수식을 활용한 문제 해결

SW설계 문제에서 수학식의 중요성 느끼고 활용 방법 익히기

01. 문제에서 사용하는 정의 이해: totient maximum

02. 첫 번째 알고리즘

03. 두 번째 알고리즘

04. 정리 문제 풀이

05. 실습 문제 풀이 & 질의 응답

<왜 수학식의 활용 예를 보는가? >

- SW/HW 시스템 설계 시 수학식 활용하여 문제 해결되는 경우 많음
- 예: 지도상의 두 지점 간 거리 계산 위해 피타고라스 정리 사용
- 적절한 수학식 활용 → (i) 좋은 성능의 시스템 설계 + (ii) 설계 시간 단축
- 이번 주 문제 풀이하며 수학식의 중요성 느끼고 활용 방법 익힘

Copyright © by Sihyung Lee - All rights reserved.

예습자료에서 본 방법의 시간 복잡도 정리

- gcd(a,b): 평균 log(min(a,b))

```
def gcd3(a, b):
    while b != 0:
        if a > b: a, b = b, a % b
        else: b = b % a
    return a
```

- n을 소인수 분해: 평균 log(n)

```
def primeFactorization(n):
    result = []
    while n % 2 == 0:
        n /= 2
        result.append(2)

    p = 3
    while p*p <= n:
        while n % p == 0:
            n /= p
            result.append(p)
        p += 2

    if n > 2: result.append(int(n))

    return result
```

Copyright © by Sihyung Lee - All rights reserved.

문제 정의: totient maximum

양의 정수 n에 대한 함수 **φ(n)**은 오일러의 **totient(토우션트)[1] 함수** 혹은 **phi(파이) 함수**라 하며

"**n보다 작은 양의 정수 중 n과 서로소인 정수의 개수[2]**"를 나타낸다. 예를 들어 n = 8일 때 8보다 작은 양의 정수 {1,2,3,4,5,6,7} 중 4개의 정수 {1,3,5,7}이 8과 서로소이며, 따라서 φ(n=8) = 4이다.

자연수 입력 **N**이 주어졌을 때 ($3 \leq N \leq 10^{17}$),

범위 **1< n ≤N** 에 속하는 모든 정수 n 중 $\frac{n}{\varphi(n)}$이 **최대**인 n을 구하는 알고리즘을 제시하시오.

※ $\frac{n}{\varphi(n)}$ 값이 같은 n이 여럿이라면, 이중 **가장 작은 n**을 해로 선택하시오.

[1]totient: '토우션트'로 발음함. 'tot'는 'that many', 'so many'를 의미하며 총합을 의미하는 'total'에도 사용됨. RSA 등의 암호 시스템도 totient 함수를 사용함
[2]두 수의 최대공약수가 1일 때 이들을 서로소(relatively prime)라 함

Copyright © by Sihyung Lee - All rights reserved.

문제 풀이 과정

① **이번 시간 문제를 잘 이해**하기 위해 간단한 경우부터 시작해 문제 이해하고 답 이끌어내 보기

문제 명확히 이해하는데 도움

이 과정에서 문제에 대한 해법도 생각해 낼 수 있음

② 이번 시간 문제에 대한 **풀이 방법 생각**

③ 생각한 풀이 방법을 보다 **효율적으로 개선**

Copyright © by Sihyung Lee - All rights reserved.

양의 정수 n에 대한 함수 **φ(n)**은 오일러의 **totient(토우션트)[1] 함수** 혹은 **phi(파이) 함수**라 하며 **"n보다 작은 양의 정수 중 n과 서로소인 정수의 개수[2]"**를 나타낸다. 예를 들어 n = 8일 때 8보다 작은 양의 정수 {1,2,3,4,5,6,7} 중 4개의 정수 {1,3,5,7}이 8과 서로소이며, 따라서 φ(n) = 4이다.

(1) **φ(2)**를 계산해 보자. 2보다 작은 양의 정수 몇 개가 2와 서로 소 인가?

(2) **φ(3)**를 계산해 보자. 3보다 작은 양의 정수 몇 개가 3과 서로 소 인가?

(3) **φ(4)**를 계산해 보자. 4보다 작은 양의 정수 몇 개가 4와 서로 소 인가?

Copyright © by Sihyung Lee - All rights reserved.

(4) 아래의 표를 사용해 **n=2~11에 대해 φ(n)을 계산**해 보자. 2번째 열의 숫자들은 n보다 작은 양의 정수들이다. 이 중 n과 서로 소가 아닌 수는 제거하고 φ(n)을 구해보자. n = 2~4에 대한 해는 문제 (1)~(3)에서 구한 값을 복사해 사용하시오.

n	n보다 작은 양의 정수 (이 중 n과 서로소인 수만 남기시오)	φ(n)	n/φ(n)	i/φ(i)가 최대인 최소의 i(≤n) 값
2	1	1	2	2
3	1, 2	2	1.5	2
4	1, 2, 3	2	2	2
5	1, 2, 3, 4	4	1.25	2
6	1, 2, 3, 4, 5	2	3	6
7	1, 2, 3, 4, 5, 6	6	1.17	6
8	1, 2, 3, 4, 5, 6, 7	4	2	6
9	1, 2, 3, 4, 5, 6, 7, 8	6	1.5	6
10	1, 2, 3, 4, 5, 6, 7, 8, 9	4	2.5	6
11	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	10	1.1	6

Copyright © by Sihyung Lee - All rights reserved.

(5) 지금까지의 계산 결과를 보면 **n이 소수(prime number)라면** φ(n)은 무엇이라고 할 수 있을까?

$$\phi(n) = n - 1$$

Copyright © by Sihyung Lee - All rights reserved.

자연수 입력 **N**이 주어졌을 때 ($3 \le N \le 10^{17}$),

범위 **1< n ≤N** 에 속하는 모든 정수 n 중 $\frac{n}{\varphi(n)}$이 **최대**인 n을 구하는 알고리즘을 제시하시오.

※ $\frac{n}{\varphi(n)}$ 값이 같은 n이 여럿이라면, 이중 **가장 작은 n**을 해로 선택하시오.

(6) 표의 3번째 열 n/φ(n) 값을 모두 계산해 보자. 이번 문제의 목표가 **n/φ(n)를 최대로 하는 최소의 n 값**을 구하는 것임을 기억하시오. 휴대폰/PC의 계산기를 활용 하시오.

(7) **N=2일 때**
1< n ≤N 범위의 정수 n 중 n/φ(n)를 최대로 하는 최소의 n 값은?

(8) **N=3일 때**
1< n ≤N 범위의 정수 n 중 n/φ(n)를 최대로 하는 최소의 n 값은?

(9) **N=4일 때**
1< n ≤N 범위의 정수 n 중 n/φ(n)를 최대로 하는 최소의 n 값은?

Copyright © by Sihyung Lee - All rights reserved.

(10) 표의 마지막 열의 값을 모두 구해 보자. n=2~4에 대한 해는 문제 (7)~(9)에서 구한 값을 복사해 사용하시오. 값을 구하면서 **이번 주 문제의 해는 어떤 특성을 갖는 값일지 생각해 보시오**.

(11) **N≥3**인 경우에 대해 생각해 보자. **소수(prime number)**가 이번 주 문제에 대한 해가 될 수 있을까? 예 또는 아니요로 답하고 **이유**도 설명하시오.

아니요…소수의 ϕ(n)값은 매우 크기 때문에 결과론적으로 결과 값이 작다

Copyright © by Sihyung Lee - All rights reserved.

(12) 이번 주 문제에 대한 해를 구할 **<u>알고리즘을 제시</u>**해 보시오 - 즉 입력 N이 주어졌을 때, **1< n ≤N** 범위에 속한 정수 n 중 n/φ(n) 값을 최대로 하는 최소의 n 값을 구하는 알고리즘을 생각해 보시오.

```
for(int i = 0; i < n; i++){
    int temp = ϕ(i) / i;
    if(answer < temp)
     answer = temp;
            }
```

(13) 제안한 방법의 복잡도는 어느 정도인가? 이 정도의 복잡도라면 N이 10^{17} 정도로 클 때도 거의 즉시 해를 구할 수 있을까? gcd(a,b) 계산은 거의 constant time이 소요된다고 가정 하시오. ~log(min(a,b))

Copyright © by Sihyung Lee - All rights reserved.

첫 번째 알고리즘

```
def totientMaximum1(N):
    nOverPhiMax = 0
    nMax = 0
    for n in range(2,N+1):
        phi = phiUsingGCD(n)
        nOverPhi = n / phi
        if nOverPhiMax < nOverPhi:
            nOverPhiMax = nOverPhi
            nMax = n
    return nOverPhiMax, nMax
```

```
def phiUsingGCD(n):
    phi = 0
    for i in range(1,n):
        if gcd3(i,n) == 1: phi += 1
    return phi
```

```
def gcd3(a, b):
    while b != 0:
        if a > b: a, b = b, a % b
        else: b = b % a
    return a
```

```
if __name__ == "__main__":
    functions = [totientMaximum1]
    for f in functions:
        for i in range(2,12):
            print(i, f(i), end=' ')
        print()
```

(14) 먼저 위 코드가 무엇을 하는지 앞에서 만든 표와 비교하며 이해해 보시오. 이해한 후에는 이를 실행해 보자. 이 함수는 TotientMaximum.py에 있다. 먼저 **N=2~11 사이의 값에 대해 계산한 값이 올바른지** 표와 비교하여 **확인**해 보시오.

(15) __main__ 아래 코드를 사용해 N=10, 100, 1000, 10000 각각에 대해 수행 시간을 측정해 보자. 사람이 느끼기에 거의 즉시 답을 얻을 수 있는 정도인가? 혹은 그렇지 않아 시간을 단축해야 하는가? 문제에서 요구하는 N의 최댓값은 10^{17}임을 고려 하시오.

```
if __name__ == "__main__":
    functions = [totientMaximum1]
    for f in functions:
        for i in range(2,12):
            print(i, f(i), end=' ')
        print()

        n, repeat = 1000, 10
        tTotientMaximum = timeit.timeit(lambda: f(n), number=repeat)/repeat
        print(f"{f.__name__}({n}) took {tTotientMaximum} seconds on average")
```

Copyright © by Sihyung Lee - All rights reserved.

첫 번째 알고리즘의 Bottleneck


```
def totientMaximum1(N):
    nOverPhiMax = 0
    nMax = 0
    for n in range(2,N+1):
        phi = phiUsingGCD(n)
        nOverPhi = n / phi
        if nOverPhiMax < nOverPhi:
            nOverPhiMax = nOverPhi
            nMax = n
    return nOverPhiMax, nMax
```

```
def phiUsingGCD(n):
    phi = 0
    for i in range(1,n):
        if gcd3(i,n) == 1: phi += 1
    return phi
```

```
def gcd3(a, b):
    while b != 0:
        if a > b: a, b = b, a % b
        else: b = b % a
    return a
```

Copyright © by Sihyung Lee - All rights reserved.

오일러의 Totient 함수 $\varphi(n)$을 보다 효율적으로 구할 수 있는 수식

$$\varphi(n) = n \times \frac{p_1 - 1}{p_1} \times \frac{p_2 - 1}{p_2} \times \cdots \times \frac{p_k - 1}{p_k}$$

$p_1, p_2, \cdots, p_k$는 n의 **서로 다른** **소인수** (prime factor, 소수인 약수)

(16) 먼저 위 수식에 작은 n 값을 대입해 풀어보며 의미와 계산법을 이해해 보자.
위 식을 이용해 φ(2)를 다시 계산해 본 후, 앞에서 φ(n) 정의에 따라 계산한 값과 비교해 보시오.

$\phi(2) = 2$

Copyright © by Sihyung Lee - All rights reserved.

오일러의 Totient 함수 $\varphi(n)$을 보다 효율적으로 구할 수 있는 수식

$$\varphi(n) = n \times \frac{p_1 - 1}{p_1} \times \frac{p_2 - 1}{p_2} \times \cdots \times \frac{p_k - 1}{p_k}$$

$p_1, p_2, \cdots, p_k$는 n의 서로 다른 소인수 (prime factor, 소수인 약수)

(17) 위 수식을 사용해 φ(3)을 다시 계산해 보시오.

$\phi(3) = 3 * 2 / 3 = 2$

(18) 위 수식을 사용해 φ(4)를 다시 계산해 보시오.

$\phi(4) = 4 * 1/2 = 2$

Copyright © by Sihyung Lee - All rights reserved.

$$\varphi(n) = n \times \frac{p_1 - 1}{p_1} \times \frac{p_2 - 1}{p_2} \times \cdots \times \frac{p_k - 1}{p_k}$$

$p_1, p_2, \cdots, p_k$는 n의 서로 다른 소인수 (prime factor, 소수인 약수)

(19) 이제 위 수식을 사용해 n=5~11에 대해 φ(n)을 다시 계산해 보자. 이렇게 φ(n)을 직접 계산해 보는 과정은 중요하다. 이를 몇 번 반복해 수식에 익숙해지면 이를 활용한 알고리즘을 생각하는데 도움이 된다.

phi(5) = 5 * 4/5 = 4

phi(6) = 6 * 2 / 3 * 1 / 2 = 2

phi(7) = 7 * 6 / 7 = 6

phi(8) = 8 * 1 / 2 = 4

phi(9) = 9 * 2 / 3 = 6

phi(10) = 10 * 4 / 5 * 1 / 2 = 4

phi(11) = 11 * 10 / 11 = 10

Copyright © by Sihyung Lee - All rights reserved.

(20) 방금 본 수식을 활용해 첫 번째 알고리즘을 개선한 알고리즘을 제시해 보시오. 이번 시간 목표가 '입력 N이 주어졌을 때, $1 < n \leq N$ 범위의 정수 n 중 $n/\varphi(n)$을 최대로 하는 최소의 n 값을 구하는 것' 임을 기억 하시오.

(21) 제안한 방법의 시간 복잡도는 어느 정도인가? 이 정도의 복잡도라면 N이 10^{17} 정도로 클 때도 거의 즉시 해를 구할 수 있을까? 숫자 i에 대한 소인수 분해는 대략 log(i)에 비례한 시간이 소요된다고 보시오.

Copyright © by Sihyung Lee - All rights reserved.

두 번째 알고리즘

```
def totientMaximum2(N):
    nOverPhiMax = 0
    nMax = 0
    for n in range(2,N+1):
        phi = phiUsingPrimeFactorization(n)
        nOverPhi = n / phi
        if nOverPhiMax < nOverPhi:
            nOverPhiMax = nOverPhi
            nMax = n
    return nOverPhiMax, nMax
```

```
if __name__ == "__main__":
    functions = [totientMaximum1, totientMaximum2]
    for f in functions:
        for i in range(2,12):
            print(i, f(i), end=' ')
        print()
```

```
def phiUsingPrimeFactorization(n):
    phi = n

    if n % 2 == 0:
        phi = phi * (2 - 1) / 2
        while n % 2 == 0: n /= 2

    p = 3
    while p*p <= n:
        if n % p == 0:
            phi = phi * (p - 1) / p
            while n % p == 0: n /= p
        p += 2

    if n > 2: phi = phi * (n - 1) / n

    return phi
```

(22) 먼저 위 코드가 무엇을 하는지 이해해 보시오. 이해한 후에는 이를 실행해 보자. 이 함수는 TotientMaximum.py에 있다. 먼저 **N=2~11 사이의 값에 대해 계산한 값이 올바른지** 표와 비교하여 **확인**해 보시오.

Copyright © by Sihyung Lee - All rights reserved.

(23) N=1,000, 10,000, 100,000, 1,000,000 각각에 대해 수행 시간을 측정하고 문제 (15)에서 첫 번째 알고리즘을 실행하며 측정했던 시간과 비교해 보자. 이번에 측정한 시간은 사람이 느끼기에 거의 즉시 답을 얻을 수 있는 정도인가? 혹은 그렇지 않아 시간을 단축해야 하는가? 문제에서 요구하는 N의 최댓값은 10^{17}임을 고려 하시오.

```
if __name__ == "__main__":
    functions = [totientMaximum2]
    for f in functions:
        for i in range(2,12):
            print(i, f(i), end=' ')
        print()

        n, repeat = 1000, 10
        tTotientMaximum = timeit.timeit(lambda: f(n), number=repeat)/repeat
        print(f"{f.__name__}({n}) took {tTotientMaximum} seconds on average")
```

Copyright © by Sihyung Lee - All rights reserved.

(24) 이번 시간 문제의 목표는 **φ(n)** 보다는 **n/φ(n) 과 직접적으로 관련**되어 있다. 이번 주 목표가 '입력 N이 주어졌을 때, 2 ~ N 범위 정수 n 중 **n / φ(n)을 최대로 하는 최소의 n 값**을 구하는 것' 임을 기억하자. 따라서 **φ(n) 에 대한 수식을** 사용해 먼저 **n / φ(n)을 수식 형태로 다시 써 보자.**

$$\varphi(n) = n \times \frac{p_1 - 1}{p_1} \times \frac{p_2 - 1}{p_2} \times \cdots \times \frac{p_k - 1}{p_k}$$

$p_1, p_2, \cdots, p_k$는 n의 서로 다른 소인수 (prime factor, 소수인 약수)

n / phi(n) = p1 * p2 * p3 … pk / p1 -1 p2 - 1….

이제부터는 **어떤 특성을 갖는 수 n이 n / φ(n)을 최대로 만드는지** 조금씩 생각해 보자. 문제 (10)에서도 이러한 특성에 대해 대략적으로 생각해 보았지만 (예: 인수가 많을수록 유리), 지금은 n / φ(n)에 대한 수식을 갖고 있으므로 더 정확하게 특성을 짚어낼 수 있다.

Copyright © by Sihyung Lee - All rights reserved.

(25) 두 개의 소인수 $p_a < p_b$ 중 어느 쪽이 $n / \varphi(n)$을 더 크게 만드는가? 다시 말해 단 하나의 소인수만 다르며 다른 모든 소인수는 동일한 두 숫자 $n_a = p_a \times p_1 \times \ldots \times p_k$ 와 $n_b = p_b \times p_1 \times \ldots \times p_k$ 중 어느 쪽의 $n / \varphi(n)$ 이 더 큰가?

(26) $n = p_1 \times p_2 \times \ldots \times p_k$ 에서 소인수 $p_2, p_3, \ldots, p_k$은 고정되어 있고 첫 번째 소인수 p_1만 선택할 수 있다면 $n / \varphi(n)$을 최대화하기 위해 어떤 값을 선택하겠는가?

(27) $n = p_1 \times p_2 \times p_3 \times \ldots \times p_k$ 에서 소인수 $p_3, p_4, \ldots, p_k$은 고정되어 있고 첫 두 소인수 p_1, p_2 만 선택할 수 있다면 $n / \varphi(n)$을 최대화하기 위해 어떤 값을 선택하겠는가? $\varphi(n)$의 방정식에서 $p_1, p_2, \ldots p_k$는 n의 **서로 다른** 소인수임을 기억 하시오.

Copyright © by Sihyung Lee - All rights reserved.

(28) 2 ~ N 범위의 정수 n 중 n / φ(n)을 최대화하는 값을 바로 찾아낼 수 있겠는가? 아래 표를 참조하시오..

n	소인수 분해	서로 다른 소인수	n/φ(n)	i / φ(i) 최대화하는 최소의 i≤n
2	2	2	2/1 = 2	
3	3	3	3/2 = 1.5	
4	2^2	2	2/1 = 2	
5	5	5	5/4 = 1.25	
6	2×3	2, 3	2/1 × 3/2 = 3	
7	7	7	7/6 = 1.1666...	
8	2^3	2	2/1 = 2	
9	3^2	3	3/2 = 1.5	
10	2×5	2, 5	2/1 × 5/4 = 2.5	
11	11	11	11/10 = 1.1	
12	2^2×3	2, 3	2/1 × 3/2 = 3	
13	13	13	13/12 = 1.0833...	
14	2×7	2, 7	2/1 × 7/6 = 2.33...	
15	3×5	3, 5	3/2 × 5/4 = 1.875	
16	2^4	2	2/1 = 2	
17	17	17	17/16 = 1.0625	
18	2×3^2	2, 3	2/1 × 3/2 = 3	
19	19	19	19/18 = 1.055...	
20	2^2×5	2, 5	2/1 × 5/4 = 2.5	
21	3×7	3, 7	3/2 × 7/6 = 1.75	
22	2×11	2, 11	2/1 × 11/10 = 2.2	
23	23	23	23/22 = 1.045..	
24	2^3×3	2, 3	2/1 × 3/2 = 3	
25	5^2	5	5/4 = 1.25	
26	2×13	2, 13	2/1 × 13/12 = 2.16...	
27	3^3	3	3/2 = 1.5	
28	2^2×7	2, 7	2/1 × 7/6 = 2.33...	
29	29	29	29/28 = 1.03...	
30	2×3×5	2, 3, 5	2/1 × 3/2 × 5/4 = 3.75	

Copyright © by Sihyung Lee - All rights reserved.

(29) N = 50일 때, 2 ~ N 범위의 정수 n 중 n / φ(n)을 최대화하는 최소의 n 값은 무엇인가?

다음 연속된 소인수를 활용 하시오: {2, 3, 5, 7, 11, 13, 17, 19, 23}. 휴대폰 or PC의 계산기를 활용하시오.

(30) N = 500일 때, 2 ~ N 범위의 정수 n 중 n / φ(n)을 최대화하는 최소의 n 값은 무엇인가?

(31) N = 1,000,000일 때, 2 ~ N 범위의 정수 n 중 n / φ(n)을 최대화하는 최소의 n 값은 무엇인가?

Copyright © by Sihyung Lee - All rights reserved.

(32) 문제 (29)~(31)의 해를 구하는데 사용한 방법의 시간 복잡도는 어느 정도인가?

이 정도의 복잡도라면 N이 10^{17} 정도로 클 때도 거의 즉시 해를 구할 수 있을까?

Copyright © by Sihyung Lee - All rights reserved.

수도 코드 써보기: 입력 N이 하나 이상 들어올 수 있으며, 이들 모두에 대한 답을 리스트에 담아 반환 필요

Copyright © by Sihyung Lee - All rights reserved.

<정리 문제>

(33) 이번 시간 문제를 뒤집어 생각해 보자. 입력 N이 주어졌을 때, 2 ~ N 범위의 정수 n 중 n / φ(n)을 최대가 아닌 **최소**로 하는 n 값을 구하고 싶다고 하자. 어떤 방법을 제안하겠는가?

(34) N = 15일 때 2 ~ N 범위의 정수 n 중 n / φ(n)을 최소로 하는 n 값은 무엇인가? 다음 연속된 소수를 활용하시오: {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}.

(35) N = 50일 때 2 ~ N 범위의 정수 n 중 n / φ(n)을 최소로 하는 n 값은 무엇인가?

Copyright © by Sihyung Lee - All rights reserved.

<정리>

- SW/HW 시스템을 설계하다 보면 기존에 누군가가 풀어 둔 수학식을 활용하여 문제를 해결할 수 있는 경우가 많다. 그 때 그 때 **적절한 수학식을 잘 찾아서 활용**한다면 **좋은 성능의 시스템**을 만들 수 있을 뿐 아니라 **시스템을 설계하는데 걸리는 시간도 단축**할 수 있습니다. 따라서 수학식을 활용하는 문제를 푼다면 반드시 먼저 검색해서 이를 효율적으로 계산할 수 있는 형태로 풀어 두었는지 확인해보자. 수학식을 활용하는지 확실하지 않은 문제더라도 원하는 해가 수학식 형태로 이미 제공되고 있는지 확인해 보자.

- 이번 주 문제에서도 φ(n)에 대한 수식이 있었기에 첫 번째 알고리즘의 시간을 단축한 두 번째 알고리즘을 얻을 수 있었고, 또한 이를 더 개선할 방법을 생각할 수 있었다.

효율적으로 알고리즘을 개선하는데 '끈질김' + '관찰'은 항상 중요

Copyright © by Sihyung Lee - All rights reserved.

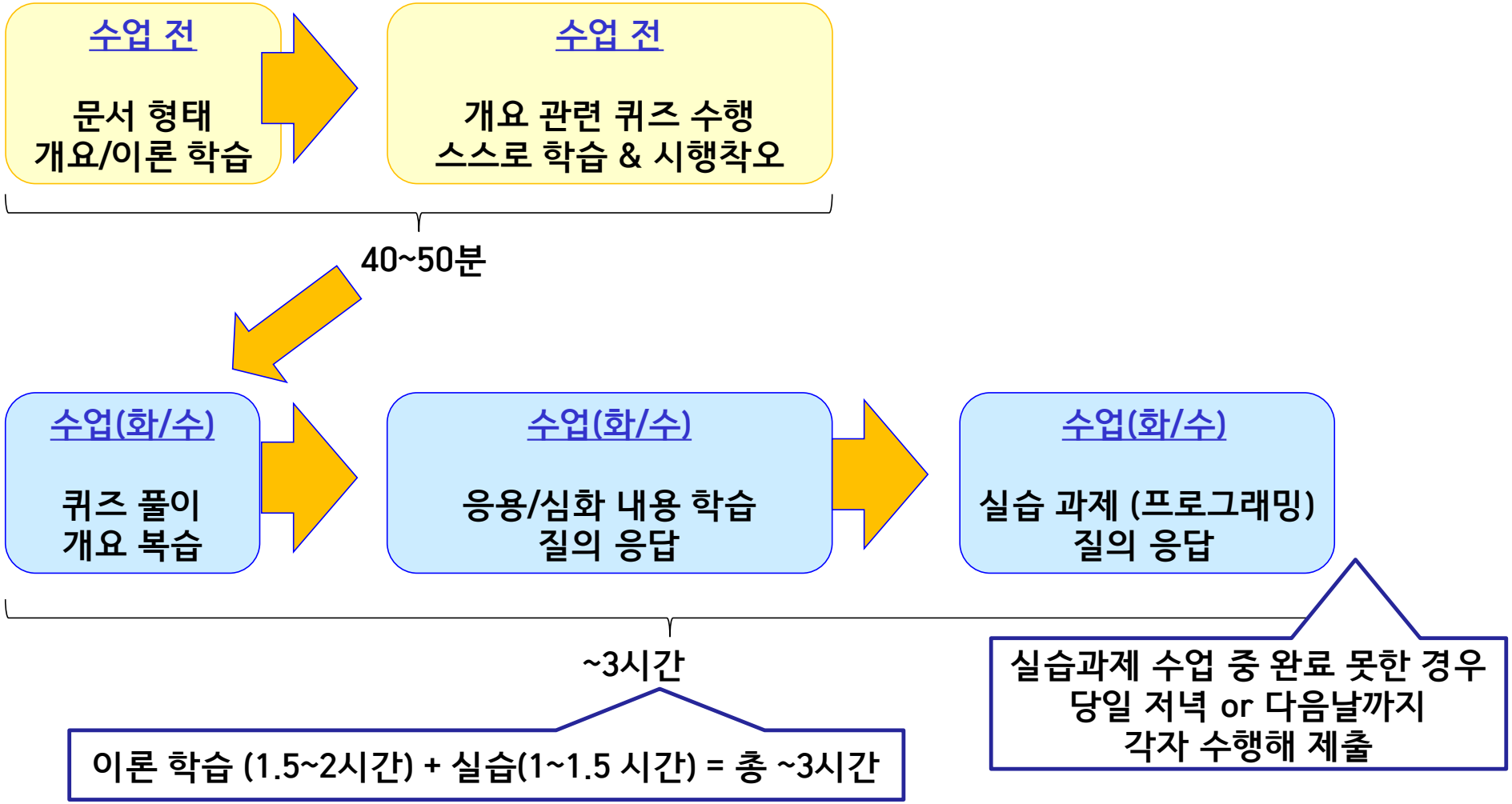
<정리>

- 이번 시간 문제도 solution space를 탐색하며 해를 찾는 문제이다. 이러한 문제를 풀 때는 가능하면 불필요한 탐색을 제거해 탐색하는 범위를 줄여가는 것이 좋다. 이 때 탐색 범위를 최소로 만드는 것은 **직접 해로가는 방법**이다 (즉 탐색하는 것 마다 100% 문제에 대한 해인 경우임). 이번 시간 문제에서도 입력 N보다 작은 모든 정수를 탐색하는 방법에서 시작하였으나 최종적으로는 직접 해를 찍어내는 방법으로 최적화하였다. 기존에도 나눗셈 tree 대신 곱셈 tree를 사용하면서 탐색 범위를 최소화하였음을 기억하자. 의외로 많은 경우에 이와 같이 탐색 범위를 최소화하는 것이 가능하므로 탐색 문제에서는 직접 해로가는 방법 혹은 그와 근접한 방법이 있는지 생각해 보자.

- 여러 다른 값에 대한 **데이터 시트 (표)**를 만들어 우리가 원하는 답에서 보이는 **패턴을 관찰**하는 것도 효율적인 방법을 찾는 데 도움이 된다.

Copyright © by Sihyung Lee - All rights reserved.

스마트 출결



Copyright © by Sihyung Lee - All rights reserved.

다음시간 & 중간고사 관련 공지

- 화요일 분반
- 다음시간(4.14): 시험 관련 공지 후 질의 응답; 예습/퀴즈 없음
- 질문 준비해 오세요.

- 다다음시간(4.21): 중간고사

- 중간고사 다음 수업(4.28)부터는 다시 예습/퀴즈 시작

Copyright © by Sihyung Lee - All rights reserved.

05. 실습문제풀이 & 질의응답

- **이번 시간에 배운 내용에 대한 실습 문제 풀이 & 질의 응답**
- **채점 방식은 지난 시간 문제 풀이때와 유사함**

- **왜 중요한가?**
- 이번 주 배운 내용을 총괄하는 문제 풀이 통해 배운 내용 활용 & 복습
- 문제 풀이 점수는 이번 주 과제 점수에 포함됨

Copyright © by Sihyung Lee - All rights reserved.

totientMaximum3(*Ns) 함수 구현 조건: totient maximum 탐색 코드 작성

- 하나 이상의 자연수 $N_1, N_2, .. N_k$ 가 입력으로 주어졌을 때, 각 N_i에 대해 2 ~ N_i에 속하는 정수 n 중 n / φ(n) 을 최대로 만드는 가장 작은 n을 구해반환

  def totientMaximum3 (*Ns):

- 입력 *Ns: 쉼표로 구분된 하나 이상의 자연수 $N_1, N_2, .. N_k$ 로, 각 N_i는 $2 \le n \le 10^{17}$ 범위의 정수
  - 위 범위를 벗어나는 값은 입력으로 들어오지 않는다고 가정 (즉 오류 처리 하지 않아도 됨)
  - 최대 1,000개까지의 입력이 들어올 수 있음

- 반환 값: 각 N_i에 대한 해를 담은 리스트
  - 입력 $N_1, N_2, .. N_k$ 에 대한 답이 순서대로 담겨 있어야 함

- 이번 시간에 제공한 코드 TotientMaximum.py의 totientMaximum3 () 함수 내에 코드 작성해 제출

Copyright © by Sihyung Lee - All rights reserved.

입출력 예: 입력으로 들어온 각 N_i에 대해 2 ~ N_i에 속하는 정수 n 중 n / φ(n)을 최대로 만드는 최소의 n 값을 리스트에 담아 반환

```
print(totientMaximum3(2,3,4,5,6,7,8,9,10,11))
```

```
[2, 2, 2, 2, 6, 6, 6, 6, 6, 6]
```

```
print(totientMaximum3(50, 500, 1000000))
```

```
[30, 210, 510510]
```

```
print(totientMaximum3(10000000,1000000000))
```

```
[9699690, 223092870]
```

```
print(totientMaximum3(10000000000,1000000000000,100000000000000))
```

```
[6469693230, 200560490130, 7420738134810]
```

```
print(totientMaximum3(1000000000000000,10000000000000000,100000000000000000))
```

```
[304250263527210, 304250263527210, 13082761331670030]
```

그 외 예제는 __main__ 아래 테스트 코드를 참조하세요.

Copyright © by Sihyung Lee - All rights reserved.

totientMinimum(*Ns) 함수 구현 조건:
2 ~ N에 속하는 정수 n 중 n / φ(n) 을 최소로 하는 n 탐색 코드 작성

- 하나 이상의 자연수 $N_1, N_2, .. N_k$ 가 입력으로 주어졌을 때, 각 N_i에 대해 2 ~ N_i에 속하는 정수 n 중 n / φ(n) 을 최소로 만드는 n을 구해반환

  def totient**Mini**mum (***Ns**):

- 입력 ***Ns**: 쉼표로 구분된 하나 이상의 자연수 $N_1, N_2, .. N_k$ 로, 각 N_i는 $2 \leq n \leq 10^7$ 범위의 정수
  - 위 범위를 벗어나는 값은 입력으로 들어오지 않는다고 가정 (즉 오류 처리 하지 않아도 됨)
  - 최대 1,000개까지의 입력이 들어올 수 있음

- 반환 값: 각 N_i에 대한 해를 담은 리스트
  - 입력 $N_1, N_2, .. N_k$ 에 대한 답이 순서대로 담겨 있어야 함

- 이번 시간에 제공한 코드 TotientMaximum.py의 totient**Mini**mum () 함수 내에 코드 작성해 제출

Copyright © by Sihyung Lee - All rights reserved.

입출력 예: 입력으로 들어온 각 N_i에 대해 2 ~ N_i에 속하는 정수 n 중 n / φ(n)을 최대로 만드는 최소의 n 값을 리스트에 담아 반환

```
print(totientMinimum(2,3,4,5,6,7,8,9,10,11))
```

```
[2, 3, 3, 5, 5, 7, 7, 7, 7, 11
```

```
print(totientMinimum(50, 500, 1000))
```

```
[47, 499, 997]
```

```
print(totientMinimum (5000,10000,50000))
```

```
[4999, 9973, 49999]
```

```
print(totientMinimum (100000,500000,1000000))
```

```
[99991, 499979, 999983]
```

그 외 예제는 __main__ 아래 테스트 코드를 참조하세요.

Copyright © by Sihyung Lee - All rights reserved.

그 외 프로그램 구현 조건

- 최종 결과물로 TotientMaximum.py 파일 하나만 제출하며, 이 파일만으로 코드가 동작해야 함
- import는 사용할 수 없음
- __main__ 아래의 코드는 작성한 함수가 올바름을 확인하는 코드로
- 코드 작성 후 스스로 채점해 보는데 활용하세요.
- 각 테스트 케이스에 대해 P(or Pass) 혹은 F(or Fail)이 출력됩니다.
- 코드를 제출할 때는 __main__ 아래 코드는 제거하거나 수정하지 말고 제출합니다. (채점에 사용되기 때문)
- **이번 시간 코드에 포함된 텍스트 파일들은 (testInput?.txt, testOutput?.txt) TotientMaximum.py와 같은 디렉토리에 두고 실행하세요. 채점에 사용되는 입출력을 담은 파일입니다.**
- **시간 테스트는 별도로 없지만, 정확도 테스트의 결과가 거의 즉시 나오지 않고 5초를 넘어간다면 fail로 봅니다.**

Copyright © by Sihyung Lee - All rights reserved.

이번 시간 제공 코드 함께 보기

- TotientMaximum.py

Copyright © by Sihyung Lee - All rights reserved.

실습 문제 풀이 & 질의 응답

- 종료 시간(17:00) 이전에 **일찍 모든 문제를 통과**한 경우 각자 **퇴실 가능**
- 실습 문제에 대한 코드는 lms 과제함에 **다음 날 23:59까지 제출** 가능합니다.
- 마감 시간까지 작성을 다 못한 경우는 (제출하지 않으면 0점이므로) 그때까지 작성한 코드를 꼭 제출해 부분점수를 받으세요.

- 실습 문제는 **개별 평가**입니다. 제출한 코드에 대한 유사도 검사를 실습 문제마다 진행하며, 코드를 건내 준 사례가 발견되면 0점 처리됩니다.
- 로직에 대해 서로 의견을 나누는 것은 괜찮지만, 코드를 직접 건내 주지는 마세요.
- 본인 실력 향상을 위해서도 **코드는 꼭 각자 직접 작성**해 주세요.

Copyright © by Sihyung Lee - All rights reserved.