라 정수일 경우에도 규칙성이 있다는 것을 설명하기 위함
  - 버전1
    - a와 n가 서로소인 경우
      - $a^{\varphi(n)} \equiv 1 \, mod \, n$
  - 버전2
    - a와 n이 서로소가 아니고 k가 정수일 경우
      - $a^{k \times \varphi(n)+1} \equiv a \, mod \, n$

소수

- 오일러 정리(Euler's Theorem) (2/6)
  - 버전1 증명 (1/2)
    - n이하 자연수에서 n과 서로소인 수들의 집합 S에 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

- a와 n이 서로소이므로 $as_k \bmod n (k = 1, 2, \cdots \varphi(n))$은 서로 다른 나머지를 가짐
- 같은 나머지를 갖는다면 $a(s_i - s_j) \equiv 0 (mod\ n)$ 이므로 $s_i = s_j$여야만 함
  하지만 a와 n이 서로소이므로 모순임
- 따라서 집합 aS를 n으로 나누면 나오는 나머지가 모두 다름

소수

- 오일러 정리(Euler's Theorem) (3/6)
  - 버전1 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

$$as_1 \times as_2 \times \cdots \times as_{\varphi(n)} \equiv s_1 \times s_2 \times \cdots \times s_{\varphi(n)} (mod\ n)$$

    - 양변을 S의 원소들로 나눔

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

소수

- 오일러 정리(Euler's Theorem) (4/6)
  - 버전2 증명
    - a가 p의 배수도 아니고 q의 배수도 아니라면 a와 n은 서로소

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

    - a가 p의 배수이나 q의 배수는 아니라고 한다면 증명은 아래와 같음(단, $a = i \times p, n = p \times q$)

$$a^{\varphi(n)} mod\ q = \left(a^{\varphi(q)} mod\ q\right)^{\varphi(p)} mod\ q = 1 \rightarrow a^{\varphi(n)} mod\ q = 1$$
$$a^{k\times\varphi(n)} mod\ q = \left(a^{\varphi(n)} mod\ q\right)^k mod\ q = 1 \rightarrow a^{k\times\varphi(n)} mod\ q = 1$$
$$a^{k\times\varphi(n)} mod\ q = 1,\ a^{k\times\varphi(n)} = 1 + j \times q \text{ (합동)}$$
$$a^{k\times\varphi(n)+1} = a \times (1 + j \times q) = a + j \times \mathrm{q} \times a = a + (i \times j) \times q \times p = a + (i \times j) \times n$$
$$a^{k\times\varphi(n)+1} = a + (i \times j) \times n \rightarrow a^{k\times\varphi(n)+1} = a\ mod\ n \text{ (합동 관계)}$$

소수

- 오일러 정리(Euler's Theorem) (5/6)
  - 버전2 증명
    - a가 q의 배수로서 $a = i \times q$이지만 p의 배수는 아닌 경우 (단, $a = i \times q, n = p \times q$)

$a^{\varphi(n)} mod\ p = \left(a^{\varphi(p)} mod\ p\right)^{\varphi(q)} mod\ p = 1 \rightarrow a^{\varphi(n)} mod\ p = 1$

$a^{k \times \varphi(n)} mod\ p = \left(a^{\varphi(n)} mod\ p\right)^{k} mod\ p = 1 \rightarrow a^{k \times \varphi(n)} mod\ p = 1$

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

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

$a^{k \times \varphi(n)+1} = a + (i \times j) \times n \rightarrow a^{k \times \varphi(n)+1} = a\ mod\ n$ (합동 관계)

소수

- 오일러 정리(Euler's Theorem) (6/6)
  - 곱에 대한 역원
    - 필요성
      - 오일러 정리를 이용하면 소수 모듈로와 합성수 모듈로를 갖는 경우에 곱에 대한 역원을 가질 수 있음
    - n과 a가 서로소일 경우
      - $a^{-1} mod\ n = a^{\varphi(n)-1} mod\ n$
        - 증명

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

    - 예제 9.7

$8^{-1} mod\ 77$을 구해보자

$$8^{-1} mod\ 77 = 8^{\varphi(77)-1} mod\ 77 = 8^{59} mod\ 77 = 29\ mod\ 77$$

소수

- 소수 생성 (1/2)
  - 소수를 생성할 수 있는 공식을 만들고자 함
  - 메르센 소수(Mersenne Prime)
    - 정의
      - 메르센 수 중에서 소수인 수
        - 메르센 수 : $M_p = 2^p - 1$
    - p가 소수라면, M_p는 소수라고 생각되었으나 반례가 존재함

$M_2 = 2^2 - 1 = 3$
$M_3 = 2^3 - 1 = 7$
$M_5 = 2^5 - 1 = 31$
$M_7 = 2^7 - 1 = 127$
$M_{11} = 2^{11} - 1 = 2047$ →합성수이므로 위배($\because 2047 = 23 \times 89$)
$M_{13} = 2^{13} - 1 = 8191$
$M_{17} = 2^{17} - 1 = 131071$

소수

- 소수 생성 (2/2)
  - 페르마 소수(Fermat Prime)
    - 정의
      - 페르마 수 중에서 소수인 수
        - 페르마 수 : $F_n = 2^{2^n} + 1 \ (n \in Z^+)$
    - F_4 까지는 검증되었으나 F_5는 소수가 아니라는 것이 증명

$F_0 = 2^1 + 1 = 3$
$F_1 = 2^2 + 1 = 5$
$F_2 = 2^4 + 1 = 17$
$F_3 = 2^8 + 1 = 257$
$F_4 = 2^{16} + 1 = 65537$
$F_5 = 2^{32} + 1 = 4294967297 \rightarrow$ 합성수이므로 위배($\because 4294967297 = 641 \times 6700417$)

목 차

- 소수
- 소수 판정
- 지수와 로그

소수 판정

- 개요
  - 암호학에서 키를 생성하기 위해 선택한 수가 소수인지 판별하기 위해 필요
  - 알고리즘 종류
    - 결정적 알고리즘(Deterministic Algorithm)
      - 같은 입력에 대해 항상 같은 출력을 주고 같은 연산 과정을 거치는 알고리즘
      - 정확하고 예측 가능한 결과를 원할 때 사용
    - 확률적 알고리즘(Probabilistic Algorithm)
      - 같은 입력에도 실행마다 다른 결과나 연산 과정을 가질 수 있는 알고리즘
      - 오차가 존재하나 빠른 결과를 원할 때 사용

소수 판정

- 결정적 소수 검증 알고리즘 (1/15)
  - 정의
    - 정수를 입력 받아 합성수인지 소수인지 판별하는 알고리즘
  - 알고리즘 종류
    - 나눔 알고리즘(Divisibility Algorithm)
    - AKS 알고리즘(Agrawal-Kayal-Saxena Algorithm)
    - 시간 복잡도

소수 판정

- 결정적 소수 검증 알고리즘 (2/15)
  - 나눔 알고리즘(Divisibility Algorithm)
    - 정의
      - 입력받는 수를 $\sqrt{n}$보다 작은 모든 양의 정수로 나누어 소수를 판별하는 알고리즘
        - $\sqrt{n}$ 보다 작은 수들 중 나누어지는 수가 있으면 n은 합성수로 판별
    - 비트-연산 복잡도
      - $f(n_b) = \sqrt{2^{n_b}} = 2^{\frac{n_b}{2}}$ (n_b는 n을 2진수로 변경하였을 때 비트 수)
      - Big-O 기호 : $O(2^{n_b})$
    - 특징
      - n_b가 커지면 비실용적
      - 소수 판별의 가장 기초적인 방법으로 사용

Divisiblity_Test (n) //n에 대해서 소수 검증

```
{
    r ← 2
    while (r < √n)
    {
     if (r | n) return "composite"
     r ← r + 1
    }
    return "prime"
}
```

소수 판정

- 결정적 소수 검증 알고리즘 (3/15)
  - 나눔 알고리즘(Divisibility Algorithm)
    - 예제 9.8

n의 비트 수를 200이라고 가정했을 때 나눔 알고리즘을 수행하는데 필요한 비트 연산의 회수는?

- $n_b = 200, f(200) = \sqrt{2^{200}} = 2^{100}$번의 비트 연산 필요
- 초당 2^{30}번의 비트 연산을 수행할 수 있는 컴퓨터라면 2^{70}초의 시간 필요
- (필요 시간$= \frac{2^{100}}{2^{30}} = 2^{100-30} = 2^{70}$초)

소수 판정

- 결정적 소수 검증 알고리즘 (4/15)
  - 나눔 알고리즘(Divisibility Algorithm)
    - 의사코드 예제

```c
#pragma warning(disable:4996)
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main() {
    int r,n;
    scanf("%d", &n);
    int flag = 0;
    r = 2;
        while (r <= sqrt(n))
        {
            if (n % r == 0)
                flag = 1;

            r = r + 1;
    }

        if (flag == 1) {
            printf("composite");
        }
        else if(flag==0)
        {
            printf("prime");
        }

    return 0;
}
```

```
7
prime
```

```
4
composite
```

```
65537
prime
```

```
3145677
composite
```

$3145677 = 991 \times 3173$

```
1000003
prime
```

```
3121793
composite
```

$3121793 = 941 \times 3317$

소수 판정

- 결정적 소수 검증 알고리즘 (5/15)
  - AKS 알고리즘(Agrawal-Kayal-Saxena Algorithm)
    - 정의
      - $(x+a)^n \equiv (x^n + a) mod\ n$를 이용하여 n이 소수인지를 판별하는 알고리즘 ($n \geq 2$인 정수)
      - 위 식이 성립하지 않으면 n은 합성수
    - 비트-연산 복잡도
      - $f(n_b) = (\log_2 n_b)^{12}$
      - Big O 기호 $O((\log_2 n_b)^{12})$
    - 특징
      - 나눔 알고리즘에 비해 큰 입력값 계산 가능함
      - 다항적 시간 복잡도를 가짐

소수 판정

- 결정적 소수 검증 알고리즘 (6/15)
  - AKS 알고리즘(Agrawal-Kayal-Saxena Algorithm)
    - 알고리즘 의사코드

AKS_Primality_Test(n) { // n이 소수인지 판별
  if (n is a perfect power){
    return "***composite***" // n이 거듭제곱수라면 합성수 판별
  }
  $r \leftarrow 2$
  while $(ord\_r(n) \leq (\log_2 n)^2)$){
    $r \leftarrow r + 1$ // r에 대한 n의 위수가 $(\log_2 n)^2$ 보다 작으면 비교 횟수가 너무 적으므로 r값 증가
  }
  for ($a \leftarrow 2$ to $\min(r, n-1)$){
    if$(1 < \gcd(a,n) < n)$ {//a와 n의 최대공약수가 1보다 크면 합성수 판별
      return "***composite***"
    }
  }

소수 판정

- 결정적 소수 검증 알고리즘 (7/15)
  - AKS 알고리즘(Agrawal-Kayal-Saxena Algorithm)
    - 알고리즘 의사코드

```
    if(n ≤ r) {
      return "prime"  // 만약 n이 r보다 작다면 이전 과정에서 충분히 걸렀다고 판단하고 소수 판정
    } //아래는  결정적 소수 검증을 위한 항등식 검증
    for (a ←1 to ⌊√φ(r) × log2 n⌋) { // 루프횟수 설정, 내림기호 사용
      if((x + a)^n ≢ x^n + a mod(x^r − 1, n)){  // 페르마의 소정리를 다항식 수준으로 확장하여 검사
        return "composite"
      }
    return "prime"
    }
}
```

소수 판정

- 결정적 소수 검증 알고리즘 (8/15)
  - AKS 알고리즘(Agrawal-Kayal-Saxena Algorithm)
    - 알고리즘 추가 예제 - n이 67일 때

AKS_Primality_Test(67) { // 67이 소수인지 판별
  if (n is a perfect power){
    return "***composite***" // 67이 거듭제곱수라면 합성수 판별
  }
  $r \leftarrow 2$
  while ($ord\_r(67) \leq (\log_2 67)^2$){
    $r \leftarrow r + 1$ // r에 대한 67의 위수가 $(\log_2 67)^2$보다 커질때 까지 r값 증가 $(\log_2 67)^2 \approx 36, r = 59$
  }
  for ($a \leftarrow 2$ to $\min(r, n-1)$){
    if($1 < \gcd(a, n) < n$) {//a와 67의 최대공약수가 1보다 크면 합성수 판별
      return "***composite***"
    }
  }

소수 판정

- 결정적 소수 검증 알고리즘 (9/15)
  - AKS 알고리즘(Agrawal-Kayal-Saxena Algorithm)
    - 알고리즘 추가 예제 - n이 67일 때

```
    if(n ≤ r) {//n = 67, r = 59 이므로 위 과정으로 소수검증 실패
      return "prime"  //
    }
    for (a ←1 to ⌊√φ(r) × log2 n⌋) { // a는 1부터 46까지
      if((x + a)^n ≢ x^n + a mod(x^r − 1, n)){  // 항등식 검사
        return "composite"
      }
    return "prime"
    }
}
```

소수 판정

- 결정적 소수 검증 알고리즘 (10/15)
  - AKS 알고리즘(Agrawal-Kayal-Saxena Algorithm)
    - 예제 9.9

n_b의 비트 수를 200이라고 가정했을 때 AKS 알고리즘을 수행하는데 필요한 비트 연산의 회수는?

- 이 알고리즘의 비트-연산 복잡도는 $O((\log_2 200)^{12})$
- 소수 판별을 위해 $(\log_2 200)^{12} = 39{,}547{,}615{,}483$의 비트 연산 필요
- 초당 10억번의 비트 연산을 수행하는 컴퓨터를 사용하는 경우 해당 알고리즘을 이용해 40초만에 소수 판별 가능

소수 판정

- 결정적 소수 검증 알고리즘 (11/15)
  - AKS 알고리즘(Agrawal-Kayal-Saxena Algorithm)
    - 알고리즘 구현

```python
#  AKS 소수 판별 메인 함수
def aks(n):
    if n == 1:
        return "composite"  # 1은 소수가 아님

    # 1: 완전 거듭제곱이면 합성수
    if is_perfect_power(n):
        return "composite"

    # 2: ord_r(n) > (log2 n)^2 를 만족하는 최소 r 찾기
    log_n_sq = math.log2(n) ** 2
    r = 2
    while True:
        if gcd(n, r) == 1 and order_mod(n, r) > log_n_sq:
            break
        r += 1

    # 3: 2 ≤ a ≤ min(r, n) 중 gcd(a, n) > 1 이면 합성수
    for a in range(2, min(r, n)):
        if 1 < gcd(a, n) < n:
            return "composite"

    # 4: n ≤ r 이면 소수로 판단 (모든 검사를 통과한 경우)
    if n <= r:
        return "prime"
```

```python
    # 5: 다항식 항등식 검증
    # (x + a)^n ≡ x^n + a (mod x^r - 1, n) 이 모든 a에 대해 성립해야 함
    max_a = int(math.floor(math.sqrt(phi(r)) * math.log2(n)))
    for a in range(1, max_a + 1):
        base = [a, 1]              # (x + a) 다항식
        lhs = poly_pow(base, n, n, r)  # 좌변: (x + a)^n mod (x^r - 1, n)

        # 우변: x^n + a
        rhs = [0] * (n % r) + [1]  # x^n → 차수가 (n mod r)
        if len(rhs) < r:
            rhs += [0] * (r - len(rhs))
        rhs[0] = (rhs[0] + a) % n  # 상수항에 a 더하기

        # 다항식 차수 맞춰 비교
        lhs = lhs + [0] * (r - len(lhs))
        if lhs != rhs:
            return "composite"

    return "prime"

# 사용자 입력 실행
print(aks(int(input('소수 판정 : '))))
```

```
소수 판정 : 67
prime
```

```
소수 판정 : 561
composite
```

```
소수 판정 : 211
prime
```

```
소수 판정 : 1000003
prime
```

소수 판정

- 결정적 소수 검증 알고리즘 (12/15)
  - 시간 복잡도
    - 정의
      - 입력 크기 n에 따라 알고리즘이 실행되는 연산 횟수의 증가율을 함수로 표현한 식
    - 다항적 시간 복잡도
      - 알고리즘의 실행 시간이 입력 크기(n)에 대해 다항식으로 표현되는 경우
      - $O(1), O(n), O(n^k)$등으로 표현(단, $k \in \mathbb{N}$)
      - 특징
        - 현실적인 시간 내에 계산 가능한 범위로 간주

소수 판정

- 결정적 소수 검증 알고리즘 (13/15)
  - 시간 복잡도
    - 지수적 시간 복잡도
      - 알고리즘의 실행 시간이 입력 크기(n)에 지수 함수식으로 표현되는 경우
      - $O(2^n), O(3^n), O(n!)$등으로 표현
      - 특징
        - 현실에서 계산 불가능한 범위로 간주
        - 입력이 조금만 커져도 계산량이 기하급수적으로 증가

소수 판정

- 결정적 소수 검증 알고리즘 (14/15)
  - 시간 복잡도
    - 계산 방법
      1. 입력 변수 정의(n, m등)
      2. 반복문, 재귀함수 구조 파악
      3. 총 연산 횟수를 수식으로 표현
      4. Big-O 표현으로 단순화

*재귀함수(Recursion Function)
함수 내부에서 자기 자신을 다시 호출하는 함수

```c
#pragma warning(disable:4996)
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main() {
    int r,n;
    scanf("%d", &n);
    int flag = 0;
    r = 2;
        while (r <= sqrt(n))
        {
            if (n % r == 0)
                flag = 1;

            r = r + 1;
    }

        if (flag == 1) {
            printf("composite");
        }
        else if(flag==0)
        {
            printf("prime");
        }

    return 0;
}
```

소수 판정

- 결정적 소수 검증 알고리즘 (15/15)
  - 시간 복잡도
    - 추가 예제 – 나눔 알고리즘의 시간 복잡도 계산
      1. 입력 변수는 n
      2. 해당 알고리즘의 반복문의개수는 한 개, 종료 조건은 while($r \leq \sqrt{n}$)
      3. 반복문은 약 $\sqrt{n}$번 반복,각 반복에서 $n \,\%\, r$시행, %의 시간 복잡도는 $O(1)$
      4. 총 연산 횟수 $\approx O(\sqrt{n}) \times O(1)$
         $= O(\sqrt{n})$

```c
#pragma warning(disable:4996)
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main() {
    int r,n;
    scanf("%d", &n);
    int flag = 0;
    r = 2;
        while (r <= sqrt(n))
        {
            if (n % r == 0)
                flag = 1;

            r = r + 1;
    }

        if (flag == 1) {
            printf("composite");
        }
        else if(flag==0)
        {
            printf("prime");
        }

    return 0;
}
```

소수 판정

- 확률적 소수 검증 알고리즘 (1/22)
  - 정의
    - 어떤 수가 소수인지 확률적으로 판단하는 알고리즘

규칙
a. 만약 검증하려는 정수가 소수라면, 알고리즘은 반드시 소수라고 판정
b. 만약 검증하려는 정수가 합성수라면, 알고리즘은 이 수를 합성수라고 판정할 확률이 $1-\varepsilon$이고, 소수라고 판정할 확률은 ε

  - 알고리즘 종류
    - 페르마 소수 검증(Fermat Primality Test)
    - 제곱근 소수 검증(Square Root Primality Test)
    - 밀러-라빈 소수 검증(Miller-Rabin Primality Test)

소수 판정

- 확률적 소수 검증 알고리즘 (2/22)
  - 페르마 소수 검증(Fermat Primality Test)
    - 정의
      - 어떤 수 n이 소수인지 여부를 파악하기 위해 $a^{n-1} mod\ n$의 값이 1이 되는지 확인하는 검증

만약n이 소수라면, $a^{n-1} \equiv 1\ mod\ n$이다$(gcd(a,\ n) = 1)$
만약n이 합성수라면, $a^{n-1} \equiv 1\ mod\ n$일 수도 있다

    - 시간 복잡도
      - $O(k^3)$(k는 비트 수)
    - 특징
      - 소수는 페르마 검증을 하면 소수로 판정
      - 소수라고 판정 날 확률은 ε
      - 합성수로 판정 날 확률은 $1 - \varepsilon$
      - 여러 밑수$(a_1, a_2, a_3 \cdots)$를 사용해서 검증하게 되면 확률 값을 개선 가능

소수 판정

- 확률적 소수 검증 알고리즘 (3/22)
  - 페르마 소수 검증(Fermat Primality Test)
    - 예제 9.10 – n이 소수지만 페르마 검증을 통과하는 경우

수 561을 페르마 검증 해보자

- 2를 밑수로 사용
  $2^{561-1} \equiv 1 \, mod \, 561$으로 페르마 검증을 통과했지만 $561 = 33 \times 17$으로 소수가 아님
- 5를 밑수로 사용
  $5^{561-1} \equiv 1 \, mod \, 561$으로 페르마 검증을 통과했지만 소수가 아님
- 7을 밑수로 사용
  $7^{561-1} \equiv 1 \, mod \, 561$으로 페르마 검증을 통과했지만 소수가 아님
- 13을 밑수로 사용
  $13^{561-1} \equiv 1 \, mod \, 561$으로 페르마 검증을 통과했지만 소수가 아님

소수 판정

- 확률적 소수 검증 알고리즘 (4/22)
  - 페르마 소수 검증(Fermat Primality Test)
    - 예제 9.10 – 추가 예제

수 561을 페르마 검증 해보자

- 17을 밑수로 사용
  $17^{561-1} \not\equiv 1 \, mod \, 561$으로 페르마 검증을 통과하지 못 했으므로 합성수임
- 19를 밑수로 사용
  $19^{561-1} \equiv 1 \, mod \, 561$으로 페르마 검증을 통과했지만 소수가 아님
- 23을 밑수로 사용
  $23^{561-1} \equiv 1 \, mod \, 561$으로 페르마 검증을 통과했지만 소수가 아님
- 29을 밑수로 사용
  $29^{561-1} \equiv 1 \, mod \, 561$으로 페르마 검증을 통과했지만 소수가 아님

소수 판정

- 확률적 소수 검증 알고리즘 (5/22)
  - 페르마 소수 검증(Fermat Primality Test)
    - 예제 9.10 – 추가 예제

수 561을 페르마 검증 해보자

- $\varphi(561) = 320$이므로 $1 \leq a < 561$중에서 320개는 $\gcd(a, 561) = 1$인 수들 320개는 모두 페르마 검증을 통과
- 561이 소수로 판단될 확률: $\frac{320}{560} = \frac{4}{7} = \varepsilon \approx 57.14\%$
- 561이 합성수로 판단될 확률 : $\frac{240}{560} = \frac{3}{7} = 1 - \varepsilon \approx 42.86\%$

소수 판정

- 확률적 소수 검증 알고리즘 (6/22)
  - 페르마 소수 검증(Fermat Primality Test)
    - 예제 9.11 – n이 소수인 경우

수 211을 페르마 검증 해보자

- 2를 밑수로 사용
  $2^{211-1} \equiv 1 \, mod \, 211$ 으로 페르마 검증 통과
- 3을 밑수로 사용
  $3^{211-1} mod \, 211 \equiv 1 \, mod \, 211$으로 페르마 검증 통과
- 5를 밑수로 사용
  $5^{211-1} mod \, 211 \equiv 1 \, mod \, 211$으로 페르마 검증 통과
- 7을 밑수로 사용
  $7^{211-1} mod \, 211 \equiv 1 \, mod \, 211$페르마 검증 통과

소수 판정

- 확률적 소수 검증 알고리즘 (7/22)
  - 페르마 소수 검증(Fermat Primality Test)
    - 예제 9.11 – 추가 예제

수 211을 페르마 검증 해보자

- 10을 밑수로 사용
  $10^{211-1} \equiv 1 \, mod \, 211$ 으로 페르마 검증 통과
- 11을 밑수로 사용
  $11^{211-1} mod \, 211 \equiv 1 \, mod \, 211$으로 페르마 검증 통과
- 13를 밑수로 사용
  $13^{211-1} mod \, 211 \equiv 1 \, mod \, 211$으로 페르마 검증 통과
- 17을 밑수로 사용
  $17^{211-1} mod \, 211 \equiv 1 \, mod \, 211$페르마 검증 통과

소수 판정

- 확률적 소수 검증 알고리즘 (8/22)
  - 페르마 소수 검증(Fermat Primality Test)
    - 알고리즘 의사코드

```
Fermat_Primality_Test(n, k)    // n이 소수인지 판별
{
  if (n ≤ 1 or n = 4)
    return "composite"

  if (n ≤ 3)
    return "prime"

  for i ← 1 to k
  {
    a ← random integer in (2, n − 2)

    if (a^(n−1) mod n ≠ 1)
      return "composite"
  }

  return "probably prime"
}
```

소수 판정

- 확률적 소수 검증 알고리즘 (9/22)
  - 페르마 소수 검증(Fermat Primality Test)
    - 알고리즘 의사코드 추가 예제

```c
int main() {
    int n, k;
    int i, j;
    int a, result;

    scanf("%d %d", &n, &k);

    if (n <= 1 || n == 4) {
        printf("composite\n");
        return 0;
    }

    if (n <= 3) {
        printf("prime\n");
        return 0;
    }
```

```c
    srand(time(NULL));

    i = 0;
    while (i < k) {
        a = 2 + rand() % (n - 3);

        result = 1;
        j = 0;
        while (j < n - 1) {
            result = (result * a) % n;
            j = j + 1;
        }

        if (result != 1) {
            printf("composite\n");
            return 0;
        }

        i = i + 1;
    }

    printf("probably prime\n");
    return 0;
}
```

```
561
1
probably prime
```

```
561
5
composite
```

```
211
100
probably prime
```

```
1000003
50000
composite
```

소수 판정

- 확률적 소수 검증 알고리즘 (10/22)
  - 제곱근 소수 검증(Square Root Primality Test)
    - 정의
      - $a\ mod\ n$에서 n이 소수이면 1의 제곱근은 ± 1이지만 합성수라면 ± 1외에도 다른 근이 있을 수 있다는 걸 이용한 검증

만약 n이 소수라면, $\sqrt{1}\ mod\ n \equiv \pm 1$이다
만약 n이 합성수라면, $\sqrt{1}\ mod\ n \equiv \pm 1$일 수도 있다

    - 특징
      - 모듈로 계산에서 n이 소수이면 1의 제곱근은 ± 1인 걸 이용
      - $a^2 \equiv 1\ mod\ n$을 만족하는 모든 a가 오직 $a \equiv 1\ mod\ n$ 혹은 $a \equiv -1\ mod\ n$일 경우에만 n이 소수일 가능성이 있음
        - a는 n과 서로소이면서 n보다 작은 정수 $\rightarrow \gcd(a, n) = 1, a < n$
      - 정의를 만족하더라도 합성수인 n이 존재 가능
        - e.g., $n = 4$일 때 $1^2 mod\ 4 = 1$이고 $3^2 mod\ 4 = 1$ 이지만 합성수

소수 판정

- 확률적 소수 검증 알고리즘 (11/22)
  - 제곱근 소수 검증(Square Root Primality Test)
    - 예제 9.12 : n이 소수인 경우

$n = 7$일 경우 제곱근 소수 검증을 해보자

$1^2 : 1\ mod\ 7 = 1,\ (-1)^2 : 1\ mod\ 7 = 1$
$2^2 : 4\ mod\ 7 = 4,\ (-2)^2 : 4\ mod\ 7 = 4$
$3^2 : 9\ mod\ 7 = 2,\ (-3)^2 : 9\ mod\ 7 = 2$
$4^2 : 16\ mod\ 7 = 2,\ (-4)^2 : 16\ mod\ 7 = 2$
$5^2 : 25\ mod\ 7 = 4,\ (-5)^2 : 25\ mod\ 7 = 4$
$6^2 : 36\ mod\ 7 = 1,\ (-6)^2 : 36\ mod\ 7 = 1$

1과 $6(= n - 1)$을 제외한 수들을 제곱해서 모듈로 연산을 했을 때 결과값이 1이 나오는 수가 존재하지 않으므로 7은 소수일 가능성이 높음

소수 판정

- 확률적 소수 검증 알고리즘 (12/22)
  - 제곱근 소수 검증(Square Root Primality Test)
    - 예제 9.13 : n이 소수의 조건을 만족하지만 합성수인 경우

$n = 4$일 경우 제곱근 소수 검증을 해보자

$1^2 : 1\ mod\ 4 = 1,\ (-1)^2 : 1\ mod\ 4 = 1$
$3^2 : 9\ mod\ 4 = 1,\ (-3)^2 : 9\ mod\ 4 = 1$

1과 $3(= n - 1)$을 제외한 수들을 제곱해서 모듈로 연산을 했을 때 결과값이 1이 나오는 수가 존재하지 않으므로 4는 소수일 가능성이 높으나, 4는 합성수임

소수 판정

- 확률적 소수 검증 알고리즘 (13/22)
  - 제곱근 소수 검증(Square Root Primality Test)
    - 예제 9.14 : n이 합성수인 경우

$n = 8$일 경우 제곱근 소수 검증을 해보자

$1^2 : 1\ mod\ 8 = 1,\ (-1)^2 : 1\ mod\ 8 = 1$
$3^2 : 9\ mod\ 8 = 1,\ (-3)^2 : 9\ mod\ 8 = 1$
$5^2 : 25\ mod\ 8 = 1,\ (-5)^2 : 25\ mod\ 8 = 1$
$7^2 : 49\ mod\ 8 = 1,\ (-7)^2 : 49\ mod\ 8 = 1$

1과 $7(= n - 1)$을 제외한 수들을 제곱해서 모듈로 연산을 했을 때 결과값이 1이 나오는 수가 존재하므로 8은 합성수임

소수 판정

- 확률적 소수 검증 알고리즘 (14/22)
  - 밀러-라빈 소수 검증(Miller-Rabin Primality Test) (1/5)
    - 정의
      - 페르마 검증과 제곱근 검증을 혼합하여 *강 의사소수를 구하는 방법

*강 의사소수(Strong Pseudoprime)
소수일 확률이 매우 큰 수

    - 특징
      - $n-1 = m \times 2^k (m$은 홀수$)$로 표현
        - n은 소수를 검증할 수
      - 밑수 a를 사용한 페르마 검증은 아래와 같이 표현

$$a^{n-1} mod\ n = a^{m \times 2^k} mod\ n = (a^m)^{2^k} mod\ n = (a^m)^{\underbrace{2^{2^{\cdot^{\cdot^{\cdot^2}}}}}_{k \text{ times}}} mod\ n \equiv 1\ mod\ n$$

소수 판정

- 확률적 소수 검증 알고리즘 (15/22)
  - 밀러-라빈 소수 검증(Miller-Rabin Primality Test) (2/5)
    - 과정
      - 초기화 단계
        - 밑수 a를 선택하고 $T = a^m mod\ n$을 계산
          - 만약 T가 $+1$이거나 -1일 경우 n은 강 의사소수라고 선언
            - 페르마 검증과 제곱근 검증을 통과
          - 만약 T가 그 이외의 값을 갖는다면 n이 소수인지 합성수인지 판단 불가
      - 1단계
        - T를 제곱해서 계산($T^2 = a^{2m} mod\ n$)
        - 만약 결과가 $+1$이 된다면 n은 합성수로 판별
          - 페르마 검증은 통과하지만 제곱근 검증은 미 통과
            - T는 1이고 앞 단계에서는 ± 1이 아닌 다른 어떤 값이었기 때문
        - 만약 결과가 -1이라면, n은 페르마 검증과 제곱근 검증을 통과하고 강 의사소수라고 선언
        - 만약 결과가 ± 1이 아닌 다른 값을 갖는다면, 소수판별 불가

소수 판정

- 확률적 소수 검증 알고리즘 (16/22)
  - 밀러-라빈 소수 검증(Miller-Rabin Primality Test) (3/5)
    - 과정
      - 2단계부터 $k-1$단계
        - 2단계부터 $k-1$단계까지 결과가 ± 1이 나올 때까지 단계 1의 과정 반복
      - k단계
        - $k-1$단계 후 절차가 끝나지 않았다면 n은 합성수라고 선언

소수 판정

- 확률적 소수 검증 알고리즘 (17/22)
  - 밀러-라빈 소수 검증(Miller-Rabin Primality Test) (4/5)
    - 알고리즘 의사코드

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

소수 판정

- 확률적 소수 검증 알고리즘 (18/22)
  - 밀러-라빈 소수 검증(Miller-Rabin Primality Test) (5/5)
    - 예제 9.15

수 561은 밀러-라빈 검증을 통과하는가?

- 밑수 a를 2로 사용하면, $561 - 1 = 35 \times 2^4$로 표현되므로 $m = 35, k = 4, a = 2$
- 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

    - 예제 9.16

27은 소수가 아닌 것을 알고 있다. 밀러-라빈 검증을 하시오

- 밑수로 2를 사용하면 $27 - 1 = 13 \times 2^1$이므로 $m = 13, k = 1, a = 2$
- 이 경우 $k - 1 = 0$이므로 초기화 단계만 수행
  $T = 2^{13} mod\ 27 = 11 mod\ 27 \rightarrow$ 합성수

소수 판정

- 확률적 소수 검증 알고리즘 (19/22)
  - 밀러-라빈 소수 검증(Miller-Rabin Primality Test) (5/5)
    - 예제 9.17

수 561은 밀러-라빈 검증을 통과하는가?

- 밑수 a를 2로 사용하면, $561 - 1 = 35 \times 2^4$로 표현되므로 $m = 35, k = 4, a = 2$
- 초기화 $T = 2^{35} mod\ 561 = 263\ mod\ 561$
- $k = 1$: $T = 263^2 mod\ 561 = 166\ mod\ 561$
- $k = 2$: $T = 166^2 mod\ 561 = 67\ mod\ 561$
- $k = 3$: $T = 67^2 mod\ 561 = +1\ mod\ 561$ → 합성수

소수 판정

- 확률적 소수 검증 알고리즘 (20/22)
  - 나눔 검증과 밀러-라빈 검증을 조합한 검증 방식
    - 방법
      1. 홀수 정수 선택(n)
      2. 알려진 소수에 대해 나눔 검증 기반의 1차 소수 판별
      3. 검증을 위해 밑수(a)로 사용할 수들의 집합 결정
         - 집합이 될 조건 : $2 \leq a < n-1, a$와 n은 서로소
      4. 밑수별로 밀러-라빈 검증, 한 단계라도 통과하지 못 한다면 1단계로 돌아가서 다른 홀수 선택
    - 필요성
      - 밀러-라빈은 확률적 검증이기 때문에 오탐이 발생할 수 있으므로 나눔 검증을 통해 작은 소수들로 나누어지는 합성수들을 배제함으로써 시간절약과 정확도 확보

소수 판정

- 확률적 소수 검증 알고리즘 (21/22)
  - 나눔 검증과 밀러-라빈 검증을 조합한 검증 방식
    - 예제 9.18 - 1

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

- 4033이하의 정수들 중 나눔 검증을 통해 $2, 3, 5, 7, 11, 17, 23$이 소수임을 확인
- 밑수를 2로 사용하여 밀러-라빈 검증을 실시
- $4033 - 1 = 63 \times 2^6$이므로 $m = 63,\ k = 6$

초기화 $\quad T \equiv 2^{63}\ (mod\ 4033) \equiv 3521 (mod\ 4033)$

$k = 1$: $\quad T \equiv T^2 \equiv 3521^2 (mod\ 4033) \equiv -1 (mod\ 4033) \rightarrow$ 소수 판별

소수 판정

- 확률적 소수 검증 알고리즘 (22/22)
  - 나눔 검증과 밀러-라빈 검증을 조합한 검증 방식
    - 예제 9.18 - 2
      - 2가 밑수인 경우가 오탐일 가능성에 대비해 3이 밑수인 경우도 계산

초기화 $T \equiv 3^{63} \ (mod\ 4033) \equiv 3551 (mod\ 4033)$

$k = 1:$ $T \equiv T^2 \equiv 3551^2 (mod\ 4033) \equiv 2443 (mod\ 4033)$
$k = 2:$ $T \equiv T^2 \equiv 2443^2 (mod\ 4033) \equiv 3442 (mod\ 4033)$
$k = 3:$ $T \equiv T^2 \equiv 3442^2 (mod\ 4033) \equiv 2443 (mod\ 4033)$
$k = 4:$ $T \equiv T^2 \equiv 2443^2 (mod\ 4033) \equiv 3442 (mod\ 4033)$
$k = 5:$ $T \equiv T^2 \equiv 3442^2 (mod\ 4033) \equiv 2443 (mod\ 4033)$ → 실패(합성수)

목 차

- 소수(Primes)
- 소수 판정
- 지수와 로그

지수와 로그

- 지수(Exponentiation) (1/6)
  - 정의
    - 거듭제곱에서 밑을 곱해야 하는 횟수를 나타내는 숫자
  - 표현
    - $y = a^x$
    - x를 지수, a를 밑이라고 명명
  - 성질
    - 지수와 로그는 역 연산 관계($y = a^x \leftrightarrow x = \log_a y$)
    - $a^x \times a^y = a^{x+y}$
    - $(a^x)^y = a^{x \times y}$

지수와 로그

- 지수(Exponentiation) (2/6)
  - 고속 지수 계산 (1/5)
    - 제곱-곱 방법(Square-and-Multiply Method)
      - 지수(x)는 n_b비트(x_0에서 x_{n_b-1})의 2진수로 간주
        - n_b비트는 지수를 2진수로 나타내었을 때의 자리 수
        - 일반적으로 아래와 같이 표현

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

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

지수와 로그

- 지수(Exponentiation) (3/6)
  - 고속 지수 계산 (2/5)
    - 방법
      - 제곱-곱 방법을 $y = a^x$에 적용

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

$y = a^x$

$$y = \boxed{a^{2^{n_b-1}} \text{ or } 1} \times \boxed{a^{2^{n_b-2}} \text{ or } 1} \times \cdots \times \boxed{a^2 \text{ or } 1} \times \boxed{a^1 \text{ or } 1}$$

- 지수를 2진법으로 표현
- 밑수를 계속해서 제곱한 후 비트에 따라 계산
  - 해당 비트가 1인 경우, 제곱수 표현
  - 해당 비트가 0인 경우, 1로 표현
- e.g., $y = a^9 = a^{1001_2} = a^8 \times 1 \times 1 \times a$

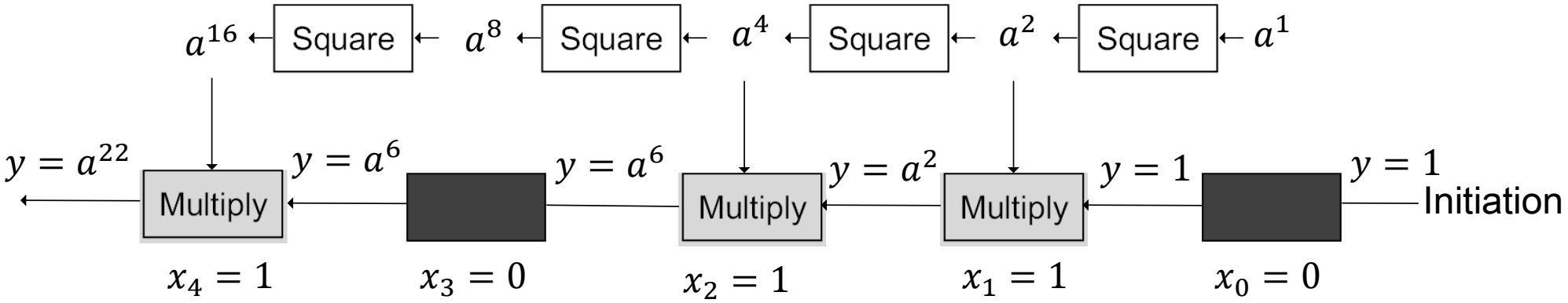
지수와 로그

- 지수(Exponentiation) (4/6)
  - 고속 지수 계산 (3/5)
    - 알고리즘 의사코드(이진수 구하는 과정 제외)

```
Square_and_Multiply (a, x, n_b)
{
  y ← 1                          // y를 1로 초기화
  for(i ← 0 to n_b − 1)          // 1부터 비트 수 −1 까지 진행
  {
    if(x_i = 1)  y ← a × y       // 비트가 1인 경우에는 y에 a를 곱한다
     a ← a^2                     // a를 제곱하면서 반복한다
  }
  return y
}
```

지수와 로그

- 지수(Exponentiation) (5/6)
  - 고속 지수 계산 (4/5)
    - 알고리즘 의사코드를 이용하여 $y = a^x (x = 22)$를 구하는 과정



- $x = 22 = (10110)_2$
- 검은 색으로 된 사각형이 의미하는 것은 곱셈이 생략된 곳
- 이전 단계의 y값이 다음 단계로 그대로 넘어가는 것을 확인 가능

$\therefore a^{22} = a^{16} \times 1 \times a^4 \times a^2 \times 1$

지수와 로그

- 지수(Exponentiation) (6/6)
  - 고속 지수 계산 (5/5)
    - 예제 9.19

$8^{15} mod\ 17$을 고속 지수 계산을 이용하여 구해보자

$8^8 mod\ 17 = 16^2 mod\ 17 = 1$
$8^4 mod\ 17 = 13^2 mod\ 17 = 16$
$8^2 mod\ 17 = 13$
$8\ mod\ 17 = 8$

- $15 = (1111)_2,\ 8^{15} = 8^8 \times 8^4 \times 8^2 \times 8^1$
- $8^{15} mod\ 17 = (8^8 \times 8^4 \times 8^2 \times 8^1)\ mod\ 17$
  $= (1 \times 16 \times 13 \times 8)\ mod\ 17$
  $= (16 \times 13) mod\ 17\ \times (1 \times 8)\ mod\ 17$
  $= 32\ mod\ 17$
  $= 15$

지수와 로그

- 로그(Logarithm) (1/18)
  - 정의
    - 어떤 양수를 만들기 위해 밑이 되는 수를 몇 번 곱해야 하는지를 나타낸 것
  - 표현
    - $x = \log_a y (a > 0, a \neq 1, y > 0)$
  - 성질
    - 로그와 지수는 역 연산 관계($x = \log_a y \leftrightarrow y = a^x$)
    - $\log_a 1 = 0,\ \log_a a = 1$
    - $\log_a MN = \log_a M + \log_a N$
    - $\log_a \frac{M}{N} = \log_a M - \log_a N$
    - $\log_a L^k = k \log_a L$

지수와 로그

- 로그(Logarithm) (2/18)
  - 지수의 역 연산 알고리즘
    - $y \equiv a^x mod\ n \leftrightarrow x \equiv \log_a y\ mod\ n$
      - $y = a^x mod\ n$을 연속적으로 계산하여 주어진 y값을 찾을 때까지 수행하는 알고리즘
      - 비트 연산 복잡도
        - $O(2^{n_b})$
    - 역 역산에 대한 전수조사 알고리즘 의사코드

```
Modular_Logarithm(a, y, n)
{
  for(x = 1 to n − 1)                  //1부터 n − 1까지 x값 시도
  {
    if(y ≡ a^x mod n) return x         // a^x mod n이 y랑 같으면 x 확정
  }
  return failure
}
```

지수와 로그

- 로그(Logarithm) (3/18)
  - 곱셈군에 대한 성질 (1/11)
    - 유한 곱셈군(Multiplicative Finite Group)
      - 유한한 개수의 원소로 이루어진 집합이면서 곱셈 연산에 대해 군의 네 가지 조건을 모두 만족하는 집합
    - 군의 크기
      - 유한군 G에 속하는 원소의 개수
      - $|G|$로 표기
      - 오일러의 φ함수를 이용해 크기 판정
    - 예제 9.20

군 $G =< Z_{21}^*, \times >$ 의 크기를 구하여라

$|G| = \varphi(21) = \varphi(3) \times \varphi(7) = 2 \times 6 = 12$
군의 원소는$\{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$

지수와 로그

- 로그(Logarithm) (4/18)
  - 곱셈군에 대한 성질 (2/11)
    - 원소의 위수*
      - a를 몇 번 곱해야 항등원을 얻는지 나타내는 가장 작은 양의 정수
      - $G = < Z_n^*, \times >$에서 원소 a의 위수는 $a^i \equiv e(mod\ n)$이 되는 정수중에서 가장 작은 정수 i(i는 군의 크기의 약수)
        - a는 곱셈군의 원소
        - 항등원 e는 1

*위수(Order)
군에서 $a^i = e$를 만족하는 가장 작은 정수 i를 원소 a의 위수라고 함
$ord(a) = i$로 표기

지수와 로그

- 로그(Logarithm) (5/18)
  - 곱셈군에 대한 성질 (3/11)
    - 원소의 위수
      - 예제9.21

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

- 이 군의 원소는 $\varphi(10) = \{1,3,7,9\}$, $|G| = 4$
- 4의 약수인 $1, 2, 4$로 원소의 위수를 계산하기 위해 3가지 수에 대해서만 거듭 제곱수를 계산

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

- 로그(Logarithm) (6/18)
  - 곱셈군에 대한 성질 (4/11)
    - 오일러 정리
      - 만약 a가 군 $G = < Z_n^*, \times >$의 원소라면 $a^{\varphi(n)} \equiv 1 \, mod \, n$
      - 예제 9.22

표가 군 $G = < Z_8^*, \times >$에 대한 $a^i \equiv 1 \, mod \, 8$의 값을 나타낼 때, 각 원소의 위수를 구하시오
($\varphi(8) = 4$이고 원소는 $1, 3, 5, 7$이라는 점에 유의)

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

오일러 정리 적용 결과

- $i = \varphi(8) = 4$일때 모든a에 대해서 $a^4 mod \, 8 = 1$
- 여러 개의 i값에 대해 x값은 1
  $\therefore$ 각 원소의 위수는
  $ord(1) = 1, \; ord(3) = 2$
  $ord(5) = 2, \; ord(7) = 2$

지수와 로그

- 로그(Logarithm) (7/18)
  - 곱셈군에 대한 성질 (5/11)
    - 원시근(Primitive Root)
      - 정의
        - 군 $G = < Z_n^*, \times >$에서 한 원소의 위수가 $\varphi(n)$과 동일한 원소
      - 특징
        - 군 $G = < Z_n^*, \times >$는 오직 n이 $2, 4, p^t, 2p^t$일 때만 원시근을 가짐
          - p는 2가 아닌 소수
          - t는 양의 정수
        - 한 군이 한 개의 원시근을 가지는 경우, 그 군에는 두 개 이상의 원시근이 존재
        - $G = < Z_n^*, \times >$가 원시근을 가지는 경우 원시근의 개수는 $\varphi(\varphi(n))$개

지수와 로그

- 로그(Logarithm) (8/18)
  - 곱셈군에 대한 성질 (6/11)
    - 원시근(Primitive Root)
      - 예제 9.23

군 $G = < Z_8^*, \times >$의 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
군 $G = < Z_8^*, \times >$은 원시근을 가지지 않음

지수와 로그

- 로그(Logarithm) (9/18)
  - 곱셈군에 대한 성질 (7/11)
    - 원시근(Primitive Root)
      - 예제 9.24

아래 표는 군 $G = < Z_7^*, \times >$에 대한 $a^i \equiv x \ mod \ 7$의 계산 값이다
군 $G = < Z_7^*, \times >$의 원시근을 구하시오

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

군의 크기는 $\varphi(7) = 6$

원소의 위수는
$ord(1) = 1 \quad ord(2) = 3 \quad ord(3) = 6$
$ord(4) = 3 \quad ord(5) = 6 \quad ord(6) = 1$

오직 두 원소 3과 5만이 $i = \varphi(n) = 6$에서 위수 6을 가짐

이 군의 2개의 원시근 3과 5를 가짐

지수와 로그

- 로그(Logarithm) (10/18)
  - 곱셈군에 대한 성질 (8/11)
    - 원시근(Primitive Root)
      - 예제 9.25

$n = 17, 20, 38, 50$ 중 어떤 n에 대해서 군 $G = < Z_n^*, \times >$이 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
  $38 = 2^1 \times 19^1$이고 19는 소수이기 때문임$(p = 19,\ t = 1)$
- $G = < Z_{50}^*, \times >$은 원시근을 가짐
  $50 = 2^1 \times 5^2$이고 5는 소수이기 때문임$(p = 5, t = 2)$

지수와 로그

- 로그(Logarithm) (11/18)
  - 곱셈군에 대한 성질 (9/11)
    - 순환군(Cyclic Group)
      - 정의
        - 한 원소 g를 반복해서 군 연산으로 적용해 생성할 수 있는 군
      - 특징
        - 만약 군 $G = < Z_n^*, \times >$이 원시근을 갖는다면 해당 군은 순환 군
          - 원시근 g에 대해, 군 전체 집합 Z_n^*은 아래와 같음

$$Z_n^* = \{g^1,\ g^2,\ g^3,\ \cdots,\ g^{\varphi(n)}\}$$

        - 만약 p가 소수라면 군 $G = < Z_p^*, \times >$은 항상 순환 군

지수와 로그

- 로그(Logarithm) (12/18)
  - 곱셈군에 대한 성질 (10/11)
    - 순환군(Cyclic Group)
      - 예제 9.26

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

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

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

3과 7은 위수가 4이므로 군 전체를 생성, Z_{10}^*의 모든 원소인 $\{1, 3, 7, 9\}$를 생성함
$\therefore Z_{10}^*$은 순환군

지수와 로그

- 로그(Logarithm) (13/18)
  - 곱셈군에 대한 성질 (11/11)
    - 군 $G = < Z_p^*, \times >$의 성질(p는 소수)
      - 해당 군의 원소는 1부터 $p-1$까지의 모든 수
      - 해당 군은 항상 원시근을 보유
      - 해당 군은 순환군이고 원소들은 g^x를 사용하여 생성 가능 (x는 1부터 $\varphi(n) = p-1$안의 정수)
      - 원시근은 로그의 밑수로 활용 가능
        - 주어진 군이 k개의 원소를 가진다고 하면 k개의 서로 다른 밑수로 계산 가능
        - 집합의 한 원소 y에 대해 $x = \log_g y$일 경우, g를 밑수로 갖는 y의 로그인 새로운 원소 x가 존재

지수와 로그

- 로그(Logarithm) (14/18)
  - 이산 대수
    - 개요
      - 정수와 소수에 대해 모듈러 지수 연산을 수행하는 것

일반 수학에서 로그와 지수

$$y = a^x \leftrightarrow x = \log_a y$$

이산 대수와 이산 로그

$$a^x \equiv b \bmod n \leftrightarrow \log_a b \bmod n \equiv x$$

지수와 로그

- 로그(Logarithm) (15/18)
  - 이산 대수
    - 성질
      - $a^x \ mod \ n$의 연산에 비해 역 역산인 $\log_a b \ mod \ n$는 계산이 복잡
      - 지수 법칙이 모듈러에서도 성립
        - $a^x \times a^y \equiv a^{x+y} mod \ n$
      - 이산 로그의 해는 위수만큼 주기성을 가지고 반복
      - Z_n*는 유한한 곱셈 군이고 생성원 a가 존재하면 $\{a^0, a^1, \cdots, a^{p-2}\}$는 모든 원소를 순환적으로 생성

로그 (Logarithm)	이산 로그 (Discrete Logarithm)
$\log_a 1 = 0$	$L_g 1 \equiv 0 (mod \ \varphi(n))$
$\log_a (x \times y) = \log_a x + \log_a y$	$L_g (x \times y) \equiv (L_g x + L_g y) (mod \ \varphi(n))$
$\log_a x^k = \mathrm{k} \times \log_a x$	$L_g x^k \equiv k \times L_g x (mod \ \varphi(n))$

지수와 로그

- 로그(Logarithm) (16/18)
  - 이산 대수를 이용한 모듈로 로그의 해 탐색 (1/3)
    - y가 주어졌을 때 $y = a^x (mod\ n)$에서 x를 구하는 법
    - 이산 대수의 도표화
      - 서로 다른 밑수별로 Z_n*에 대한 표 사용
      - 군 $G = < Z_7$*,× >에 대한 이산 대수표

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

지수와 로그

- 로그(Logarithm) (17/18)
  - 이산 대수를 이용한 모듈로 로그의 해 탐색 (2/3)

<군 $G = < Z_7^*, \times >$에 대한 이산 대수표>

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

  - 예제 9.27

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

위의 표를 이용하여 아래와 같이 계산할 수 있음
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

- 로그(Logarithm) (18/18)
  - 이산 대수를 이용한 모듈로 로그의 해 탐색 (3/3)
    - 이산 대수를 기초로 한 알고리즘 사용
      - n이 커질 경우 이산 대수 성질과 이산 대수표를 사용해 $y = a^x (mod\ n)$을 푸는 것은 불가능
        - 이산 대수 문제를 풀기 위해 기본적인 성질을 활용한 여러 가지 알고리즘들이 개발
        - e.g., 디피-헬만 키 교환(Diffie–Hellman Key Exchange)
      - 전수 조사 알고리즘보다는 효율성이 높으나 모두 소인수분해 복잡도 정도의 수준 유지

Thanks!

민 윤 홍(muh4316@gmail.com)